University of Plymouth

PEARL **https://pearl.plymouth.ac.uk**

04 University of Plymouth Research Theses 01 Research Theses Main Collection

1995

Extension and hardware implementation of the comprehensive integrated security system concept

Morrissey, Joseph Patrick

http://hdl.handle.net/10026.1/336

http://dx.doi.org/10.24382/3619
University of Plymouth

All content in PEARL is protected by copyright law. Author manuscripts are made available in accordance with publisher policies. Please cite only the published version using the details provided on the item record or document. In the absence of an open licence (e.g. Creative Commons), permissions for further reuse of content should be sought from the publisher or author.

The Extension and Hardware Implementation of the Comprehensive Integrated Security System Concept

by

Joseph Patrick Morrissey

B.Eng. (Hons)

A thesis Submitted to the University of Plymouth
in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

School of Electronic, Communication and Electrical Engineering
Faculty of Technology

November 1995

COPYRIGHT STATEMENT

This copy of the thesis has been supplied on the condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the author's prior written consent.

LIBRARY STORE

The Extension and Hardware Implementation of the Comprehensive Integrated Security System Concept

Joseph Patrick Morrissey

B.Eng.(Hons)

The current strategy to computer networking is to increase the accessibility that legitimate users have to their respective systems and to distribute functionality. This creates a more efficient working environment, users may work from home, organisations can make better use of their computing power. Unfortunately, a side effect of opening up computer systems and placing them on potentially global networks is that they face increased threats from uncontrolled access points, and from eavesdroppers listening to the data communicated between systems. Along with these increased threats the traditional ones such as disgruntled employees, malicious software, and accidental damage must still be countered.

A comprehensive integrated security system (CISS) has been developed to provide security within the Open Systems Interconnection (OSI) and Open Distributed Processing (ODP) environments. The research described in this thesis investigates alternative methods for its implementation and its optimisation through partial implementation within hardware and software and the investigation of mechanisms to improve its security.

A new deployment strategy for CISS is described where functionality is divided amongst computing platforms of increasing capability within a security domain. Definitions are given of a: local security unit, that provides terminal security; local security servers that serve the local security units and domain management centres that provide security service co-ordination within a domain.

New hardware that provides RSA and DES functionality capable of being connected to Sun microsystems is detailed. The board can be used as a basic building block of CISS, providing fast cryptographic facilities, or in isolation for discrete cryptographic services. Software written for UNIX in C/C++ is described, which provides optimised security mechanisms on computer systems that do not have SBus connectivity.

A new identification/authentication mechanism is investigated that can be added to existing systems with the potential for extension into a real time supervision scenario. The mechanism uses keystroke analysis through the application of neural networks and genetic algorithms and has produced very encouraging results.

Finally, a new conceptual model for intrusion detection capable of dealing with real time and historical evaluation is discussed, which further enhances the CISS concept.

Contents

Abstract i

Contents ii

List of Figures vi

List of Tables viii

Acknowledgements x

Declaration xi

Glossary of Abbreviations xii

1. INTRODUCTION 1

1.1 Introduction 2

1.2 Aims and Objectives of the Research 5

1.3 Thesis Structure 6

2. INTRODUCTION TO CRYPTOGRAPHY 9

2.1 Introduction 10

2.2 Terms Used in Cryptography 10

2.3 Block and Stream Cryptosystems 11

2.4 Symmetric and Asymmetric Cryptosystems 15

2.5 Choosing a Cryptosystem 28

2.6 A Cryptosystem for CISS 32

2.7 Security Mechanisms Involving Cryptography 34

2.8 Key Management and Distribution 38

2.9 Conclusion 43

3. COMPREHENSIVE INTEGRATED SECURITY SYSTEM 44

3.1 Introduction 45

3.2 Threats, Services and Mechanisms 45

3.3 Conceptual Model of CISS 52

3.4 Security Management Information Base 55

3.5 The Ten Agents of CISS 59

3.6 Security Management Centres 64

4. 3-L ARCHITECTURE 68

4.1 Introduction 69

4.2 Three Level Architecture 70

4.3 Domain Management Centre 73

4.4 Local Security Server 82

4.5 Local Security Unit 87

4.6 Common Services 93

5. HARDWARE IMPLEMENTATION OF CISS SECURITY MECHANISMS 100

5.1 Introduction 101

5.2 Sun Interfaces 103

5.3 SBus 106

5.4 Interface Board 112

5.5 Processing Board 121

5.6 Hardware Implementation of the RSA Cryptosystem 125

5.7 Hardware Implementation of the DES Cryptosystem 139

6. SOFTWARE IMPLEMENTATION OF CISS SECURITY MECHANISMS 150

6.1 ENCDEC Device Driver 151

6.2 Long Integer 159

6.3 RSA 164

6.4 DES 169

6.5 Conclusion 171

7. USER AUTHENTICATION MECHANISM FOR A LOCAL SECURITY UNIT 172

7.1 Introduction 173

7.2 User Authentication 173

7.3 Overview of the Investigation 180

7.4 Application of Keystroke Authentication within CISS 206

7.5 Conclusion 207

8. MONITORING AGENT SPECIFICATION 209

8.1 Introduction 210

8.2 The Basic Model 216

8.3 Conceptual Monitoring Agent 220

8.4 Conclusion 240

9. CONCLUSION 241

9.1 Achievements of the Research Programme 242

9.2 Limitations of the Research 243

9.3 Suggestions and Scope for Future Work 244

References 245

Appendix A: DES and RSA 260

Appendix B: ENCDEC Board 272

Appendix C: Software 295

Appendix D: Papers 324

List of Figures

FIGURE 2-1 ELECTRONIC CODEBOOK 12

FIGURE 2-2 CIPHER BLOCK CHAINING 13

FIGURE 2-3 CIPHER FEEDBACK 14

FIGURE 2-4 OUTPUT FEEDBACK 14

FIGURE 2-1 SYMMETRIC CRYPTOSYSTEM 16

FIGURE 2-2 DES ALGORITHM 18

FIGURE 2-3 ASYMMETRIC CRYPTOSYSTEM 23

FIGURE 3-1 CISS CONCEPTUAL MODEL 52

FIGURE 3-2 TEN AGENT INTERACTION 60

FIGURE 3-3 SECURITY MANAGEMENT CENTRE 64

FIGURE 4-1 THREE LEVEL ARCHITECTURE 71

FIGURE 4-2 DMC TO DMC VIA TTP 76

FIGURE 4-3 X509 CA HIERARCHY 77

FIGURE 4-4 MULTI-LAYER SSID 79

FIGURE 4-5 NON-REPUDIATION ROUTES 85

FIGURE 4-6 SECURE DUMB TERMINAL 87

FIGURE 5-1 SYSTEM UNIT AND SBUS COMPONENT GAP 106

FIGURE 5-2 SBUS SYMMETRIC SYSTEM 107

FIGURE 5-3 L64853A 112

FIGURE 5-4 OVERVIEW OF INTERFACE BOARD 120

FIGURE 5-5 COMMAND WORD 123

FIGURE 5-6 ENCDEC BOARD OVERVIEW 124

FIGURE 5-7 PQR LOGICAL INTERFACE 127

FIGURE 5-8 PQR STATES 128

FIGURE 5-9 PQR INTERNAL ARCHITECTURE 132

FIGURE 5-10 DES CHIP 140

FIGURE 5-11 DES PIPELINING 146

FIGURE 5-12 ECB PROCESS 147

FIGURE 5-13 CBC PROCESS 149

FIGURE 6-1 DEVICE DRIVER OVERVIEW 152

FIGURE 6-2 MATHEMATICAL MODULE 159

FIGURE 6-3 FRAMING WITHIN A CRYPTOSYSTEM 165

FIGURE 7-1 TEXT ENTRY SCREEN SHOT 182

FIGURE 7-2 LINEARLY AND NON-LINEARLY SEPARABLE PROBLEM 184

FIGURE 7-3 BIOLOGICAL NEURON 185

FIGURE 7-4 MODEL NEURON 185

FIGURE 7-5 MULTI-LAYER PERCEPTRON MODEL 187

FIGURE 7-6 NEURAL NET LEARNING SCREEN SHOT 190

FIGURE 7-7 BASIC NETWORK REF 2 RESULTS 193

FIGURE 7-8 ROULETTE WHEEL 200

FIGURE 7-9 SCALING FUNCTION 201

FIGURE 7-10 CROSSOVER 202

FIGURE 7-11 SCREENSHOT OF GA TRAINING 203

FIGURE 8-1 STAGES OF INTRUSION DETECTION 215

FIGURE 8-2 MONITORING AGENT: BASIC MODEL 217

FIGURE 8-3 PROFILER 228

FIGURE 8-4 PROFILE MONITORING 230

FIGURE 8-5 ANALYSIS WITHIN CISS 237

List of Tables

TABLE 2-1 SYMMETRIC CRYPTOSYSTEMS 32

TABLE 2-2 ASYMMETRIC CRYPTOSYSTEMS 33

TABLE 3-1 SECURITY SERVICES 49

TABLE 3-2 SECURITY MECHANISMS 51

TABLE 3-3 SAMPLE SERVICE STRUCTURE 51

TABLE 4-1 AGENTS USED WITHIN 3-L 72

TABLE 4-2 3-L DIVISION OF LABOUR 72

TABLE 5-1 SBUS PHYSICAL DIMENSIONS 106

TABLE 5-2 MEMORY MAPPING FOR SBUS INTERFACE 122

TABLE 5-3 RSA HARDWARE 125

TABLE 5-4 ADDRESS TRIGGERED COMMANDS 128

TABLE 5-5 INTERNAL COMMANDS 129

TABLE 5-6 DES STATUS REGISTERS 141

TABLE 5-7 DES SINGLE COMMANDS 142

TABLE 5-8 DES CONTINUOUS COMMANDS 143

TABLE 5-9 DES REGISTER ADDRESS 147

TABLE 7-1 RATIO 1:1 RESULTS 194

TABLE 7-2 RATIO 4:1 RESULTS 194

TABLE 7-3 TWO REFERENCE RESULTS (REF1 AND REF2) 196

TABLE 7-4 TWO REFERENCE RESULTS (REF2 AND REF3) 196

TABLE 7-5 GA BASIC RESULTS 204

TABLE 7-6 GA EXP 1 RESULTS 205

TABLE 7-7 GA EXP1 DIGRAPHS 205

TABLE 7-8 GA EXP2 RESULTS 206

TABLE 7-9 GA EXP2 RESULTS 206

TABLE 7-10 GA EXP2 DIGRAPHS 206

TABLE 8-1 ACTIONS 224

TABLE 8-2 MONITORING MEASUREMENT 224

TABLE 8-3 SAMPLE SYSTEM RESOURCES 225

TABLE 8-4 SAMPLE ACTIVITIES 226

Acknowledgements

This thesis contains aspects of three years work within the Network Research Group in the School of Electronic Communication and Electrical Engineering, University of Plymouth, Plymouth, England.

I would like to thank the following.

- Peter Sanders, my Director of Studies, most importantly for finding me sponsorship and for all the subsequent support he has given me during my time within the Group.
- Dr Colin Stockel, my Supervisor, for his extreme patience during the writing of this thesis, for all the ideas and time he has given to me during the three years of research, and for all the FREE cups of tea.
- The newly appointed, Dr Steven Furnell, for his help and close collaboration.
- Other members of the Network Research Group.
- Finally, to my parents, that have encouraged and supported me over the past 25 years.

Declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other university award.

Relevant scientific seminars and conferences were regularly attended at which work was often presented; external institutions were visited for consultation purposes, and several papers prepared for publication.

The work presented in this thesis is solely that of the author.

Signed.....

Date.....12/2/96.....

Glossary of Abbreviations

AA	Association Agent
CA	Certification Authority
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CISS	Comprehensive Integrated Security System
DES	Data Encryption Standard
DMC	Domain Management Centre
ECB	Electronic Codebook
FAR	False Acceptance Rate
FRR	False Rejection Rate
GA	Genetic Algorithm
IB	Interface Board
IDEA	International Data Encryption Algorithm
ISO	International Standards Organisation
IV	Initial Vector
LAN	Local Area Network
LSIB	Local Security Information Base
LSS	Local Security Server
LSU	Local Security Unit
MA	Monitoring Agent
MAC	Message Authentication Code
MDC	Manipulation Detection Code

MMU	Memory Management Unit
MS-DOS	Microsoft Disk Operating System
OPENA	Operational Environment Agent
OS	Operating System
OSI	Open Systems Interconnection
OFB	Output Feedback
PB	Processing Board
PEM	Privacy Enhanced Mail
PGP	Pretty Good Privacy
PofD	Proof of Delivery
PofO	Proof of Origin
PofR	Proof of Receipt
PofS	Proof of Submission
RA	Recovery Agent
RSA	Rivest, Shamir, and Adleman
SAA	Security Administrator Agent
SMA	Security Mechanisms
SMC	Security Management Centre
SMIB	Security Management Information Base
SMIBA	Security Management Information Base Agent
SSA	Security Services Agent
SSid	Security Services Identification Certificate
TTP	Trusted Third Party
UA	User Agent

Chapter One

1. Introduction

This chapter gives a brief introduction to the need for security within open distributed environments. It also describes the aims and objectives of the research and gives a brief introduction to each chapter within this thesis.

1.1 Introduction

Civilisations have been built upon a single foundation - knowledge. The exploitation of natural resources, the management of workforces, the governing of economies, the creation of weapons, all require knowledge; knowledge of a process, knowledge of a method, knowledge of a thing. Within the twentieth century the possession of knowledge bequeaths more power than ever before and with its new found value it has acquired a new name - data. Data is this century's most valuable commodity; it has no material substance, residing in an ethereal realm of charge within the memory of computers. Computers, themselves a major influence on the shape of our society, must protect the resource that moulds our society. The increased connectivity which computers have attained within the past two decades and the wider availability of computing power have placed a potential weapon within reach of many members of society, and as technology improves this weapon is likely to be extended to an ever increasing proportion of society.

As we enter the final years of the twentieth century the changes that have taken place within our society, especially within the past three decades, were effected by the increasing exposure of the public to information technology. Data storage, manipulation and delivery have all become faster and easier for the user as technology has improved and demand has increased. More people are beginning to use the power that information technology can provide and equipment manufacturers have attempted to blur the technology that service the user, by providing easy interfaces through which it is possible for anybody to very quickly attain a basic level of competence and remove the illusion that computers were meant just for the 'techies.'

It is the ultimate goal of modern information interconnection strategies to provide global connectivity of people and computers. This goal may have appeared unobtainable a decade or so ago, but recent activity within the information technology community makes it appear that it may not just be possible, but now very probable.

The latest charge within the boundaries of information technology comes under the banner of the "information super highway," the term made famous by vice-president Al Gore of the USA [1], [2]. It is the "sound-byte" by which the programme to lead the United States data communication infrastructure into the next century is known. The infrastructure will comprise of an optical fibre network extending to the home of every citizen within the USA, so providing much increased capacity and greater accessibility for all. As with so many innovations from America it is now being adopted within the European Community.

As networks and the consequential connection of computers with their stored data becomes increasingly common through the introduction of standards such as the International Standards Organisation Open System Interconnection (ISO-OSI) reference model [3], the security of that data becomes ever more important. Current techniques in security have proved themselves to be ineffective when confronted by credible hackers [4], [5], [6]. As more and more people gain access to networks and the computers connected to them, as current policy within the EC and USA indicates they will, then there is likely to be more people tempted to pursue illegal activities. Many may wish to instigate malicious damage while others may only inadvertently cause damage. Whatever the reason, the action is illegal and security upon the computer systems must be capable of dealing with these attempts.

To combat the increased security risks that placing sensitive information upon open computer networks with ever increasing numbers of users introduces, new methods for security must be found. The model upon which this research is based is called the Comprehensive Integrated Security System[7], CISS. The ISO have been setting standards for open interconnection of computer systems over the last two decades and the CISS model attempts to provide the required security to meet the ISO recommendations [8], [9]. CISS is an adaptable security system capable of being added to existing computer systems to bring their current level of security up to an acceptable level dictated by the specific organisation.

Increasingly it is becoming easier for individuals to pick up the necessary skills to hack or cause malicious damage within computer systems. As development tools for business packages have appeared to help business, so to have development tools for virus writers. Which makes it quicker and easier to produce viruses that can subsequently be released to cause damage to a system's data. Recently an application 'SATAN' [10] has become available through the internet, ostensible for the use of security administrators, that can be used to detect weaknesses (such as access control and illegal assumption of privileges) in the set up of UNIX systems. As more of these applications are made available, there will be a proliferation of people with the necessary skills to effectively conduct attacks upon the computer systems, that are becoming ever more important to everyday life.

1.2 Aims and Objectives of the Research

The main aims of this research project were to further the CISS model by the investigation of suitable security mechanisms and the production of modules that would aid in the development of a CISS prototype. The specific objectives are:

1. the identification of weaknesses within the CISS model and the refinement of said model;
2. the optimisation of a CISS software prototype through implementation of functionality within hardware;
3. development of a new management strategy for CISS within a local area network and specification of terminal security;
4. research into, and the production of, new hardware for the development of a prototype CISS implementation;
5. the production of equivalent functionality that the objective 4 hardware provides in software for greater flexibility of CISS;
6. research and development into new mechanisms and concepts that will aid in the security of a local security unit.

1.3 Thesis Structure

This thesis describes the research conducted in the pursuit of the above aims and objectives.

Chapter 2 begins with a general description of cryptography and the modern strategies for its use that form the basis of all security upon modern computer systems. Without cryptography, security would not be feasible across a distributed environment as secure communication would not be practical. The theory and basis for modern security policies are described along with various security mechanisms which when co-ordinated within a coherent policy provide strong security.

Chapter 3 briefly describes the principles that lie behind the CISS security model. The need for security within an OSI environment is discussed along with how CISS can provide this in a practical manner. CISS is the basic security model that is developed within the proceeding chapters. It provides security through the correct ordering of security mechanisms and the use of secure protocols.

Chapter 4 describes a management policy for the implementation of CISS upon a local area network (LAN). LANs provide a good division between the relatively (physically) secured communication and a more hostile WAN where a communication medium can potentially be tampered with. Organisations usually view their area of responsibility extending as far as their own LAN. This chapter uses the mechanisms and concepts described in the previous two chapters to provide an overall coherent policy that is flexible enough for configuration in a heterogeneous environment, supporting diverse security policies. The management

uses a hierarchical structure formed from local security units (LSU), local security servers (LSS), and domain management centres (DMC).

Chapter 5 describes the hardware built for a CISS prototype that has been implemented upon a Sun workstation. The board is the first of its kind for Sun workstations, providing facilities for symmetric and asymmetric cryptographic algorithms whose use was described in chapter two. The development of the prototype board was accomplished in three stages. The use of this board will speed up a CISS implementation as described within chapter four and can be used in the provision of security mechanisms described in chapter two.

Chapter 6 describes the necessary software drivers supplied for use with the prototype board and a description of the software written to provide the necessary cryptographic algorithms, that can be used on systems without access to the hardware implementations. The software and hardware are fully compatible and ready for use within a prototype CISS. The software supplied as part of a mathematical module for CISS is also described.

Chapter 7 describes the investigation of a new technique for access control and user identification mechanisms that can be adapted for use by many computer systems. Its adaptability makes it very suitable for application within CISS, specifically the local security unit as described in chapter four. The new method analyses the keystroke characteristics of an individual from samples of static text strings, so that when presented with further keystroke samples they can be classified correctly for user authentication. The keystroke recognition has been accomplished through the use of neural networks and genetic algorithms for pattern recognition and data sorting respectively.

Chapter 8 expands the functionality of the monitoring agent within CISS. Most of the security mechanisms used by CISS to provide protection to the computer system on which it is implemented can be thought of as barriers. For example, user authentication blocks an individual's attempts to access a system. The monitoring agent is expanded to include system activity monitoring through appropriate auditing techniques and user profiling.

Finally, chapter 9 gives the main conclusions that can be gathered from the research. The main achievements are reviewed together with its limitations, and suggestions for further development of the CISS model and management architecture.

Chapter Two

2. Introduction to Cryptography

This chapter begins with an explanation of terms that are commonly used within the field of cryptography, then describes the two groups of cryptographic systems. The five most popular and best respected systems are summarised. The requirements of a cryptographic system are discussed with respect to implementation and application within CISS and two cryptosystems are chosen for implementation within CISS. Finally, a brief introduction to the use of cryptographic techniques within non-confidential security mechanisms is given.

2.1 Introduction

Cryptology is the science of secure communication in spoken, written, or increasingly, electronic form. The word Cryptology derives its meaning from the Greek words 'kryptos' meaning 'hidden' and 'logos' meaning 'word'. Cryptography is commonly associated with the 'classic' written communiqués of war [11]. More recently it is being used in less dramatic but more lucrative scenarios, such as banking, pay-television or mobile communications. Effective security within distributed systems is only possible through the use of cryptography. If it were necessary to secure distributed systems through tamper proof technology then it would be too expensive for most organisations to afford.

For a long time cryptographic techniques were solely used to ensure confidentiality of data, but with more importance being placed upon the electronic documents stored within computer memory, cryptographic techniques are being applied within proof mechanisms, to provide identification, authentication and integrity. As information technology becomes even more a part of everyday life the need for the implementation of strong cryptographic techniques within effective security procedures grows.

2.2 Terms Used in Cryptography

A *cryptographer* is a person who finds ways of ensuring the security of data whereas a *cryptanalyst* tries to obtain the meaning of data manipulated by the cryptographer. A *cryptosystem* is a way of ensuring data can be stored and transmitted confidentially. Data in an unsecured form is known as *plaintext* and the process of concealment is *encryption*; encrypted data is known as *ciphertext* and the process of revelation is *decryption*. Most

cryptosystems require knowledge by the user of a *key* which is vital to the correct encryption/decryption of the data. The raw effectiveness of how difficult a cryptosystem is to break by a cryptanalyst is determined by calculating the time it would take for every possible key to be used in decrypting a ciphertext. This form of attack is known as *brute force.* If a cryptanalyst is constrained to use a brute force attack the cryptographer has done his job well. For example, if a cryptosystem has a key length of 56 bits there are over 7×10^{16} possible keys, each decryption and analysis of the result for a given ciphertext might take a microsecond, therefore, to test all keys would take around 2000 years.

2.3 Block and Stream Cryptosystems

There are two ways in which cryptosystems can be used in communication, as *block cryptosystems* or as *stream cryptosystems.* A block cryptosystem takes as its input a block of plaintext which it replaces with a block of encrypted data. Block cryptosystems are defined by the finite size of the plaintexts which they can encrypt and their use of buffers to store data. Stream cryptosystems work on streams of data such as voice or video data where encryption must be done 'on the fly,' generally encrypting/decrypting a bit at a time. It is possible to convert a block cryptosystem into a stream cryptosystem, but, it is not possible to convert a stream cryptosystem into a block cryptosystem due to the time dependency of a stream cipher.

2.3.1 Block Cryptosystem Modes of Use

Block cryptosystems are generally used in one of four ways [12]: electronic codebook(ECB); cipher block chaining(CBC); cipher feedback(CFB); and output feedback (OFB).

2.3.1.1 Electronic Codebook

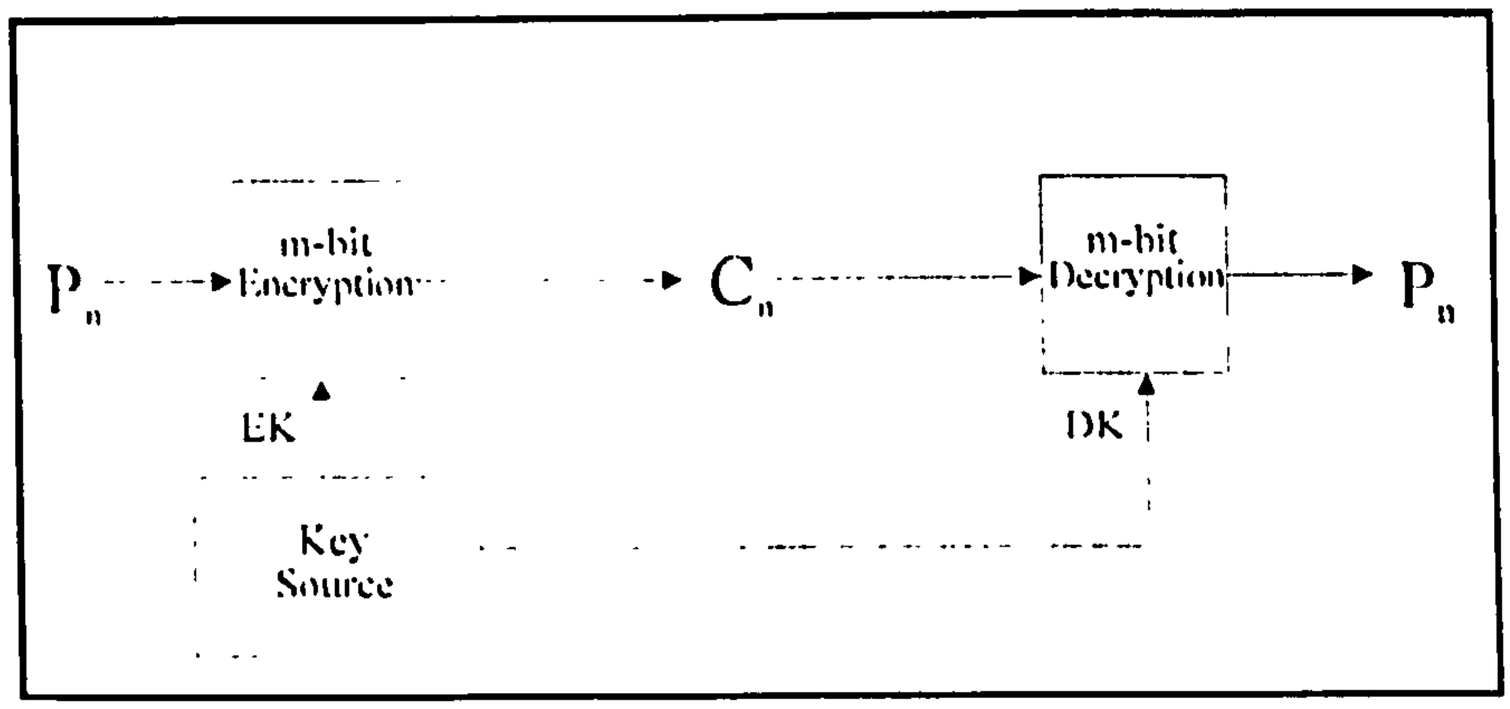


Figure 2-1 Electronic Codebook

The Electronic Codebook (ECB) mode of operation produces one-to-one mapping of the m-bit blocks, where m is the size of a block that the cryptosystem can process. Any errors occurring in an m-bit block during storage/transmission is confined to that block. When the plaintext is not a multiple of m bits in length it is necessary to include padding before encryption, which leads to an expansion of the ciphertext. This padding necessitates framing information to be sent in the ciphertext.

When a cryptosystem is used in ECB mode any message that has a regular pattern throughout its length, such as a header that appears in the first line of every page, may exhibit patterns in its ciphertext. A cryptanalyst could then potentially use these patterns to attack the ciphertext, the Cipher Block Chaining mode of operation avoids this problem.

2.3.1.2 Cipher Block Chaining

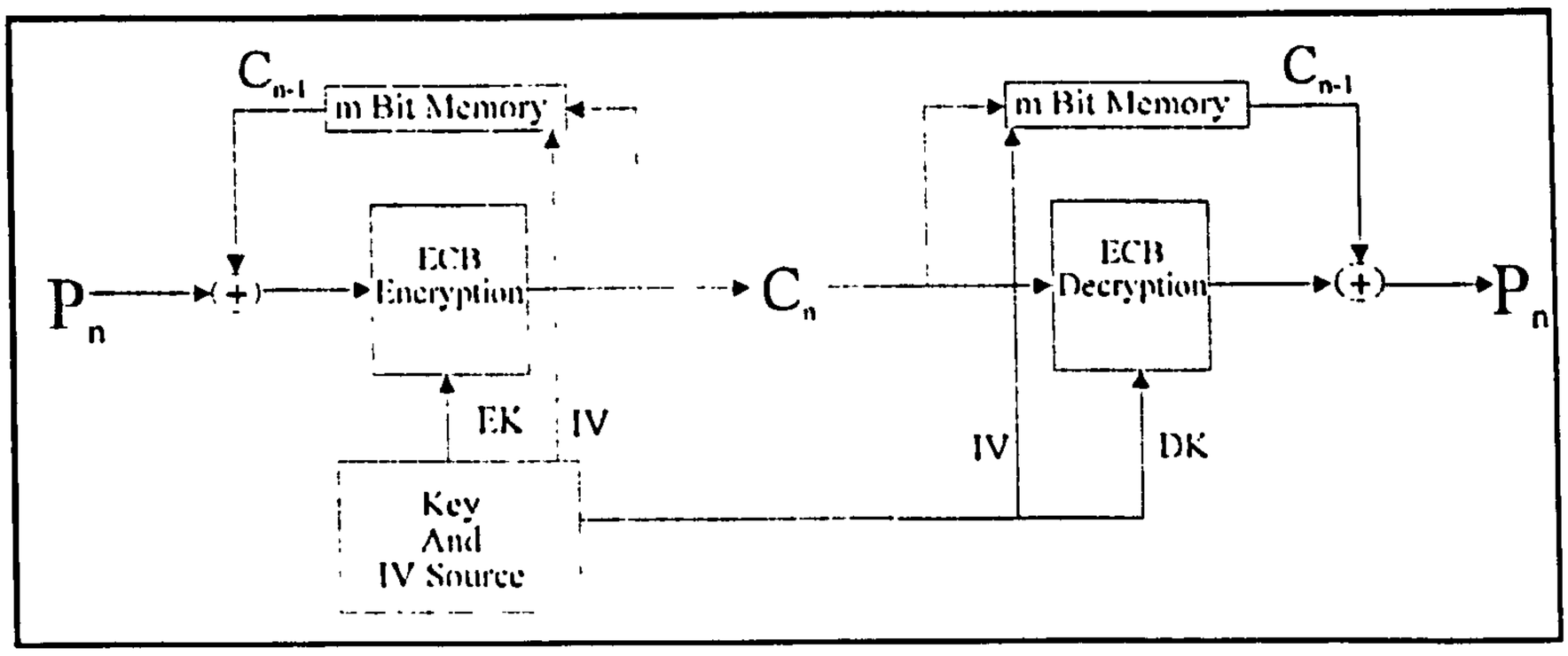


Figure 2-2 Cipher Block Chaining

Figure 2-2 shows how to use a block symmetric cryptosystem in CBC mode. All but the first m-bit block of plaintext is exclusive ORed with the previous m-bit block of cipher text. Since there is no previous block of cipher text for the first block of plaintext an *Initial Vector* (IV) is chosen and held within the m-bit memory. The initial vector is a random m-bit number and must be communicated along with the decryption key (DK) to all parties wishing to decrypt the ciphertext. Decryption of a ciphertext encrypted using CBC is accomplished through decryption of the ciphertext and XORing the result with the previous ciphertext.

Because all following blocks of ciphertext are dependent upon previous blocks of ciphertext, any error occurring in the ciphertext during communication will cause all following blocks to be decrypted incorrectly.

2.3.1.3 Cipher Feedback and Output Feedback

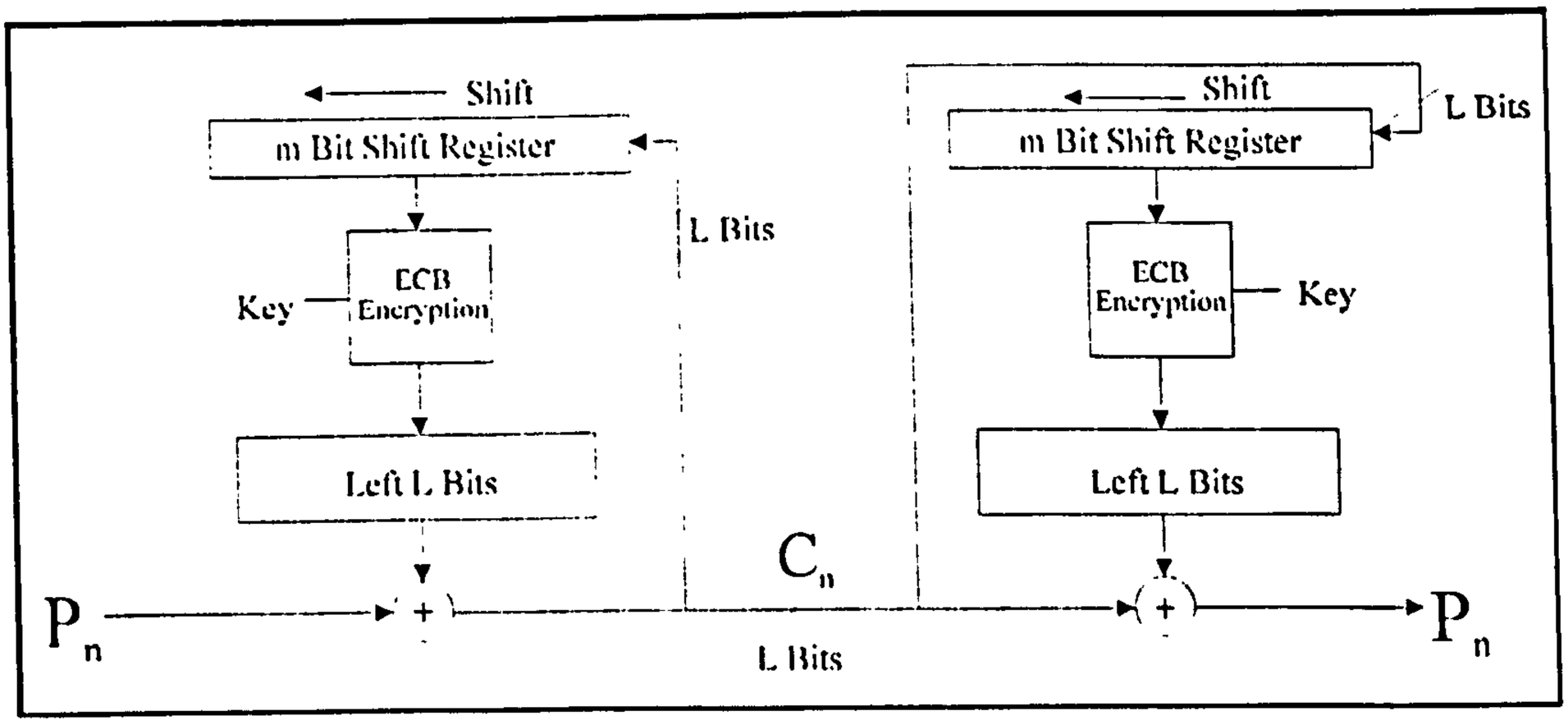


Figure 2-3 Cipher Feedback

Cipher feedback can be used as a stream cipher. Before encipherment L (the number of bits to be fed back) and an initial vector of m bits must be chosen. Like CBC any error occurring in the cipher text during transmission will cause all following packets to be decrypted incorrectly.

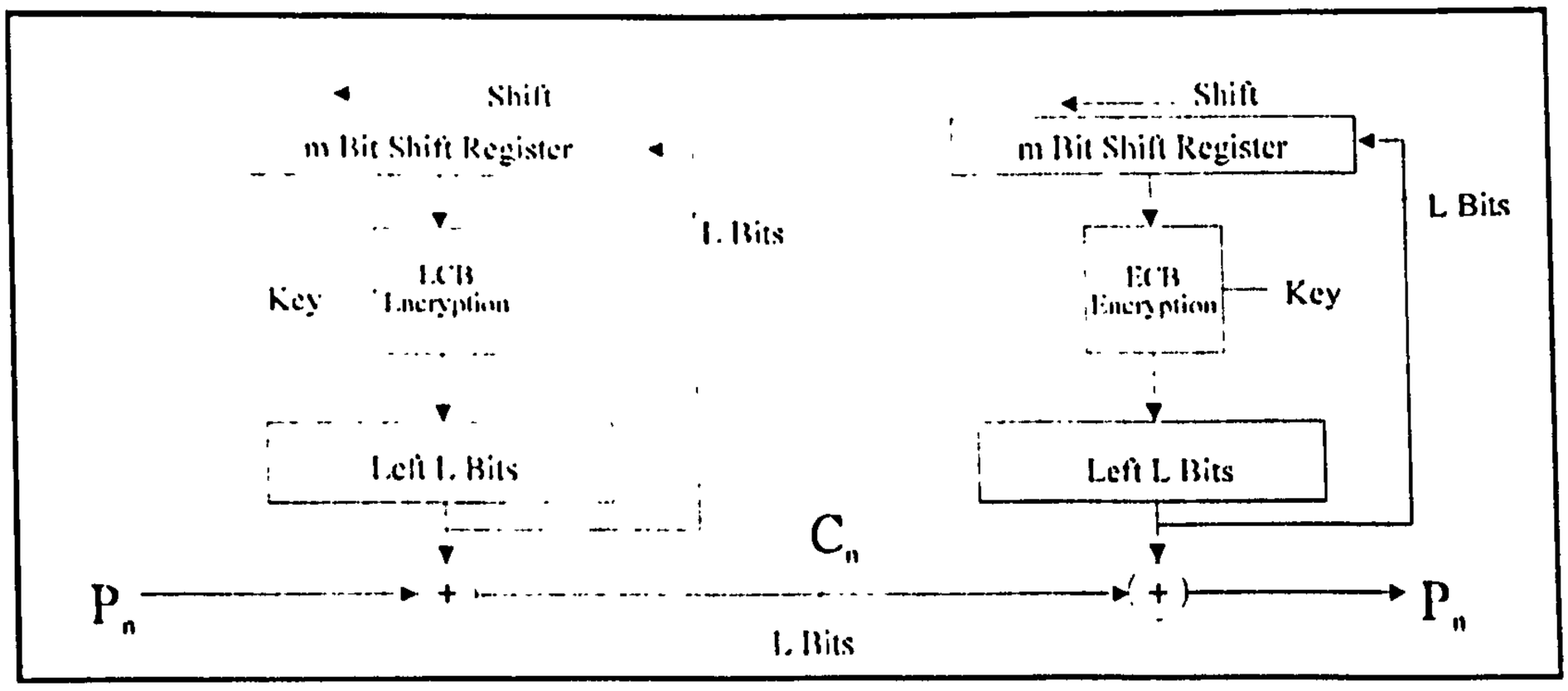


Figure 2-4 Output Feedback

The fourth mode of operation involves using the symmetric block cryptosystem as a pseudo-random number generator. This mode is useful for video or audio communication

where small errors in the data are acceptable and like ECB any error occurring in the stream of cipher text is not perpetuated throughout the following decryption, as all blocks of m bits in OFB are independent.

2.4 Symmetric and Asymmetric Cryptosystems

In 1949 Claude Shannon released a crucial paper in modern cryptography, "Communication theory of secrecy systems," [13]. In it he placed all cryptosystems in two groups; those cryptosystems whose strength depend upon the inadequacy of computational ability at any given time - *computationally secure* systems, or those that no matter how much computing power is available cannot be broken - *unconditionally secure* systems. All modern cryptosystems are computationally secure, the 'one-time pad' cryptosystem [14], is the only known unconditionally secure system

Most modern cryptosystems can be further divided into one of two groups, *symmetric* or *asymmetric*. A symmetric cryptosystem requires only one key. This key is used in the encryption and decryption calculations giving no differentiation between the encrypter and the decrypter. In asymmetric cryptosystems the encryption and decryption calculations use separate but related keys, thus differentiating between the encrypter and decrypter.

2.4.1 Symmetric Cryptosystems

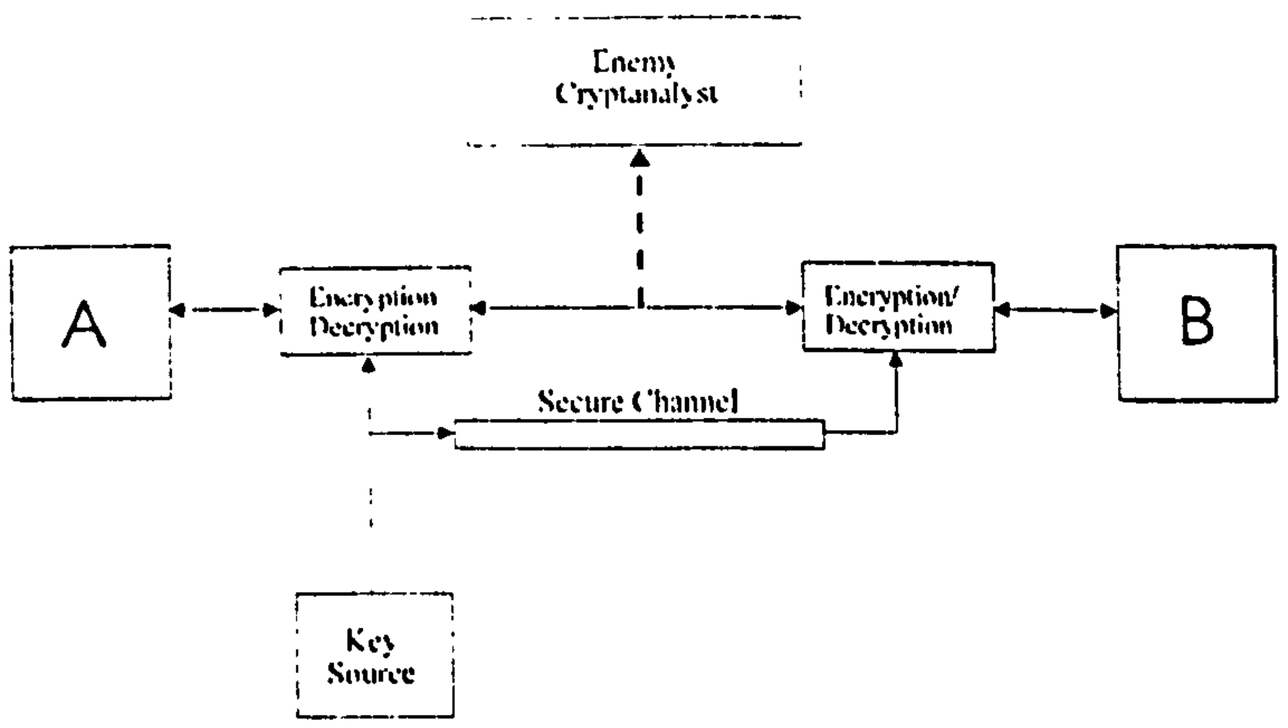


Figure 2-1 Symmetric Cryptosystem

If a party A wishes to communicate with a second party B, confidentially, using a symmetric cryptosystem, then both A and B must know the key that is in use. Once the key is known to both parties confidential duplex communication can be performed. The most immediate problem with the use of a symmetric cryptosystem is how to communicate the key to be used confidentially between the two parties. This leads to the necessity of having a 'secure channel' along which keys can be communicated secretly. The necessity of having a secure channel gives this system its second name, a *secret key cryptosystem*.

The necessary existence of a secure channel begs the question, why not send the data across it? The secure channel and its provision led to research into a second class of cryptosystem which does not require the channel - asymmetric cryptosystems, see section 2.4.2.

The next three sections detail the symmetric cryptosystems considered for use within CISS.

2.4.1.1 DES

In 1974 the US NBS(National Bureau of Standards) sent out a call for cryptosystems to be used as a Data Encryption Standard (DES) and a cryptosystem submitted by IBM was accepted [15]. DES is itself based upon a cryptosystem developed by IBM prior to 1974, their Lucifer cipher [16]. The Lucifer cipher had a key length of 128 bits, however, DES has a key length of only 56 bits. In 1977 a Senate select committee ascertained that the NSA (National Security Agency) was instrumental in the reduction of the DES key length (previously denied by IBM and NBS). The NSA has also classified the design principles that IBM used for part of the cryptosystem.

Shannon[13] provided suggestions to guide the design of new cryptosystems, primarily the principles of *diffusion* and *confusion.* Diffusion removes the statistical nature of the plaintext from the ciphertext. Confusion reduces the ability to draw statistical information between the ciphertext and the plaintext. DES attains diffusion through transposition and confusion is attained through substitution. When transposition and substitution are used in combination the new cryptosystem is referred to as a product cryptosystem.

In its native mode of operation (ECB) the DES cryptosystem takes a 64-bit input and gives a 64-bit output that is dependent upon a 64 bit key, of which, only 56 bits are used and the remaining eight make up an odd parity check. DES can be used in any of the modes described in section 2.3.1. DES is a product cipher that also incorporates bit-by-bit modulo 2 addition. The DES algorithm is shown below in Figure 2-2.

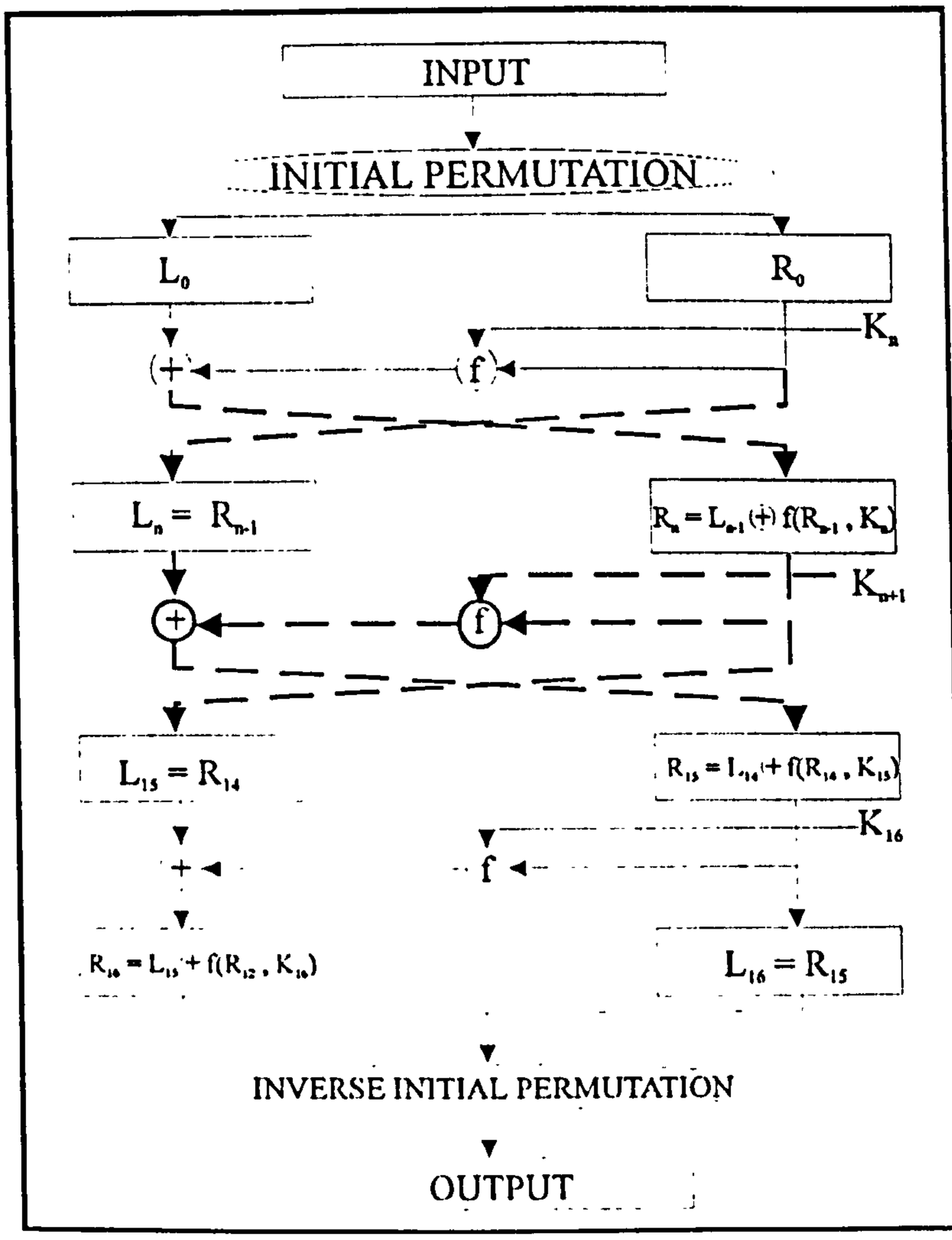


Figure 2-2 DES Algorithm

DES requires an input block of 64 bits, which undergoes an *initial permutation*, *IP*. The input block is then split into two blocks of 32 bits, the left 32 bits (L_n) and the right 32 bits (R_n) . Blocks L_n and R_n undergo 16 complex key dependent computations, known as the 16 rounds of DES, before going through an inverse of the initial permutation *IP-1*.

DES's designers deserve a lot of credit, even with all the suspicion aimed at the cryptosystem (originating from the fact that it is government approved) nobody has found a back door in the DES design, and nobody has been able to devise a strategy much better than a brute force attack to obtain the secret key.

The main source of concern about DES is its relatively small key. The DES key is 56 bits long giving 2^{56} possible keys, because of the speed and cost of DES hardware the production of a machine that breaks normal DES through brute force would cost $10million dollars[17]. However, DES can be significantly improved through techniques such as double or triple DES which gives an effective key length of 112 bits making it substantially more expensive to crack.

It is common knowledge that there are certain keys that if used reduce the security of DES, there are however only 4 weak keys and 60 semi-weak keys[14]. The weak keys of DES produce identical encryption and decryption algorithms, i.e. a ciphertext produced using a weak key can be deciphered by encrypting it again. The semi-weak keys are a group of key pairs that encrypt a plaintext into the same ciphertext, i.e. one key in a key pair can decrypt a ciphertext generated using the other. It should be noted that there are only 64 compromising keys out of 2^{56} possible keys.

The DES cryptosystem is probably the most widely used cipher at present, protecting the confidentiality of transactions involving trillions of dollars each year, as recommended by the American Bankers Association[18]. The technical details of the DES algorithm were released in 1977, some 18 years later it is still a very secure cryptosystem, despite the numerous attempts to break it [19].

2.4.1.2 Escrow Technology

In the 1990s the United States NSA developed a new block cryptosystem and set cryptographic policy that will have far reaching consequences to cryptography. The Clipper chip, as the cryptographic chip designed by the NSA has become known, employs the secret 'Skipjack' algorithm[20]. The government has proposed the use of this chip in communication devices used to deal with the government.

The Clipper chip takes a 64-bit input block and transforms it into a 64 bit output block, the transformation being dependent upon an 80-bit key. The Skipjack algorithm was designed by NSA cryptographers, and unlike many other modern cryptosystems, NSA have not allowed the algorithm to be made public; citing that the design techniques employed in the Skipjack algorithm are representative of algorithms used to protect classified data. However, NSA and chosen experts from the field of cryptography [21], have evaluated the cryptosystem and reported that it provides a high level of security. One consequence of the secrecy of the Skipjack algorithm is that only certain companies are allowed to produce Clipper chips, thus setting up a monopoly.

Annoying as it may be to the cryptographic community that the Skipjack algorithm is kept secret, it is not the main flaw that many find with the Clipper chip. Every Clipper chip that is produced will have its own device unique 80-bit key [22]. This 80-bit key is divided into two 40-bit keys and stored separately with two escrow agents, giving law enforcement agencies the capability of listening in on any communications using the Clipper chip. To obtain the device's unique keys law enforcement agencies must first get a court order. The use of Clipper chips are at this moment voluntary but because of the amount of business the

US government generates they could mandate Clipper when dealing with them, effectively making it a standard. The government has proved its intentions of using the Clipper chip by ordering 50 000 Clipper equipped devices.

As with all cryptographic technology within the USA its exportation is subject to the Defence Trade Regulations [23], [24] and as a consequence it is at the moment illegal to export Clipper chips.

2.4.1.3 International Data Encryption Algorithm (IDEA)

This cryptosystem is similar to DES in that it is a symmetric block cryptosystem. IDEA [25] takes a 64-bit block and outputs a 64-bit data block, the result depends upon the 128-bit key used. Unlike DES which uses substitution and transposition, IDEA uses a mixture of three algebraic groups whose operations are mixed:

1. exclusive ORs;
2. addition modulo 2^{16}(ignoring any overflow);
3. multiplication modulo $2^{16}+1$(ignoring overflow).

These groups were chosen so that when implemented in hardware or software they would be very fast. Implementations of the IDEA algorithm tend to run faster than DES in hardware and software [26]. For example, on a 33-Mhz 386, IDEA encrypts at a speed of 880Kbits/s, while an equivalent DES program attained encryption speeds of 584Kbits/s. CRYPTECH (a security manufacturing company from Belgium) produce cryptographic chips including a DES chip that runs at 10Mbits/s. Although DES uses functions that are

optimised in hardware, IDEA was specifically optimised for implementation on current technology so it is capable of running at 880Kbits/s in software and at 177Mbits/s in hardware, which is considerably faster than DES.

The IDEA algorithm can be used in any of the modes that DES can be operated in, such as CBC, ECB CFB and OFB. The algorithm has been patented [27] by Ascom-Tech in Europe and has a patent pending in America.

Because of the newness of IDEA there have been few accounts of attacks against it and no accounts of successful ones. It has been proved that it is resistant to differential cryptanalysis[28], which is one of the more successful cyptanalytic attacks devised. Differential cryptanalysis exploits differential characteristics in ciphertext from known differences in the plaintexts to probabilistically determine possible keys.

The relative newness of IDEA has meant it has found limited application, although it can be found within security applications such as Pretty Good Privacy (PGP) [29].

2.4.2 Asymmetric Cryptosystems

In 1976 a paper crucial to modern cryptography was published by Diffie and Hellman, entitled "New Directions In Cryptography," [30] in which they suggested a way to set up secure systems that require no communication of a secret key. Though Diffie and Hellman went some way to showing that such a scheme was possible they did not present a workable solution, that was finally solved inl 1978 by Rivest, Shamir and Adleman as discussed in Section 2.4.3.

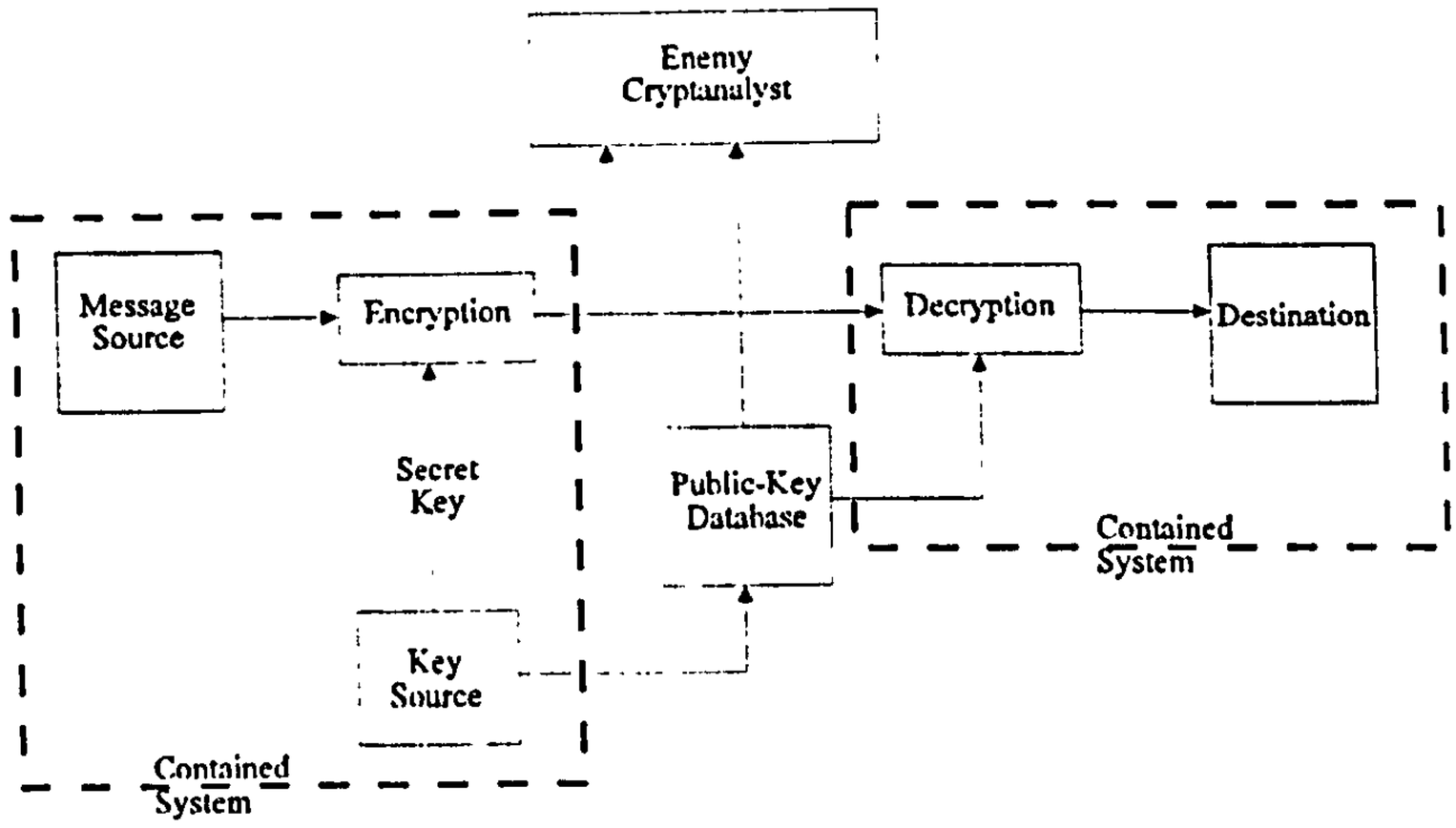


Figure 2-3 Asymmetric Cryptosystem

Asymmetric cryptosystems require two keys: a private key and a public one. The public key may be made available to anyone without reducing the security of the cryptosystem. These cryptographic systems are also known as *public key cryptosystems*. For a public key cryptosystem to be acceptable, computation of the private key from the public one should take enough time that an attempt would take longer than the useful life time of the information encrypted.

If party A wishes to send party B a message confidentially using a public key cryptosystem, party A would obtain party B's public key either from a database or through a request to B, and use the public key to encrypt the plaintext. Only party B should have the related private key and therefore only party B should be able to decrypt the ciphertext. To obtain duplex confidential communication between A and B each party must know two keys, their own private key and the other party's public key.

The public-key cryptosystems considered for use within CISS are detailed in the following sections.

2.4.3 RSA

In 1978 Rivest, Shamir and Adleman published "A Method For Obtaining Digital Signatures And Public-Key Cryptography," [31] in which they presented the first proposal for a one-way trapdoor function. This is now known as RSA (Rivest, Shamir, Adleman).

The seven steps of the RSA algorithm are:

1. generate two large prime numbers (p, q);
2. compute the modulus (n) = p * q;
3. compute $\phi(n) = (p-1)*(q-1)$;
4. select an odd integer e that is relatively prime to $\phi(n)$;
5. compute d as the multiplicative inverse of e, modulo $\phi(n)$;
6. make the Public key known, P = (e, n);
7. keep the Private key, S = (d, n).

The first part of the RSA algorithm requires the generation of two large prime numbers *p* and *q*, which can be accomplished by using Fermat's Little Theorem[32] to repeatedly test random numbers for compositness. If a random number passes the test several times to different bases, it is very probably prime.

The two large primes p and q are then multiplied together to give the modulus n that is then used in all future RSA calculations using this key. Euler's totient $\phi(n)$ [32] is then calculated, using the prime factors of the modulus n, resulting in the totient being equal to (p-1)(q-1). Using Euclid's algorithm, see Chapter 6, a random integer e is selected so that it is co-prime with $\phi(n)$. Euclid's extended algorithm, see Chapter 6, is used to determine d, the modulo inverse of e modulo $\phi(n)$. The key set (e, n) is published while (d, n) is kept secret.

Confidential communication is performed using the public-key of the recipient.

$$C = m^e \pmod{n}$$

The recipient decrypts the ciphertext C using his private key.

$$m = C^d \pmod{n}$$

At the moment the only strategy for breaking the RSA cryptosystem is to be able to factorise the known modulus into the two prime factors that were used in the creation of the key. The security of the RSA cryptosystem therefore depends upon the difficulty in factoring large integers for its security. The best and quickest methods for accomplishing long integer factorisation still work in exponential time when presented with integers larger than 500 bits in length. For example, a 664-bit number takes 10^{23} [33] steps to factorise, so if a theoretical computer does a million steps per second, then using a million such computers it will take four thousand years to factorise.

RSA has been adopted by the cryptographic community as the unofficial public-key crpytosystem standard [14], its official endorsements include the ISO specification for

digital signatures[34] along with the French Banking Community[35]. RSA is also found within PGP, various programs that implement Privacy Enhanced Mail[36] (PEM), and other security products.

2.4.4 ElGamal

ElGamal is a public key cryptosystem [37] that utilises the key distribution scheme developed by Diffie and Hellman [38]. The Diffie and Hellman key distribution scheme depends upon the function,

$$y = \alpha^{r} \bmod p$$

For example:

A and B wish to share a secret K_{AB}. A has a private key *Sa* and B has a private key *Sb*.

Two constants are known by both parties α and p.

A computes, $y_A = \alpha^{Sa} \bmod p$

B computes, $y_B = \alpha^{Sb} \bmod p$

Calculating *Sa* or *Sb* from knowing α, p, y_A and y_B is a discrete logarithm problem and as is the factorisation of long integers it is a computationally difficult problem to solve, requiring exponential time for solutions.

y_A is communicated to B.

y_B is communicated to A.

Both A and B can now calculate K_{AB}.

$$K_{AB} = \alpha^{SaSb} \bmod p$$

$$= y_A^{Sb} \bmod p$$

$$= y_B^{Sa} \bmod p$$

y_A and y_B can be kept in a database along with α and p.

K_{AB} can be used as a key for a secure communication session.

The ElGamal cryptosystem works in the following manner. If party A wishes to send a confidential message M to party B. A first obtains B's public key y and the constants α and p. Party A then calculates a secret K which is relatively prime to p-1. A then generates and sends to B a ciphertext with two distinct parts, C1 and C2.

$$C1 = \alpha^k \bmod p$$

$$C2 = y^k M \bmod p$$

B can reproduce the plaintext M through using his private key Sb.

$$M = C2/C1^{Sb} \bmod p$$

To attain the same level of security as RSA achieves it is necessary to use a public key file and ciphertext twice the size of that used in RSA, the reason being that the algorithm itself requires twice the information. The ElGamal public key cryptosystem is as yet un-patented. However, it is possible that use of ElGamal would violate Diffie-Hellman key exchange patents in the USA and Canada.

The National Institute of Standards and Technology (NIST) have adopted a variant of ElGamal in its Digital Signature Standard[39] (DSA) demonstrating the cryptographic community's faith that has been placed in the security of this algorithm.

2.5 Choosing a Cryptosystem

Selection of the best cryptosystem for a security system is a matter of compromise between several characteristics, some of which are detailed below.

2.5.1 Security

The security of a cryptosystem cannot be calculated accurately in any mathematical form and therefore security derived from cryptosystems is dependent upon belief. Belief that a cryptographic algorithm is secure grows with the continued appearance of failed attempts to break it. Therefore, cryptosystems that stand the test of time can be regarded as having proven security, as well as can be accomplished without mathematical proofs.

2.5.2 Cost

A primary concern for most potential users of a cryptosystem is how much the technology will cost them and the cost of a cryptosystem is influenced by two things:

1. patent;
2. and form of implementation.

Most cryptographers that develop a cryptosystem do so because they are interested in the lucrative patent laws and subsequent royalties a successful cryptosystem will bring them.

Use of a cryptosystem in a country where it has been patented requires payment of royalties to those people holding the patent, though payment is negotiable and is sometimes dispensed with. An active patent makes the cost of a security system using the cryptographic technique more expensive.

It is a general case that implementations of cryptosystems in software are cheaper than purchasing a hardware implementation. Development of a cryptosystem in hardware requires greater effort on behalf of the manufacturer who subsequently charges more for his product. For a hardware implementation of a cryptographic algorithm it must first be analysed and optimised for hardware, designs must be drawn up, masks for chip manufacture made and finally chips must be manufactured. The entire process requires an initial input of capital usually upwards of six figures. However, a single person or team of programmers can produce coding optimised for current computer systems at a cost far below that necessary for hardware development.

One of the hidden costs of any system is that of maintenance and upgradeability. Because hardware is specifically designed to take the load off the computing platform it was designed for, any improvements in the technology of the computing platform will not be felt in functions where cryptographic hardware is used. Therefore, a security communication system utilising hardware for cryptographic processes must purchase new hardware if it wishes to stay abreast of improvements in technology and retain competitiveness with rivals. However, in cryptosystems implemented within software this is not the case. It has become common practice for software developers to offer revised versions of software to be provided at a greatly reduced price when technology has improved. Even if new versions of

the software were not provided, improvements in the speed of computers for instance would improve the performance of software cryptosystems.

2.5.3 Speed

Speed of a cryptosystem depends upon the algorithm it employs and the technology used in its construction. As a rule symmetric algorithms tend to be simpler than asymmetric ones to implement in digital electronics, using operations based around byte word sizes, bit-shifting and logical operations that are continually optimised. Asymmetric cryptosystems, such as those described in section 2.4.2, require long integer modular exponential calculations which are notoriously slow in digital electronics due to the large number of operations required. A modular exponentiation using repeated squaring [40] (as described in chapter six) requires approximately $O(\beta^3)$ steps, where β is the bit length of the integers used within the calculation. Due to the increase in speed of technology and specialist algorithms[41] for factorisation a secure public-key length should be greater than 400 bits, giving approximately 64 million steps. Symmetric algorithms, such as those described in section 2.4.1, require a constant number of steps significantly less than that required in the long integer calculations of asymmetric systems. The smaller number of steps in symmetric cryptosystems makes them faster than the asymmetric cryptosystems.

2.5.4 Published and Unpublished Algorithms

When a cryptosystem has been designed its creators must decide whether or not to release details of the cryptosystem's algorithm. If a cryptosystem derives its security from the secrecy of its algorithm then details of the algorithm must be kept confidential. However,

this is difficult as confidentiality must be maintained by the developers, the engineers that design the implementations (hardware/software), and the manufacturers. Once the device is sold (whether as hardware or software) it must keep its design secret against attempts of reverse engineering. Therefore, a cryptosystem that derives its security from secrecy of its algorithm has many points at which it may be attacked. However, if the security of a cryptosystem is not dependent upon its algorithm's secrecy all these potential weaknesses have been eliminated.

A cryptosystem that can have details of its algorithm released to the public allows potential users/buyers to:

1. analyse the algorithm and test its security;
2. check that 'trapdoors' are not present allowing third parties access to confidential data.

The longer a cryptosystem survives such public scrutiny the greater the belief in its security and therefore the more confident potential buyers are to use and buy such a cryptosystem. The augmentation of public confidence by publishing algorithms has made it more common for cryptographers to release details of their cryptosystems to the public.

2.5.5 Asymmetric or Symmetric cryptosystems

Symmetric cryptosystems require the use of secure channels to communicate session keys between parties. However, asymmetric systems do not require one, thus eliminating a potential weak point for a cryptanalyst to exploit. Therefore, using symmetric cryptosystems in a distributed communications environment involves greater care over key management, and a subsequent increase in the size and complexity of key exchange

protocols. Because asymmetric cryptosystems use long integer arithmetic the necessary hardware designs are more complex and subsequently producing more expensive hardware devices. Symmetric systems require simpler designs making them cheaper.

2.6 A Cryptosystem for CISS

Increasingly, security policies have been relying upon the use of both an asymmetric and a symmetric cryptosystem. The symmetric cryptosystem is used for the bulk of data encryption for communication, such as files or messages. The asymmetric algorithm is used in key distribution and proof mechanism which are described in sections 2.7 and 2.8.

Symmetric Cryptosystem	DES	IDEA	SkipJack
Key Length (bits)	56	128	80
Block Length (bits)	64	64	64
Year of Origin	1974	1992	1990
Algorithm	Published	Published	Un-Published
Hardware Encryption Speeds	10Mbits/sec	177Mbits/sec	15Mbits/sec
Software Encryption Speeds on 386	584Kbits/sec	880Kbits/sec	NA
Security	Known Weak keys. size of key small.	Suspect keys found. Analysis limited	Escrow agencies hold device keys. Analysis limited

Table 2-1 Symmetric Cryptosystems

Asymmetric Cryptosystem	RSA	ElGamal
Public key	e, n	α, p, y
Cipher text	c	c1, c2
Year of Origin	1978	1985
Algorithm	Published	Published
Hardware (CRYPTECH)	32Kbits/s	<32Kbits/s
Software 386	6Kbits/s	<6Kbits/s
Security	Dependent on long integer factorisation	Dependent upon discrete logarithms

Table 2-2 Asymmetric Cryptosystems

Based upon sections 2.4 and 2.5 DES and RSA best fit the needs for CISS.

1. Both DES and RSA have been around for sufficient time that the belief in their security is almost universal.
2. The cost of implementing DES in hardware is relatively low, in part due to the age of the algorithm and its large continuing demand.
3. Although there is a patent on the RSA algorithm, it is only valid in the USA and therefore development of a security system using RSA outside of the USA will not require the payment of royalties.
4. The current research into DES is revealing new methods of using it that give it greater security [42], [43] and reduce the insecurity present in such a small key.
5. RSA requires less memory for storage of public keys and cipher texts than ElGamal.

The other popular cryptosystems were discounted because:

1. ElGamal requires larger public keyfiles for the same relative security as RSA. Also, to send a plaintext securely ElGamal expands the ciphertext to twice its original size whereas no expansion is necessary in RSA. Decryption in ElGamal requires an extra division step than RSA;

2. IDEA is too new. Analysis of the algorithm is nowhere near as extensive as that of DES. When the new algorithm has been thoroughly analysed it is very promising as a replacement for DES. It is patented[27] in Europe and therefore may be expensive to use commercially;
3. escrow technology is not a viable option, not only can it not be purchased outside of the United States but letting the government hold keys to every Clipper chip may not be accepted by companies purchasing CISS.

2.7 Security Mechanisms Involving Cryptography

In communication there are several main security services other than confidentiality that cryptography helps to provide within an overall security policy:

1. authentication;
2. integrity;
3. and non-repudiation.

2.7.1 Hashing Algorithms

A hashing algorithm produces a finite length bit string of set size from a potentially infinite one.

$h = \mathrm{H}(M)$, where M may be of any length but h is of length n dependent upon the hashing algorithm.

A good hashing algorithm has three characteristics: it is hard to calculate M given h; it is easy to calculate h given M; and it is difficult to calculate another message that gives the

same value *h* as H(*M*). One thing that should be noted about hashing algorithms is that they suffer from multiple mappings. Since the input message may be infinite, and the output is only beneficial if it is of a relatively small length, more than one input will achieve any specific result. Therefore, the finite length output should be within a range great enough that generation of any specific result from the hashing algorithm takes more effort than the benefits of obtaining it.

Hashing functions are generally used within the provision of authentication and data integrity, they are particularly useful when used in conjunction with public-key cryptography to supply digital signatures, see section 2.7.2. Examples of one way hash functions are MD5[44] and the Secure Hash Algorithm (SHA)[45].

A hashing algorithm that is key dependent is known as a message authentication code (MAC) and can be constructed through using DES or any other block cryptosystem in CFB mode. The exact method is specified in ANSI X9.9[46]. This is useful for providing fast authentication mechanisms. For authentication to take place both the sender and receiver must be in possession of the secret key used by a MAC.

Integrity of a message can be determined by use of a manipulation detection code (MDC), which is a hashing algorithm used to provide integrity. It can be used independently of authentication and confidentiality, where data sensitive to manipulation but not necessarily secret is being communicated.

2.7.2 Digital Signatures

Digital signatures[47] are used in the provision of authentication. A digital signature is an item of data that can be attached to a digital message to provide an indication of who the message creator was. It is analogous to a normal signature on a paper document and is meant to be used in the same way.

Most of the public-key cryptosystems were developed in the pursuit of digital signature algorithms. For example, RSA and ElGamal were first proposed as algorithms for digital signatures. There is no confidentiality required, nor is there a secret key shared between the two parties as is necessary in the case of a MAC. A typical sequence involving digital signatures can be seen below.

- Party A generates public key pair AP and AS, and publishes AP;
- Party A generates a document (doc) that it wishes to prove to B has been generated by A through use of a signature.
- $A \rightarrow B$: Aid, doc, signature = $E_{AS}(H(Aid, doc))$;

  Aid is an identifier identifying A as the sender so that B can obtain the correct public key for decryption of the signature. H() is a hashing algorithm used to reduce doc to a bit length suitable for quick public key encryption.
- B obtains the correct public key AP and decrypts the signature. If D_{AP}(signature) is the same as the result of B's own H(Aid, doc) calculation then it can be assumed that A has signed it and did indeed generate it.

A necessary assumption in the use of digital signatures is that A's private key is only known to A. If anyone else knew it then they could sign documents pretending to be A. Due to the dependency of the hash value upon every bit of information contained within Aid and doc instead of being used as a signature it is also possible to use it as an indication of integrity.

2.7.3 Non-Repudiation

A non-repudiation service is one which removes the ability from an entity of denying that they have committed an act that they have done. A non-repudiation service relies upon the generation of evidence, which is used to provide proof of certain facts.

There are two important non-repudiation services that have been defined in the ISO-OSI Basic Reference Model Part 2: Security Architecture; proof of origin, and proof of delivery. Proof of origin can be attained through the use of digital signatures, proof of delivery requires the use of a delivery service.

Proof of origin is generated by the originator of a message. The proof of origin may be a signature that can be stored or an entire message encrypted using an asymmetric private key. The proof is usually stored by the recipient of the message in case the originator denies generation of the message. Proof of delivery is generated by a delivery authority and sent to the originator of the message as proof that the originator's message has been placed in the message store of the recipient, thus preventing the delivery service from denying delivery.

There are four phases of non-repudiation [48].

- Evidence generation:

  Evidence is generated by an evidence generator or the initiator of a process that requires proof. An Evidence generator is a party trusted to provide accurate information, they may also provide services of evidence recording or verification.

- Evidence transfer, storage and retrieval:

  Evidence is transferred between evidence generators and evidence users, or to and from storage at the request of evidence users.

- Evidence Verification:

  An evidence user requests from an evidence verifier (this is usually an evidence generator) that the evidence provided is good enough in case of a dispute.

- Dispute resolution or arbitration:

  An arbitrator collects evidence from all parties involved in a dispute and resolves it.

2.8 Key Management and Distribution

When cryptographic techniques are used it is important not to weaken the overall cryptographic scheme by placing the secret/private keys in danger from attacks. To prevent the unwanted acquisition of keys by third parties an effective and secure key management policy is necessary. This policy will determine three things:

1. generation of keys;
2. storage and access of keys;
3. communication of keys.

Key generation should be conducted secretly and randomly within the valid range of keys for a specific cryptosystem. Any potential weaknesses that are known about the generation process of keys should be tested for and eliminated.

The communication of keys within a public key cryptosystem is simple. Only the public keys of a key pair should ever be communicated between computer systems and because the private key is indeterminate with knowledge of the public one it remains safe. However, in the event a symmetric key system is being used it is necessary for two parties to know what the secret key in use is. This produces a co-ordination problem, as it is inadvisable to communicate a secret key in any clear form. Therefore, it is now generally accepted that a proper policy for confidential communication requires the use of both a public-key cryptosystem and a secret key system. The secret key is communicated between the two parties by encrypting it with the private key of one and the public key of the other.

An example of a protocol for the communication of a secret key using a public key cryptosystem with authentication is shown overleaf:

- Party A generates public key pair AP and AS, and publishes AP;
- Party B generates public key pair BP and BS, and publishes BP;
- Party A generates ciphertext CT from plaintext PT using symmetric key SK,

  $CT = E_{SK}(PT)$;

- A→B : Aid, CT;

  Aid is an identifier identifying A as the sender so that B can obtain the correct public key for decryption of the ciphertext CT.

- A→B : key = $E_{BP}(E_{AS}(SK))$;
- B calculates SK = $D_{AP}(D_{BS}(key))$;
- B calculates PT = $D_{SK}(CT)$.

A problem associated with the distribution of public keys is in trusting that the public key obtained does in fact belong to the party that it is supposed to. This problem can be solved through the use of a Trusted Third Party (TTP). A TTP is an independent organisation that has no interest in one communicating party over another. TTPs are generally expected to be provided by the government or by a large telecommunications provider such as British Telecom (BT).

TTPs can become involved in any of three ways within a communication protocol [49]:

1. off-line: these TTPs are certification authorities or key generation authorities which may not be involved in every instance of communication. The TTP generates evidence that can be used by the communicating parties to check authenticity etc;

2. on-line: these TTPs may be delivery authorities that act as intermediaries for communication, capable of providing proof of origin and proof of receipt and may assist the two parties in verifying certificates etc;
3. in-line: in this case the TTP acts as a monitor and intermediary performing exchanges of information etc.

The authentication and integrity of cryptographic keys between two communicating parties using an off-line TTP is dependent upon the use of *certificates*. A certificate is an item of data that can prove a certain fact due to the trustworthiness of the TTP that generated it. Each TTP will have its own asymmetric cryptosystem key pair, a private key that does not leave the physical locality of the TTP and which may be used to sign certificates to prove their validity, and a public key that is available to all users of the TTP to check signatures.

An example of the fields found within a TTP certificate is shown overleaf.

- ID number:
  this is the TTP certificate number.
- Certificate Signature Algorithm:
  this field is used to indicate which signature scheme was used by the TTP to sign the certificate.
- Issuer's Name:
  this field holds the unique/distinguished name of the TTP.
- Period of Validity:
  this field contains the date on which the certificate expires.
- Unique name:
  this field contains the unique/distinguished name of the entity for which the certificate is used to identify.
- Entity Public Key:
  this field contains the public-key to be used in any communication that requires its use.
- TTP signature:
  this is the signature placed upon the certificate by the TTP using the signature scheme indicated in 'Certificate Signature Field.' The signature is an assurance that the certificate has not been tampered with and is generated using the TTP private key.

The three main services that a TTP may supply are the following.

1. Generation and distribution of session keys.

   If confidential communications are necessary between two parties using a symmetric cryptosystem then they may opt to have the certification authority generate a random secret key. In this case the TTP would generate the session key and then communicate the keys confidentially using the public-keys of the two parties to encrypt the secret key.

2. Generation and distribution of certificates.

   The initial information used in the generation of a certificate must be gathered with great care, and should therefore be done through conventional means such as the use of written requests or personal meetings. The distribution of certificates is accomplished through the storage of the certificates in an openly accessible database maintained by the TTP, generally indexed through a party's unique name.

3. Revocation of certificates.

   After a specified elapsed time a certificate must be withdrawn and re-issued, or if there has been some abuse of the system certain certificates might be re-issued, or simply removed.

TTPs can be used in non-repudiation acting as evidence generators/stores/arbitrators. Non-repudiation and certification are security services that come under the title of *notary services*, these are services that require an independent witness to a process for the attainment of security.

2.9 Conclusion

This chapter has described the uses for which cryptographic techniques can be used within computer security. Although cryptography is a large part in any security system it is not the only part. A discussion of the further necessary attributes are discussed in the next chapter, where CISS is introduced.

The cryptosystems chosen for implementation within CISS are RSA and DES, chosen for the reasons given in section 2.6. These cryptosystems can provide the necessary functionality for authentication, integrity, proofs and confidentiality. The techniques described in sections 2.7 and 2.8 will be used in the provision of effective security within CISS.

Chapter Three

3. Comprehensive Integrated Security System

This chapter presents the fundamental concepts of CISS. The chapter begins by discussing the security threats that distributed systems face. A description of services and mechanisms to counter the potential threats are presented. The CISS conceptual model that describes how CISS supplies the services and mechanisms is given, followed by descriptions of the CISS knowledge base and the ten agents that provide the functionality necessary for the security system. Finally, the management structure for CISS within a domain is discussed.

3.1 Introduction

The CISS architecture [7], [50] was developed to protect computer systems within an open distributed environment against threats of a potentially malicious nature. The exact threats are detailed below in section 3.2. An open distributed environment is one in which geographically dispersed terminals, users, applications and other computing resources process and communicate information between them. These types of systems are becoming much more common as users begin the migration out of central offices and into the home, and as organisations begin to make the best use of resources through greater accessibility to its users.

The majority of existing computer networks were not designed with the necessary security features in which to provide protection of computing resources within a distributed environment. CISS was conceived to provide the additional security features that are missing from many of today's networks. Its exact configuration is dependent upon the security policy within the organisation which it is being implemented.

3.2 Threats, Services and Mechanisms

The security implemented within an organisation is dependent upon its security policy. A security policy is created from analysis of:

- which threats should be protected against;
- the services that the security system should provide to counter the potential threats;
- and the degree of protection to allocate to system resources.

3.2.1 Threats

Threats to a computer system are realised in one of two forms of attack, *passive* and *active*. A *passive* attack attempts to glean useful information from eavesdropping or monitoring transmissions. An *active* attack involves some form of modification, creation, prevention or interrogation of data and services.

Threats and subsequent attacks present a greater security risk within a distributed environment where information is transmitted over large distances and in greater volume. For example, the threat of an intruder within a closed system would be lessened by the need for the intruder to gain physical access to an entry point to the system. Whereas, in an open system access is potentially available from terminals in geographically dispersed locations that may not be physically secured.

The threats that must be considered are as follows.

- Masquerade.

  A masquerade occurs when one entity pretends to be another, this form of attack usually involves other forms of active attacks. For example, authentication sequences can be recorded and replayed. Recording would be a passive attack while replay is an active attack. Masquerade is a threat present from legal or illegal: user; software; hardware.

- Illegal Associations.

  An illegal association is one in which there has been a violation of the authentication and authorisation policies in place for the system. This may involve legal/illegal users/software. An attack may result in the creation of an account for an individual

that is not allowed such an account under the current policies, thus setting up the potential for illegal associations between said user/software and system resources.

- Non-Authorised Access.

  Non-authorised access is one in which an entity violates the access control mechanisms of a security system. This threat may involve a legal/illegal user/software. This threat is clearly demonstrated when a hacker obtains illegal access to a system and has therefore violated access control. Total access control can be obtained through modification of control files or by masquerading as a privileged user.

- Denial of Service.

  This is where a legal user/software is denied legitimate access to system resources/services. An attack may be caused by illegal/legal users/software engineering a way to tie up system resources, an example would be the monopolisation of processing time. Accidental damage or deterioration of equipment can also lead to denial of service.

- Repudiation.

  This occurs when an entity denies an action such as originating, delivering, receiving, or transmission of a message when they have actually performed said action.

- Leakage of Information.

  This takes the form of interception of data leading to a loss of confidentiality, or identification of data which results in loss of anonymity. Leakage of information can be caused by illegal/legal users/software/hardware.

- Traffic Analysis.

  Although there may be no leakage of information, a passive attack involving analysis of traffic flow is a security threat. Analysis of traffic can produce statistics on size, source, destination, frequency of communications between parties. This information could then be used in guessing the nature of the communication. The threat of this attack is present from illegal/legal users/software/hardware.

- Invalid Message Sequencing.

  This form of attack is active and involves the capture of transmitted data for the purpose of eliciting an illegal effect from the system through re-ordering or replay. The threat of this attack is present from illegal/legal users/software/hardware.

- Data Modification.

  This is an active attack that either through intentional or un-intentional action causes a modification in information being transmitted or stored. It results in a loss of integrity. The threat of this attack is present from illegal/legal users/software/hardware.

- Deduction of Information.

  Although access to certain database segments may be restricted information from those segments may be garnered through the interrogation of the information unprotected in a database. This form of attack is referred to as deduction by inference. The threat of this attack is present from illegal/legal users.

- Illegal Modification of Programs.

  Is the introduction of some form of software that modifies the operation of legitimate software. The threat of this attack is present from illegal/legal users/software/hardware.

3.2.2 Security Services

To combat the threats described in section 3.2.1 a security system makes available services that can be used to provide the necessary security. The table below lists the security services that are used to counter the threats listed in section 3.2.1.

Threat	Service
Masqueraders	• Authentication
Illegal Associations	• Access Control
Non-Authorised Access	• Access Control • Authentication
Denial of Service	• Quality of Service
Repudiation	• Non-repudiation
Leakage of Information	• Message Confidentiality
Traffic Analysis	• Traffic Flow Confidentiality
Invalid Message sequencing	• Connection Integrity
Data Modification	• Data Integrity
Illegal Modification of Programs	• Access Control

Table 3-1 Security Services

What follows is a description of the security services used to counter the threats in the table above.

- Authentication:

  is used to identify a user/software/hardware to other entities. This can be accomplished through digital signatures, MACs, secret identifiers.

- Access Control:

  determines whether an entity has legal access to system resources. This is accomplished through the use of an *access control matrix*. If an entity attempts to access a system resource, the access matrix is consulted by the security system. If the

point of intersection in the matrix indicates right of access then the entity is allowed to continue.

- Connection Integrity:

  is used to prevent such attacks as replay. This is generally accomplished through the use of time-stamps, or sequencing numbers. Time-stamps indicate a specific time, such as the time a message is sent. If the recipient receives a message outside of a time period described by the security policy as being valid it is considered an attack. Sequence numbers are used to order packets that may arrive in different orders due to connectionless communication.

- Data Integrity:

  prevents the modification of data going undetected. The usual method for accomplishing this is through the use of integrity checkers such as hash algorithms, MDCs, etc.

- Message Confidentiality:

  is to prevent loss of confidentiality. It is usually accomplished through the use of cryptography.

- Non-Repudiation:

  helps to prevent the denial of various truths. This is generally accomplished through the generation of proofs, and the use of TTPs to provide notary services.

- Quality of Service:

  is the active monitoring of the state that a connection/association is in. If a denial of service, or loss of service is detected then efforts to establish service is attempted.

- Traffic Flow Confidentiality

  prevents the analysis of traffic. Traffic flow confidentiality involves the use of *traffic padding* or *routing control*. Traffic padding is used to either pad out individual messages or to insert messages with data that have no other function than to confuse any analysis of the traffic. Routing control can provide prevention of traffic analysis through organising a messages route so that only those segments of a network that are physically secured are used and so cannot be eavesdropped upon.

3.2.3 Mechanisms

The security services themselves are implemented through the correct ordering and application of security mechanisms. The mechanisms employed within CISS are given in Table 3-2. Table 3-3 gives some examples of which security mechanisms constitute a security service.

En	Encipherment
DS	Digital Signature
AC	Access Control
DI	Data Integrity
AE	Authentication Exchange
TP	Traffic Padding
RC	Routing Control
Nt	Notarisation

Table 3-2 Security Mechanisms

X indicates a mechanism is present and O indicates its absence.

Service	En	DS	AC	DI	AE	TP	RC	Nt
Peer Entity Authentication	X	X	O	O	X	O	O	O
Data origin Authentication	X	X	O	O	O	O	O	O
Access Control Service	O	O	X	O	O	O	O	O
Data Confidentiality	X	O	O	O	O	O	O	O
Non-repudiation Origin	O	X	O	X	O	O	O	X
Non-repudiation delivery	O	X	O	X	O	O	O	X

Table 3-3 Sample Service Structure

The majority of the security mechanisms in Table 3-2 are described in Chapter 2.

3.3 Conceptual Model of CISS

The way that CISS supplies an organisation with the necessary security services and mechanisms can be described through the conceptual model of CISS which can be seen in Figure 3-1 below.

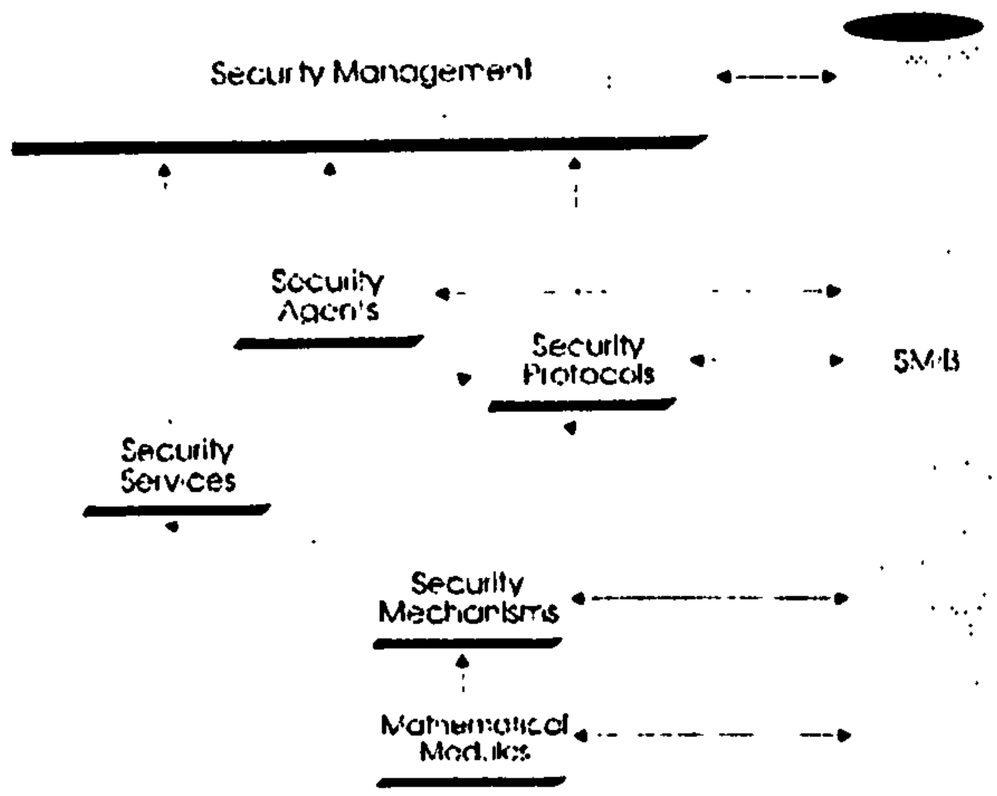


Figure 3-1 CISS Conceptual Model

The various segments of the conceptual model are discussed below.

3.3.1 Mathematical Module

The *mathematical module* is a set of basic mathematical functions that can be used to provide the cryptographic functions used in the majority of security mechanisms described in section 3.2.3. The provision of basic mathematical functions allows CISS to implement new cryptographic algorithms as and when they become available.

For example, all public key cryptosystems are based upon the same techniques, those of modular exponentiation. Therefore, the provision of standard low level functions for modular exponentiation allows CISS to implement either of the public-key cryptosystems described in Chapter 2, and potentially all others based upon the same techniques. The cryptosystem actually used within a security service will be dependent upon the individual organisation's security policy which governs the configuration of CISS. However, the fact that either cryptosystem can be provided makes the mathematical module approach very flexible.

3.3.2 Mechanisms and Services

The security services and mechanisms are those described in section 3.2.2 and 3.2.3.

CISS provides flexibility of implementation through the selection of security services that are used within secure communication/processes. The individual selection of security services used by various entities within a system is necessary because of the differing amounts of security an organisation's security policy will require for individual system resources. For example, a user/process may use weak but efficient security mechanisms within the security services used within electronically sent memos, while electronic funds transfers between departments within an organisation may require slower stronger mechanisms. The exact strength of a service is determined via the security policy and the technological limitations that are placed upon the user/process. The configuration of the services and mechanisms that are used by entities are located in security profiles stored within the knowledge base of CISS - the security management information base.

3.3.3 Agents

Because a security system is a very complex and potentially large software construction problem CISS has been divided into ten functional elements known as agents. The agents are responsible for the co-ordination, supply and control of security services to users/processes. The agents' functionality is described in section 3.5.

3.3.4 Security Protocols

The *security protocols* are sequences of events that two or more parties follow to accomplish a task. It determines a safe and secure way in which the two parties may accomplish their task. All parties involved in a protocol must be aware of the correct order of events to follow. CISS provides protocols for the interaction of users/software and the agents that make up its functional portion. Protocols govern such things as key exchanges, contract signing, peer to peer authentication, etc. Protocols for secure communication can be found in Schneier[14] and Muftic[51], the agent protocols are specified in Muftic[7].

3.3.5 Security Management Information Base

The security management information base (SMIB) is the CISS knowledge store. It contains information on the configuration of CISS through the selection of mechanisms, services, protocols and the subsequent limitations upon the variables in their use as specified by a security policy. The SMIB contains information on authorised users, authentication data, user entity capabilities and privileges. See section 3.4 for details of the SMIB.

3.3.6 Security Management

The most abstract part of the CISS conceptual model is the segment necessary for correct management of CISS. The ISO security architecture [8] divides management functions into four parts.

1. *System Security Management:*

   deals with event handling, such as the keeping of audit trails and system recovery.

2. *Security Service Management:*

   selects and invokes the security mechanisms to provide security services.

3. *Security Mechanisms Management:*

   manages the cryptographic keys and the necessary parameters for correct invocation of the cryptographic mechanisms.

4. *Security of OSI Management:*

   provides the necessary management exchange of information through security management protocols.

The management functions are provided by CISS through the SMIB and the various agents and can only be configured by the security administrator of CISS.

3.4 Security Management Information Base

The SMIB is central to the functioning of CISS. The SMIB contains six basic conceptual segments which are given below.

- The identification segment:

  contains the identity of an entity, for example the user's login name. Indexed by the user's name is an item of authentication information such as a password for basic authentication.

- The extended security segment:

  stores the security profiles of users/resources. These profiles store the mechanisms and services that are to be used to counter security threats. The profiles for individual objects are necessary for comprehensive implementation of a security policy.

- The secure associations segment:

  maintains information on current active associations. Monitoring of the associations is important for the detection of denial of service and other potential security violations.

- The extended access control segment:

  holds information on the privileges of users. It contains the access rights of the user, and groups the user may belong to. Privileges describe the users ability to conduct actions such as creation, modification, and deletion of system resources. The authentication mechanisms that are necessary for access to system resources may also be stored here.

- The security log:

  is where events of significant interest to security that occur on a computer system are recorded for further analysis. The analysis of the security log is usually conducted by the security administrator.

- The confidential segment:

  private segment used to store sensitive data for active entities, such as temporary secret/private keys.

Because of the sensitivity of the information present within the SMIB it is an option of the security administrators to allow its entries to be selectively encrypted. However, this form of protection leads to an increased amount of processing and consequently greater delay when accessing the information stored within the SMIB.

The use of security profiles by the SMIB provides a flexible solution to creating different levels of security according to policy or user's personal preferences. For example, the user security profiles that are stored within the SMIB may contain the preferred encryption algorithm for confidentiality, or the preferred digital signature mechanism, or even the size of public key used (a smaller key would mean quicker encryption/decryption speeds and less waiting time for the user). The configuration of CISS (controlled through the parameters stored within the SMIB) according to the local security policy maintains limits on personal preferences. For instance, a user would be denied a public-key pair if they attempted to violate the minimum or maximum public-key length set by the active security policy which is stored as a parameter within the SMIB.

The modularity of CISS allows greater flexibility in the implementation of security requirements, with the possibility for quick modifications to its implementation through adjustments to the entries within the SMIB. This modularity is extended to the services and mechanisms that can be sequenced according to the preferred options of the security

administrator and local security policy through alteration of the respective entries within the SMIB.

Security services are formed through the correct ordering of security mechanisms. The order of security mechanisms within security services is held within the SMIB's security profiles. The security protocols are used to order the mechanisms and services so that no security holes appear through the way they are used, as opposed to the individual mechanism's or service's innate level of security. An example of the way mechanisms and services are sequenced in CISS is described below.

The mathematical module provides numerous mathematical functions including:

exponential calculation (EXP);
modulus calculation (MOD);
bit shift operation (BSH);
bit substitution operation (BSUB);
exclusiveor operation (XOR).

A (Mechanism) is comprised of (Math module function1), (Math Module function2), etc.

RSA is a digital signature mechanism
HASH is an integrity mechanism.
DES is an encipherment mechanism.

RSA is comprised of sequenced EXP and MOD.
HASH is comprised of sequenced EXP, MOD and XOR.
DES is comprised of sequenced BSH, BSUB and XORs.

A service is provided through the use of selected mechanisms.

A (Service) is comprised of (Mechanism 1), (Mechanism 2), etc.

POO is the notary service proof of origin
CON is a confidentiality service.

PofO is comprised of sequenced HASH and RSA mechanisms.

CON is comprised of the DES mechanism.

A protocol would then use the services in a sequenced step of actions to accomplish a specific task such as:

confidential communication with proof of origin (CON-COM-PofO);

CON-COM-PofO is comprised of sequenced CON and PofO services.

3.5 The Ten Agents of CISS

The ten CISS agents are:

1. user agent (UA);
2. security administration agent (SAA);
3. security services agent (SSA);
4. security mechanisms agent (SMA);
5. security management information base agent (SMIBA);
6. operational environment agent (OPENA);
7. association agent (AA);
8. inter-domain communications agent (IDCA);
9. monitoring agent (MA);
10. recovery agent (RA).

The interaction of these agents can be seen in Figure 3-2 overleaf.

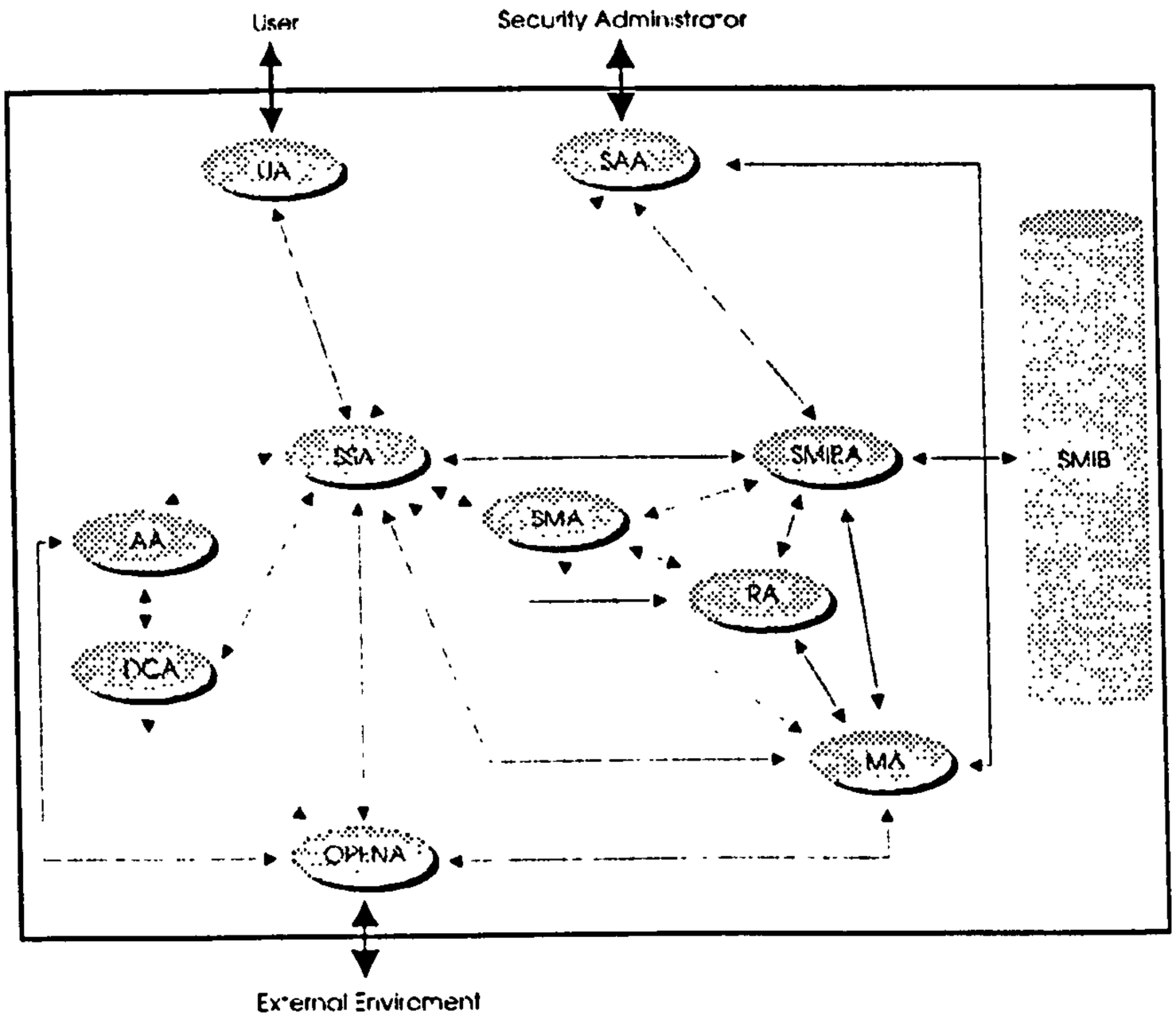


Figure 3-2 Ten Agent Interaction

As can be seen there are three access points to CISS, the UA, the SAA, and the OPENA.

3.5.1 User Agent

This is the interface of CISS through which the users can access its security mechanisms and services directly via a request. Dependent upon the security policy some security services will be provided irrespective of the user's wishes (see the OPENA description below). The user interface is primarily responsible for:

- providing user access to the SSA;
- formatting the user requests into valid calls for service from the SSA, and the presentation of data from the SSA to the user;
- maintaining a temporary log of user requests, helping against data inference;
- prompting the user for more information at the SSA's request;

- assisting users in their own security management requirements, such as file encryption for storage.

3.5.2 Security Administration Agent

This is the second interface to CISS and is solely used by the security administrator. The division of functionality between the UA and the SAA has come about due to the necessity of making the UA a multi-user agent producing complex code, the separation produces simpler code for the SAA. The main functions of the SAA are:

- interfacing between the security administrator and CISS;
- configuring SMIB segments, which implement the security policy;
- accessing the SSA on behalf of the security administrator;
- co-ordinating and co-operating with the MA and the RA.

3.5.3 Operational Environment Agent

This is the final interface through which CISS communicates with the outside world. Its main function is interfacing the operating system (OS), applications, to the SSA for the provision of security services. The OPENA is made up in part by the application programming interface (API). The interface allows applications to access CISS services giving CISS greater flexibility for implementation within existing systems. The OPENA interacts with the AA and IDCA for communication with entities external to the local security domain.

3.5.4 Security Services Agent

The SSA is the central agent through which all services are requested. The SSA has contact with all other agents and should therefore be fully optimised as it has the potential to be a processing bottleneck. The SSA selects the security mechanisms that a requested security service is made up from, and with assistance from the security mechanism agent, implements the service. The SSA must restrict the use of security services by user entities to those that they are entitled to use, either through their privilege or the capability of the system they are using.

3.5.5 Security Mechanisms Agent

The SMA interfaces the security mechanisms to the SSA for the provision of security services. Therefore, it accepts control and data commands from the SSA. The SMIBA provides information on the parameters that a mechanism can accept; for example, depending on the security policy in place an RSA key length of 512 may be too small or may be excessively safe.

3.5.6 Security Management Information Base Agent

The SMIBA is responsible for interfacing the SMIB to the other agents within CISS. It is the only agent with direct access to the SMIB. All the security parameters held within the SMIB that the SSA requires must be obtained through requests to the SMIBA. As this is the only agent that interacts with the SMIB and the SMIB will be under heavy use this is a potential bottleneck for CISS activity.

3.5.7 Association Agent

The AA is responsible for the associations between agents within the local security domain. It must make sure that associations are initiated with the correct security and that security is maintained throughout the association. At termination of the association the AA must make sure that the agents complete a proper termination by examining a state table kept within the SMIB secure associations segment. The agent is also responsible for detecting denial of service subject to quality of service conditions.

3.5.8 Inter Domain Communication Agent

The IDCA is responsible for communication with entities external to the local security domain. Therefore, it must negotiate the required parameters for a secure connection with an external entity. Such negotiations may involve the type of cryptographic algorithm to be used, the hashing algorithm, size of key. The IDCA interacts with the AA and the OPENA for the provision of secure inter domain communications.

3.5.9 Monitoring Agent

The monitoring agent monitors the activity of CISS, primarily the actions of the SSA, and logs the events within the SMIB. The historical events that are stored within the SMIB are only accessible by the security administrator for the preparation of auditing logs. The monitoring agents role is enlarged to encompass entity profiling in Chapter 8 where a monitoring model is presented.

3.5.10 Recovery Agent

The recovery agent is responsible for the recovery of CISS when faults either at component level or procedural occur, caused either maliciously or accidentally. The RA must detect when faults occur and place CISS into a secure state. The error is noted by the MA for the attention of the security administrator. This is especially useful for procedural errors which can be used to weaken CISS's security through repeated use, potentially leading to denial of service to legitimate users. The biggest potential problem that the RA will have to deal with is a fault occurring within the SMIB. In this situation the RA would have to halt CISS, producing a complete denial of service and notify the security administrator.

3.6 Security Management Centres

The suggested management structure for a CISS security system involves the use of *Security Management Centres* (SMCs) [7]. The SMC controls the security for all objects within its security domain, which is defined as the entities subject to a single security policy and a single authority. The SMC's influence extends only as far as the organisation's local area network as this is usually a good division point between the outside world and the relatively secure environment upon the organisation's LAN.

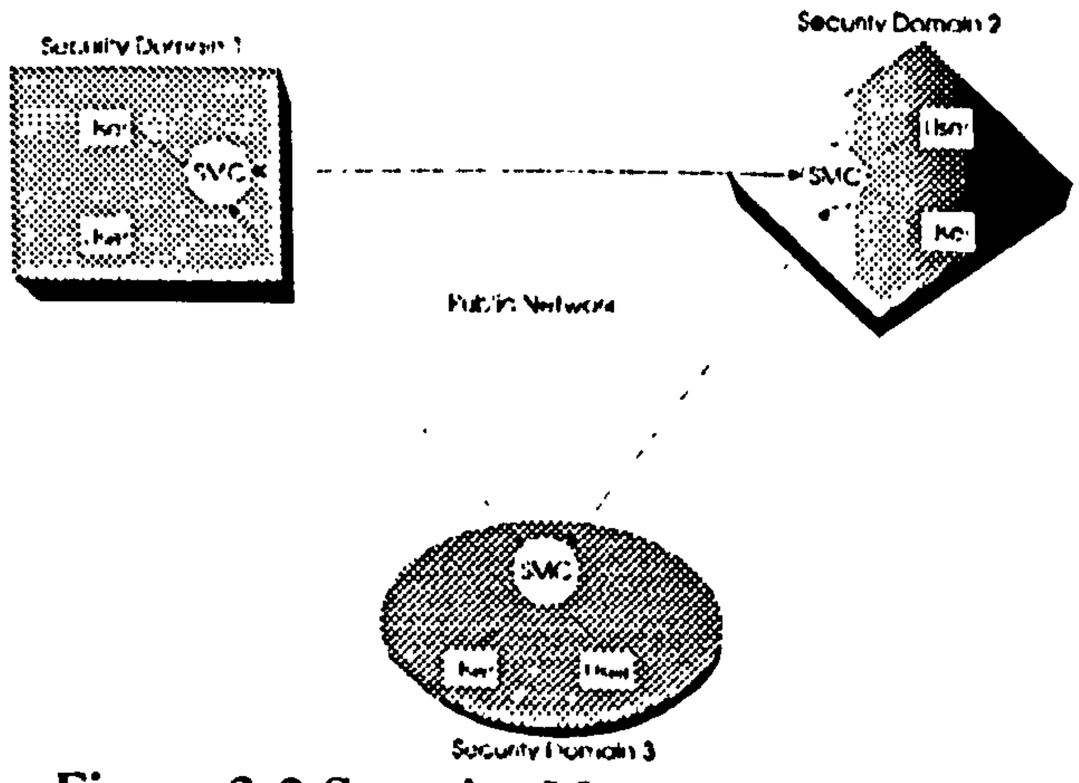


Figure 3-3 Security Management Centre

The SMC as with all entities within the conventional CISS setup, has access to the ten agents that make up CISS and an SMIB in which to store operational information for the management of CISS. The services and mechanisms that are provided to the SMC through the CISS agents are used to provide management functions to the domain and all other CISS secured entities within it. Each SMC implements the security policy of the security domain providing hosted clients with their security needs.

The main security functions that the SMC provides to users and the security administrators include the following.

- Logging and Auditing Security Relevant Events.

  Audit information is analysed here for potential privilege abuse and intrusion detection.

- Inter-domain Communications.

  When a user wishes to perform some function manipulating an entity within a different security domain the SMCs of each domain must negotiate what services are required for secure communication over the public unsecured network.

- Generation, Storage and Distribution of Cryptographic Keys.

  To maintain confidentiality within the domain it is necessary to generate keys to a set policy and then use secure protocols for their storage and distribution.

- Domain Notary Service.

  The SMC must act as a *Trusted Third Party* (TTP) for the users hosted upon it. This means that it must provide the functions, registration, notarisation,

certification services and public key verification along with public-key distribution for all users.

- Trusted entry point.

  Firewalls [52], are used for access control to local area networks through the Internet. This method of access control should be implemented in CISS via the SMC which will act as a single point of entry and exit to the security domain, allowing the implementation for efficient security logging and access control.

- Domain Communications.

  If a user wishes to perform a remote operation upon a machine that is within the same security domain, the SMC must make sure that all security requirements are met according to the policy laid out within the domain.

- Access Control.

  Through the use of the SMC's SMIB the access rights and privileges of individuals may be determined so that unauthorised manipulation of domain entities can be secured against.

- Authentication.

  Every person that wishes to use CISS functionality must first be authenticated by the SMC through information stored within the SMC's SMIB.

One of the problems of having a single point of control, like an SMC, is that it produces a potential bottle neck that can adversely affect system performance. One way of alleviating this problem is through distributing the central processing elements functionality. Alternatively, a very powerful central machine can be used to meet the demand for services but the reduced performance return of using a large central machine makes the latter

solution less attractive in large networks, the former solution is presented in the next chapter.

Chapter 4

4. 3-L Architecture

The chapter begins by describing the topology of a new management scheme for CISS. The CISS agents and services that the SMC provides are divided amongst the three levels of the new scheme. The three levels of the architecture are then described, along with the way in which the SMC' services are provided. Finally, the common services that the three levels provide are summarised.

4.1 Introduction

A security domain generally extends as far as an organisation's local area network (LAN), which provides a suitable boundary at which responsibility for security is clearly divided between that of the organisation and that which is not. Under ideal circumstances the computer systems for which the single authority is responsible would be homogenous (i.e. common computing platforms, and processing capabilities) and the connections between these systems would be totally secure. However, the reality is that in most cases a security domain will be made up of vastly differing machines with differing capabilities and physical conditions, potentially spread across a large geographical area where the security of a communication line cannot be assured. Therefore, the security requirements of a domain should not start at its boundary with the outside world, but should begin at the user terminal.

Because of the differing capabilities of machines within a single organisation the current model for CISS is inadequate as it demands far too much processing power from potentially very limited machines. Within this chapter a new architecture still abiding by the CISS concept is suggested. The work is theoretical but some of the requirements that it suggests for a workable system are investigated in later chapters. The modularity of the new architecture and the flexibility it provides is better suited for a real world application, allowing the security to be determined on a machine by machine basis

The necessity for security at the user terminal is being demonstrated through other research currently undertaken within the Network Research Group at the University of Plymouth into the acquisition of user passwords across a local area network [53]. The method being

investigated requires a PC, an Ethernet card and a custom program that enables the PC to run in 'promiscuous mode,' where the PC looks at all packets upon the Ethernet. The custom program then filters the information looking for passwords that are in plaintext; for instance when a mail client accesses a mail server the user must login and enter a password to read their mail, but with many readers the password is sent in plaintext. An attacker using a promiscuous PC on a LAN can store the captured passwords and login names to obtain full access to the user's account.

4.2 Three Level Architecture

There are two problems with the conventional deployment of the CISS agents.

1. In a heterogeneous network there may be machines that are incapable of providing the necessary processing power that is required by the ten agents of CISS.
2. The use of a central SMC to provide all management functions for a domain creates a bottle neck.

The objective of the three level architecture (3-L architecture) is to provide management of security services within the boundaries of a security domain in a distributed environment by replacing the central SMC and altering the deployment of the CISS agents.

The 3-L architecture is comprised of a domain management centre (DMC), multiple local security servers (LSSs) and a local security unit (LSU) upon each user terminal. Multiple LSUs would be hosted upon a single LSS, the number of LSSs within an organisation would be dependent upon its size and its internal policy for computer asset organisation. However, it is envisaged that a single DMC will enforce domain wide policy while each

logical division within an organisation such as accounting, personnel, sales, procurement, will have an LSS for enforcement of divisional security policy. Upon each data terminal that users may access the organisation's computer system an LSU will be present. The three levels interact within the boundaries of the security domain to meet the requirements of the organisations security policy for activities within the security domain and without.

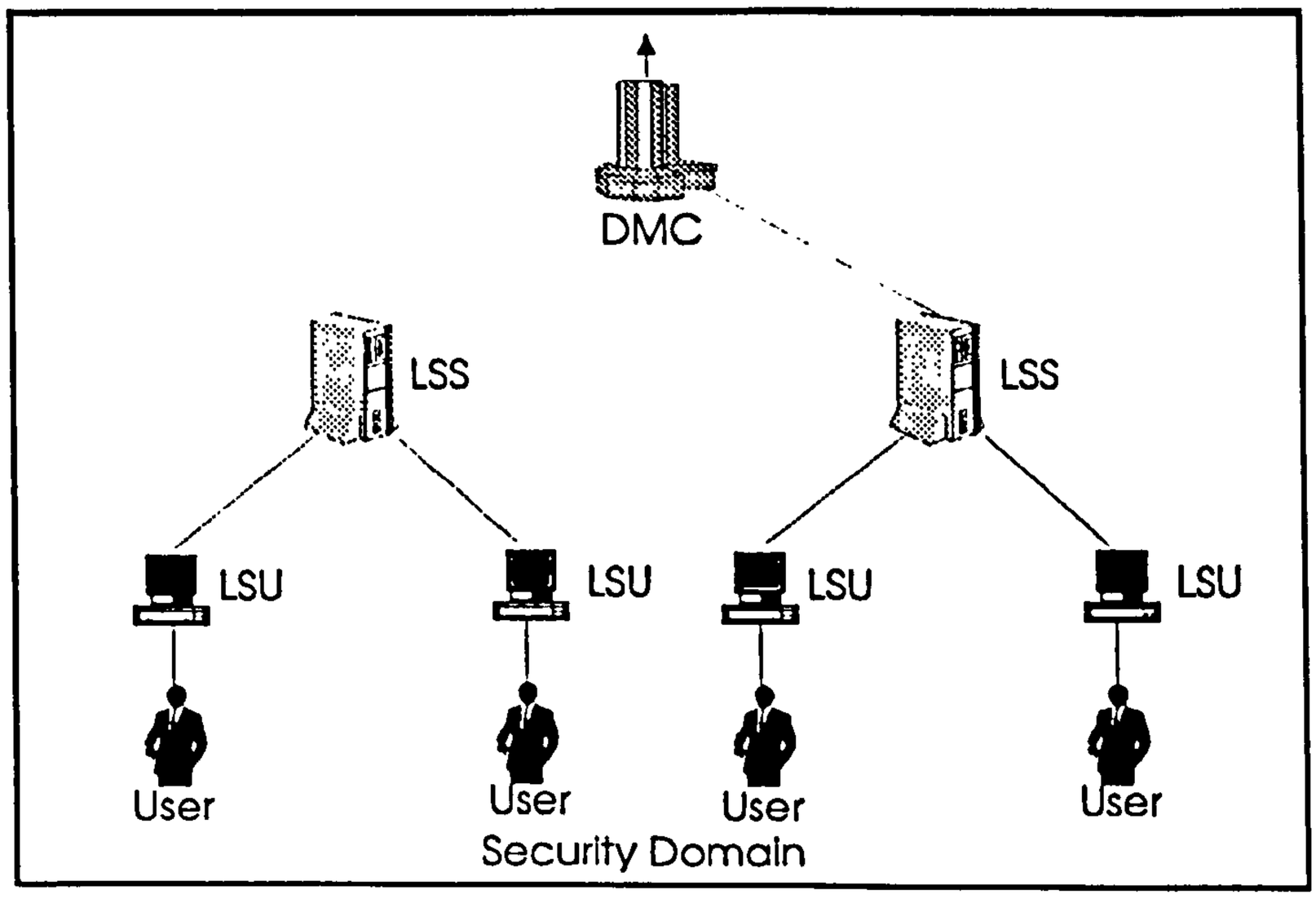


Figure 4-1 Three Level Architecture

Each higher level within the 3-L architecture is aware of the capabilities of the immediate lower level units that are connected to it, information concerning the hosted security modules would be kept within the SMIBs of the LSSs and DMCs and would require an extension to the security profiles for software and hardware components discussed within Chapter 3. This knowledge allows the upper levels to determine whether services requested by lower level units meet with the required security laid down by the domain security policy and therefore whether the operation can go ahead, or if the operation requires more security

than is available the operation is prevented by the higher level entity and the reason is logged with the requesting unit.

Each of the level entities that make up the three level architecture are built from the agents that are available within CISS. The CISS agent list for each level is given in the table below.

X-Present, O-Absent

	DMC	LSS	LSU
UA	O	O	X
SAA	X	X	O
SSA	X	X	X
SMA	X	X	X
SMIBA	X	X	O
OPENA	X	X	X
AA	X	X	O
IDCA	X	O	O
MA	X	X/O	X/O
RA	X	X/O	O
ADDITIONAL	O	O	X

Table 4-1 Agents Used within 3-L

Some of the SMC's duties are common amongst the three levels and some are specific to a certain level, the table below sums up the division of labour for the 3-L architecture.

X-Provided, O-Not Provided

Security Service	DMC	LSS	LSU
Logging and Auditing Security Relevant Events	X	X	X
Generation, Storage and Distribution of Cryptographic Keys	X	X	X
Domain Notary Service	O	X	O
Inter-Domain Communications	X	O	O
Trusted Entry Point	X	O	O
Domain Communications	O	X	O
Access Control	X	X	O
Authentication	O	X	X

Table 4-2 3-L Division Of Labour

The 3-L architecture relies heavily upon the use of public-key and secret-key cryptosystems; every user, LSU, LSS and DMC has their own public cryptographic key pair used for authentication. The use of the LSU, LSS and DMC key pairs are for security service management only, while the user's public-key pair is used to attain personal liability, accountability, etc.

4.3 Domain Management Centre

The domain management centre is the highest authority within the three layer architecture and is comprised of most agents within the basic CISS model. For all intents and purposes it acts as an SMC to entities requesting associations with entities on the opposite side of a security boundary. The DMC is the only level entity within a security domain that has the direct use of the IDCA. As with the SMC the DMC is analogous to a packet switching centre in data networks. It is therefore possible to control traffic flow between a CISS secured domain and its external environment.

Within the DMC's SMIB each LSS has its own security profile in which its inter domain communication properties are listed i.e. the mechanisms and services to be used during contact with entities outside the local security domain. Also within the extended access control segment an access list is maintained that may hold attributes that refer the DMC to the LSS of interest for instructions or a detailed list of external entities that may form an association with an LSS/LSU. In this way the DMC replaces the role of the SMC in its analogous role of an Internet firewall.

For example, a business wishes to place its accounting machines upon a LAN so that its accountants may access them from remote sites across the domain. However, the administration is worried that the placement of the machines upon the LAN will put them at risk from attackers external to the security domain. Therefore, a parameter is entered into the machines profile held within the extended access control segment of the DMCs SMIB indicating no external associations are allowed with the accounting machines. As the DMC is a required route to the machines within the security domain for external connections the accounting machines attain greater access control.

The DMC is responsible for the implementation of the security policy within a domain. To accomplish this the DMC is capable of modifying the SAA or rather the SMIB entries that determine the security mechanisms used by the various services and protocols of the LSSs, the divisional security policies will vary between the boundaries set by the domain wide security policy. For maximum security, configuration of the 3-L architecture should only be possible from the DMC console. Therefore, the machine that requires the most stringent access control and authentication requirements within a domain is the DMC. Of the functions the DMC provides, the hardest to gain access to and the greatest privileges to use will be the one evoked for configuration of SMIB segments within the 3-L architecture.

4.3.1 Inter Domain Communications

One of the primary jobs the DMC has is the managing of interdomain communications. Within a security domain the 3-L DMCs are implemented by organisations to look out for their own interests through the implementation of their security policy. Therefore, DMCs by necessity have a policy of distrust to all entities outside of their domain. This policy of

distrust makes communication difficult between two DMCs when there must be a negotiation of protocols between them. For trust to exist between two DMCs an arbitrator is necessary to distribute cryptographic keys and confirm identities. DMC to DMC communication can be facilitated through the use of a TTP public key certification service[54] which alleviates the problems of authenticity and integrity of public-keys used by an unknown DMC. TTPs may also provide evidence storage facilities (as described in Chapter 1) for audit trails and notary services, the exact proofs/records a TTP keeps is dependent upon how active a role the TTP takes in the facilitation of secure communications between the two parties. For some of the notary services to be provided the TTPs must be in-line or on-line, potentially producing bottlenecks for system activity during requests from multiple TTP users. Therefore, the 3-L architecture will generally use TTPs only in an off-line capacity, generating the necessary certificates for key distribution etc.

Every DMC has its own public key pair, with the private key never being transmitted across any communication medium. The public keys of the DMCs will be registered with a TTP along with the DMC's unique name and any other information that the TTP deems necessary for public-key certification, this information will then be signed by the TTP and the certificate stored or communicated to the DMC.

An example of certification can be seen in Figure 4-2, below.

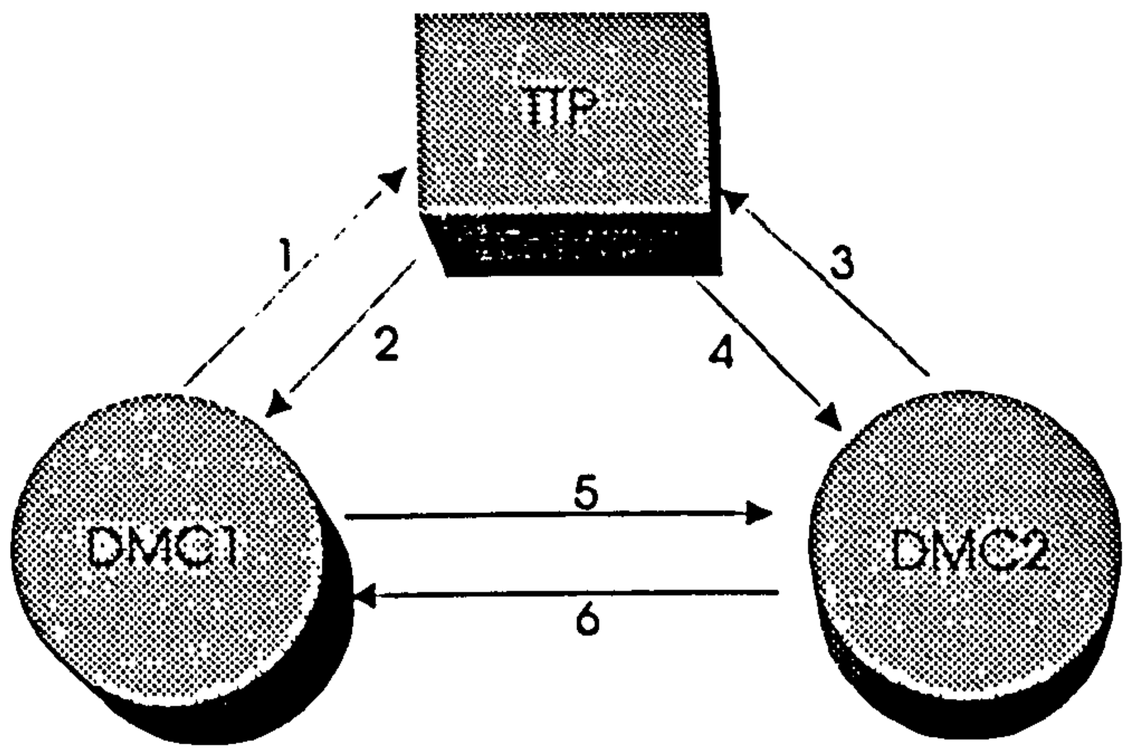


Figure 4-2 DMC to DMC via TTP

1. Request certificate from TTP.
2. TTP sends DMC1's requested certificate.
3. Request certificate from TTP.
4. TTP sends DMC2's requested certificate
5. DMC1 requests service of DMC2 passing its certified public key.
6. DMC2 responds to DMC1 passing its own certified public key.

Two DMCs may not subscribe to the same TTP, making it impossible for either DMC to trust the public key certificate of the other. To overcome this problem the X500[54] series of recommendations suggest a hierarchical structure for TTPs providing certification services. The X500 series recommendations are a set of standards for a distributed directory. Because public key certificates are almost impossible to forge at present, it is safe to store certificates in an open distributed database such as the X500 directory. The X509[54] recommends that certification authorities certify each others public keys in a hierarchical fashion. The following diagram shows a suggested certification authority structure. Using the notation of the X509 standard:

X<<Y>> which is a certificate, X certifies the public key of Y.

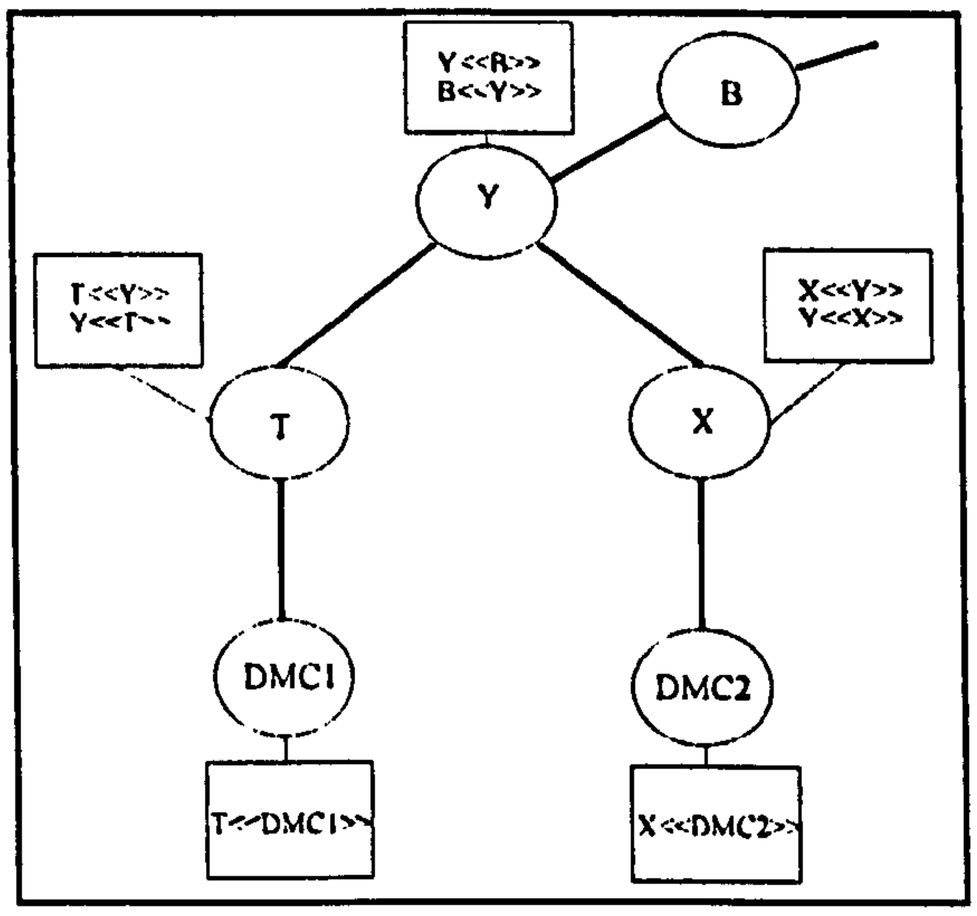


Figure 4-3 X509 CA Hierarchy

For example, using Figure 4-3 the two DMCs can obtain each others public key certificate through a chain of certificates.

DMC1 obtains Y<<T>>, Y<<X>>, X<<DMC2>>

DMC2 obtains X<<Y>>, Y<<T>>, Y<<DMC1>>

A DMC registering with a TTP for the first time must be given the TTPs public key securely so that no-one can substitute a false key. This can be accomplished through delivery of the key within a written letter, securely transported disk or smart card. Once a DMC can trust the TTP's public-key, obtaining further updated keys can be done via the TTP database as a self-signed certificate or from a higher level TTP within the X509 hierarchy.

Level entity peer to peer authentication may be accomplished through the use of an X509 authentication scheme, as detailed above. Using the LSSs to certify the LSUs public-key and the DMC to certify the LSSs public key. The DMC being the highest level within the

3-L architecture cannot obtain a true certificate (one provided by a higher TTP certification authority) and therefore lower levels must trust the DMC.

4.3.1.1 Security Services id certificates

The SSid is a special form of certificate that one entity provides to another within the 3-L architecture. If an entity has successfully petitioned another entity for a service, it will be presented with an SSid, the requesting entity must then present the certificate to the service provider whenever they require continued use of that service. A standard SSid certificate can be seen below.

- SSid header:
  this is used to signify that what follows is an SSid certificate.
- Security Service ID Number:
  this is the DMC service number.
- SSid Type:
  indicates the type of SSid, i.e. access, session etc.
- Requesting Entity ID:
  this field holds the domain unique/distinguished name of the requesting entity.
- Period of Validity:
  this field contains the period of time for which the certificate is valid.
- Server Signature:
  this field contains the server signature of the information above.

The next three sections are added by the requesting entity when presenting the certificate back to the server for further use of the service.

- SSid continuance header:
  this is used to signify that what follows is the attachment.
- Time-Stamp:
  current time of use.
- Requesting Signature:
  this is the signature placed upon the certificate by the requesting entity signing all sections including the provided server SSid section.

Dependent upon the security policy implemented within the domain it may only be necessary for these certificates to be used occasionally rather than continuously within a secure process. The presentation of the SSid back to the service provider provides proof that the requesting entity is the same entity that successfully petitioned the server in the first place. This is similar to the ticket used in Kerberos [55].

The use of this type of certificate is particularly useful as it gives authenticity and integrity without the overhead for encrypting the information completely. In many cases management information within security systems do not require confidentiality but only authenticity and integrity. However, if security policy requires confidentiality then it can be accomplished by encrypting the appropriate sections within the certificate.

Another thing that should be noted is the ability to stack SSids, in that the association of an LSU within one domain will result in a multi-layer SSid. Figure 4-4 shows an example of stacked SSids.

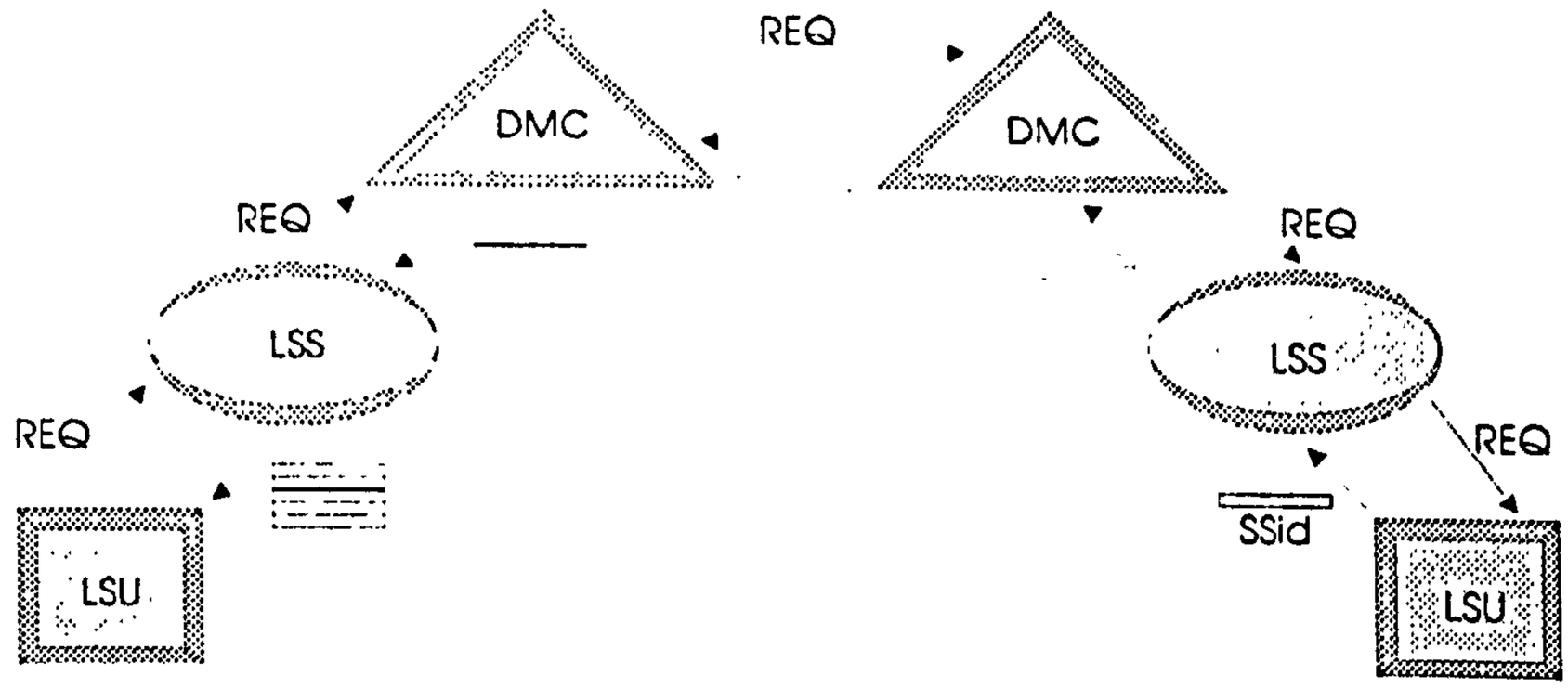


Figure 4-4 Multi-layer SSid

The SSids are useful for auditing purposes and security supervision functions, once recorded they provide a comprehensive map of user activity.

4.3.1.2 Protocol for a DMC to DMC Connection Using a TTP or Certification Authority

$E_X()$ is an encryption process using the key X and applied to the data within the brackets.

H() is a hashing process applied to the data within the brackets.

DMC1 asymmetric keys : S1-private, P1-public.

DMC2 asymmetric keys : S2-private, P2-public.

hd - header containing necessary information, such as time stamp etc

1. DMC1 → TTP : (hd, REQ_{CERT1}, sig) where sig = $E_{S1}(H(hd, REQ_{CERT}))$
2. TTP → DMC1 : ($CERT_{DMC2}$)
3. DMC1 checks TTP signature on the certificate and the validity of the certificate, if both are correct it will cache the certificate.
4. DMC1 → DMC2 : (hd, REQ_{CON}, sig) where sig = $E_{S1}(H(hd, REQ_{CON}))$
5. DMC2 examines the request and attempts to obtain DMC1's public key

   DMC2 → TTP : (hd, REQ_{CERT}, sig), where sig = ES2(H(hd, REQ_{CERT}))
6. TTP → DMC2 : ($CERT_{DMC1}$)
7. DMC2 checks the TTP signature on the certificate and then the validity of the certificate, it then checks the validity of the DMC1 REQ_{CON}. Within REQ_{CON} there will be a minimum set of requested security services, if DMC2 can agree upon a common set of security services it sends an SSid with a generated secret key S_S, else DMC2 will send back its minimum set of security services for renegotiation.

IF common-set = TRUE **THEN**

DMC2 generates a symmetric key S_s

DMC2 → DMC1 : (hd, $RESP_{con}$, $SSid_{con}$, $E_{PI}(S_s)$, sig),

where sig = $E_{SI}(H(hd, RESP_{con}, SSid_{con} E_{PI}(S_s)))$

where $RESP_{con}$ contains the negotiable variables

ELSE

DMC2 → DMC1 : (hd, $RESP_{con}$, sig)

where sig = $E_{SI}(H(hd, RESP_{con}))$

$RESP_{con}$ contains DMC2's minimum security requirements

ENDIF

8. In the case of renegotiation, DMC1 will again submit a REQ_{con} if it can meet DMC2's minimum requirements.

DMC1 presents the SSid inter-domain association ($SSid_{con}$) whenever it requires a service from DMC2. If confidentiality is not required for DMC to DMC communication then the E_{ss} process can be ignored. The security services required by the DMC's security policy may require notary services, in which case they might have to be provided by a TTP which would then be used in an on-line or in-line function. The above protocol does provide proof of origin in that the signatures that the DMCs produce can be used at a later time as evidence for a DMC's part in the communication.

Access to inter-domain communications is dependent upon a user's right to those communications. The user's rights are determined via their entry within their LSS's SMIB, which is determined by the organisation's security policy and security manager. It may be

the case that certain users will not be allowed access to inter-domain communications because it is not necessary as part of their job description and they would therefore be tying up resources allocated to other users that do require the service. In the same fashion as many organisations organise phone privileges allowing or denying, local calls, national calls and international calls dependent upon need.

The co-ordination and synchronisation of management keys within the security domain is the responsibility of the DMC. The DMC ensures synchronisation by regular distribution of LSS public-key certificates to all LSSs or their respective entries within an X500 directory if one is being used. The certificates are distributed in off-peak hours so that system performance is not adversely affected. The DMC may demand the generation of new keys from its hosted LSSs according to the security policy of the domain. The time for which keys may be valid will be set at convenient intervals dependent upon specific security policies and noted within the distributed certificates. The interval for generation and distribution could for example be weekly, allowing the distribution of the certificate list to be accomplished on Sunday nights, which in all likelihood is the quietest night in most organisations.

4.4 Local Security Server

The local security server provides the framework for secure communications across the LAN and has a part in all of the LSU's activities. The LSS can be thought of as a file server and any references to files will be examined by the LSS. It enforces the protocols to be used as indicated by the DMC and subsequently entered into the LSS's SMIB. The LSS treats the DMC as a TTP and it is envisaged that the DMC should be a physically secured

machine to which access and the obtaining of system privileges is restricted to minimal personnel. The DMC should be solely controlled by the security section of the organisation.

The LSS provides secure services such as secure file transfer, notary services, etc, to the LSUs that it hosts. For secure communication outside of the security domain the LSU contacts the LSS which then contacts the DMC on the LSU's behalf.

4.4.1.1 Protocol for LSS to LSS communication

$E_X()$ is an encryption process using the key X and applied to the data within the brackets.

H() is a hashing process applied to the data within the brackets.

LSS1 asymmetric keys : S1-private, P1-public.

LSS2 asymmetric keys : S2-private, P2-public.

hd - header containing necessary information, such as time stamp etc

1. LSS1 receives request from a hosted LSU.
2. LSS1 $\rightarrow$LSS2 : (hd, REQ_{CON}, $E_{P2}(S_S)$, sig), where sig = $E_{S1}(H(hd, REQ_{CON}, E_{P2}(S_S)))$

   S_S is the symmetric key set up for connection between LSS1 and the LSU.
3. LSS2 checks the digital signature using the distributed LSS public key list, see section 4.7.2, examines the REQ_{CON} against information within its SMIB.

   - **IF** access granted **THEN**

     LSS2 $\rightarrow$ LSS1 : E_{SS}(hd, $SSid_{CON}$)

     ELSE

     LSS2 $\rightarrow$ LSS1 : (hd, neg)

where neg may be a renegotiation of connection variables or an $SSid_{DEN}$ refusal of connection.

ENDIF

4.4.1.2 Protocol for LSS to DMC Confidential Communication

1. LSS receives a request for a connection to the DMC.
2. LSS → DMC : (hd, REQ_{con}, ES2(S_s), sig),

   where sig = E_{s1}(H(hd, REQ_{con}, E_{s2}(S_s))
3. The DMC checks the signature using the entries within its SMIB, and checks the variables chosen within the request, it and then checks the access rights of the request.

   - **IF** access granted **THEN**

     DMC → LSS1 : E_{ss}(hd, $SSid_{con}$)

     ELSE

     DMC → LSS1 : (hd, neg)

     where neg may be a renegotiation of connection variables or an $SSid_{DEN}$ refusal of connection.

     ENDIF

4.4.2 Domain Notary Services

The LSS will provide the notary services within a domain, specifically those of non-repudiation with proof of delivery and proof of origin. These are of particular importance within a working environment as decisions must be made on information that is communicated between the various employees of an organisation. A message received

containing mis-information can lead to the use of false information to make business decisions. For example, a devious employee could provide a co-worker with false information that may lead to a bad business decision resulting in financial loss. In an environment with proof of origin the information can be traced back to the originator and the necessary disciplinary actions taken.

Also, the ISO-OSI Basic Reference Model-Part 2: Security Architecture [8], defines two forms of a non-repudiation security service; non-repudiation with proof of origin and non-repudiation with proof of delivery. Therefore, for compliance with OSI it is necessary that these two services be provided by the DMC and LSSs within the 3-L architecture. Figure 4-5 below shows the provision of non-repudiation mechanisms within a local security domain.

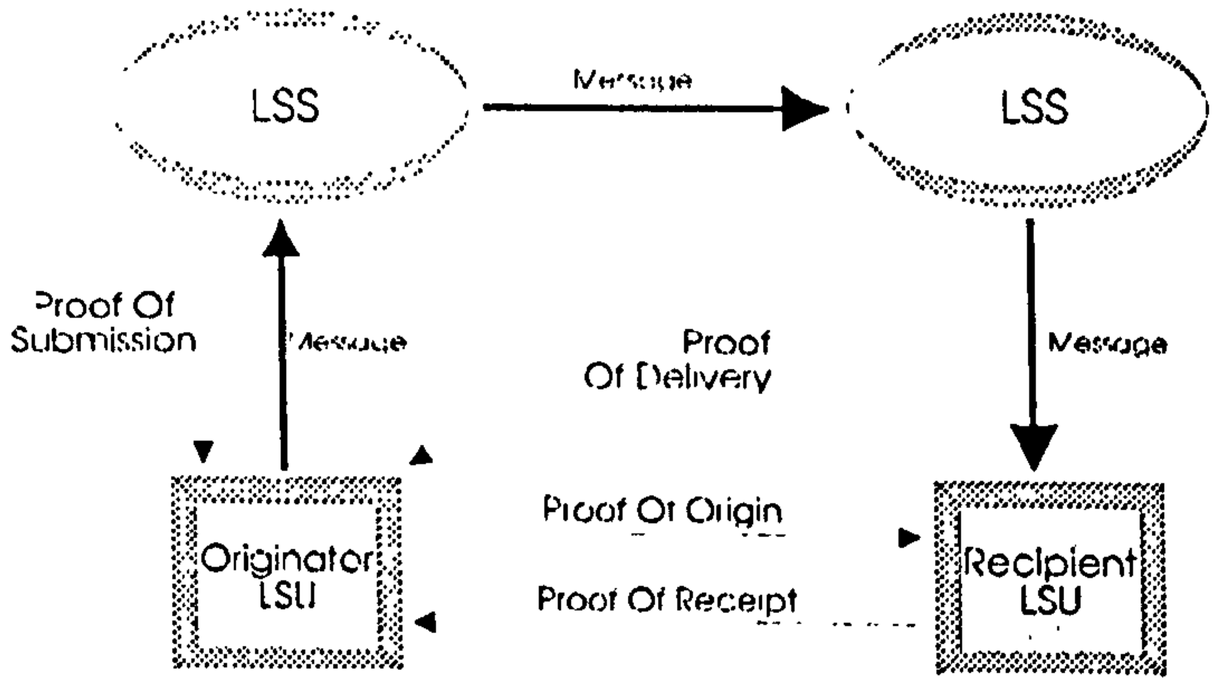


Figure 4-5 Non-repudiation Routes

Proof of origin (PofO) is provided throughout the system by digital signatures. Where proof may be needed at a later date the document is stored with its attached signature.

(hd, m, sig.) where $sig = E_{SO}(H(hd, m))$

where hd is a header that must contain the following fields as a minimum; time stamp, originators id, and hashing mechanism used. S_O is the Originators private key, and m is the message.

Proof of delivery (PofD) is generated by the delivering LSS once the LSU has acknowledged receipt of the message.

$$PofD = E_{SD}(H(hd, m))$$

hd is the message header, m is the message sent and S_D is the LSS's private key.

Other suggested non repudiation services that should be included are those of receipt and submission:

Proof of receipt (PofR) is generated by the recipient's LSU

$$PofR = E_{SR}(H(hd, m))$$

hd is the message header, m is the message sent, and S_R is the receipt

Proof of Submission (PofS)is generated by the senders LSS

$$PofS = E_{SS}(H(hd, m)),$$

hd is the message header, m is the message, and S_S is the LSS's private key

Non-repudiation services can be requested by users whenever a transfer of data takes place, the proofs are kept by the user in case of need against false claims. Proof of origin is always implemented within the 3-L architecture as the authentication of entities within a distributed environment is one of its main tasks.

User A wishes to send a memo to user B but it is a memo that B must receive for A to keep their job. Therefore, A invokes its LSU with a request for the non-repudiation service of receipt, proof of receipt implies proof of delivery and submission. User A may retain his proof of receipt for use if B denies getting the memo.

4.5 Local Security Unit

The local security unit is situated upon the user terminal, it is concerned primarily with the confidentiality of data and the authentication of users. Before we discuss the local security unit the use of dumb terminals in networking needs to be addressed. Due to their lack of processing power and therefore their inability to maintain confidential communications between themselves and their host, they provide a small security risk. In the case of dumb terminals it must be assumed that the physical links between the terminal and the host is secured, either through cryptodevices [56] at either end of the connection, or physical impediments, for example concrete encasement of the line.

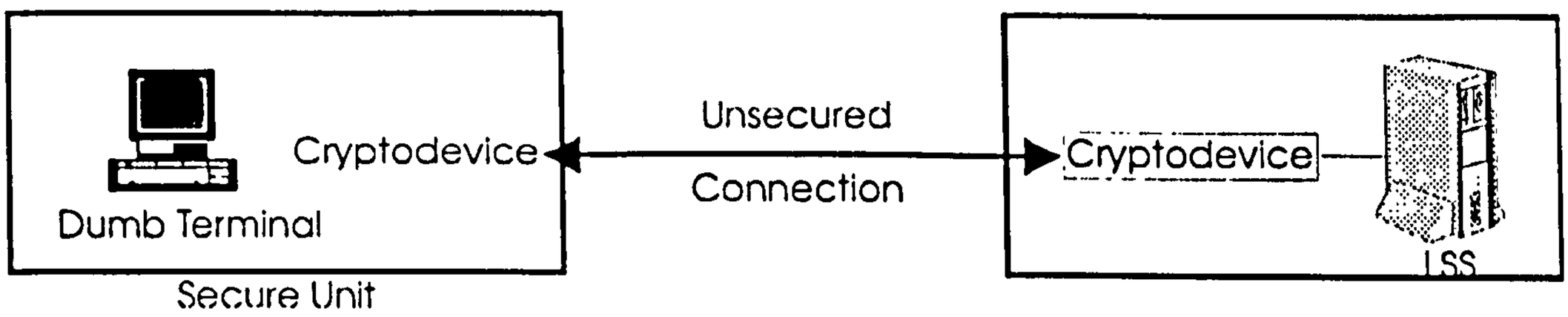


Figure 4-6 Secure Dumb Terminal

In the event of dumb terminals being used within the 3L architecture the LSU will reside on the host and the security methods for confidentiality mentioned above must be relied upon

to provide the confidentiality of data communicated between the terminal and the host, the use of cryptodevices for confidentiality could also be used to obtain a data integrity service. The alternative to the use of cryptodevices is to limit the services that dumb terminals can access, denying them any access to sensitive information. This would be accomplished through restrictions indicated via the security profile for the LSU within the LSS SMIB.

However, with the downward spiral of the cost of processing power, and the movement towards distributed processing it can be assumed that most recently procured user terminals upon LANs will not be dumb terminals, with the most common approach being the use of PCs. How much processing power these terminals have is extremely variable and dependent upon such things as the computer purchase policy of the company, or the position of the user within the company. Therefore, the 3-L architecture loads these machines the least by limiting the necessary processing for security that must be accomplished upon them.

Within the 3-L architecture there are many LSUs within a domain, this division of processing power is a more efficient way of meeting the needs of users than through the use of a large central processing element. Most of the cryptographic processing will be present on the LSUs, it is therefore important that the LSU provides fast cryptographic functionality. The cryptographic functions settled upon in Chapter 2 were DES and RSA it is these processes that must be provided within a fast implementation. Of the two media, software and hardware, implementation in dedicated hardware generally runs faster. However, as mentioned in Chapter 2, hardware has a limited useful life time and replacement costs are much higher than for software. For every function implemented in

hardware a software function should also be provided. The software version may run considerable slower than the hardware but the fact that it is available in software, capable of being run on potentially any machine, is a good selling point and also necessary for complete compatibility.

4.5.1 User Authentication Within the LSU

The LSU is the access point of users to the system and it is therefore logical that it be at this point that user authentication take place. User authentication is usually conducted at several sites within an organisation's physical location. Even if there is no set policy for authentication it may still take place implicitly through the suspicion of people that work within the organisation towards strangers walking into offices etc. The points usually associated with the presence of authentication techniques are also those areas of traditional access control:

- entry to a building;
- entry to a room;
- access to a terminal.

Authentication at the first two boundaries are primarily concerned with the physical security of equipment, securing against theft or damage. The third site at which authentication takes place is concerned with the authentication of an individual attempting to use a service.

At the beginning of a user's session the OPENA of the LSU will be called by the systems login process via the CISS API. When this happens the LSU first establishes a secure

connection with its LSS, where the user identification data is stored. It then attempts to obtain an SSid authentication certificate, this certificate can later be used by the LSU if further identification is necessary, however, the certificate will be invalidated upon suspicious user activity.

4.5.1.1 Local Security Information Base

The LSIB is a temporary storage cache for information relevant to the specific user upon the data terminal such as the information that is required by the SSA of the LSU, once the user has logged off the LSIB is cleared. The clearing of the LSIB should be done thoroughly as any residual information such as cryptographic bit sequences representing keys may be retrieved by an attacker through scanning the storage media. Erasure of sensitive information stored by computer requires over-writing of the memory area it occupied. The National Computer Security Centre considers three over writes to be adequate for the job of deleting sensitive information.

4.5.1.2 Protocol for Confidential Communication Between LSU and LSS

At the start of any communication between the LSU and LSS it is necessary to negotiate a session key that can be used for the encryption of all sensitive data passing between the two levels, ensuring the confidentiality of all data passed between the levels.

$E_X()$ is an encryption process using the key X and applied to the data within the brackets.

H() is a hashing process applied to the data within the brackets.

LSU asymmetric keys : S1-private, P1-public.

LSS asymmetric keys : S2-private, P2-public.

hd - header containing necessary information, such as time stamp etc

1. LSU generates symmetric session key S_S.

2. LSU → LSS : (hd, REQ_{conn}, $E_{P2}(S_S)$, sig),

   where sig = $E_{S1}(H(hd, REQ_{conn}, E_{P2}(S_S)))$

3. LSS checks the signature sig by using the protocol described in Chapter 2.

   - **IF** sig = TRUE **THEN**

     LSS → LSU : $E_{SS}(hd, SSid_{conn})$

     ELSE

     LSS → LSU : (hd, $SSid_{DEN}$)

     {Termination of Protocol}

     END IF

The specified symmetric key can now be used to encrypt all communication between the LSU and LSS, or it can be used to encrypt only especially sensitive information. Once again this is policy dependent. However, it is advisable to not overuse encryption since even when symmetric cryptosystems implemented in hardware run in the range of tens of Mbits per second, any unnecessary activity needlessly slows down the system.

It can be assumed that the LSU who received the $SSid_{conn}$ for a requested connection using the secret key E_{SS} is the only entity with knowledge of that key, therefore providing implicit authentication via the use of the E_{SS} key.

4.5.1.3 Protocol for User Authentication

The suggested protocol for user authentication at an LSU is where implicit authentication of an LSU is not used.

$E_X()$ is an encryption process using the key X and applied to the data within the brackets.

H() is a hashing process applied to the data within the brackets.

LSU asymmetric keys : S1-private, P1-public.

LSS asymmetric keys : S2-private, P2-public.

hd - header containing necessary information, such as time stamp etc.

1. User starts up the user agent.
2. LSU challenges the user according to the authentication mechanism specified in the LSIB.
3. LSU sets up confidential session with LSS as described above.
4. LSU → LSS : (hd, E_{Ss}(INF), sig) where sig = E_{S1}(H(hd, E_{Ss}(INF))).
5. LSS checks the signature, obtains INF and then compares the information INF with that contained within its SMIB. In the event that it is unable to find the information within its SMIB, it may interrogate other LSSs in the assumption that the user attempting to login is in fact a non-local user.

   - IF INF = SMIB-INF THEN

     LSS → LSU : (hd, $SSid_{ACPT}$)

   ELSE

     LSS → LSU : (hd, $SSid_{DNY}$)

     {termination of protocol}

ENDIF

6. LSU informs user of successful login

The use of smart cards in the authentication process can help increase the relative security of the user authentication mechanism. A smart card can be used by the system to store a user's private key. Whenever the user attempts to access the system they must present the card with its encoded information and processing capability. Of course there would also be other authentication mechanisms but here too the smart card can be of use by storing the relative information required by these mechanisms for identification. The only thing that limits the use of smart cards at this time is the limited processing power that they can provide. However, increased processing power is only a matter of time and eventually all the information that is necessary to identify an individual could be stored upon a single card and more importantly be processed. Therefore, no vital information will ever need to be communicated off the smart card and any necessary cryptography that needs to take place can take place on the card.

Once a user has been authenticated the user's security information such as his asymmetric key pair and security preferences that are stored within the LSS's SMIB can be down loaded into the LSIB. The users asymmetric key pair is used to provide personal accountability.

4.6 Common Services

These services are provided through the co-ordination of all three layers of the 3-L architecture.

4.6.1 Access Control

Access control is determined by information contained within the SMIB segment - extended access control. There is a segment entry for every user, LSU, LSS and DMC contained within the collective SMIBs of the domain. The DMC maintains access information upon inter-domain communications while the LSS maintains access information upon LSUs, files and passive devices.

An access control segment will maintain at least four attributes per entity.

1. Distinguished Name.

   The label assigned to the entity by which it can identified.

2. Access Rights.

   A discrete measure of the access capabilities of active entities. Each entity within the 3-L architecture will be given access rights, the greatest access rights being granted solely to security administrators.

3. Required Access Rights.

   The access rights required by an entity to gain access to the named entity.

4. Authentication Details

   Any authentication requirements that are necessary for access to be granted, such as additional authentication mechanisms, use of smart cards, local environment conditions etc.

The authentication details and access right requirements form the conditions that must be met for access to an entity to be granted to an active entity.

4.6.2 Logging and Auditing Security Relevant Events

Auditing is the recording of specific events for future reference and is a necessary part of any security system, since it provides a means for intrusion detection and accountability. The simple knowledge that a system has auditing capabilities can be enough to keep authorised users from abusing their privileges. It can be used as a historic record, providing a review of past events, or as a real-time event log used in the examination of user activity patterns. Audit logs have traditionally been used as a record of past events that can be used to track any misuse of the system by hackers [57]. To this end it has been used in the determination of which user's accounts a hacker uses when on the system. This is accomplished through the inspection of account activity and detection of discrepancies, such as periods of activity outside of work hours (a common indicator of illegal user account activity). Auditing also helps in the discovery of what an attacker is doing while illegally logged onto a system, giving the system administrator the ability to assess any damage caused by the attacker.

The DMC will act as the auditing collection point for the 3-L system. Auditing is accomplished by all three levels of the 3-L architecture, but the majority of analysis will be provided by the DMC or a specialised audit analysis machine.

The DMC has the sole responsibility of logging the inter-domain communication information some examples of potential auditing information may be:

1. attempted association;
2. incorrect termination of association;
3. transport errors.

The LSS has the responsibility of logging the domain communication as well as the object manipulation and user information some examples of auditing information are:

1. user login events;
2. object manipulation;

   for example

   creation, deletion, modification of file objects;
3. attempted violations of object security.

The local security unit has the responsibility of operating environment relevant information such as the use of specific operating system (OS) commands or discretionary functions that the security administrator feels require auditing. Examples of LSU audit information are:

1. use of volatile commands;
2. CPU time;
3. hours of activity.

The DMC should be capable of loading all auditing information during off-peak and carrying out a detailed off-line analysis of the days events looking for any activity that matches previous patterns of unsecure activity. Such examples may be:

1. repeated login failures;
2. login of users from unusual physical locations;
3. activity at unusual hours through the day;

which can be done using an expert system [58]. It may not always be necessary to do this and it should therefore be at the discretion of the security administrator. The level to which

the analysis of auditing information goes down to (the lowest level being OS commands of the LSU) needs to be discretionary as user activity on a LAN can be large and therefore the auditing information can become unwieldy. The security administrator who is suspicious of some strange activity must be capable of requesting a full system audit where analysis does go down to the lowest level.

The monitoring agent is absent from the LSU because it is envisaged that the OPENA with its API will interact with the existing operating system to provide the necessary low level auditing information.

4.6.3 Generation, Storage and Distribution of Cryptographic Keys

The DMC and LSSs act as certification authorities within a security domain; each LSS certifies that a specific user and LSU has a specific public key and distributes the certificate within the domain. The DMC certifies that each LSS has a specific public key and releases the list of LSS certificates at regular intervals. The list may be stored in one distributed database as suggested in the X509 recommendations, separately by each LSS or presented upon request by a level entity.

All LSSs within a domain maintain, or have access to the certificates that certify the current public keys of all other LSSs and there local users. If a user requests a confidential communication via its LSU with another user within the same domain they request the service from their local LSS (LSS1). LSS1 obtains the certificate for the second LSS (LSS2), as indicated by the address passed to it by USER1 in their request. LSS1 contacts LSS2 and obtains the certificate containing USER2's public key, this information is then

passed back to USER1. Negotiation of a session symmetric key can take place confidentially and the communications between USER1 and USER2 occur afterwards using the symmetric key decided upon within the negotiations. The LSUs do not now need the intervention of the LSSs for further communication since the LSUs OPENA uses the secure channel of communication for application communication.

The problem of key generation and distribution is most acute at the initial set-up of a system when using the 3-L architecture and at proceeding additions of new level entities such as LSUs and LSSs.

The addition of new users can be accomplished using the same process that the existing operating system uses. The CISS API will interact with the new user account creation process and can either be called by it, or call it after a request by a security administrator. The normal OS required information would then be gathered along with the additional information that CISS requires for all new accounts. The necessary CISS information will include the length of modulus for generation of a key pair, the services that the user will require or have access to, the user's access privileges, etc.

Adding a new LSU will require the registering of the LSU with the local LSS. For maximum security the LSS key pair would have to be generated at the LSU and then the public part keyed into the LSS by a security administrator. The security administrator would also key in the LSS's public-key, thus allowing the LSS to communicate confidentially with the new LSU. Adding a new LSS requires similar steps as that of adding a new LSU except the keying in of public key parts is done at the LSS and DMC levels.

Once an initial set of keys has been initialised within the 3-L architecture the continuing management of the keys can be done in-line, that is over the network. Keys can be generated and distributed using the current keys to confidentially communicate new keys. Every level of the hierarchy is capable of generating cryptographic keys eliminating the necessity for communication of private keys over the network.

Chapter 5

5. Hardware Implementation of CISS Security Mechanisms

This chapter describes a new board that has been developed for Sun microsystem computers with SBus capability and is meant for use in a CISS prototype. The chapter begins by presenting the options available for hardware connection to Sun systems followed by a description of the desired qualities connected hardware should possess. The board was built in three parts which are described: a generic SBus adapter board; a processing board and finally the hardware cryptosystems.

5.1 Introduction

The cryptographic algorithms involved within the required security mechanisms of CISS (see Chapter 3), and therefore necessary as part of the three level architecture, are complex and processer intensive. However, they are very important and play an essential part in authentication, confidentiality, integrity and proof mechanisms. Any usable security system, such as the one being developed through the CISS projects of the Network Research Group, must provide these services and also conform to a set of simple rules that help designers construct effective products.

- The product must be easy to use:

  how easy a system is to use is mainly determined by the interface that is presented to the user and the study of human computer interaction has led to the belief that graphical interfaces are best; providing the user with a multitude of windows, buttons, and menus. The plethora of *Graphical User Interface* (GUI) design kits and their popularity, such as the Microsoft Visual range [59], [60], confirm the direction user interfaces are taking. By designing an apparently simple GUI it is possible for the user to accomplish complicated tasks easily.

- The product must be of value to the user:

  the value or usefulness a user finds in a product depends upon the application. The system that is in development within the Network Research Group is CISS, designed to be useful to people requiring data protection in a static environment i.e. upon a single computer system or in an active environment such as the communication of data across networks.

- The product must run efficiently and quickly:

  the product must run efficiently upon most machines in use at present, in that it should make good use of memory and available resources. There is little point in producing a system that can only be run upon a top end machine (as there will be few buyers). The speed at which an application runs is very important as most people that work with computers are familiar with the frustration associated with waiting for a system to boot up. Therefore, it is necessary to optimise a system so that it produces as little waiting time as possible upon computer systems that have a variety of processing capabilities.

In CISS the most obvious point at which processing time is a potential problem is during the use of security mechanisms that require cryptographic processes and it is here that a hardware implementation would benefit the system's speed most. It is also at this point that the differences between old machines and new machines (i.e. their comparative technological levels and processing capabilities) become greatly exaggerated as they perform intensive processing of the complex algorithms. To combat both these problems hardware was designed to remove the bulk of cryptographic processing from the machines themselves, and place it within custom boards that can provide all the cryptographic functionality required by CISS.

Shepherd [61] suggested that a complete implementation of the CISS agents should be provided within hardware. This is unlikely in practice as such a solution would be of considerable expense to any organisation interested in the use of CISS, due to the high level

of expenditure that would be necessary at initial purchase and any custom hardware built to provide the entire CISS functionality would very soon become outdated by the rapid advancement in chip fabrication and consequently processing power. It would be much better to provide modular security devices that may be pieced together around existing systems, in effect creating a suite of security products, and strengthening the modular concept of CISS and its additive potential to existing systems.

This chapter describes a card (named ENCDEC) developed for the Network Research Group's CISS projects to help speed up the system by making available hardware implementations of necessary cryptographic algorithms.

5.2 Sun Interfaces

The implementation of the CISS concept under discussion was developed upon a Sun microsystem SPARC workstation in UNIX. Sun provide three interfaces for the connection of hardware to their systems.

- RS 232 [62];

  The RS-232-C standard is a technical specification laid down by the Electronic Industries Association for the interface between *Data Communications Equipment* (DCE) and *Data Terminal Equipment* (DTE), for example the RS 232 interface would be used in the connection of a modem to a personal computer.

The standard can be used for communications within the range of zero upto 2500 bytes per second and it can be used in an asynchronous or synchronous mode for serial data communications.

- SCSI [63];

Small Computer Systems Interface is ANSI standard X3.131 wherein a peripheral bus and command set are defined. SCSI is an eight bit I/O bus that allows the addition of peripherals such as disk drives, tape drives and printers with minimal disruption to the existing host system. The SCSI bus provides both asynchronous and synchronous handshaking. Asynchronous operation requires a handshake after every byte transferred, synchronous operation transfers a series of bytes before each handshake giving a maximum transfer speed of 5 Mbytes.

The interface allows up to eight devices to be connected, of which, any two (an initiater and a target) may use the interface to transfer data between themselves at any given moment. The SCSI architecture uses a bus arbitration scheme that awards priorities to the different devices. The SCSI bus has eighteen signals; nine for control, and nine for data.

- SBus [64].

This is a proprietary connection system similar to the Local Bus system in PCs. The SBus specifications were published by Sun in late 1989, detailing the I/O bus developed by Sun microsystems themselves. I/O devices such as video cards, cards utilising digital signal processors, and graphics accelerators have migrated to this

connection system due to its high performance and open architecture. However, the SBus is a new I/O connection system and with the physical size of a single SBus board being approximately 8cm by 14cm it is best suited for VLSI technology.

5.2.1 Choice of Interface

From the interfaces available on the Sun system the ENCDEC device was built for connection to the SBus, for the following reasons.

1. The SBus is internal to the system box which potentially provides the best physical security because there are no electronic components immediately accessible (including wires). Most system boxes can be locked to prevent anyone opening up the system box, for reasons of theft or vandalism etc.
2. The SBus can attain speeds up to 25MHz giving it the ability for very high speed applications and a peak datarate of 100MB per second at 32 bits. The SBus also has extended transfer capabilities allowing it to transfer data in 64 bit words giving it a maximum capacity of 200MB per second.
3. SBus devices have the ability to obtain direct memory access (DMA) by taking control of the bus and completing transfers directly to/from memory.
4. The SBus has a 32 bit data bus as opposed to the 8 bit bus of SCSI and the serial line of RS-232.

5.3 SBus

5.3.1 Physical Aspects

The SBus is a board level expansion bus that is meant for use within areas that have limited space available. It was designed for components that are surface mounted quad flat packages. The SBus uses 82 signals for information transfer and control, with an additional 14 power and ground connections, giving 96 connections in all.

The SBus uses a high density 96-pin connector; SBus cards come in two sizes *single* or *double* width, with one or two connectors respectively.

Type	Total Length	Total Width
Single	146.7mm	83.82mm
Double	146.7mm	170.28mm

Table 5-1 SBus Physical Dimensions

The SBus is located within the system unit and therefore has strict restrictions placed upon its physical dimensions and those of the components placed upon the board.

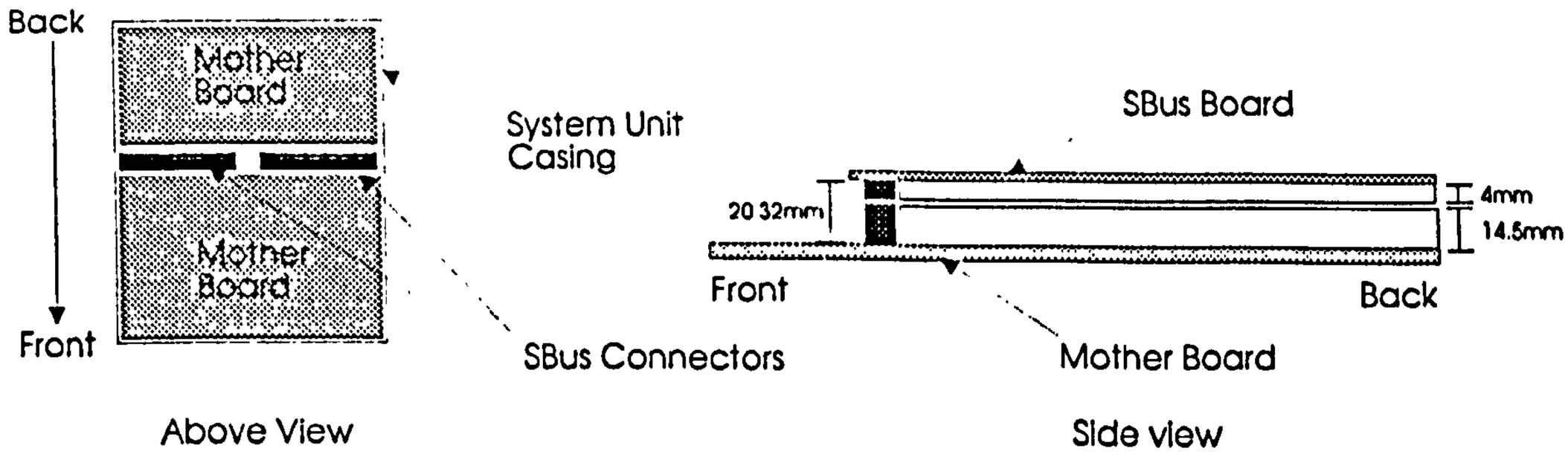


Figure 5-1 System Unit and SBus Component Gap

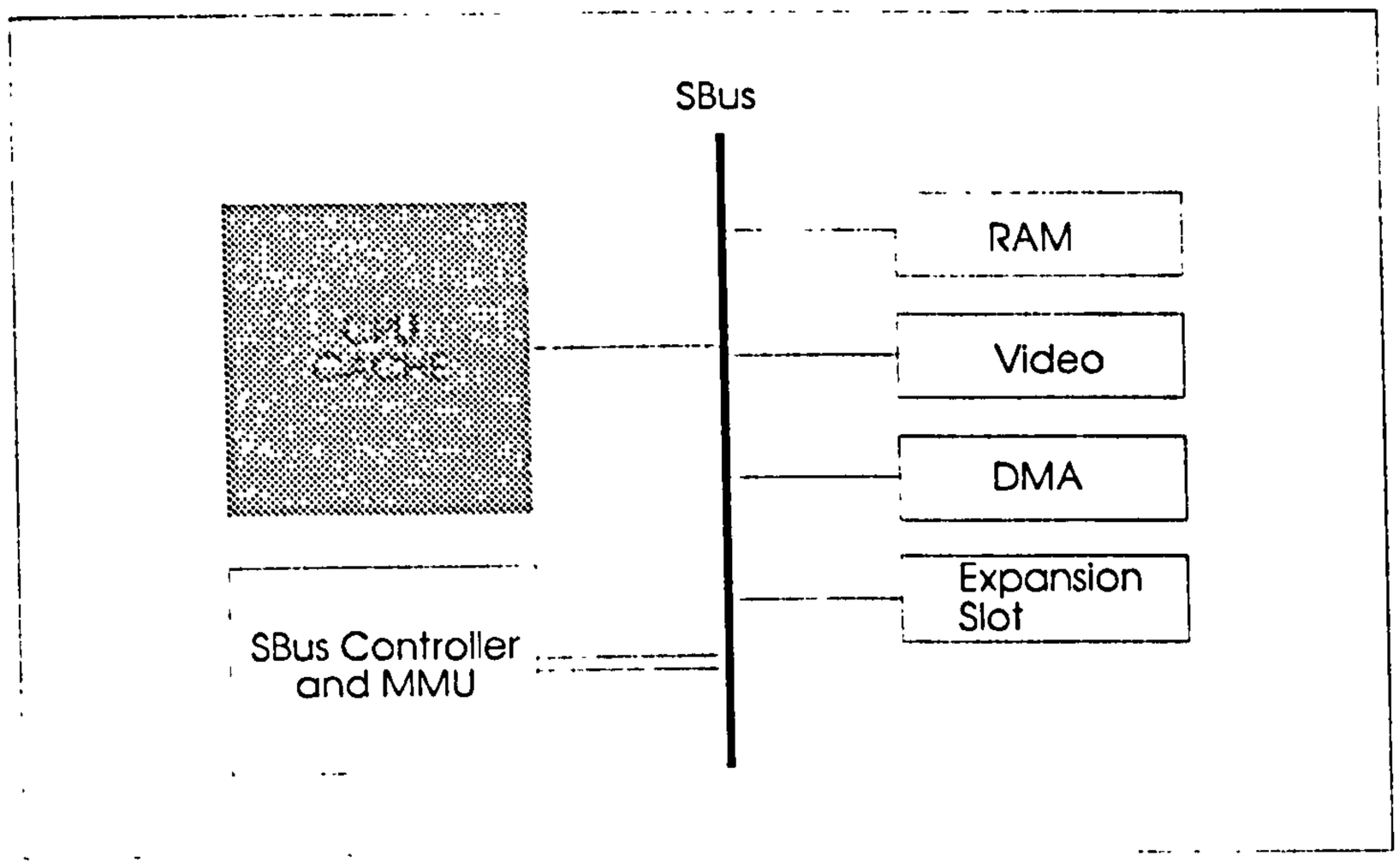


Figure 5-2 SBus Symmetric System

5.3.2 Direct Virtual Memory Access

Devices connected to the SBus place a virtual address on the bus, prior to any transfer of data, which is translated by the memory management unit of the system into a physical address. This is known as *Direct Virtual Memory Access* (DVMA), which is a slight modification of *Direct Memory Access* (DMA). DVMA helps simplify the operating system and software memory management tasks of a computer system. When virtual memory addressing schemes are used, an address translation is required to convert the virtual address into a real/physical address. The functional unit within the Suns that does this is the SBus controller and memory management unit (MMU), see Figure 5-2.

The direct accessing of any memory location by devices connected to the system, which DVMA allows, is a powerful memory access scheme that allows connected devices to obtain information they require the moment they require it. There is no need for buffering

information (which can greatly slow down an interfacing scheme) between the device and the system's memory.

5.3.3 Masters and Slaves

Boards that are built to connect to the SBus come in two varieties *masters* and *slaves*. A master device on the SBus will request control of the SBus from the SBus control unit; it is the job of the SBus control unit to arbitrate between requests from masters that wish to use the bus, and to prevent overuse by any one device. Once the master device has control of the bus it is then in control of the reading and writing of data to and from system memory through the use of the system's memory management unit. Once the master has completed its transfer to/from memory it relinquishes control of the bus, making it available for other devices. A slave device connected to the bus waits until a bus master decides to read data from or write data to it.

Master devices can maximise the throughput of data transferred across the bus. They are capable of doing this because they can operate upon data that they have previously read into their own memory, and then request control of the bus so that a result may be written back to system memory. There is no need for the system or SBus controller to check on the device's state by polling it, an action that would be necessary if the device was an SBus slave. It is therefore clear that an SBus master can obtain the best speeds due to its potential to take control of the SBus when it is needed. However, because a bus master requires greater functionality than a slave when interfacing to the bus, the circuitry of the device gets a little more expensive and slightly more complex but these negative aspects are easily compensated for by the increased capabilities of the device.

5.3.4 Bus Cycle

As with any bus, there are only two activities that take place across it, the reading and writing of data to or from memory. Irrespective of whether the memory happens to be a device register or a location in RAM, all locations are memory mapped and specified by a single physical address. The basic read and write SBus cycles can be found in Appendix B. The SBus uses a data word of 32 bits and a big-endian addressing scheme, meaning that the significance of bytes in a word decreases as the address of the bytes increase. Data transfer can be conducted using data of length 8, 16, 32, or 64 bits during a single cycle. When 64 bits is transferred the master device must time-multiplex the Rd, Siz(2:0) and PA(27:0) lines for transferring the most significant 32 bits of the double word.

Data transfer can occur in one of three ways across the SBus.

1. Standard.

   The normal transfer is as described previously and follows the basic read/write cycles shown in Appendix B. The basic cycle for data transfer is split into two parts the translation cycle and the slave cycle.

   The translation cycle begins when the master device asserts its Bus Request (*BR) line and on detection of a Bus Grant (*BG asserted) signal, initiated by the SBus controller, places the virtual address on the data lines which are subsequently sampled by the SBus controller. The master must also assert the data size lines (SIZ(2:0)) and the Read Write indicator (Rd), then it drives the data lines D(31:0) if it is writing to the system or another SBus device.

The bus controller asserts the Address Strobe line (AS) once it has made the address translation, indicating the beginning of the slave cycle. It then places the physical address on the address lines (PA(27:0)) and asserts the Slave Select (SEL) line to indicate which SBus slave is to be accessed. If information is being read by the master then the slave must place the information on D(31:0), keeping it stable until RD falls.

2. Atomic transfers.

   When the master does not release the BR line at the SBus controllers acknowledging BG assertion, it is indicating its need for an atomic transfer. In this mode the master retains control of the bus for two or more bus cycles. This capability allows for more efficient transfers of large amounts of data.

3. Burst mode transfers.

   Burst mode transfers allow the master to transfer multiple data words within one bus cycle. The master does this by transmitting the base address for the data at the start of the transfer and then relies upon the slave to place the data in the order it receives it. Burst modes can be used in two/four/eight and sixteen word bursts.

It would be beneficial for any device built for connection to the SBus if they could transfer data in all of the three methods above. Methods two and three give a greater bandwidth and throughput than one, increasing the speed at which a device can access any required data.

5.3.5 Desired SBus Capabilities

From an examination of the SBus characteristics described previously, it was decided that the following qualities were necessary when implementing the ENCDEC card.

1. The ENCDEC device would take up one SBus slot using a single SBus card. This was determined by price as well as the convenience of having the remaining SBus slot free within the available system for envisaged future upgrades.
2. The ENCDEC device would be a bus master rather than a slave, giving it the best possible throughput and potential processing efficiency.
3. The ENCDEC device should be capable of transferring data in any of the three modes described in section 5.3.4, giving it the greatest possible data transfer rate for the system.

The next decision made was on the method of construction for the device. Although the SBus is meant to be used by devices with surface mount technology, it was decided due to pressures of cost and available equipment that the prototype board should be constructed using wire wrap and DIP (Dual In Parallel) CMOS technology chips. Because of the size of DIPs it was also decided that the device would have to be constructed as a prototype in two halves, one half would be the interface board that would connect to the Sun workstation and the other half would provide general processing capabilities to allow custom cryptographic chips to be easily incorporated.

5.4 Interface Board

5.4.1 SBus Interface

The necessary logic for the connection of a device to the SBus is rather complicated. This reason has led to products being developed that can aid the hardware developer in the provision of interface logic for SBus devices. One such product is the L64853A SBus DMA controller [65], produced by LSI Logic. For a developer to produce their own circuitry to interface to the SBus would require considerable effort and expense in the fabrication of the needed circuitry. A custom design would however be beneficial for a finished product as there would then be less off-the-shelf chips available for the construction of a rival product.

The L64853A Internal block diagram can be seen below.

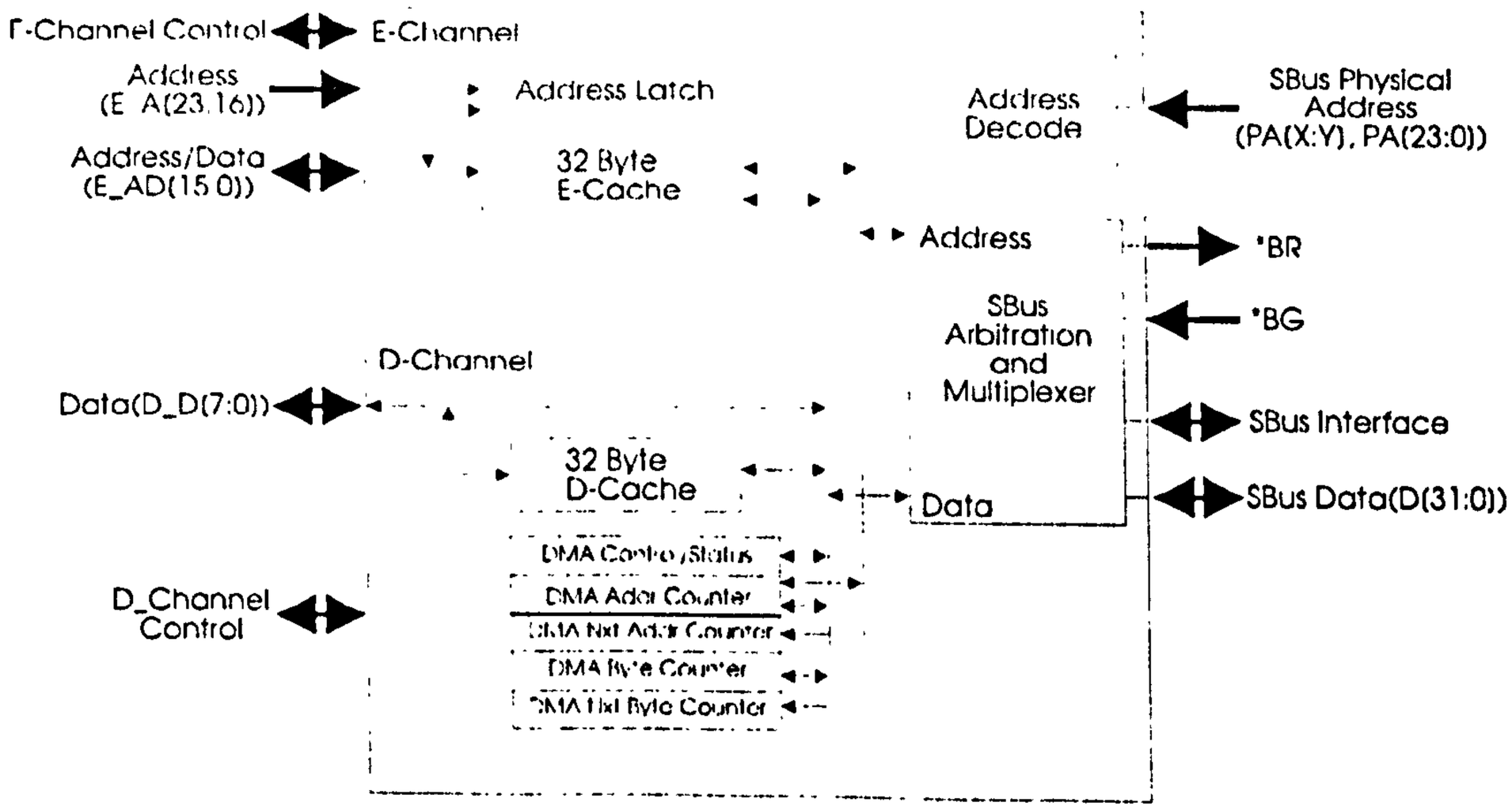


Figure 5-3 L64853A

As can be seen from Figure 5-3 the DMA controller has one interface to the SBus and two interfaces to devices connected to it, the E-channel and the D-channel. The E-channel has

16 bit access to the DMA controller and the D-channel has eight bit access. The key features of the L64853A controller are:

1. it supports byte, halfword, word or four word burst transfers;
2. it has a maximum transfer of 8.3 MBytes/second at 25 MHz, assuming an SBus latency of up to 40 clock cycles;
3. provides buffering of data unpacking or packing the data into bytes or half words;
4. has bus master capabilities for data transfer.

These features meet most of the desired SBus interface capabilities discussed earlier in section 5.3.5.

The E-channel allows 16 or 8 bit transfers to the DMA memory cache, with a higher priority than the D-channel. This means that if two requests for services come in from the peripheral channels then the E-channel will be served first. The E-channel supports DMA read and write operations in master mode and register read/writes in slave mode. The E-channel is programmed using external registers which provide byte counters, control/status etc.

The D-channel allows only 8 bit transfers to the DMA memory cache, as with the E-channel SBus read and writes are conducted in bus master mode whereas register read and writes are in slave mode. The DMA controller provides registers for controlling the DMA actions; a control status register, address counter, next address, byte counter, and next byte counter.

Because the DMA controller provides the registers for the D-channel, and use of the E-channel would require the addition of several external registers and control circuitry for 16 bit access to the memory cache, it was decided to use the D rather than the E-channel for connection to the processing board.

The D-channel internal registers are written to by the OS device drivers that control the input and output of user data to the card, the device drivers provided for the ENCDEC prototype are described within Chapter 6. When an SBus master writes or reads the controller's registers the controller is operating as an SBus slave. There is no way for a D-channel peripheral to read or write the DMA controller's internal registers easily.

5.4.1.1 Register Read and Writes (Slave Mode)

The internal and external registers of the D-channel peripheral are read and written in the following manner.

1. The SBus master that wishes to read or write to the D-channel registers places the virtual address upon the SBus lines D[31:0] which is then translated into a physical address by the SBus controller's MMU. The physical address is placed upon the PA[27:0] lines and a *SEL is asserted that corresponds to the DMA controllers SBus slot.
2. The DMA controller reads the PA[X:Y] lines and interprets whether an external or internal register is identified.
3. Access to external caches are done using a transfer size of one byte. There is no buffering of the data within the D-channel cache, it is written straight to the D-channel peripheral.

4. The DMA controller asserts *ACK1 informing the SBus controller that Slave-mode operation is over.

5.4.1.2 External D-channel registers

The only external register that is provided by the interface card is a transfer status register that informs the D-channel peripheral whether the interface board is in the process of reading or writing data. The external register is a latch (LATCH) that can be polled by the processing board.

5.4.1.3 D-Channel Read (Master Mode)

This is the transference of data from an SBus device to the D-channel cache.

1. The D-channel peripheral asserts the D_REQ line indicating that it wishes to make use of the DMA transfer facilities. The control/status register has been programmed by the device driver software indicating that a read operation is to take place, the device driver software would also have programmed the address register.
2. DMA controller asserts *BR to request control of the bus. When the SBus controller asserts *BG the DMA controller places the virtual address stored in the address counter register on the D[31:0] lines.
3. The SBus controller's MMU translates the virtual address into a physical address and places it on PA[27:0]
4. When *ACK2 is asserted the DMA controller reads the D[31:0] lines until all four words are transferred into the D channel cache.

5. The DMA controller then asserts *D_ACK and *D_WR and begins transfer of the data to the D-channel peripheral (the processing board in the case of the ENCDEC prototype).
6. The data in the D-channel cache is then written on the D_D[7:0] lines by the DMA controller until transfer of the cache data is complete.

5.4.1.4 D-Channel Write (Master mode)

This is the transference of data from the D-channel cache to the system memory or to another SBus device.

1. The D-channel peripheral (ENCDEC processor) asserts D_REQ to request a DMA transfer from the DMA controller.
2. DMA controller asserts *D_ACK and *D_RD in response to the ENCDEC processor.
3. The DMA controller reads data on the D_D[7:0] lines into the D_cache till it is full.
4. DMA controller asserts *BR and awaits the *BG signal from the SBus controller.
5. The DMA controller places the contents of its address counter register onto lines D[31:0].
6. The SBus controller's MMU translates the virtual address and places the physical address onto line PA[27:0].
7. The DMA controller outputs the contents of its D-channel cache onto the SBus D[31:0] lines.

To read in multiple blocks of 32-bits of data it is necessary to use the following procedure:

1. Load up the D-channel control and status register setting EN_CNT = 1.
2. Load the transfer size into Byte Counter
3. Load the starting address into the D-channel address counter register
4. Initiate the D-channel controller transfer
5. Once the byte counter reaches zero the DMA controller will assert the SBus *INTREQ signal signifying the end of the transfer.

5.4.1.5 Hardware Implementation of Master Mode Transfers

Bearing in mind the operation of the DMA controller during DMA transfers, the hardware implementation to facilitate these activities consists of a RAM chip, a ripple counter and various logic chips. The RAM gives a maximum storage capacity of 8K bytes, and acts as a memory for data, that is about to, or has just been, transferred along the SBus D[31:0] lines. For the duration of the DMA controller's assertion of its *D_RD or *D_WR lines, indicating a data transfer to/or from the RAM chip, the D_D[7:0] lines must remain stable; on the rising edge of the DMA controller's *D_RD/*D_WR signal the ripple counter's output is incremented once, and subsequently addresses the next byte to be accessed in the RAM chip.

The DMA controller provides the logic necessary for use of an external or internal (default) identification PROM more details of the PROM are given in the next section.

5.4.1.6 FCode PROM

Every device that connects to the SBus must have a *Programmable Read-Only Memory* (PROM) that identifies the device, if a PROM is not found at the base address of an SBus slot then the slot is ignored. The PROM may contain additional software written in Forth that allows the device to be used in the booting sequence of the machine or for self diagnostics. The self diagnostic functions and general debugging of a device can be done prior to reboot by placing the Sun in Open PROM Forth Toolkit [66] mode which uses the Forth interpreter in the unit's boot ROM, and running the boards FCode [67] routines directly through toolkit commands. The Forth toolkit is extremely useful and a relatively powerful tool, it can be used to access all memory locations of the Sun workstation. Best use of the tool for debugging is accomplished through Forth programs that can be written to run under the toolkit and conduct extensive debugging functions.

During a Sun station's booting up process the system probes all SBus slots and attempts to access the devices address 0, at which an FCode PROM should sit. The address range given over to the PROM should be in the range of 30-60 bytes for simple cards, but potentially the entire physical address range of the SBus slot can be available.

The structure of information within the FCode PROM is

- Header:
    - magic number;
    - version number;
    - length;
    - checksum;
- body (contains FCode program);
- end token.

The information stored within the FCode PROM of the ENCDEC card can be seen below:

```
FCode-version1
hex
   " encdec,josw"        NAME
   " encdec,1"           MODEL
   3 0 INTR
   MY-ADDRESS 10000 + MY-SPACE XDRPHYS
   c XDRINT XDR+
   MY-ADDRESS 20000 + MY-SPACE XDRPHYS XDR+
   4 XDRINT XDR+
   " reg" ATTRIBUTE
end0
```

- NAME defines a name by which the device can be matched with the correct OS driver.
- MODEL defines the model of the card for the manufacturer's purpose.
- INTR defines the priority level of the device's interrupt signal within the OS.
- The lines beginning with MY-ADDRESS are used to provide the address for the property reg that identifies where the device's registers are located. The first MY-ADDRESS segment identifies the DMA controller register, the second identifies the peripheral register.

The first two and final line are used by a tokenizer to produce a file that can then be downloaded to a PROM burner, and therefore do not exist in the FCode PROM. The tokenizer converts Forth source code into FCode binary format that can be read by the systems open boot ROM.

The PROM containing the FCode information is the FCODE chip found in Appendix B, Figure B-2. The active low input on the DMA controller ID_CS is pulled high using a 4.7k

ohm resistor indicating that an external ROM is to be used and not the default internal identification number of 0xFE810102. The SBus controller accesses the FCode PROM placing 00 on PA[X:Y] the register select lines of the DMA controller.

5.4.2 Concluding Remarks on the Interface Board

The interface board provides an SBus interface with DVMA master capabilities. It has the potential of serving multiple peripherals Along with the DVMA registers it has an 8-bit register for direct communication with the system. Up to 8 KBytes of data may be stored on the card a capacity that can be easily expanded, earlier versions of the card could store twice as much.

The interface board is a general purpose interface to the SBus and provides a quick and easy way of attaching a peripheral with general processing capabilities to it. The interface to the SBus interface board is simple and easy to use and requires only 4 control signals.

The Interface board connections can be seen below in Figure 5-4.

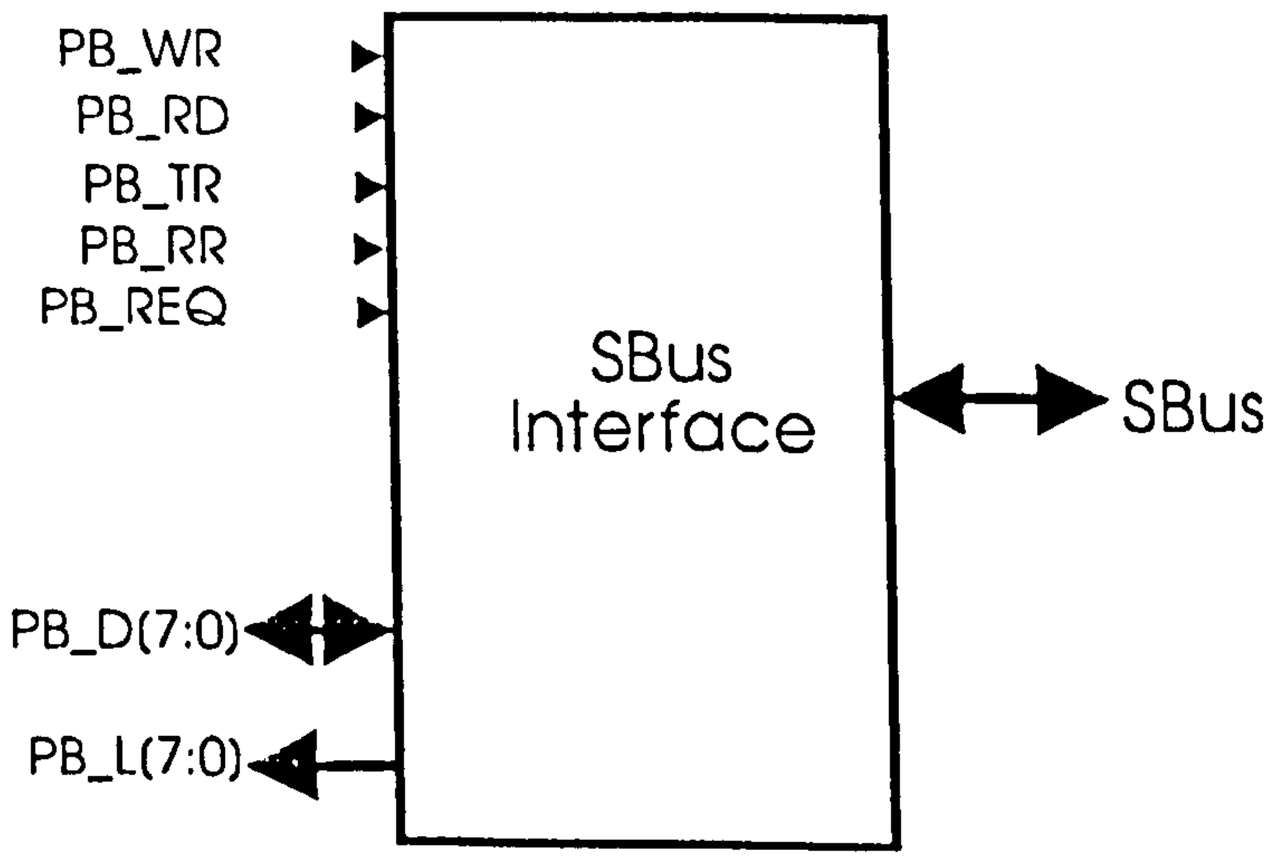


Figure 5-4 Overview of Interface Board

5.5 Processing Board

With the interface complete and ready for use it was necessary to decide how the information would be processed once transferred to the interface board. The original concept for this project was the optimisation of CISS by implementing the slowest parts of CISS within hardware, however, the project that was undertaken in conjunction with this, that would have produced a complete CISS prototype in software ran behind schedule and did not get a system running in time for a proper analysis to take place.

With the original concept of this project in mind it was decided that the processing board should be as adaptable as possible so that implementation of CISS functionality could be accomplished quickly and relatively easily. The best way of providing a general purpose platform for developing a prototype is to use a micro-processor and implement any of the functions that are required through firmware.

The processor chosen to be at the heart of the processing board (PB) was a Motorola 68000, the circuit diagram of the PB can be found in Appendix B, Figure B-3.

In its default state the PB is idle and awaiting data to be written to the interface board. While in its default state the PB polls the external register of the interface board, examining the state of PB_01. A low PB_01 indicates that there is no fresh data stored within the interface board's memory, A high PB_01 indicates that data is held within the interface boards memory ready for reading by the processing board. The processing board memory mapping for accessing the interface board can be seen in Table 5-2.

Address	Description
0x4280001	Ram Write
0x4300001	Ram Read
0x4400000	Register Read

Table 5-2 Memory Mapping for SBus Interface

All information is read from the interface board using one address to access the interface memory (INT RAM) during an *interface read*, at the end of a successful interface read the ripple counter (RIP) addresses the next byte of data, making it available for the next interface read. A single address is also used to access the interface memory during an *interface write*, at the end of a successful interface write the ripple counter addresses the next interface memory location ready for data to be stored on the next write cycle.

Data sent to the PB via the interface board is entered in a standard form (PB-packet), it is made up of a six byte header and a variable length data segment.

- Header
  - 1 byte- Function
  - 3 bytes- Function dependent
  - 2-byte- Message_length
- Message
  - Variable length data

Once the PB detects a high PB_01 it reads the first six bytes of data in the interface RAM, which make up the PB-packet header, and writes it to its on board memory at address

0x200000, 8500 bytes of memory are reserved for storage of the PB-packet beginning at address 0x200000.

There are currently six functions that are implemented upon the ENCDEC prototype:

1. load DES key;
2. DES;
3. load RSA key;
4. RSA;
5. multiplier load;
6. modular multiplication.

The *function byte* of the header informs the PB which of the implemented functions are to be used.

7	6	5	4	3	2	1	0
RSA	DES	KEYLD	MULT	DECRYPT	MULTLOAD	BEGIN	END

Figure 5-5 Command Word

The BEGIN and END bits are necessary for the DES cryptographic process and their use is described in section 5.7.4.

Once the header has been read into PB memory the message_length section is passed to an accumulator and the data section is loaded into the PB memory, beginning at address 0x200006.

The *function* byte of the packet header is then used to call the subroutine that implements the indicated function. The firmware for the PB can be found in Appendix B. Once the subroutine has completed its actions the data is ready to be written to the interface board in an interface write. Any modifications to the length of the PB-packets data portion will have been recorded in the PB-packets *message_length*. The *message_length* is loaded into an accumulator and used to transfer the correct number of bytes to the interface memory. The PB then requests a DMA access to the SBus from the interface board by asserting PB_REQ.

Once the information has been written from the interface board to system memory the PB returns to its default state.

The following sections 5.6, and 5.7 describe the implementation of the RSA (see section 2.4) and DES (see section 2.5) algorithms within the ENCDEC board.

An overview of the ENCDEC prototype can be seen below in Figure 5-6.

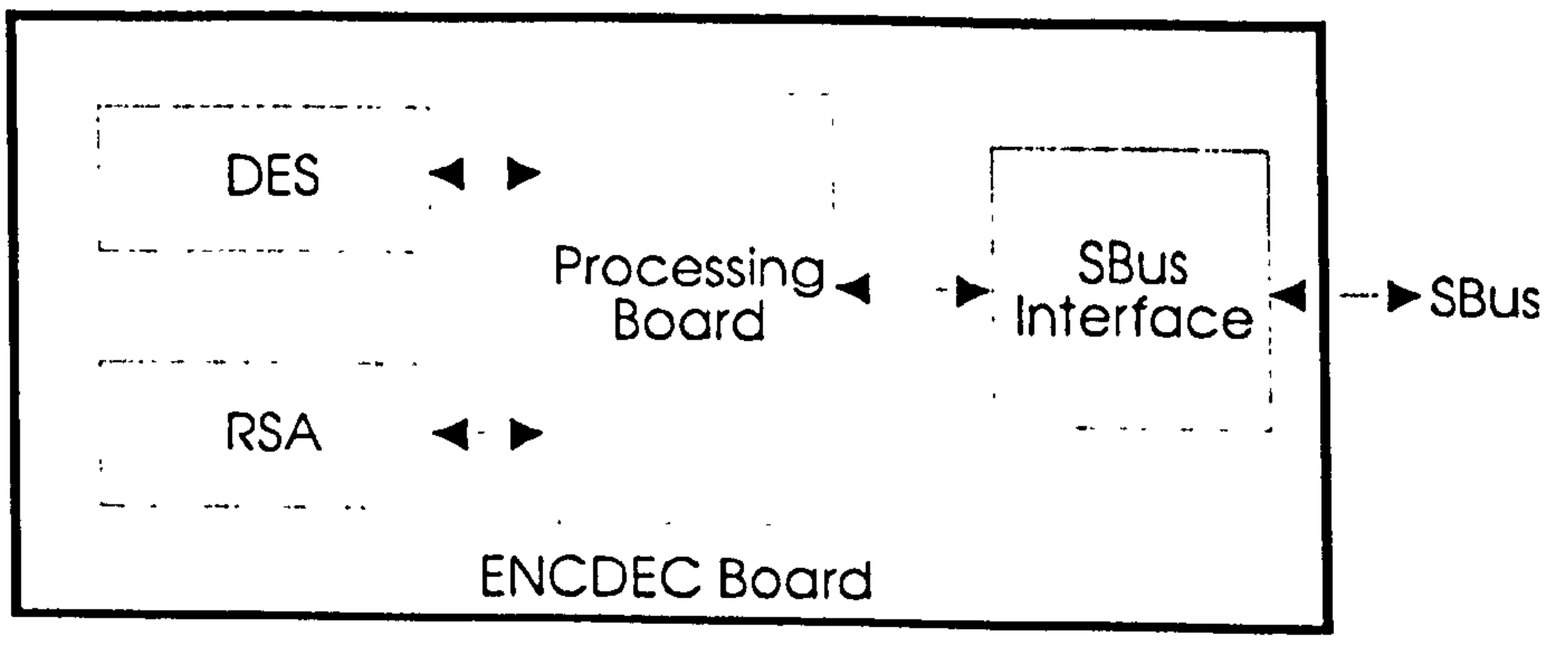


Figure 5-6 ENCDEC Board Overview

The *function* byte of the packet header is then used to call the subroutine that implements the indicated function. The firmware for the PB can be found in Appendix B. Once the subroutine has completed its actions the data is ready to be written to the interface board in an interface write. Any modifications to the length of the PB-packets data portion will have been recorded in the PB-packets *message_length*. The *message_length* is loaded into an accumulator and used to transfer the correct number of bytes to the interface memory. The PB then requests a DMA access to the SBus from the interface board by asserting PB_REQ.

Once the information has been written from the interface board to system memory the PB returns to its default state.

The following sections 5.6, and 5.7 describe the implementation of the RSA (see section 2.4) and DES (see section 2.5) algorithms within the ENCDEC board.

An overview of the ENCDEC prototype can be seen below in Figure 5-6.

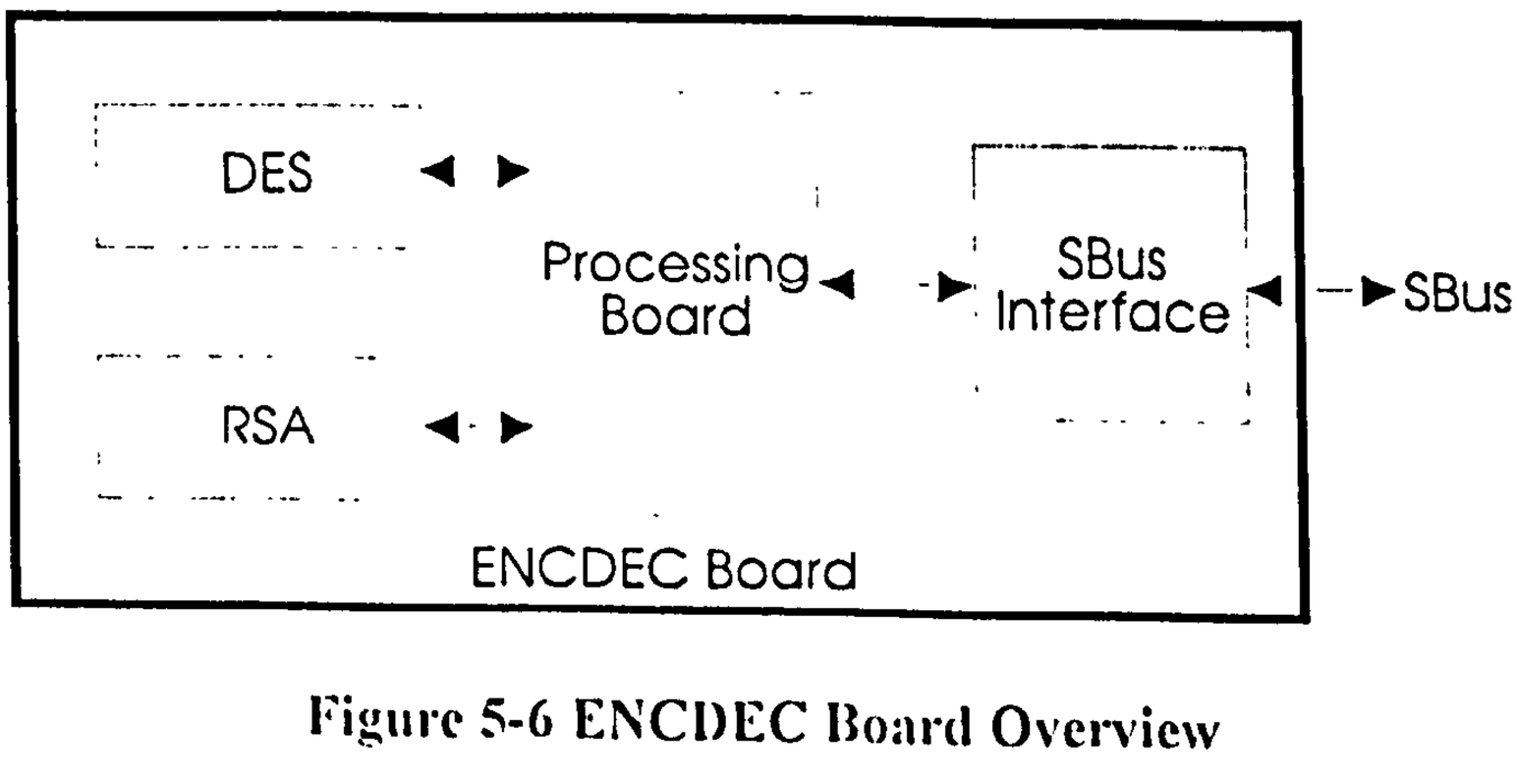


Figure 5-6 ENCDEC Board Overview

5.6 Hardware Implementation of the RSA Cryptosystem

5.6.1 Review of RSA Hardware

In section 2.7 there was a brief discussion on the reasons behind the slow implementation of the RSA algorithm, and the chapter was concluded stressing the need for a hardware implementation of RSA if the algorithm was to be used within a security mechanism. Below is a partial list of available RSA chips taken from [68], [69].

Company	Baud rate (K) per 512 Bits	Bits Per Chip
AT&T	19	298
British Telecom	5.1	256
Calmos Syst. INC	28	593
Cryptech	32	512
Cylink	6.8	1024

Table 5-3 RSA Hardware

Current research within the Network Research group includes an optimised modular exponentiation VLSI implementation for RSA [70] with expected results of 111K bits per second using a 506 bit modulus.

5.6.2 Cryptech PQR 512 RSA Hardware Implementation

Of the few manufacturers that produce chips capable of RSA calculations by far the best that can be found at this present time (with a product available for sale) is CRYPTECH. What follows is a description of the CRYPTECH chip and the chosen strategy for

implementing RSA calculations upon it, the ENCDEC implementation of the chip can be found in Appendix B Figure B-4.

CRYPTECH produce the PQR 512 [71] ASIC (Application Specific Integrated Circuit) in a QFP 100 (Quad Flat Package), requiring surface mount technology. It has been designed to optimise modular exponentiation, which is the operation that makes up the bulk of the RSA algorithm:

$$y = x^e \bmod n;$$

where y, x, e and n are integers.

The PQR 512 has been built to allow it to be connected in parallel with other 512s, giving it a greater key handling range.

CRYPTECH combine two of the PQR 512s with an 8kx8 static RAM (Random Access Memory) chip to make the PQR 1024 module, giving enhanced performance and twice the 512's key length handling. The static RAM of the PQR module is divided into key and data segments of 512 blocks each, there are 8 blocks of memory for both data and key storage.

A single PQR 512 can handle keylengths of up to 513 bits, encrypting at 32kbits/s, while the PQR 1024 can handle keylengths of up to 1033 bits at 16kbits/s.

PQR 512

$32 \leq \text{Keylength} \leq 513$

PQR 1024

$32 \leq \text{Keylength} \leq 1033$

Because the PQR 512 can only handle keylengths below 513 bits in length it was decided to use the PQR 1024 module. A keylength of 512 bits is now considered the minimum key length that should be used in RSA [72]. The security of RSA depends upon the difficulty of factorisation, therefore, having a larger key makes RSA more secure, see Chapter 1 for more details. Although the security provided by a key of 1033 bits is at the current level of processor speed and factorisation techniques overkill, hardware must be designed to have a reasonably long useful life since hardware is generally expensive to upgrade.

The external interface of the PQR 1024 is similar to that of a 8kx8 RAM. Its physical dimensions are 73 mm by 63 mm.

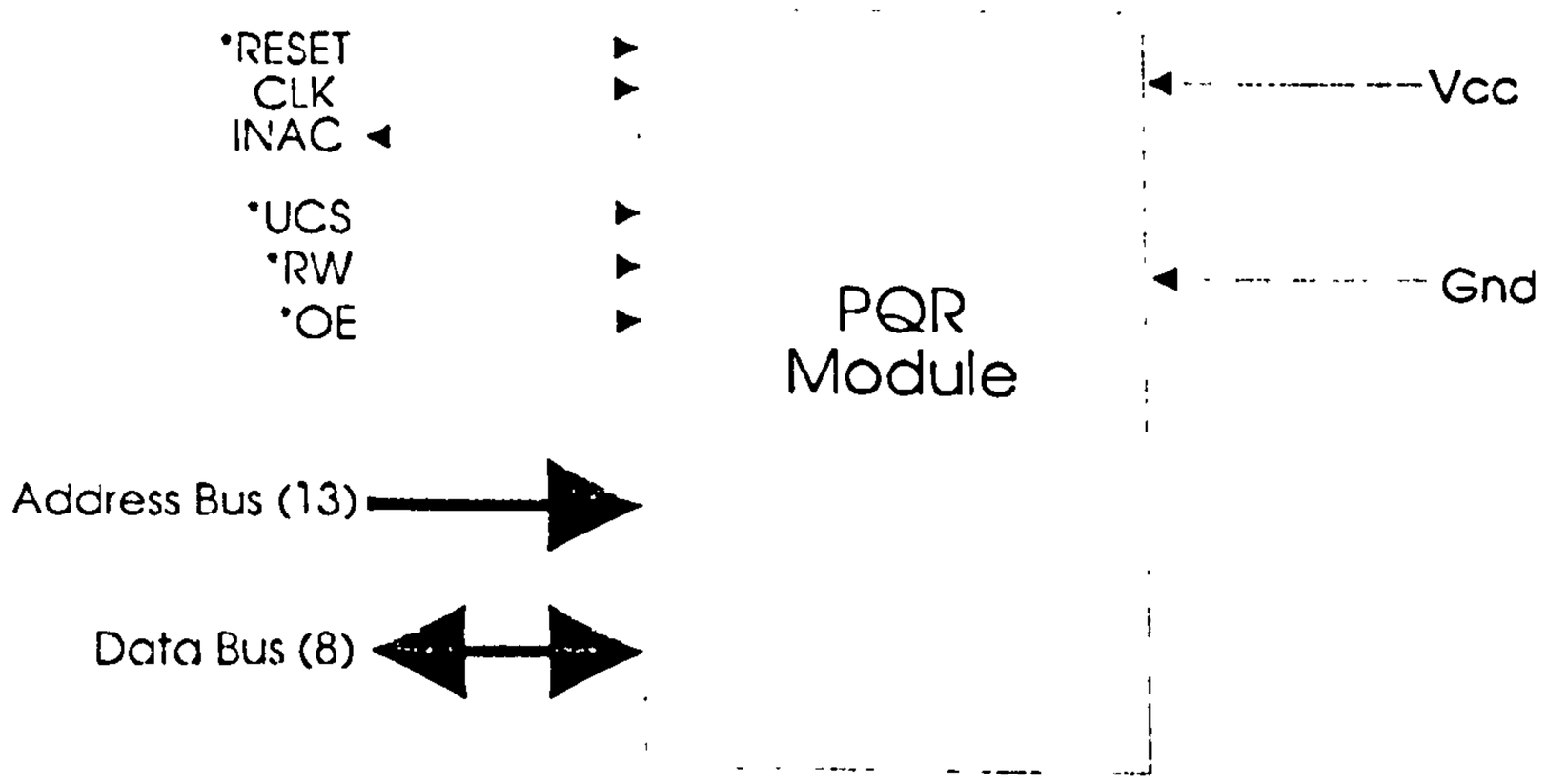


Figure 5-7 PQR Logical Interface

The PQR 1024 module is CMOS (Complementary Metal Oxide Surface) technology which accepts TTL level signals on all pins but the CLOCK input.

The PQR module can be in any of 4 different states. The state it is in is controlled through 6 address triggered commands.

ppp- indicates a block of key memory.

Mapped Address	COMMAND	A12-A0	FUNCTION
0xA0200C	ABORT	1 xxxx xxxx x110	Software Reset
0xA02004	PLEND	1 xxxx xxxx x010	Terminate Key Access
0xA02006	INTEN	1 xxxx xxxx x011	Unmasks INAC
0xA0200E	INTDI	1 xxxx xxxx x111	Masks INAC
0xA02008	START	1 pppx xxxx x10x	Execute Internal Program
0xA02000	PLORE	1 pppx xxxx x00x	Resets A Block Of Key Memory

Table 5-4 Address Triggered Commands

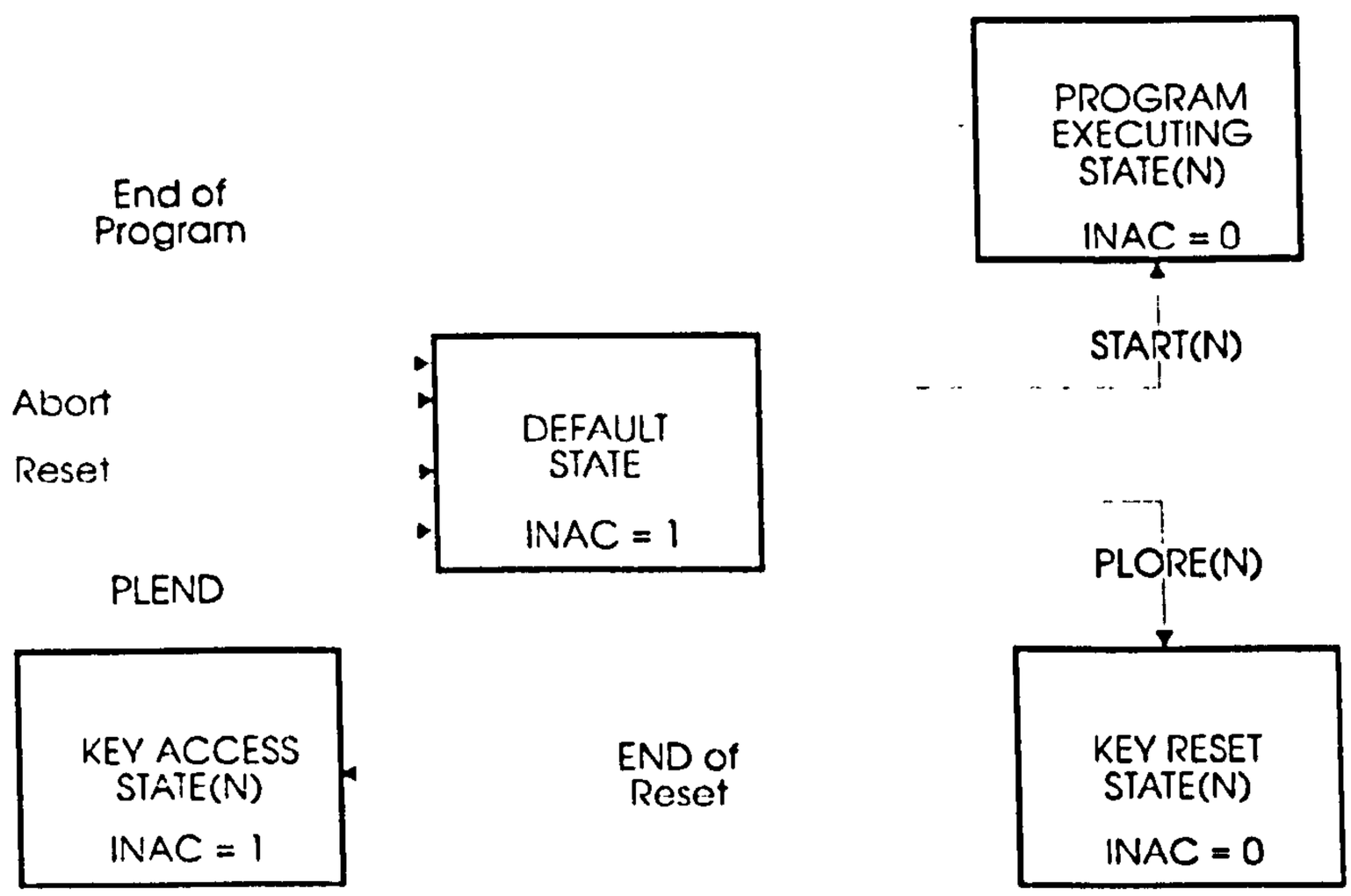


Figure 5-8 PQR States

1. *Default State* : The chip enters this state just after a reset, and on normal termination of a internal program execution or key access.
2. *Key Reset State* : This is where a block of key memory is reset.

3. *Key Access State* : In this state the PQR module allows the reading and writing of key memory blocks.
4. *Program Execution State* : In this state the PQR module executes an internal program made up of 11 internal commands.

Table 5-5 below shows the 11 internal commands,

MNEMONIC	FUNCTIONALITY
END XXX	Terminate Program
INITMOD XXX	Initialise the number in the internal register as the modulus
LOADOUT VAR	Load data from internal register to data_memory block VAR
LOADIN VAR	Load data from data_memory block VAR into internal register
MULTPL VAR	Multiply data_memory block VAR with the internal register, result in internal register
SQUARE VAR	Raise the data in the internal register to a power of 2
SHRIGH XXX	One bit shift right of internal register
SHLEFT XXX	One bit shift Left of internal register
EXPON VAR	Use VAR to produce the next four steps in an exponent calculation based on Knuth(see section 5.6.4.3)
PRESET VAR	Used to indicate the amount in bits that is to be moved from or too the internal register

Table 5-5 Internal Commands

Mnemonic is an 8 bit number;

<Command><Variable>
< 1 byte >

where command is the upper 4 bits and VAR is the lower 4 bits. If the command is followed by an XXX then the lower 4 bits should be set to 0000.

5.6.3 RSA Packet

Information is sent to the PB as described in section 5.5. The packet configuration sent to the board for an RSA process is described below.

- Header
    - Function
    - Blank
    - Mod_length
        Length of the modulus used in the modular exponentiation calculation.
    - Exp_length
        Length of the exponent used in the modular exponentiation calculation.
    - Message_length
- Message
    - data
        variable length

An RSA process using the ENCDEC card is split into two phases:

1. RSA Keyload;
2. RSA Calculation.

5.6.4 RSA Keyload

The first step in an RSA calculation when using the ENCDEC board is the loading of an RSA key. An RSA keyload is initiated by the RSA and KEYLOAD bits being set within the function byte of the RSA-packet header. The *data* field within the message portion of the RSA-packet is used to store the modulus and the exponent, in that order.

Two areas in the PB's memory have been set aside for the storage of the modulus defined in the *data* field. 130 bytes have been allocated from the base address of 0x202134, for storing an unmodified modulus, and another 130 bytes from the base address 0x202328 is allocated for the twos complement of the modulus, see section 5.6.5.3 for further explanation.

The modulus and the exponent defined within the RSA-packet are used in the construction of a program that is used by the PQR module to perform calculations. The program is formed from the 11 internal commands described in section 5.6.2, and is stored within one of the PQR's key blocks during the PQR's key access state, which it enters after receiving the PLORE addressable command In the case of the ENCDEC card the key storage block used for program storage was 0, access to the memory block is ended once the PQR receives the PLEND signal, returning the PQR module to its default state. Once the program is loaded and the PLEND signal sent it is not possible for the program to be read again, therefore, the risk that an attacker capable of somehow halting an RSA calculation being capable of reading a keypart is removed.

The program created and written to the PQR's key memory for calculating a modular exponentiation can be divided into four parts:

1. load and initialise the modulus;
2. load the message component of the calculation;
3. perform the calculation;
4. output the result.

5.6.4.1 Load and Initialise the Modulus

The internal architecture of the PQR processor can be seen in Figure 5-9.

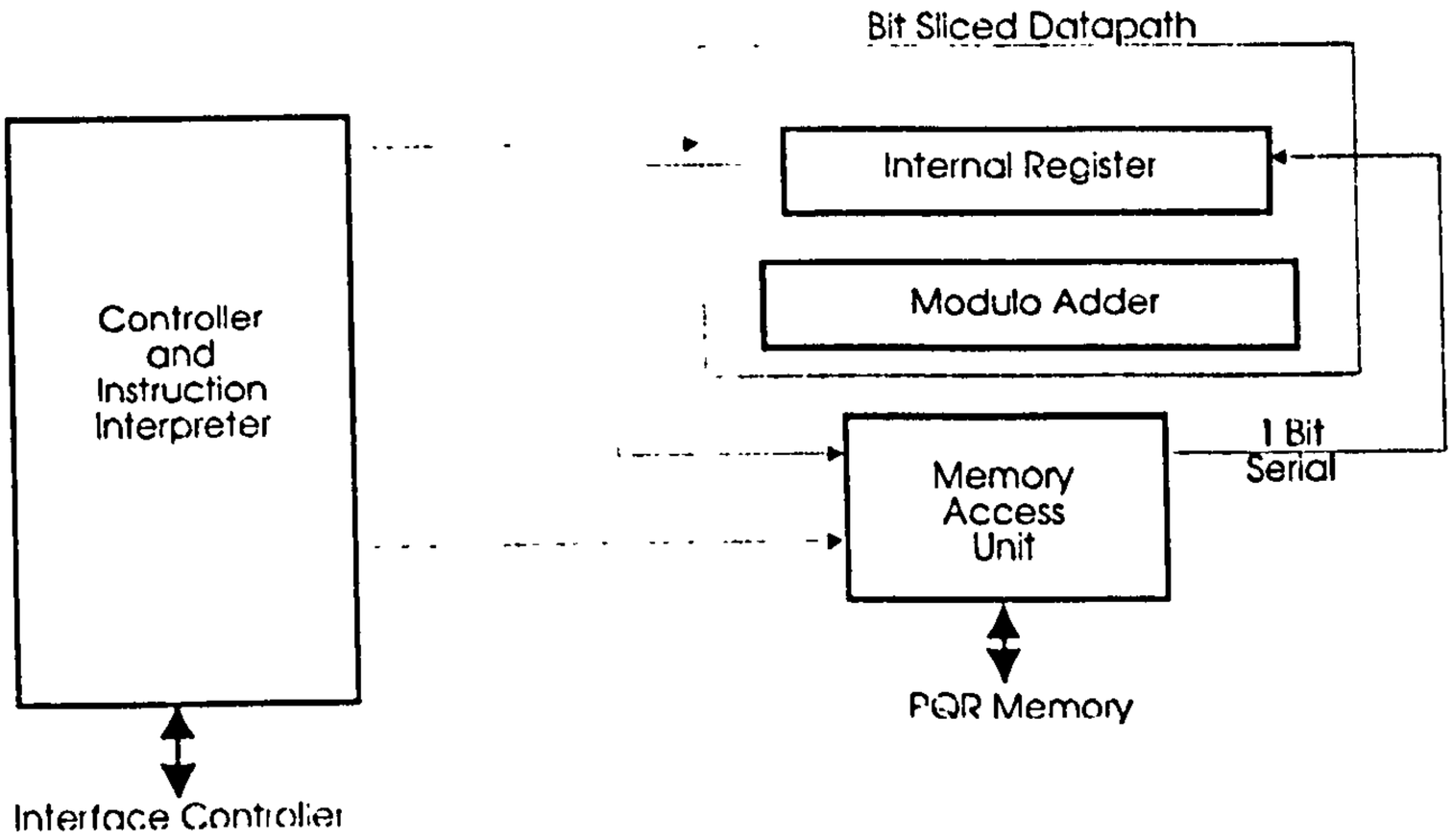


Figure 5-9 PQR Internal Architecture

The internal register is designed for input/output purposes, it is a serial shift register connected to the memory access unit. Bits are shifted in a *least significant bit* (LSb) to *most significant bit* (MSb) direction, the PQR 1024's internal register is 1040 bits long.

Prior to any modular exponentiation a modulus must be initialised to the PQR. There are two preparation steps that the modulus must undergo for correct initialisation.

1. The value in the internal register to be initialised must be the two's complement of the actual modulus.
2. The first bit of the complemented modulus must be placed upon the 3rd bit place of the internal register.

The two's complement of the modulus is calculated by the 68000 through an exclusive ORing of the modulus with a value equal to 2^{KL}, where KL is the effective length of the modulus calculated as the length of the modulus in bits up to and including the MSb equal to 1, and the addition of 1 to the resulting number. The length of the complemented modulus is equal to the effective length.

Example

modulus = 0110 0011 1110 1101

therefore, the effective length = KL = 15

2's complement = (110 0011 1110 1101) XOR (111 1111 1111 1111) + 1

001 1100 0001 0010 + 1

001 1100 0001 0011

The two's complemented modulus is stored by the 68000 in data memory block 7 during the RSA keyload phase in preparation for initialisation when the PQR program is run in phase 2 of the RSA calculation. The complemented modulus is aligned upon the first byte of memory and any free memory in the data memory block is cleared.

A copy of the two's complemented modulus is stored in the PB memory allocated for such storage. The complemented modulus is held in preparation for the final calculation of the modular exponential calculation, see section 5.6.5.3.

The complemented modulus stored within data memory block 7 is left shifted into the internal register through the use of PRESET and LOADIN 7 internal instructions.

Rather than creating a program to shift the exact number of bits equal to the modulus length a more general strategy was chosen for loading data into the internal register. Data was loaded into the register placing the MSb of the data in the MSb position of the register, this was accomplished by shifting in data 1040 bits at a time, the same length as the internal register of the PQR 1024, and then using the SHRIGH and SHLEFT commands to properly align the data within the register in accordance to the specific needs of the PQR processor (see below). Due to the design of the PQR module the preset value must be equal to the number of bits to be loaded into the register minus two,

PRESET = number of bits to be loaded -2

therefore, for the strategy decided upon the PRESET is set to 1038.

Because of the design of the PQR module the modulus must be aligned upon the 3rd MSb position. To accomplish this two SHRIGH commands are used to position the modulus; use of SHRIGH places a zero in the left most bit position of the internal register.

The command INITMOD is then given which transfers the complemented modulus within the internal register to a modulus register.

5.6.4.2 Load the Message Component

The value of the message component must be less than the value of the modulus. At this stage of the ENCDEC's processing the data will have been correctly framed ensuring that

its value is within the necessary parameters, for details of framing see Chapter 6. The message data stored within block 0 of data memory is loaded into the internal register by use of the PRESET and LOADIN 0 commands. As in the previous section the preset value must be two less than the length of data to be transferred, the strategy previously employed to load the modulus into the internal register is again adopted here. However, this time the data must be aligned so that its MSb is aligned upon the second bit position of the register, therefore, only one SHRIGH is used.

Once the correct alignment of the data has taken place a copy of the data is transferred to data memory block 0, ready for use during the calculation phase.

5.6.4.3 Calculation

The PQR module can implement two mathematical operations square and multiply. These are initiated through their respective internal command calls SQUARE and MULTIPLY. An exponentiation calculation is represented by a series of multiply and square operations. To calculate exponentiations the PQR module uses a left to right algorithm.

For example,

$$R = m^e$$

The left to right algorithm:

- begin with the most significant bit of the exponent e, set accumulator a to 1;
    - each 1 is a square and multiply operation, $a = ma^2$;
    - each 0 bit is a square operation, $a = a^2$;
- $R = a$.

Therefore,

if the exponent is equal to,

0010 1100 0101

By making a = m we can skip the most significant bit equal to 1 and then carry on as usual.

The list of operations is

S, SM, SM, S,S,S,SM,S,SM

where S = square, an SM = square and multiply

The data is first loaded from data memory block 0 into the internal register using the strategy already described. The internal register now acts as the accumulator for performing the exponentiation. The S and SM operators are calculated for the first nibble of the exponent and the relevant PQR internal commands are used, i.e. SQUARE and MULTIPLY, both these functions use the data stored in data block 0 for manipulation of the accumulator. Every other nibble of the exponent is used as the argument for the internal command EXP which automatically generates the necessary S and SM operations for a nibble.

5.6.4.4 Result

Because of the PQR design the result of a calculation is 1 bit longer than the modulus and fulfils the following rule,

$0 \leq \text{result} < 2 * \text{modulus}$.

The value remaining in the internal register at the end of the calculation phase is the result. The result is not on the correct byte boundaries and it is therefore necessary to re-align the result so that its least significant bit is correctly byte aligned within the internal register; this can be accomplished through the use of SHRIGH commands. The number of SHRIGH instructions necessary is found by using the following algorithm.

```
IF ((KL+1) mod 8) EQUAL TO 0
        do not shift right
ELSE
        SHRIGH (8-(KL+1) mod 8) times
END
```

The result in the internal register is then read out using the PRESET and LOADOU 1 command. The program is terminated using the internal command END, returning the PQR processor into its default state. The result of the calculation can now be read externally from data block 1, however, it must be tested for having a greater value than the modulus, if it has a greater value the modulus must be subtracted from the result to obtain the result mod (m).

5.6.5 RSA Calculation

The second phase of an RSA calculation is indicated by the setting of the RSA bit within the RSA-packet header *function* byte. The *data* field of the RSA-packet contains the data to be encrypted/decrypted The data has been framed so that it is a multiple of the modulus defined in the key load phase.

The RSA calculation can be divided into two sections:

1. loading of data;
2. control of the PQR module;
3. final calculation of the result.

5.6.5.1 Loading of Data

The data to be encrypted/decrypted in the modular exponentiation calculation is written to block 0 of PQR data memory, in blocks equal to the modulus length, by the 68000, the data is aligned upon the first byte of memory and any free space in the memory block is cleared, i.e. set to zero. The PQR data memory block 0 is memory mapped at the PB address 0xA0000.

5.6.5.2 Control of the PQR module

Once the data has been written to the PQR module the 68000 sends the addressable command START0, indicating that the PQR module should run the internal command program stored within the key memory area 0. The 68000 then polls the PQR INAC line, the INAC line shows when the PQR module leaves the program execution phase, when the INAC line goes low indicating the end of execution the 68000 reads the result of the PQR calculation.

5.6.5.3 Final Calculation of the Result

The result read from the PQR module is compared to the modulus stored at 0x202134, if the result is less than the modulus it is written to the PB memory reserved for the *data* field of the RSA-packet, over the original data. If the result is larger than, or equal to, the modulus the result is added to the complimented modulus stored at address 0x202328 and then written to the memory reserved for the *data* field.

The three steps of the second phase are run until all the data within the RSA-packet has undergone the RSA calculation, once this is done the PB writes the RSA-packet back to the interface board as described in section 5.5.

5.7 Hardware Implementation of the DES Cryptosystem

5.7.1 CRY121C102

The DES chip chosen for use within the ENCDEC prototype was the CRYPTECH CRY121C102 [73]. The DES chip provides high data encryption speeds, of around 20Mbits/second, and its easy interfacing. The DES chip comes in two packages a 48 pin DIP and a 68 pin PLCC, for prototyping purposes we used the 48 pin DIP. The ENCDEC implementation of the DES chip can be found in Appendix B Figure B-5.

The DES chip is logically divided into two parts:

- an internal DES processor;
- an input/output shell that interfaces with the internal processor and the rest of the world.

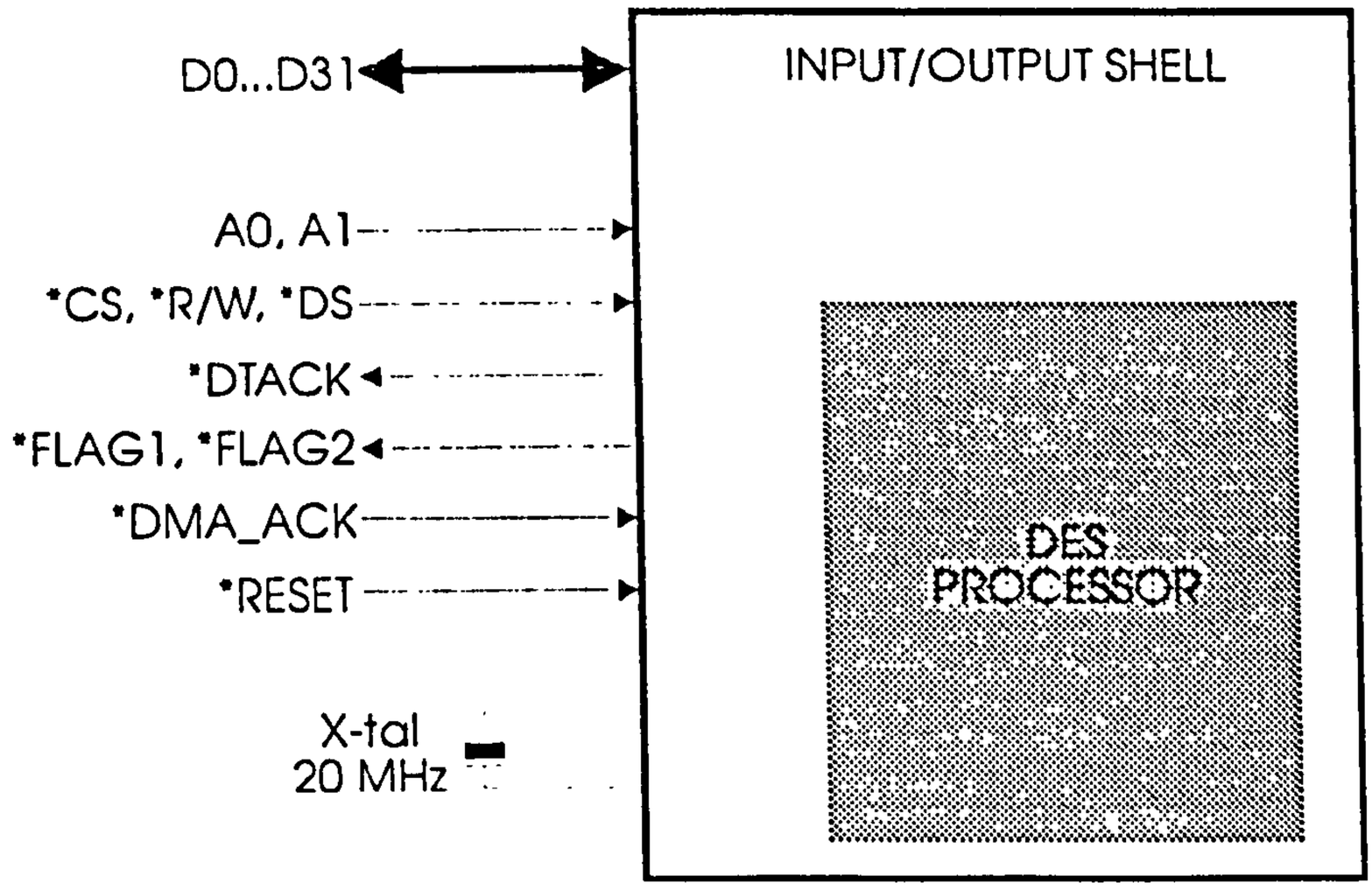


Figure 5-10 DES Chip

The DES processor is manipulated through the use of its numerous registers.

- Two input data registers:

  that are used for the buffering of data awaiting processing by the internal DES processing segment of the chip.

- An output data register:

  holds the result of a DES calculation performed on plaintext in a previous calculation.

- A command and a command modify register:

  the command register is a temporary buffer for the commands used in the programming of the DES chip, see below. The command modify register holds data that modifies the commands held in the command register. The contents of the modify register is not cleared once read by the internal DES processor.

- Four configuration registers:

  all four registers are addressed at the same location. Lines D7 to D4 are used in the selection of a specific register, while D3-D0 are written into it. Through these registers it is possible to configure the bus width, ROM key load, the Flag 1 request line and the Flag 2 request line.

- Three status registers

  The status registers are used by the external processor to check on the I/O shell, the internal DES processors control logic and the status of the internal DES processors key registers.

The various registers are accessed by the use of the input lines A1 and A0.

A1	A0	Register
0	0	Data
0	1	Configuration
1	0	Command
1	1	Command Modify

Table 5-6 DES Status Registers

Single Commands

Hex	Mnemonic	Description
EC-EF	LEK1-LEK4	Load external key in key register 1-4
C8-CB	SLK1-SLK4	Select key 1-4 for en/decryption
D0	RSD	Reset all work registers
D8	RST	Reset all work and key registers
A8	DES	Execute DES on work register 1
A9	DESI	Execute DES^{-1} on work register 1
89	NOP	Dummy command
C0	LIV	Load initial vector
C2	RFV	Read final vector

Table 5-7 DES Single Commands

The single commands are provided for the initialisation and subsequent clearing up processes that must be done before and after the processing of data. The single commands may also be used in the provision of isolated commands that must be used while data is still secure within the chip.

Continuous Commands

Command	Modifier	Mnemonic	Description
00	0	CE_ECB	Pipelined encryption ECB
00	1	CE_CBC	Pipelined encryption CBC
00	2	CE_CFB8	Pipelined encryption CFB8
00	3	CE_OFB8	Pipelined encryption OFB8
10	E	CE_CFB1	Pipelined encryption CFB1
10	F	CE_OFB1	Pipelined encryption OFB1
01	8	CD_ECB	Pipelined decryption ECB
01	9	CD_CBC	Pipelined decryption CBC
01	A	CD_CFB8	Pipelined decryption CFB8
01	B	CD_OFB8	Pipelined decryption OFB8
10	E	CD_CFB1	Pipelined decryption CFB1
10	F	CD_OFB1	Pipelined decryption OFB1
02	8	MRE_ECB	Minimum Response ENC
03	8	MRD_ECB	Minimum Response DEC

Table 5-8 DES Continuous Commands

The continuous commands are used for bulk encryption/decryption, a continuous command stays valid as long as data is being read in and out of the chip. A continuous command is stopped by placing new information into the data register and then overwriting the continuous command with a new command such as NOP (no operation).

5.7.2 DES Packets

It was felt unnecessary to provide the cipher feedback and output feedback modes of operation as they were originally developed for stream cipher applications. The two remaining modes electronic codebook and cipher block chaining were settled upon to provide the bulk of cryptographic processing in confidentiality mechanisms due to their faster speed of encryption at the cost of minor amounts of padding. A description of the DES modes of operation can be found in Chapter 2.

Information that is written to the ENCDEC board for a symmetric cryptographic process is presented in packets. Each packet comprises of a header and the message, the header is six bytes long comprising of four one byte words and one two byte word.

- HEADER
  - Function
  - For future implementation
  - Mode

    8-bit word indicating the DES mode to be used:

    ECB = 0x08

    CBC = 0x04

    OFB = 0x02

    CFB = 0x01
  - For future Implementation
  - Message_length
- MESSAGE
  - Data

5.7.3 DES key load

When using the board in a DES operation the key must first be loaded into one of the ten key memories. Any future operations can use the key and IV by simply referencing the key location 0-9. The processing board has set aside on board storage for 10 DES keys and their initial vectors that are necessary for some of the DES modes of operation. The key storage base address is at 0x203000 and has been allocated 240 bytes of memory. 160 bytes of memory are used in the storage of the key and IV. The other 80 bytes are used in the storage of working IVs, i.e. the IV used in the encryption/decryption of data sent to the board in multiple packets. Once the keys are stored upon the board they may be used to speed up operations that require numerous plaintext messages to be encrypted/decrypted, by avoiding the continual loading of new keys.

The assembly code function *keyld* is used to store the key and is called if the DES and KEYLOAD bits are set within the DES packet header FUNCTION word. In the keyld phase of a DES operation the first eight bytes of the DATA field will be the DES key and the second eight the IV.

5.7.4 DES Calculation

The second phase of a DES operation is the actual encryption/decryption process, where plaintext/ciphertext is sent to the board for DES processing. The MESSAGE section of the DES packet will contain the information to be processed.

The ENCDEC card must be notified when the first packet and last packet of plaintext or ciphertext is sent for DES encryption/decryption. This is accomplished by the setting of the BEGIN and END bits in the header FUNCTION word.

- For encryption:

  the first packet must be indicated so that the correct IV is used in the cryptographic process, for instance the second packet of five being encrypted with the DES mode CBC would need to use the stored IV as opposed to the original that was sent to the board. The final packet must be indicated so that the packet may be padded out. The ciphertext will be up to eight bytes longer than the plaintext, the expansion is necessary because of DES processing blocks of 64 bits at a time. Before the final packet of a plaintext message is encrypted it is padded, this operation makes the plaintext length a multiple of 64 bits, the final byte of the

plain-text is the length of the necessary padding in bytes, if the plaintext is already a multiple of 64 bits the message is padded by 64 bits and the final byte holds the value eight; all the padded bytes except for the final one are set to a random byte sized word. The size of the ciphertext is then stored in MESSAGE_LENGTH of the DES packet header, see section 5.7.2.

- For decryption:

  as with the encryption process it is necessary to know what packet of the plaintext is being processed so that the correct IV is used, and the final packet must be indicated to the ENCDEC board so that the padding information can be removed and the plaintext returned to its original size. The size of the plaintext is then stored in the MESSAGE_LENGTH of the DES packet header.

The continuous functions were used to program the actual encryption/decryption routines of the DES chip. The continuous commands have been highly pipelined giving a better performance than the single commands. The pipelining means that while data is undergoing the cryptographic process the output of the previous process is being made available while the next input is being entered.

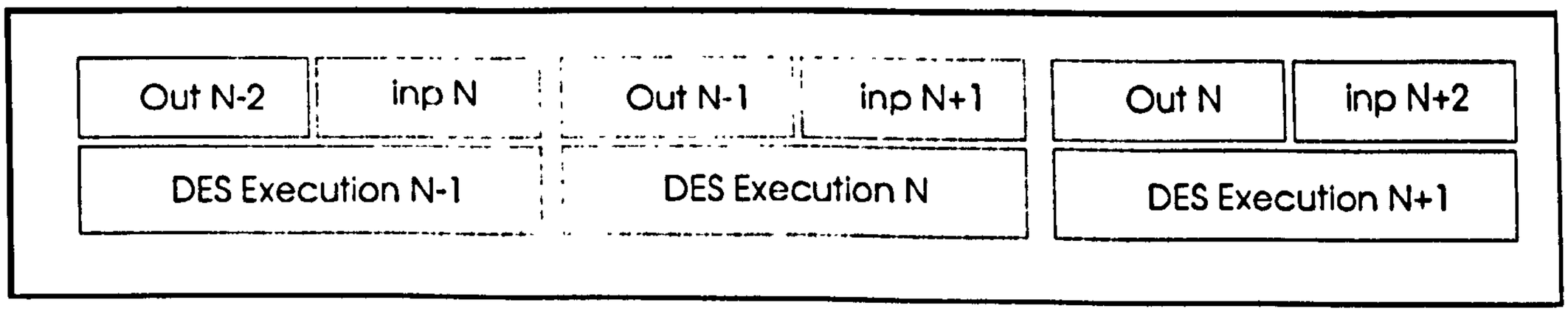


Figure 5-11 DES Pipelining

The process for ECB using the DES chip can be seen in the Figure 5-12. The ECB DES mode is chosen by setting the P_ECB bit in the MODE word of the DES packet header.

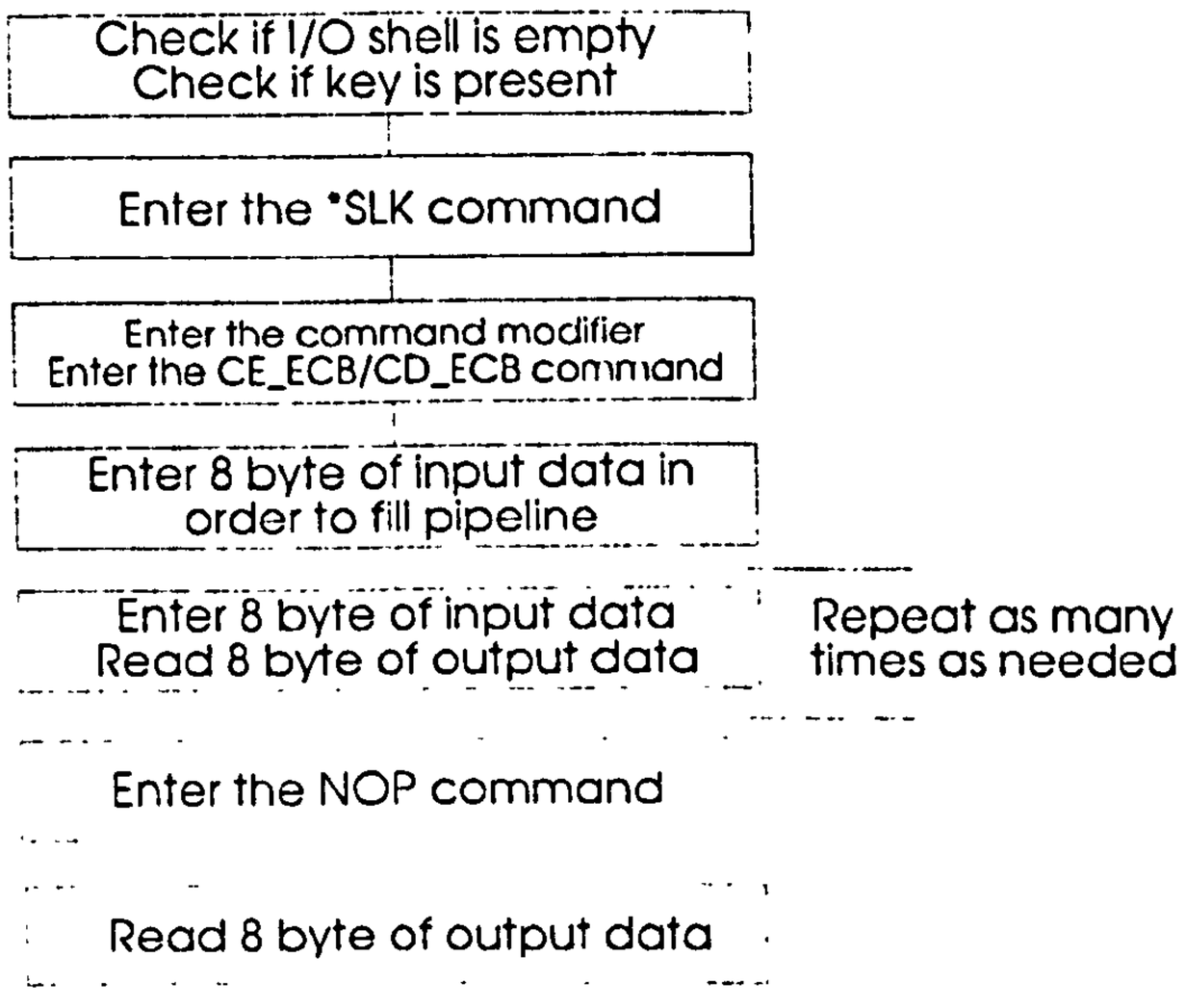


Figure 5-12 ECB Process

The key is loaded into the DES chip by the routine *lkey*, the *lkey* function loads a key into the internal key register 1, and then indicates to the DES processor that it is to use the key contained in register 1 for future DES calculations. 8 bytes of data are then written to the DES chips I/O register. The registers memory map can be found in Table 5-9.

Address	Description
0x800000	Input/output register
0x800002	Command/Status1 register
0x800004	Command Modify/Status3 register
0x800006	Configuration/Status2 register

Table 5-9 DES Register Address

Once the keys have been loaded the command modify register is loaded with the value 0x00 and the DES command of CE_ECB or CD_ECB are stored within the command register, depending on whether a encryption or decryption is required. Once the DES chip is correctly programmed the 68000 enters the *enc* routine. By polling the status registers of the DES chip it is possible to obtain information about the current state of the chip. The registers are located at the same address as the configuration and command registers; status registers are accessed on a read cycle while the others are accessed on write cycles. The *enc* routine loads eight bytes of data into the input register of the DES chip which are then processed by the internal DES processor. The 68000 polls the DES status register 1 first to check whether an error has occurred during the calculation and then to see if there is a valid DES output to be read. An error will only occur if there is a hardware fault with the ENCDEC card. When there is a valid word stored in the output register the word is read from the DES chip and stored at the same block of memory as the original 8 bytes. The data from the DES packet is written to the DES chip till all data has been processed, it then awaits writing to the interface card.

The CBC mode of operation is chosen by setting the P_CBC bit in the mode word of the DES packet header. The process of implementing the DES CBC mode using the DES chip is summarised in Figure 5-13.

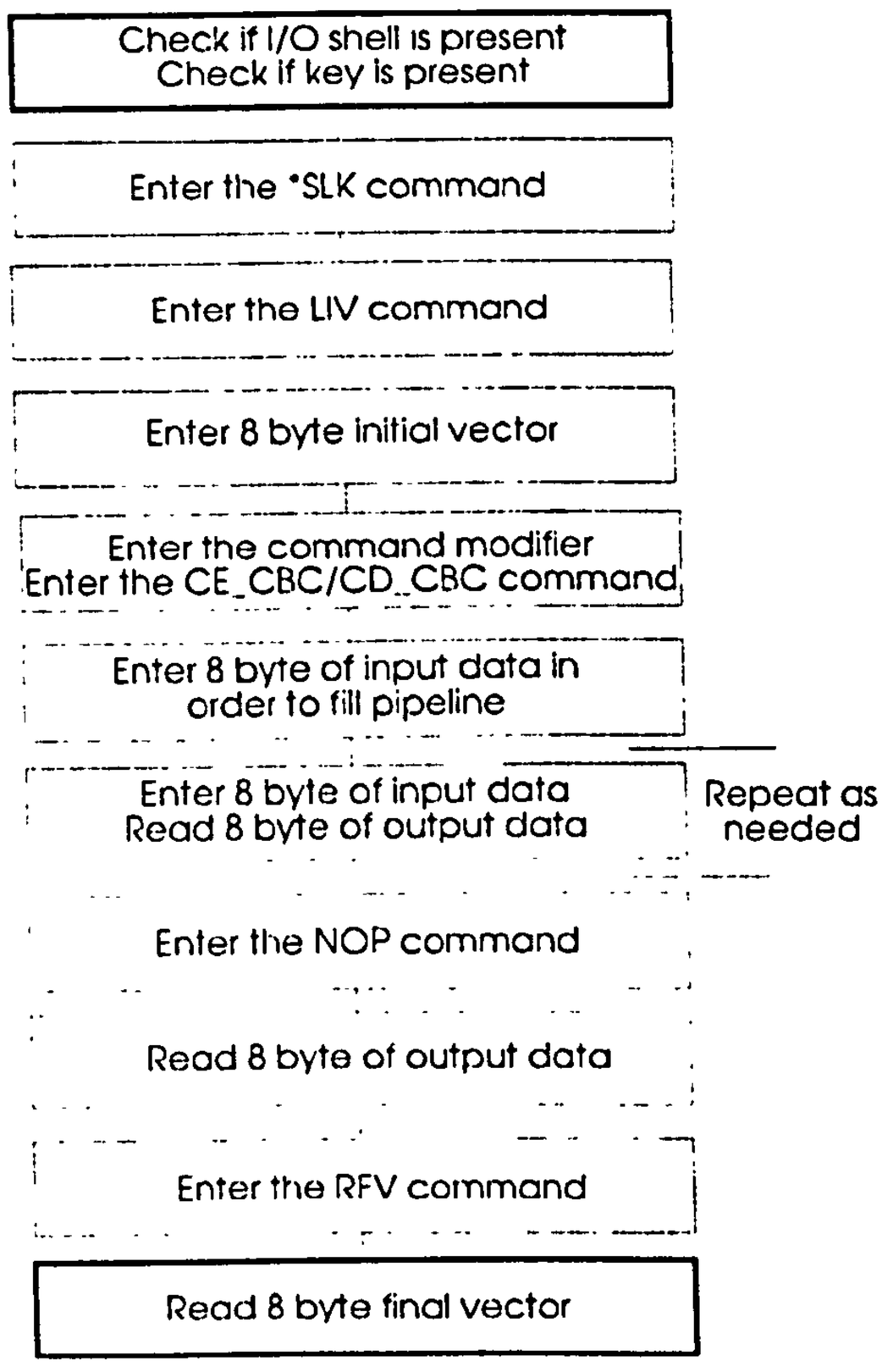


Figure 5-13 CBC Process

The 68000 checks the START bit in the DES packet header FUNCTION word. If the start bit is high then the initial IV is loaded into the DES chip proceeding the command LIV, that tells the DES chip the next block of data written to its input register will be an initial vector. If the START bit is not set then the working IV will be written to the DES chip. The command modify register has the value 0x01 written to it and the command P_CBC is written to the DES command register. The function *enc* is called and the procedure is as described above.

Chapter 6

6. Software Implementation of CISS Security Mechanisms

This chapter describes the software written for inclusion within CISS. The chapter begins by describing the ENCDEC device driver, used to support the ENCDEC board. The framing protocols for DES and RSA are detailed with a brief description of the most commonly used functions provided for hardware or software implementation of the cryptosystems.

6.1 ENCDEC Device Driver

UNIX [74] is made up of three parts; the UNIX kernel, user processes and a system of data files stored in secondary memory. The kernel is at the centre of the system, it controls all access to the CPU, primary memory, and I/O devices. The user processes request resources from the kernel on behalf of the users, and the kernel assigns the resources, such as processing time, as requested.

Input/output (I/O) in UNIX is handled very simply, in that all I/O is presumed to be conducted upon files, some of which, in fact, will be hardware devices. The kernel maintains a overall filesystem that maps user requests onto the necessary system resource, be it an area in secondary memory or a device connected to the system. The UNIX file system is made up of:

1. regular files;
2. directories;
3. special files;
4. symbolic links etc.

The Files that are used for accessing devices are the special files that are collected within the '/dev' directory. A list of all devices that may be attached to the UNIX system are indexed via two arrays the *cdevsw* (switch table for character devices)or *bdevsw* (switch table for block devices). A block device is capable of storing data in fixed block sizes and later retrieving them. Devices that must read data in variable size blocks are character devices. The devsw tables contain entries to a device driver's 'top half,' these are the routines that are called by the user processes. The 'bottom half' routines are called on the assertion of an

interrupt by the device. Because the ENCDEC prototype board does not meet the specifications to be a block driver it is configured as a character device.

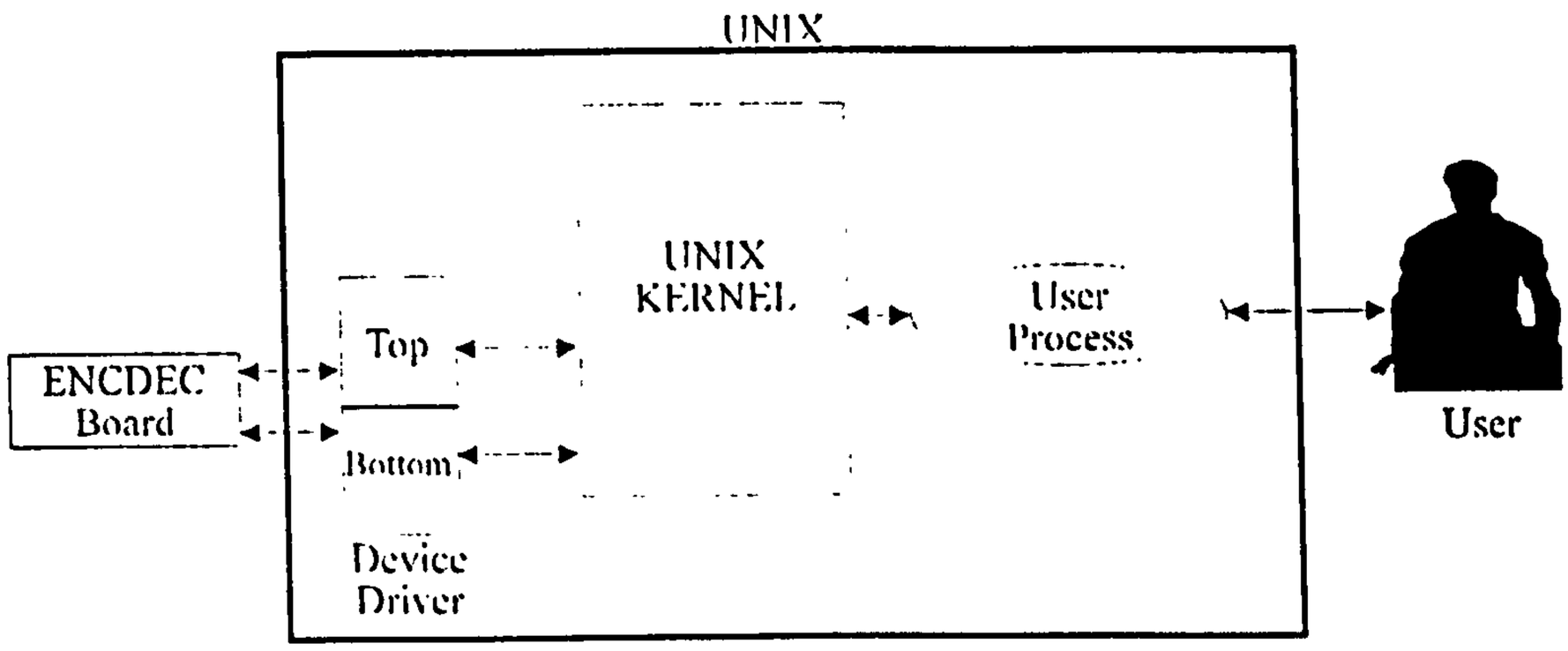


Figure 6-1 Device Driver Overview

The ENCDEC cdevsw entry is

{

encdec_open, encdec_close, encdec_read, encdec_write, encdec_ioctl, encdec_reset, encdec_select, encdec_mmap, (struct streamtab *) encdec_str)

}

These are known as the entry points to the device driver.

There are two ways in which a device driver can be loaded within the UNIX kernel and its entry points placed within the devsw lists, during autoconfiguration and through the use of *modload*. All physical devices upon a UNIX system have two structures assigned to them *dev_info* and *dev_ops*.

dev_info:

- *devi_parent*: pointer to dev_info structure that corresponds to the parent device, in the case of the ENCDEC board the parent is SBus device;
- *devi_next*: pointer to dev_info structure of any sibling devices, system dependent;
- *devi_slaves*: a pointer to the dev_info structures of any child devices;
- *devi_name*: a pointer to a character string that is contained within the FCode PROM of the device, in the ENCDEC board's case this is "encdec,josw";
- *devi_nreg*: number of registers the device has, the ENCDEC board has four registers;
- *devi_nintr*: number of interrupts the device has, the ENCDEC board has a single interrupt;
- *devi_intr*: a pointer to dev_intr structure that hold information on the devices interrupt level;
- *dev_driver*: a pointer to the dev_ops structure of the device;
- *devi_data*: a pointer to private data;
- *devi_nodeid*: id number of this device from the FCode PROM.
- *devi_unit*: logical unit number of this device.

dev_ops:

- *devo_rev*: device driver revision number;
- *devo_identify*: a pointer to the device driver's _identify() routine, see below;
- *devo_attach*: a pointer to the device driver's _attach() routine, see below.

A device driver can be configured into the UNIX kernel through the use of the *config()* command adding the dev_ops structure of the device_driver to the ops list which the kernel maintains. When the kernel starts execution it begins its boot time initialisation including its autoconfiguration process. In open boot PROM systems this involves the probing of SBus slots for FCode PROMs, and the creation of a *dev_info* structure list. Once all SBus slots have been probed, and a dev_info structure created for each device found, the system attempts to attach a device driver to the device and fill the *dev_ops* field of the dev_info structure. The kernel does this by using the _identify() routines held within its ops list to identify the device drivers to which physical devices belong.

The second way to introduce a device driver to the UNIX kernel is to use the modload command. The modload command uses the initialisation routine that must be part of a loadable device driver and the vd pseudo driver that is part of the kernel. The vd driver is passed a vdldrv structure that is used to load a device driver.

vdldrv:

- *drv_magic*: magic number that tells the vd driver the type of loadable module, in the ENCDEC board's case this is set to VDMAGIC_DRV indicating that it is a device driver and not a pseudo driver;
- *drv_name*: a pointer to the device_driver's name, "encdec";
- *drv_dev_ops*: a pointer to a dev_ops structure initialised by the device_drivers initialisation routine.
- *drv_bdevsw*: this is a pointer to the bdevsw structure defined by the device_driver in this case in is set to NULL.
- *drv_cdevsw*: this is a pointer to the cdevsw structure defined by the ENCDEC device driver.
- *drv_blockmajor*: indicates the line in the bdevsw list kept by the kernel that is the device drivers entry, in the ENCDEC board's case this is set to 0, it has no preconfigured entry.
- *drv_charmajor*: as the drv_blockmajor except within the cdevsw list, set to 0 as there is no preconfigured entry for the ENCDEC board.

The device driver provided with the prototype board is one that allows the loading of device_drivers into the kernel. This eases any debugging that is necessary and does not require the kernel to be remade.

6.1.1 Autoconfiguration Support

If a device driver is configured into the system and loaded during autoconfiguration, at boot time, it requires two routines *identify* and *attach*. These routines must be present even if the device is not to be configured during autoconfig as they are called by the vd driver during the loading process of loadable drivers.

6.1.1.1 encdec_identify()

When this routine is called, it is passed a single argument which should be the name of the ENCDEC device contained within the device's FCode PROM. The name of the ENCDEC prototype board is "encdec,josw". If the routine is called with the correct name, the fact is recorded within a static variable and a successful identification is returned to the system.

6.1.1.2 encdec_attach()

On the successful identification the encdec_attach routine is called. The attach routine:

1. allocates device driver state memory, the data structures required for operation of the device;
2. maps the device registers into kernel memory and stores the addresses into data structures, the addresses are obtained from information within the dev_info structure for the encdec device, these would include the ENCDEC external register, and internal registers;
3. alerts the system to the highest blocking priority for DVMA mapping;
4. adds an entry within the kernels interrupt handling routine for the driver's interrupt service;
5. initialises the device.

6.1.1.3 encdec_vdcmd()

This is the initialisation routine called by the vd-pseudo driver during modload, modunload and modstat. Three commands are catered for by the routine:

- VDLOAD: The driver initialises the ENCDEC board according to any configuration information passed to the initialisation routine by modload and links in the vd entry point struct with that of the cdevsw set aside for loadable drivers.
- VDUNLOAD: The initialisation routine verifies that the ENCDEC board is not in the middle of an operation and if it isn't it is unloaded and the necessary cleanup operations are done.
- VDSTAT: The initialisation routine fills in the vds_modinfo data structure detailing the current status of the ENCDEC board.

6.1.2 Driver Entry Points

6.1.2.1 encdec_open()

The ENCDEC device can be accessed through the special file it has been assigned within the '/dev' directory. Before any operation can be carried out using the device it, must first be opened. When a user process uses either open() or fopen() C commands with the special filename as an argument the encdec_open() routine is called. The encdec_open() routine checks to see if the device has been opened previously. If it has been an error is reported, otherwise the device is initialised and inhibited from being opened by another process. The routine then returns control, indicating success to the user process.

6.1.2.2 encdec_close()

The encdec_close() routine is called whenever the C functions fclose() or close() are used and the special device file is indicated. The encdec_close() routine tidies up the device and

then closes it down, and signals the system telling it that the device cannot be used until it is reopened.

6.1.2.3 encdec_read() and encdec_write

A user process uses any of the file read()/write() routines within the C language giving the device special filename as an argument to read or write information to the ENCDEC board.

For example:

a user program might call the *read*() system call;

read(fileno, address, nbytes);

the kernel then calls the drivers *read*() routine;

encdec_read(dev, uio).

The process for I/O using a device driver to interface with a device can be seen below.

1. Read or write routine is called.
2. The appropriate entry point routine is called and initiates activity on the device while blocking any other attempts to access the board.
3. The CPU passes control to some other process.
4. The device completes its action and issues an interrupt request, the interrupt is serviced by the kernel and the encdec_intr() routine is called.
5. The interrupt handler starts up the process that called the device entry point routine and control is passed back to the user process that requested the I/O service.

The encdec_read() and encdec_write() routines call a kernel support routine physio() that breaks up large transfers of data into segments that the device can handle. The physio() routine in turn calls the encdec_strategy() routine described below for the actual transfer of data to and from the board.

6.1.3 Interrupt Support

6.1.3.1 ENCDEC_intr()

When this routine is used it takes the appropriate action to deal with the interrupt called. In the case of the ENCDEC board, there are two reasons for an interrupt to occur:

1. an error has occurred upon the board;
2. a request for SBus control and subsequent DVMA is being made for the completion of a requested read/write.

In the first case the board is reset and any actions that were being carried out must be resubmitted. In the second case the device interrupt request is reset and control is returned to the system. The user process that requested the read/write service is then sent a signal to show successful completion.

6.1.3.2 ENCDEC_poll()

The encdec_poll() routine is called when there are several devices sharing the same interrupt priority. The poll() routine checks the ENCDEC board's register to verify whether it has requested the interrupt. If the ENCDEC board did request the interrupt then the poll routine calls the encdec_intr() routine.

6.1.4 Private Support Routines

6.1.4.1 ENCDEC_strategy()

The encdec_strategy routine provides common code for the _read and _write routines in the transfer of data to and from the board. The encdec_strategy routine is called by the kernel support routine physio.

6.2 Long Integer

Most public cryptosystems require long integer arithmetic. The integer size is usually anywhere up to 1024 bits in length and to deal with this problem a basic library of long integer functions was developed, to provide the functions for the mathematical module of CISS. All cryptographic routines can be constructed from the basic operators addition, subtraction, multiplication, and division. The diagram below shows the hierarchy for the provision of security services within the CISS architecture.

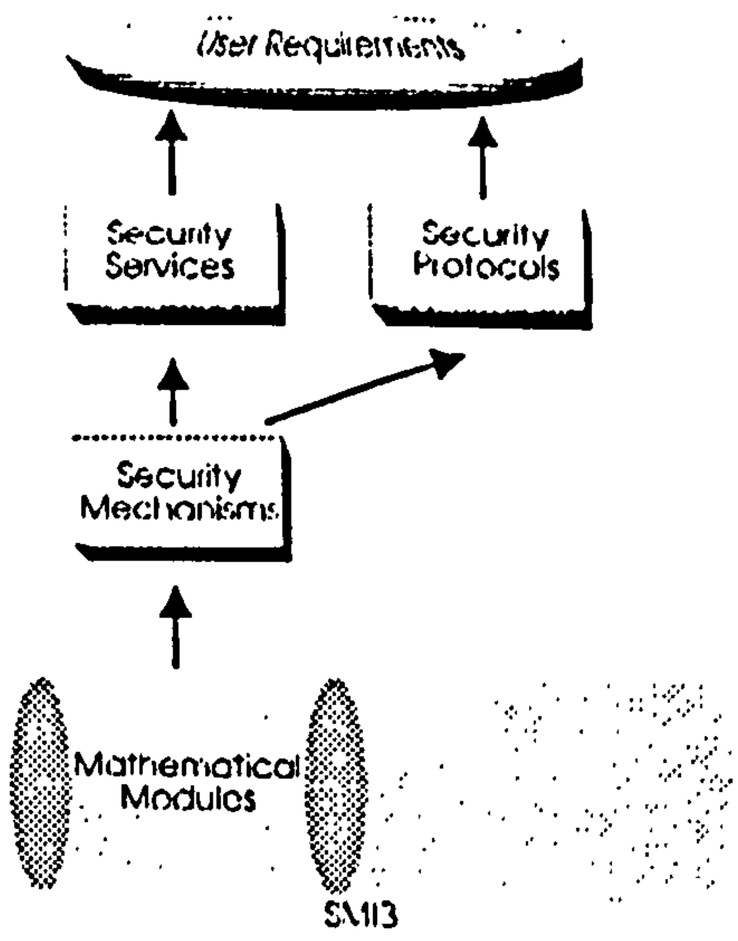


Figure 6-2 Mathematical Module

The long integer libraries have been written for the high level languages C and C++ which provide easy porting to the UNIX system, for which a prototype CISS has been developed. It is therefore necessary to provide optimised versions of these to any mathematical modules that are attempting to provide the cryptographic algorithms used within CISS through software. The long integer library is included upon a disk with this thesis and the header file for the long integer modules can be found in Appendix C.

The C++ library defines a class 'long integer' and overloads the operators so that programs that use the long integer library can simply use the normal operators when manipulating long integers, making program generation using the developed library tidier and easier to read. However, in C the operators cannot be overloaded and separate functions have been developed to provide the needed functionality, C libraries were necessary for current compatibility with UNIX.

6.2.1 The Four Basic Algorithms

The four basic algorithms of +, -, / and *[75], used by the mathematical module developed are detailed in Appendix C. Because computers are mainly optimised to deal with words that are multiples of 8 bits in length the long integer library uses numbers of base 256.

6.2.2 Further Operations

With modifications the four basic algorithms were made capable of dealing with negative integers. The basic algorithms for mathematical operations can be represented as:

1. S = add(A, B) (Addition);
2. D = subtract(A, B) (Subtraction);
3. P = multiply(A, B) (Multiplication);
4. Q,R = divide(A, B) (Division).

What follows are the most essential algorithms found within the long integer library that allows the implementation of the RSA algorithm.

6.2.2.1 Modulus

The calculation of the modulus is accomplished through the use of the basic division algorithm. Given two integers $A = (A_1, A_2, A_3,.....A_n)_b$ and $B = (B_1, B_2, B_3,.....B_n)_b$ we wish to form the radix-b modulus of A mod B. We use the basic division algorithm discarding the quotient and using the final $M = (A_{m+1}.....A_{m+n})_b$ / d, (d is found in Appendix C).

1. P, R = divide(A, B)
2. M = R

The algorithm gives us the function:

M = mod(A, B) (Modulus).

6.2.2.2 Modular Exponentiation

Exponentiation is performed using the left to right algorithm as described in section 5.6.4.3.

Given three integers,

$A = (A_1, A_2, A_3, \ldots A_n)_2$,

$B = (B_m, B_{m-1}, B_{m-2}, \ldots B_0)_2$,

and $M = (M_1, M_2, M_3, \ldots M_k)_2$.

We wish to calculate $E = A^B \bmod M$

1. Acc = 1, j = m, where m is the first non-zero position.
2. If $B_j = 1$ goto 9
3. If Bj = 0
4. Acc = multiply(Acc, Acc)
5. Acc = mod(Acc, M)
6. j = j - 1
7. If j < 0 goto 15
8. goto 2
9. Acc = multiply(Acc, Acc)
10. Acc = multiply(Acc, A)
11. Acc = mod(Acc, M)
12. j = j - 1
13. If j < 0 goto 15
14. goto 2
15. E = Acc

The algorithm provides the function:

E = mod-exp(A, B, M) Modular Exponentiation.

6.2.2.3 Greatest Common Divisor

This is also called Euclid's Algorithm and it is used within the RSA algorithm has been demonstrated in Chapter 2. Given two positive integers (A, B) we wish to find their greatest common divisor. Euclid's algorithm gives us the function,

X = GCD(A, B), where X is the greatest divisor.

1. s = A, t =B
2. If S ≤ 0 goto 7
    3. d = s
    4. s = mod(t,s)
    5. t = d
    6. goto 2
    7. X = d

6.2.2.4 Multiplicative Inverse

The extended Euclid algorithm is also required within the RSA algorithm, it allows us to calculate the multiplicative inverse of A modulo n, where A* u ≡ 1 (mod M).

Update(a, b)

1. temp = b
2. b = subtract(a, multiply(y, temp))
3. a = temp

Euclid's extended algorithm, u = inverse(A, M)

1. g = M, h = A, w = 1, z = 0, v = 0, r = 1
2. if h ≤ 0 goto 6
3. y = Q, where Q,R = divide(g, h)
4. update(g, h), update(w, z), update(v, r)
5. goto 2
6. u = mod(v, M)

6.3 RSA

The functions provided for the provision of RSA functionality obey the following naming policy:

EC_function_name() is a software function;

ED_function_name() is a function that utilises the ENCDEC board functionality.

Both the RSA software library and the ENCDEC board access routines have been written in C and C++, the header files for both can be found in Appendix C.

6.3.1 RSA Framing

Because the RSA algorithm works on blocks of data, the size of which is dependent upon the modulus used for the RSA calculations, it is necessary to frame data that is encrypted. Framing can also help against the use of a replay attack using captured encrypted frames. There is no policy for framing and it is system dependent, the framing strategy chosen for

the ENCDEC board is compatible with the CRYPTECH policy [76]. Framing is necessary for block cryptographic systems, as data to be encrypted will rarely be of the correct size and will therefore need at least a basic form of filling and the information to indicate the amount of filling that was used.

A message to be encrypted is therefore seperated into discrete packets whose size is the same as the modulus of the key pair used and whose numerical value is less than that of the modulus.

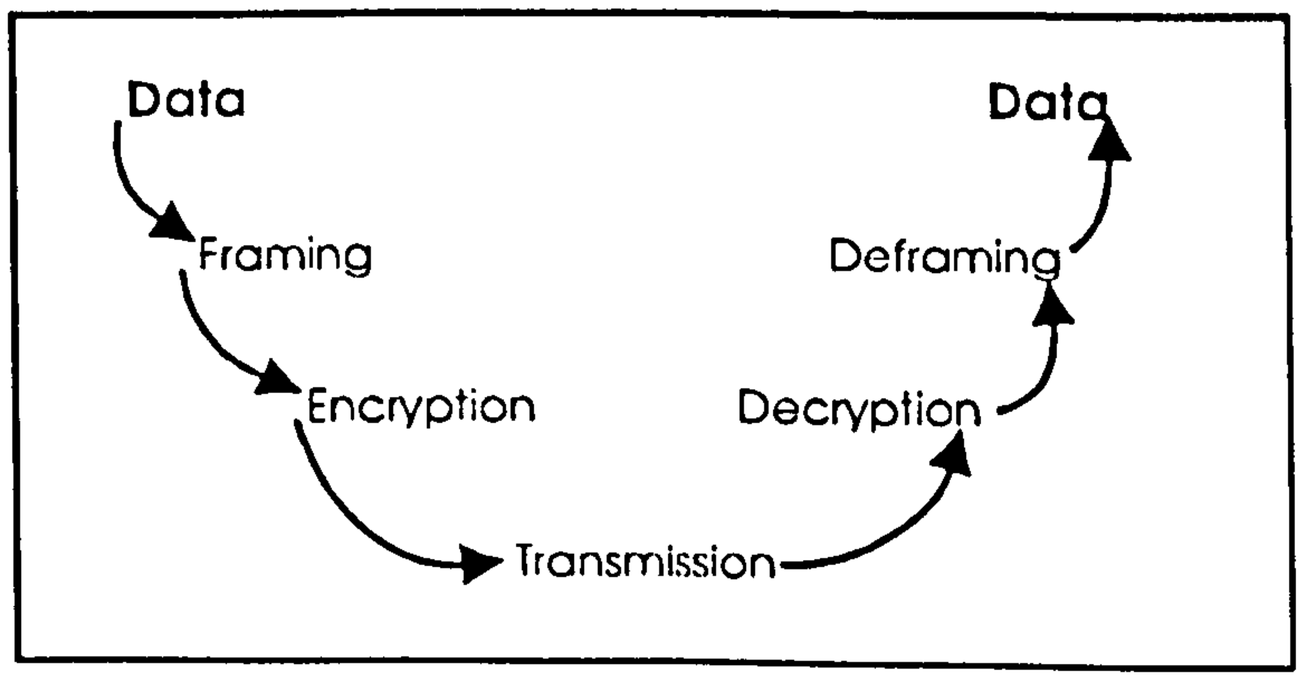


Figure 6-3 Framing Within a Cryptosystem

A frame is made of four segments:

1. header: 2 bytes;
2. framing information: variable;
3. fillup data: variable length;
4. message: variable length.

The frame header is made up of 2 bytes. Byte 1 must be one less than the most significant byte of the modulus, making sure that the overall numerical value of the frame is less than the modulus. Byte 2 indicates what information is contained within the information field of the frame. The byte 2 value is indicated below

MSbit
1. set to 0
2. set to 1
3. optional field present
4. application data present
5. timestamp present
6. length field present
7. status&routing field present
8. frame order field present

LSbit

The optional field and application data field are one and four bytes long respectively. The timestamp is four bytes long and used to help combat replay attacks. The length field is used to indicate the length of the message portion of the frame in the last frame of the stream. The status&routing field is 1 byte in length, the first nibble indicates the position of the frame within a stream and the second may be used for routing information but is unused at this moment.

A first nibble of value 0001 indicates the first frame.

A first nibble of value 0010 indicates the final frame.

A first nibble of value 0011 indicates a single frame.

The frame order field is used at the decryption end of the RSA process in the sequencing of frames in a stream, it is a modulo 256 counter, using a single byte.

The fillup portion of a frame is the data used to pad out the frame so that it is of the same length as the modulus. The framing protocols fills the message with random bytes which are removed from the message deframing by using the length field of the information portion of the message.

The final portion of a frame is the message portion, the actual data that is to be sent confidentially.

6.3.2 Key Generation

The key generation for an RSA key pair is described within Appendix A. The routines written for the facilitation of key generation are:

- _pz_verify()

  Tests a long integer a specified number of times for primality.

- _pz_gen()

  Generates a prime number with a specified first nibble, using a specified number of primality tests.

- _pz_pair_gen()

  Generates a pair of strong prime numbers, with a specified length and first nibble for their product.

- _pz_key_step()

  Produces an RSA key pair, from two prime numbers.

6.3.3 Key Storage

RSA keys are stored within files that can be encrypted by a CISS private key or held secure by some other means. Two files are required for an RSA key pair, both files hold one exponent and the modulus. The exponent and modulus both occupy fields of 92 bytes length.

A key structure, as defined within the header files found in Appendix C, consists of two segments of type long integer, an exponent and the modulus.

- _keyfile_trafo()

  Stores the contents of a keyfile in a key structure.

- _keystruct_trafo()

  Transforms a key structure into a keyfile, with specified name.

A keyfile looks like:

Exponent 92 bytes	Modulus 92 bytes

6.3.4 Encryption/Decryption

Information to be encrypted can be passed to the RSA functions as a data structure or as a file. The two functions used for encryption/decryption are:

- _file_crypt()

  Encrypts/decrypts the contents of a file and writes the result of the RSA calculation to another file.

- _data_crypt()

  Encrypts/decrypts the contents of a data structure writing the result of the RSA calculation to another data structure.

6.4 DES

The functions provided for DES functionality obey the following naming criteria:

ED_function_name() are the functions interfacing with the ENCDEC board;

EC_function_name() are the functions implementing DES in software.

Framing for the DES process is simple. The final byte of a message contains the number of bytes used to fill the final DES cryptographic block. The padding is filled with random numbers.

6.4.1 Key Generation

The key is generated through the use of the,

- _generate_des_key(),

  function, that generates a DES key through the selection of 8 random bytes.

When the key is generated an initialisation vector is also generated (see Chapter 1, which describes the use of IVs within DES modes of operation).

6.4.2 Key Storage

The two functions are.

- _des_key_write_to_file()

  Stores the DES key and IV within a file according to the format shown below.

- _des_key_read_from_file();

  Reads the contents of a DES key file and stores the key and IV into a DES key structure.

A DES key file is made up of two parts

8-byte DES key	8-byte IV

6.4.3 Encryption/Decryption

Encryption/Decryption using the DES algorithm is performed by:

- _des_file();

  Encrypts/decrypts the contents of a file using the DES algorithm and stores the results within a file.

- _des_data();

  Encrypts/decrypts the contents of a data structure using the DES algorithm and stores the result within another data structure

6.5 Conclusion

The software routines written to accomplish the RSA and DES algorithms are the mirror of the routines written to interface with the ENCDEC prototype, allowing a good development platform on which security can be built, with easy interchangeability during development. The long integer library, of which a few of the most important functions were described, provides the required functionality for the mathematical module within CISS using RSA as its asymmetric cryptosystem. The software implementation of RSA was constructed using the long integer library.

Chapter 7

7. User Authentication Mechanism for a Local Security Unit

This chapter describes new research into an authentication mechanism that can be used by the LSU of the 3-L architecture. The chapter begins by describing the basic concepts user authentication and keystroke analysis. The experiment is then presented in three parts: sampling; analysis and optimisation. The results from the analysis and optimisation sections of the experiment are discussed. Finally, the application of keystroke analysis within CISS is discussed.

7.1 Introduction

Secured computer systems aim to protect the data processing services they provide and the data they store within memory. The first line of defence that any computer system has against malicious attacks is to deny access to potential attackers. The only way that this can be done is to identify the person that is attempting to log onto the system, and thereby determine whether they have a legitimate right to services.

Obtaining access to a computer system through coercion of a legitimate user is a possibility that no authentication mechanism can deal with, unless the system is sensitive to duress. This sensitivity is not always enviable as it can lead to antagonising an attacker and endangering the user; imagine an armed robbery where a voice activated lock will not open because the bank manager is under extreme stress. This chapter does not attempt to deal with such an occurrence and it is felt that the only way to combat this type of attack is through the implementation of a good security policy.

In Chapter 4 the 3-L architecture was described and the LSU defined. One of the most important functions that the LSU has is in the provision of user authentication mechanisms.

7.2 User Authentication

An individual possesses three qualities that may be used to identify them [77]:

- what the individual knows;
- what the individual owns physically;
- what the individual is.

Any or all of these qualities can be used in a user authentication mechanism, with the strongest authentication mechanisms using more than one of these qualities.

7.2.1 What They Know

Authentication systems exploiting this quality generally require the individual to know a secret password which must be presented when challenged. Other systems may use a database that contains detailed personal knowledge about its users. For example, whenever someone attempts to login they are asked random questions, such as, birthdates of offspring, names of relatives, favourite item of music, and the answer given is compared to those in the database.

The password is the most common authentication mechanism in use at present. When a user logs into a computer system he is usually challenged for a password, this password is usually cryptographically protected and kept in a database indexed by the user's login name. If users choose their passwords according to common security policy [78], then the password mechanism is very secure, especially when augmented with a limited amount of unsuccessful logon attempts (i.e. a "three tries and you are out" policy).

However, it has been shown in various studies [79], that no matter how theoretically secure a password mechanism is, its overall security is compromised very easily by:

- users picking easily guessable words such as, "password," "drowssap," or using words derived from their login names, or found in dictionaries. Dictionary words are

particularly dangerous because programs exist to crack user accounts [80], by simply using an exhaustive search technique to obtain passwords;

- because some users communicate their passwords to other users, or write them down near their terminals, for example the BT [81] case.

Even with its faults, the password mechanism is still in wide use as the sole form of user authentication on many computer systems. The low cost of implementing a password mechanism and the familiarity users have with the mechanism makes certain that for the foreseeable future it will continue to be the most popular form of authentication.

7.2.2 What They Own

In this situation the legitimate user will have a *token* that must be presented when challenged. The token usually takes the form of a magnetic stripe card or increasingly a smart card [82]. The difference between the two cards is a matter of intelligence, the smart card, as its name implies, has some intelligence in the form of on-board processing capabilities. The present smart card technology is split into two main areas: active cards and second generation cards. Active cards are meant to be used in a stand alone mode where power and processing is done on-board, while second generation cards are passive and require a card reader which provides power. However, most cards in use fall into the area between the two.

The use of tokens has increased in everyday life, with almost everybody carrying one (e.g. Automatic Teller Machines cards). Tokens used in an authentication mechanism require the purchase of specialised equipment. For example, in the case of magnetic stripe cards, not

only must a magnetic stripe card reader be purchased for all physical locations where user authentication is deemed necessary, but so must cards for all users on the system. This initial outlay of capital on specialised equipment can make a token scheme prohibitive in most environments except those that require high security, such as government installations, banks, etc.

With the continuing reduction of processor size, smart cards are being used in 'see through' authentication mechanisms [83], that do not necessarily require a specialised reader but require the purchasing of smart cards for all users.

Tokens used in an authentication mechanism can provide good security, but users are fallible and may lose or forget them. The token may also be stolen, seriously compromising security.

7.2.3 What They Are

When a biological characteristic can be used to identify an individual it is called a *biometric.* Biometrics can be divided into two groups, *active* and *passive* [83]. A passive biometric is a physical characteristic that may be scanned with no effort by the subject other than the exhibition of said characteristic, e.g. finger-prints, or facial features. An active biometric is hand writing [84], or voice patterns [85].

In a biometric authentication system an attacker can only hope to imitate the characteristic that the mechanism uses to identify legitimate users. Some of the biometrics that have been researched are human faces, speech, finger prints, body odour and iris patterns. Due to the

technologies employed in biometric measurement the field is relatively new and has not found many applications in existing authentication mechanisms..

However, biometrics offer potentially the best form of user authentication since they measure actual characteristics of what a user is. As with any authentication mechanism they are not fool proof. In passive biometric authentication mechanisms the user must present a part of the body to identify themselves. It is of course possible for an attacker to obtain the necessary body part. However, the effort and will on the part of the attacker must be far greater than that necessary, for example to steal a slip of paper with a user's password on it. It is also possible for two people to have similar active characteristics and therefore fool an active biometric authentication mechanism, but, if an attacker happens to have a dissimilar active biometric they must make an extreme effort to mimic the user.

Biometrics used in an authentication mechanism offer several advantages:

- requires very little training for users to use (grandma friendly);
- there is no risk of losing, or having a token stolen as there is in a token mechanism;
- no risk of having an item of knowledge (password) becoming known.

However, biometrics also have several disadvantages:

- all passive systems require processing equipment either audio, visual or chemical that can be very expensive;
- the processing of biometric data can be very processer intensive;
- the information to deceive a system may be acquired from the user through sampling;
- they are never perfect, recognition of a subject is generally given as a degree of confidence.

Biometric authentication systems utilise the only measurement that actually tests the identity of a person. It can be argued that the first two qualities merely identify the knowledge of a password and the token itself, not the person themselves.

Potentially the most useful mechanism was one which exploited a biometric measurement, either active or passive. Most biometric measurements such as voice recognition, finger print classifiers, and vein pattern recognition require very specialised equipment, which in most cases is very expensive. This chapter investigates and discusses a cheap solution for an active biometric security mechanism, that can easily be introduced into most existing systems using the most common authentication mechanism - the password.

7.2.4 Typing Analysis

In 1975 Spillane [86], proposed that just like handwriting, typing styles were an active biometric measurement. When analysing typing, various factors can be measured, for example:

- inter-keystroke timing (the timing between consecutive key depressions);
- depression time (the length of time a key is depressed);
- pressure of key depression.

Further research [87], [88], [89] into the area of typing analysis has confirmed Spillane's proposal. An advantage of typing analysis over other biometric systems is that it minimises some of the general disadvantages present in them:

1. it potentially requires no extra equipment;
2. most of the data processing can be done in an initial learning phase.

Typing analysis is accomplished in one of two ways: *statically* or *dynamically*.

Static analysis is conducted upon text strings that do not change. There are two points at which such analysis could significantly augment the security of a computer system; in the initial authentication of a user (i.e. at login) and whenever a volatile command is used. At *login* there are usually two points that are ideal for static typing analysis, at the entry of *user name* and the corresponding *password*. A volatile command is one which when used can be potentially harmful. They are commands that generally delete or modify data stored within computer memory, such as the 'deltree' or 'del' commands in MS-DOS [90].

Dynamic analysis is conducted upon previously unknown text strings that are entered by a user. The entry characteristics can be stored and analysed later or they may be analysed in real-time, allowing continuous supervision of a session.

The effectiveness of a user authentication mechanism can be measured using two figures; the *false rejection rate* (FRR) and the *false acceptance rate* (FAR). The FRR represents the percentage of legitimate users actually classified by the user authentication mechanism as attackers. The FAR is the percentage of attackers classified by the mechanism as legitimate users.

The use of static typing analysis in a security system can be used to increase the security of the most widespread authentication mechanism; the password. It is to this end that the proposed static typing classifier was investigated.

7.3 Overview of the Investigation

The investigation conducted into typing analysis can be split into three phases:

1. sampling;
2. classification of typing samples;
3. data optimisation.

What follows is a description of the three phases.

7.3.1 Sampling

7.3.1.1 Sample Collection

The typing samples collected for the investigation were taken using an IBM PC with a 386-33MHz DX processor using a standard UK configuration Qwerty keyboard. Through examination of other research in the area of typing analysis [87], [88], [89], [91], [92], it was decided that the inter-keystroke timing was the best discriminator of people's typing styles, and one of the few useful typing characteristics for static analysis. The sampling program was written in C, utilising the DOS keyboard interrupts to time the intervals between key depressions.

7.3.1.2 Reference Texts

Every subject that took part in the experiment entered four set reference text strings. The four strings were as follows:

REF1 : "*PRINT*";

REF2 : "*TRANSFERENCE*";

REF3 : "*RED SKY AT NIGHT*";

REF4 : "*RUGBY PLAYERS ODD SHAPED BALLS*";

At the end of each text entry the subjects had to depress the *return* key.

The text strings were chosen to make good use of the keyboard, such as cross digraphs that use both halves of the keyboard, REF1 was chosen for familiarity to the test subjects, as it is one of the most common words that a computer literate person will type, all four reference texts use the three main rows of keys on a Qwerty keyboard and REF4 is unusual in the combination of words and digraphs.

The subjects were forced to use static text strings to make classification of their samples as difficult as possible; common sense dictates that typing styles are exaggerated when users type in familiar text strings such as their names. The choice of set strings of text represents a worst case scenario for typing analysis of passwords (where all users have the same password) and the real situation of volatile commands that are set by the operating system being used and common to all users. The four text strings were also graduated in length to evaluate what effect it had on classification of samples.

Before the test subjects began typing they were told:

1. speed was a factor;
2. any mistakes made during entry would cause the entry to be invalid and they would have to enter the whole text string again.

A screen shot of the entry system can be seen below

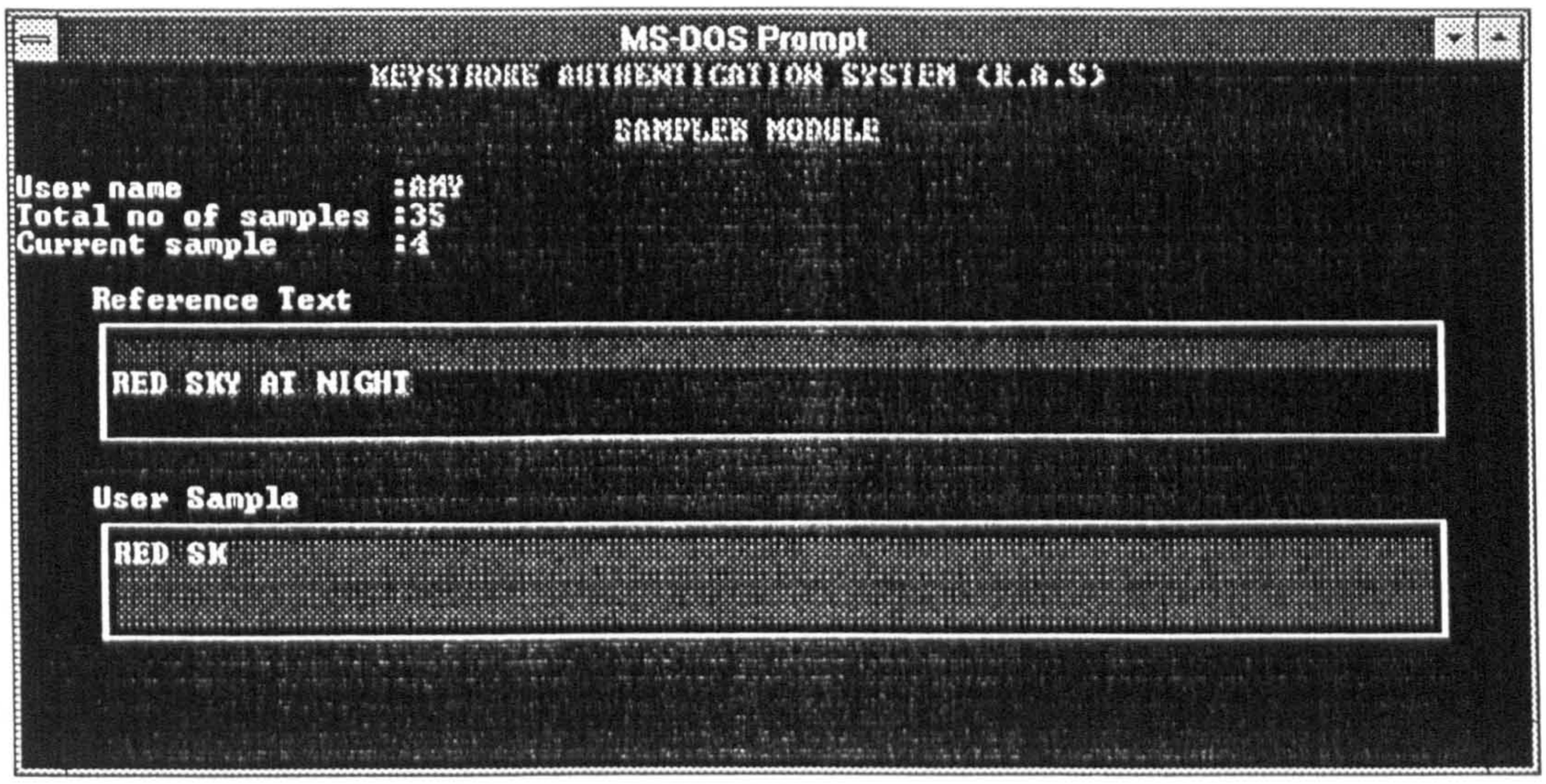


Figure 7-1 Text Entry Screen Shot

Fifteen subjects took part in the experiment, who were all reasonably familiar with typing, coming from a computing background. Every subject entered each text string 35 times, making all entries at one 10-15 minute sitting. The inter-keystroke times were recorded by the sampling program and stored within files.

7.3.2 Classification of Typing Samples

7.3.2.1 Current Classification Techniques

Classification of patterns conventially falls into three methods.

1. nearest neighbour techniques;
2. statistical techniques;
3. artificial intelligence (AI) pattern recognition schemes.

Nearest neighbour techniques[93] utilise distance metrics such as Hamming, or Euclidean, to determine the distance of an unclassified pattern to classified patterns. A threshold distance needs to be set so that unclassified patterns that have a distance below the threshold are classified as a recognised pattern.

Statistical techniques such as the Bayes classifier [94], require that a probability density function be chosen, such as the Gaussian or Normal models; co-variance, standard deviation, and mean of the training set are calculated and the unclassified pattern is tested. As in the nearest neighbour techniques it is necessary to calculate a threshold value above which an input pattern is classified as a recognised pattern.

AI techniques for pattern recognition employ the use of neural networks for example ADALINE networks [95], or the multi-layer perceptron [96]. These take vector inputs and produce a vector output that is used as an indication of classification.

When the first two techniques are applied to typing analysis they can work on the separate digraphs or the entire sample as a whole. When working upon separate digraphs a percentage threshold must be chosen above which the entire sample is classified as coming from a legitimate user.

From the three techniques of classification it was chosen to investigate the use of the multi-layer perceptron. The reason for choosing the multi-layer perceptron lies in the fact that it is one of the few classification techniques that can solve problems involving non-linear

separability, and the potentially infinite variety of individual typing styles suggests that typing analysis is a non linear problem.

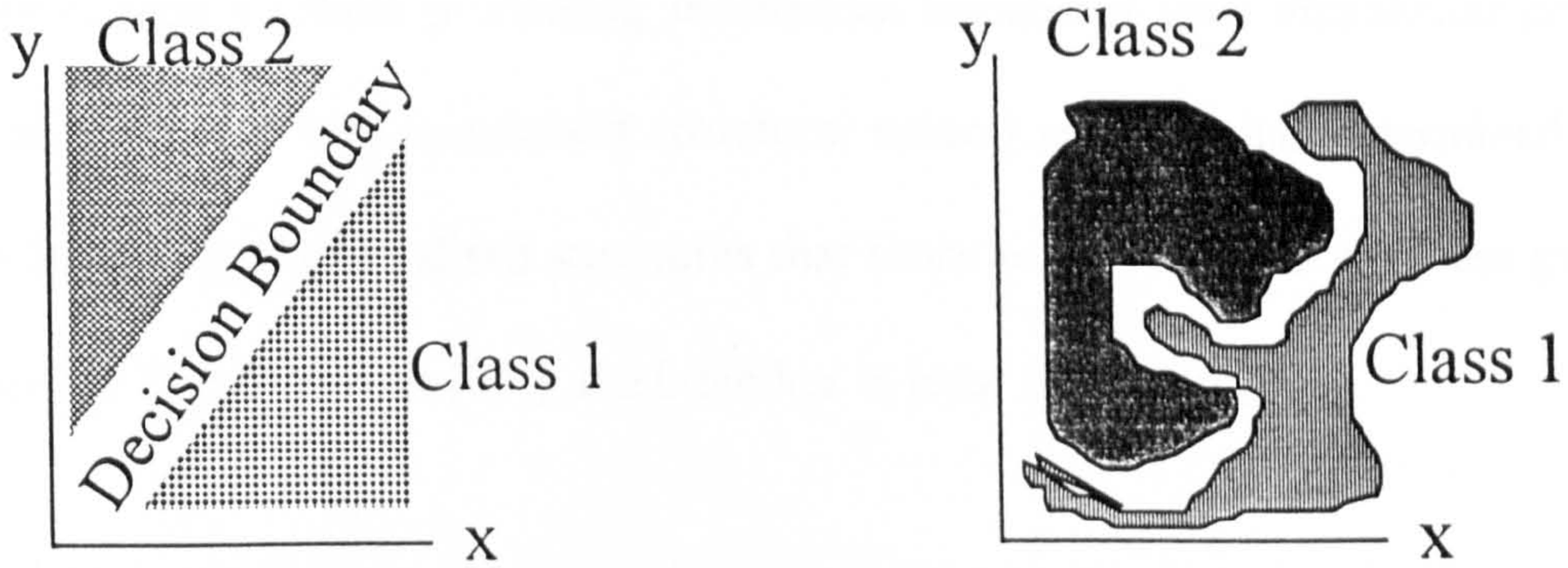


Figure 7-2 Linearly and Non-Linearly Separable Problem

7.3.2.2 The Perceptron

The human brain exhibits several qualities of which hardware designers and software programmers wish computers were capable. Although computers can calculate problems quicker than a human, they lack the robust nature that evolution has made part of the brain. A brain is capable of processing many instructions at the same time, this *parallel processing* allows us too walk, talk, and process visual data, simultaneously, without a noticeable deterioration in performance. Damage in part of the brain does not result in a complete shutdown, known as *fault tolerance*, continuing damage results in *graceful degradation.* The brain also has the ability to *learn* solutions to problems, it is not necessary for it to be programmed for all eventualities. It is because of these abilities of the human brain that research in the development of artificial intelligence based on principles learnt through neurobiology has become mainstream. One of the biggest areas of research is the field of *pattern recognition*, whether in seismic activity or fluctuations in the stock market.

Wherever it is possible that a pattern may exist a researcher will try to point an artificial intelligence algorithm at the problem.

The brain is such a robust processing mechanism because it uses *distributed processing*. Instead of building a very specialised structure, natural selection has determined that it is better to build lots of generalised structures that share work. In the brain these generalised structures are called neurons [97], the brain has at least 10^{10} neurons.

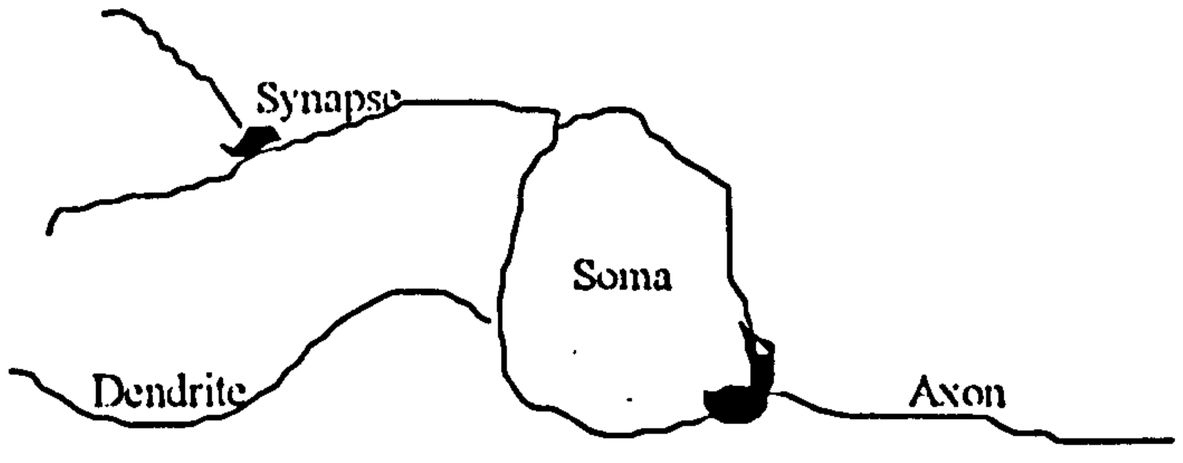


Figure 7-3 Biological Neuron

In 1943 McCulloch and Pitts proposed a model of a biological neuron. It was not till 1962 that simple collections of the basic neuron model became known as *Perceptrons* [98].

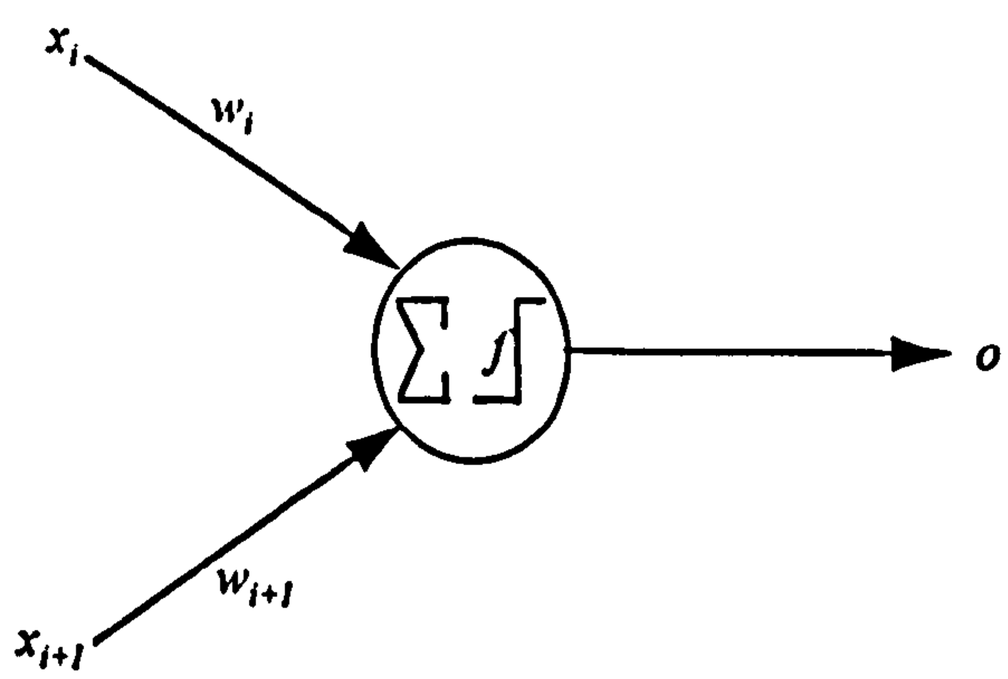


Figure 7-4 Model Neuron

The output *o(t)* of a perceptron is calculated through a feedforward algorithm, w_i is the weight of the ith input, x_i is the ith input and ϕ is the threshold value.

$$o(t) = f\left(\sum_{i=0}^{n} w_i(t)x_i(t) + \phi(t)\right)$$

where,

$$f(x) = 1 \quad x > 0$$

$$f(x) = 0 \quad x \leq 0$$

The perceptron gets better at solving a problem as its weights are adjusted through a learning algorithm. In supervised learning this is done by comparing the output of the perceptron to that wanted.

If *o(t)* is correct, $w_i(t+1) = w_i(t)$

If $o(t) = 0$, and it should be 1, $w_i(t+1) = w_i(t) + x_i(t)$

If $o(t) = 1$, and it should be 0, $w_i(t+1) = w_i(t) - x_i(t)$

Perceptrons can provide solutions for only linear problems, (for example a simple non-linearly separable problem is the exclusive-OR) because of this problem perceptrons took a back seat in research. It was not until 1986 that they made a re-emergence when Rumelhart, McClelland and Williams[99], put forward their learning rule for multi-layer perceptrons, and thus gave perceptrons the ability to solve non-linearly separable problems.

7.3.2.3 The Multi-Layer Perceptron

A multi-layer perceptron has an input layer, an output, and at least one hidden layer which is not directly connected to the outside world. The topology of a multi-layer perceptron can be seen in Figure 7-5.

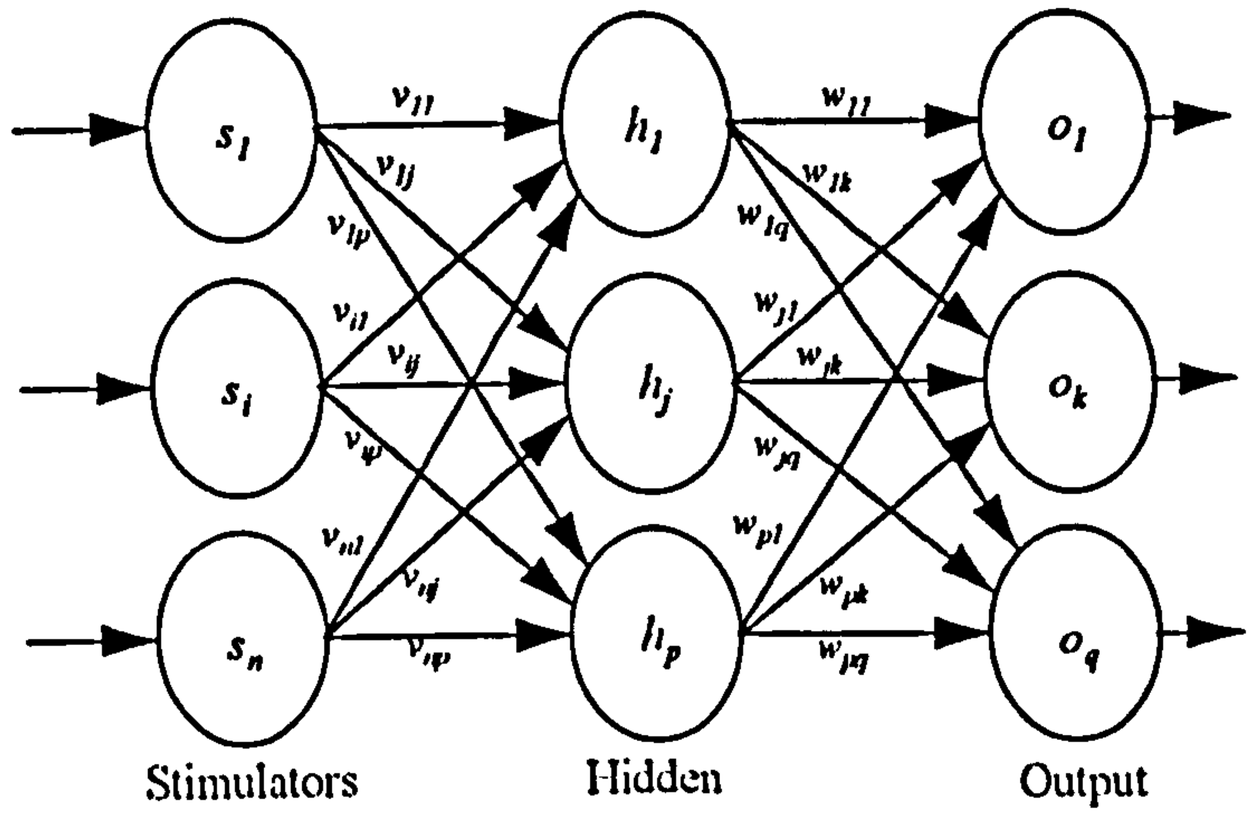


Figure 7-5 Multi-Layer Perceptron Model

The output of the stimulator layer is s_i where $1 \le i \le n$.

The output for the hidden layer nodes is calculated using,

$$h_j = f\left(\sum_{i=1}^{n} s_i v_{ij} + \theta_j\right)$$

where $1 \le j \le p$, θ_j is the threshold value for node j in the hidden layer, v_{ij} is the input line weighting.

Unlike the simple perceptron the multi-layer perceptron's output function is a sigmoid, to allow for a modified learning algorithm.

$$\text{where } f(x) = \frac{1}{1+e^{-kx}}$$

The output from the output layer nodes is calculated using,

$$o_k = f\left(\sum_{i=1}^{p} h_j w_{jk} + \Gamma_k\right),$$

where $1 \le k \le q$, Γ_k is the threshold value for node k of the output layer, w_{jk} is the input line weighting.

The learning algorithm employed for the multi-layer perceptron used in the experiment was the 'back-propagation' algorithm [99]. The back propagation algorithm tries to minimise the error of each node's output. The algorithm starts calculating errors from the output layer and then propagates the error back to the hidden layer till the errors for all nodes is known.

The back-propagation algorithm.

1 : The error is calculated for each output node o_k such that the error of the output node d_j is

$$d_k(t) = o_k(t)(1 - o_k(t))(T_k(t) - o_k(t))$$

for all k where T_k is the expected output of node o_k.

2 : The output error e_j for each node in the hidden layer is calculated using

$$e_j(t) = s_j(t)(1 - s_j(t))\sum_{k=1}^{q} w_{jk}(t) d_k(t)$$

for all j.

3 : Adjust the weights connecting the output and hidden layers

$$\Delta w_{jk} = \alpha\, h_j(t) d_k(t)$$

for all j, where α is a positive constant controlling learning rate

4 : Adjust the input weights to the hidden layer

$$\Delta v_{ij} = \beta\, s_i(t) e_i(t),$$

for all i, where s_a is the output from the input nodes and β is a positive constant controlling learning rate.

5 : Adjust the threshold of all nodes:

For output nodes

$$\Delta \phi_k(t) = \alpha\, d_k(t);$$

For hidden layers

$$\Delta \Gamma_j(t) = \alpha\, e_j(t).$$

As has been previously stated the back-propagation algorithm attempts to find the minimum error, however, early in the training cycle it is possible for the perceptron net to get stuck in a local minima; to combat this an extra term is introduced, the momentum term. This term is added to the weight adaptation equation in steps 3 and 4 of the learning algorithm.

For example the step 3 equation becomes

$$\Delta w_{jk} = \alpha\, h_j(t) d_k(t) + m(w_{jk}(t) - w_{jk}(t-1))$$

where m is a positive constant called the momentum factor.

7.3.2.4 Neural-Net software

So that the principles of the multi-layer perceptron could be more readily understood it was decided that the software should be written from scratch to implement the neural nets. The language used was C++, the header files generated can be seen in Appendix C.

Three classes were defined: node, path and net. An arbitrary perceptron net was built by defining the overall structure of the perceptron net in the object net; the net class describes the structure of the nodes and their functions, stimulators, thinkers, responders. The net object holds a list of nodes that are used to describe the make up of all nodes within the net, containing information such as threshold values etc. The node class also contains a list of path object that are the weighted input connections to the node. The code used to generate the multi-layer perceptrons can be found on a disk included in this thesis.

While the neural nets were being trained it was possible to observe the error values of the output node on a graph. This gives an idea of how well the neural net is performing.

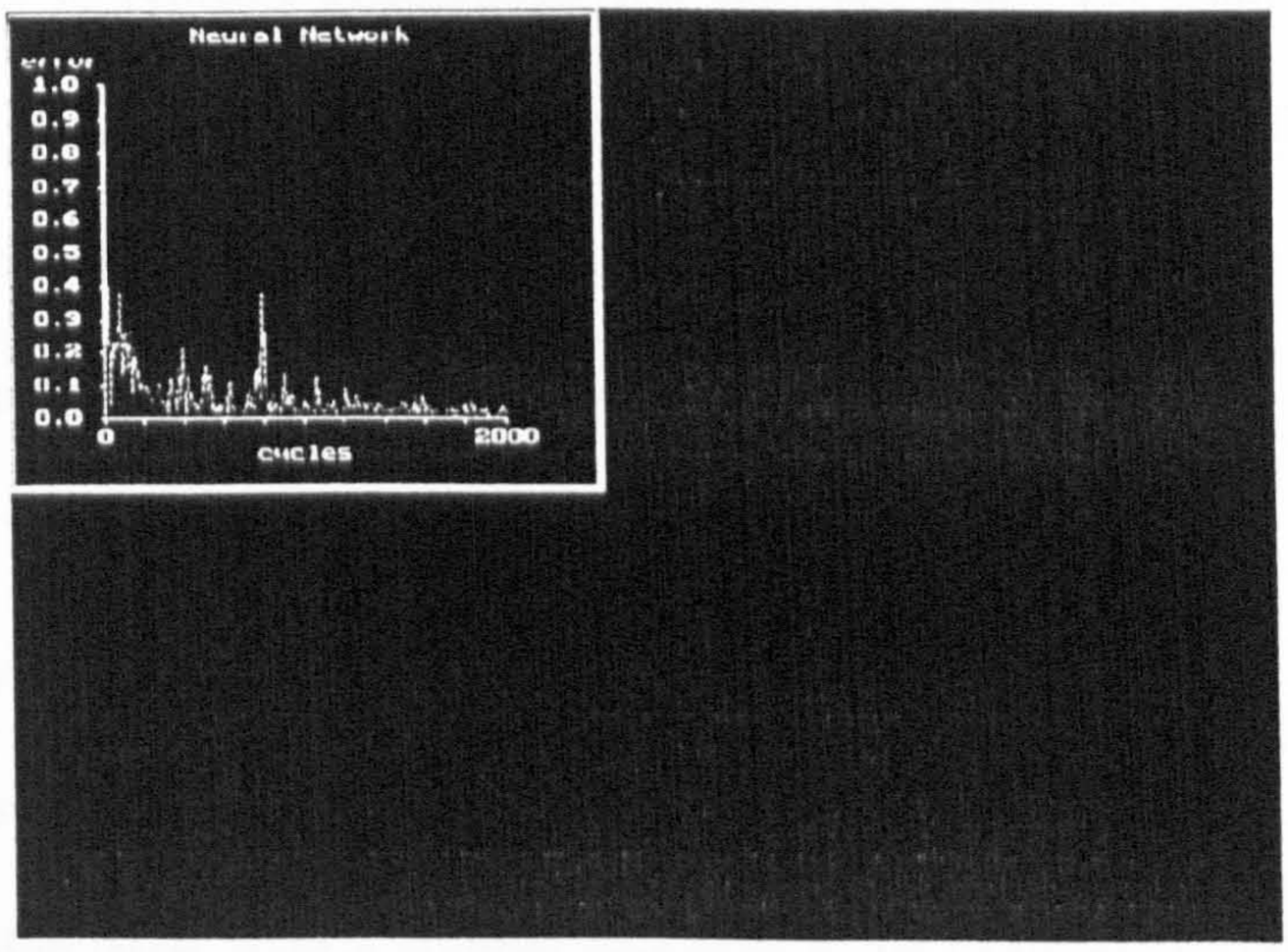


Figure 7-6 Neural Net Learning Screen Shot

A multi-layer perceptron network was created for each REF text for every test subject. Each test subject's networks were taught to distinguish their keystroke timings from the other test subjects. The neural net, if working perfectly, would give an output of 1 if the specific user's timings were the input vector, and a 0 if the input vector was one of the other 14 test subjects. However, multi-layer perceptrons rarely give perfect outputs so a

threshold scheme had to be set. It was decided to evaluate the efficacy of three threshold values 0.7, 0.8, and 0.9, any value output by the network above these thresholds was taken as recognition of a specific test subject.

7.3.2.5 Learning Ratios

A neural net learns through repeated exposure to training vectors. It was decided that the first twenty-five samples of every user would be used as a training set for the neural net. The final ten samples of every test subject were used to evaluate the effectiveness of the trained multi-layer perceptron. The positive training set (comprised from the first 25 samples of the subject to be identified by the neural net) were un-filtered, it is possible to filter training inputs to remove uncharacteristic samples, using for example a kohonen net [100], to identify those samples that are not characteristic and excluding them from the training process. However, it was felt that it would be more beneficial to see how the multi-layer perceptron performed on raw data.

The basic topology of the initial multi-layer perceptron was:

- input nodes equal to the number of digraphs in the REF text;
- one hidden layer, with a number of nodes equal to twice the input nodes;
- a single output node.

Initially a pilot study was conducted into how long the training cycle of the multi-layer perceptron should be. It was soon discovered that anything below 2000 cycles gave poor results, with results improving as the number of training cycles was increased. No benefit was found from training the neural net beyond 4000 cycles.

Therefore, the neural net was trained with 4000 presentations of training samples. The training vector used for each cycle was chosen randomly from the training set. However, when the results were evaluated they gave an FRR of 100% and an FAR of 0% for all threshold values. Obviously the multi-layer perceptron was not receiving a large enough proportion of positive samples from the user it was meant to identify. To combat this lack of true samples a learning ratio bias was introduced into the network training phase.

$$T : F,$$

where T is the number of true samples picked randomly from the training set of the specific subject to be identified, and F is the number of samples picked randomly from the other subjects' training sets.

Therefore, a ratio of 1:1 presents one true sample for every false sample.

Gradually increasing the learning ratio of a multi-layer perceptron trained to recognise REF2, using the basic network topology, can be seen in Figure 7-7 below.

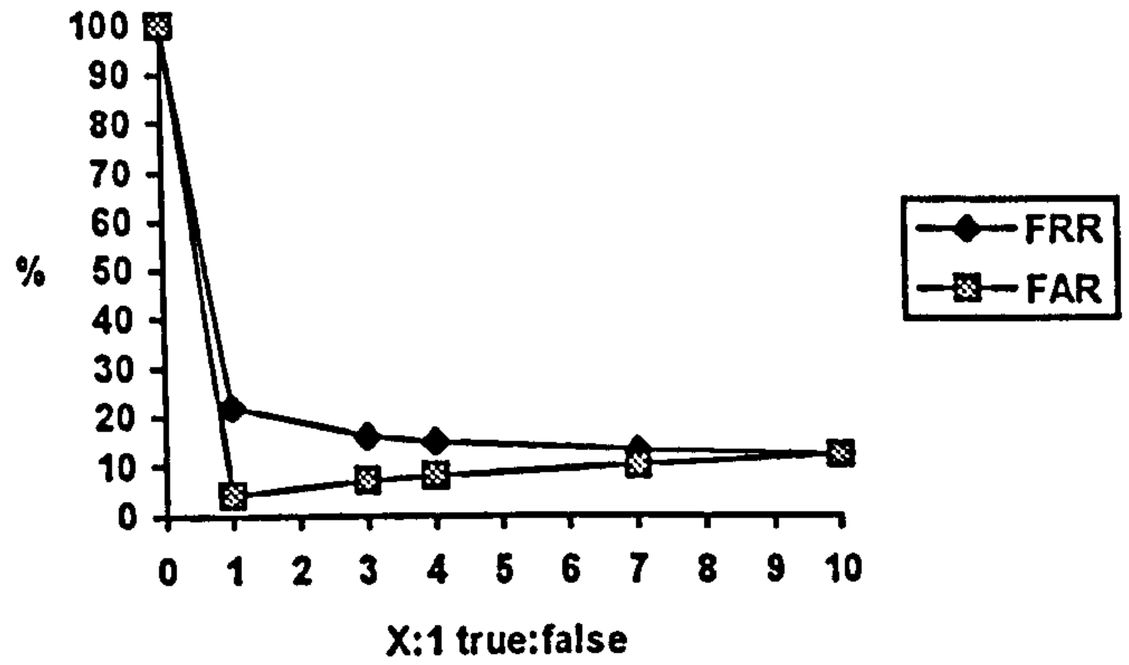


Figure 7-7 Basic Network REF 2 Results

As can be seen from the results, a decision must be made on the type of classification error to be minimised. A low FRR produced by larger learning ratios causes the network to give a higher FAR. A low FRR is wanted when the static analysis of typing is directed at volatile commands, since it is assumed that the majority of users initiating a volatile command are legitimate users. When analysis is conducted at login, security demands a low FAR, since access control wishes a minimum of illegitimate users to gain entry to a system.

7.3.2.6 Network Topology

The Kolmogrov mapping neural network existence theorem [101], states that it is never necessary to have more than three layers in a multi-layer perceptron i.e. an input layer, 2 hidden layers and an output layer, to solve a classification problem. Of course it is very hard to find the specific three or two layer configuration that can solve a particular problem.

An attempt was made to find the best configuration for a multi-layer perceptron to solve the typing classification problem. Numerous two hidden layer topologies were investigated but

the slight benefit, if any, that they made to a reduction in the two indicators (FAR and FRR) did not compensate for the additional processing time in the training phase of the network The effect that changing the number of nodes in a single hidden layer was also investigated, but here too no improvement was found on the basic net topology.

.

7.3.2.7 Basic Results

The networks were evaluated with the final 10 typing samples of each subject, giving for test purposes 150 legitimate attempts and 2100 attempts of illegitimate access.

Using the basic network topology and a learning ratio of 1:1 gave the results below.

	0.7	0.8	0.9
REF 1	12	8	4
REF 2	4	4	3
REF 3	5	4	2
REF 4	3	2	2
Average	6	5	3

FAR given as %

	0.7	0.8	0.9
REF 1	30	42	58
REF 2	22	25	34
REF 3	24	27	34
REF 4	53	38	43
Average	32	33	42

FRR given as %

Table 7-1 Ratio 1:1 Results

A ratio of 4:1 gave the results below.

	0.7	0.8	0.9
REF 1	24	21	15
REF 2	8	6	5
REF 3	8	6	5
REF 4	4	3	2
Average	11	9	6

FAR given as %

	0.7	0.8	0.9
REF 1	10	15	26
REF 2	15	19	25
REF 3	21	23	27
REF 4	28	32	40
Average	19	22	30

FRR given as %

Table 7-2 Ratio 4:1 Results

As can be seen in tables 7-1 and 7-2 above, a moderate increase in the learning ratio helped the neural network reduce the FRR for all REF texts. It is interesting to note that the FAR results are very much better than the FRR for networks trained with the 1:1 ratio. The better FAR results were still present when the network was trained using a 4:1 ratio, however, the difference between the two indicators was considerably reduced.

The low FAR made the network appear more suitable for login security, as the FRR is still a little high for use in a volatile command security mechanism. However, as was discussed above, the FRR can be reduced through larger learning ratios but at the cost of increasing the FAR.

7.3.2.8 Use of Multiple Reference Texts

Since the neural network analysis seemed more suited to a login security mechanism it was decided to investigate the use of two reference texts, thus emulating the login process where a username and password are used. This is a worse case scenario where all users have the same name and password.

Two neural nets were employed in this two reference strategy, one for the login name and the other for the password. Every attempt of authentication to the system is placed into one of three categories depending on the combined result of the neural nets.

- (R)ejected - both samples were rejected by their respective neural networks.
- (L)ow - one of the two samples was rejected.
- (H)igh - Both samples were accepted

REF 1 and REF 2

	R	L	H
0.7	72	26	2
0.8	76	22	2
0.9	82	17	1

Impostor attempts (%)

	R	L	H
0.7	3	29	68
0.8	4	33	63
0.9	9	39	52

User Attempts (%)

Table 7-3 Two Reference Results (REF1 and REF2)

REF 2 and REF 3

	R	L	H
0.7	87	13	1
0.8	89	11	0
0.9	92	8	0

Impostor Attempts (%)

	R	L	H
0.7	4	38	58
0.8	5	40	55
0.9	7	50	43

User Attempts (%)

Table 7-4 Two Reference Results (REF2 and REF3)

As can be seen, it is possible to increase the security significantly, using a threshold value of 0.9 and a requirement of at least an L category result, the FRR can be brought down to 7% and the FAR to 8%.

7.3.2.9 Classification Conclusion

The results from phase 2 of the experiment are very encouraging. It must be remembered that this method of user authentication is not suggested as a total solution but is meant as a way of increasing the level of security on a password mechanism cheaply and relatively painlessly in terms of the complexity of adding it to the system. The experiment was implemented in such a way as to make life as difficult as possible for the classifier, and from the results obtained the multi-layer perceptron has coped very well.

Two conclusions that can be drawn from the results given by the neural net are:

1. that the optimum text length for identification purposes involving keystroke analysis is greater than 5 and less than 31, optimally around 13-20 letters long;
2. the combination of two texts in a login phase would provide adequate security for almost any login mechanism.

7.3.3 Data Optimisation

The next part of the investigation was an attempt to improve the results of the basic multi-layer perceptron through optimisation of the input vectors. Starting from an initial conjecture that:

"each individual has specific digraphs that are particularly characteristic of their typing style,"

the possibility of identifying characteristic digraphs was researched.

The optimisation technique must sort out which of the digraphs used in a REF text are characteristic of the subject to be identified, it was decided that a genetic algorithm [102] would be used for optimisation, as this was a new application within security.

7.3.3.1 Genetic Algorithm

As with the perceptron the genetic algorithm (GA), came about through research into an aspect of the real world; evolution, and more specifically the mechanism behind it, that provides the necessary small increments in performance for natural selection to work, genetics. Evolution is a natural sorting mechanism that seeks to optimise performance of living entities, it relies upon the fact that the more successful individuals of any group will pass on their genetic information to their children. In the process of breeding there is a chance that the genetic material passed onto the next generation will be mutated, introducing a characteristic that was not previously present in the groups genetic pool. If this mutation is a contributing factor to the success of an individual then it is likely to be passed onto the next generation.

Genetic algorithms are slightly different from other algorithms that perform search and optimisation procedures in several ways:

- GAs conduct their search from several points and not from a single point;
- GAs use probabilistic rules and not deterministic ones;
- GAs need only know the objective function and no other auxiliary information;
- GAs need only a coding of the parameter set and not the parameters themselves.

7.3.3.1.1 Parameter Set Coding

Genetic algorithms require that the parameter set be coded into a finite length string. The parameter set in the case of our investigation will be the digraphs present in each reference text. The string representing the possible digraphs is known as a *gene* and the values it

takes on are called *alleles.* The gene is used to calculate the input vectors that are used for a neural net of the basic topology. A bit value of 1 in the gene means that the corresponding digraph is used, a bit value of 0 and it is not used.

For example:

REF 2=TRANSFERENCE[return];

therefore a bit representation will have 12 bits and an allele value from 1 to 2^{12};

a gene that has an allele value of $1A3_H = 000110100011$;

will use the timing from digraphs 'NS', 'SF', 'ER', 'CE' and 'E[return]' as the input vector to the multi-layer perceptron.

At the beginning of the search a population size is chosen which can be any size greater than two. Each member of the population has a genotype, the genotype being the total genetic information for an individual member of a group. The population in the case of typing analysis is made up of *genotypes* that contain the gene describing the digraphs to be used by the neural network. For the initial generation the genotypes had randomly selected allele values.

7.3.3.1.2 Selection

The next step is to discover which individuals of a population are better at the task of identifying a test subjects keystroke timing,. this is achieved by giving each member of a generation a *fitness* value, the higher the fitness, the better the individual was at identifying the subject. To obtain the fitness of an individual a multi-layer perceptron was constructed using the basic topology and trained using only those digraphs selected by the digraph gene.

Each perceptron net was trained using 4000 presentations of samples from the training set, the resulting network was then presented with 200 randomly chosen samples from the training set to obtain an average fitness of the individual,

$$\text{average_fitness} = \frac{\sum_{n=1}^{200} 1 - d_n}{200}$$

where d_n is the output error of the neural network on the nth presentation of a sample.

Once the fitness has been calculated for all individuals in the current generation a probabilistic function selects which individuals will reproduce. Those individuals with higher fitness values have a better chance than those with lower fitness values of contributing their genetic coding towards the next generation. Selection is performed in our case by a roulette wheel function. Each individual of the current generation is given a portion of a roulette wheel in proportion to their fitness value, a random number is then generated and the section of the roulette wheel the number falls into indicates the individual picked to become a parent. This process means that there is always a chance that the most successful individual of a generation will not contribute to the next generation. A partner is then chosen for the first individual picked in the same manner.

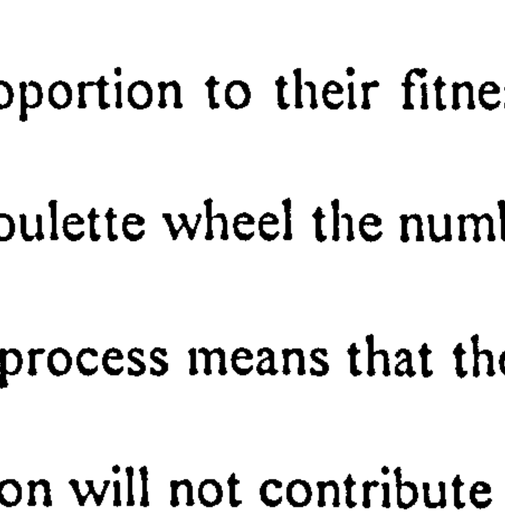


Figure 7-8 Roulette Wheel

7.3.3.1.3 Fitness Scaling

Fitness scaling is introduced because of the chance of having extraordinary individuals in early generations that can monopolise the selection process. These individuals may produce better results but they might not be able to produce the best. So a way must be found to curb there large selection chances. To do this a function is introduced and can be seen in the Figure 7-9.

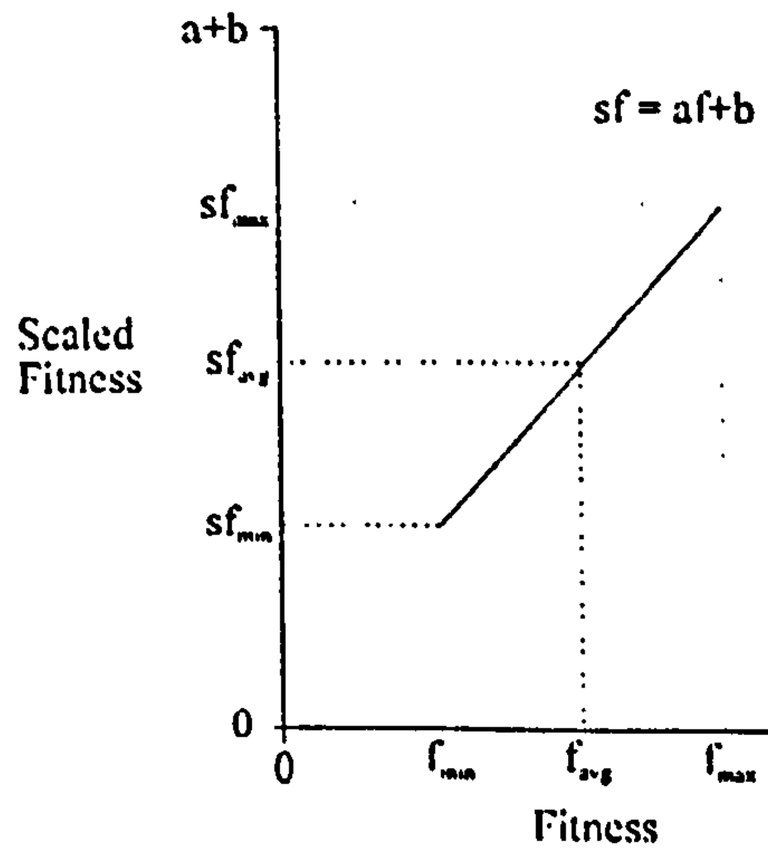


Figure 7-9 Scaling Function

This function curbs the fitness of extraordinary individuals in early generations and exaggerates the difference of close individuals in later generations.

7.3.3.1.4 Reproduction

Once a set of parents has been picked the next step is for them to reproduce. Reproduction is accomplished through the division of the two digraph genes and their subsequent recombination into two offspring, this is collectively known as *crossover*. However,

crossover may not always occur and in such a situation the offspring are just copies of their parents, the probability of crossover occurring is equal to *pcross.*

In crossover a random number is generated, R_A where,

$$1 \leq R_A \leq L-1,$$

and L is the length of the digraph gene.

The random number (R_A) is then used as a division point for the two digraph genes and each parent contributes one part of their split gene to one of the two offspring.

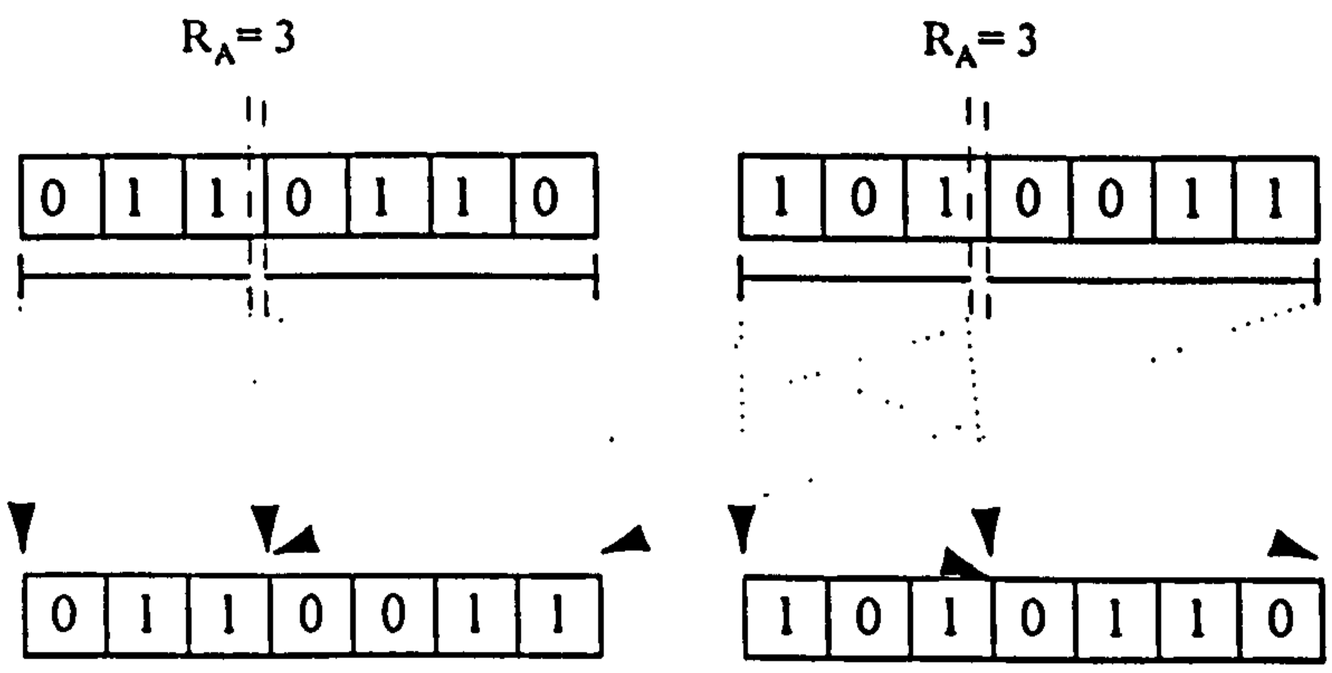


Figure 7-10 Crossover

Once the recombination is over, there is a chance per bit location that a mutation might have occurred. The chance for a mutation is small but necessary since it can contribute genetic sequences which may not have been present in the initial generation and have no way of coming about through crossover. If a bit in the gene sequence has mutated it has simply reversed its value i.e. a 1 changes to a 0, and vice versa.

7.3.3.1.5 Results of a GA

As with the perceptron, genetic algorithms rarely give perfect answers due to their probabilistic nature. To obtain a result the genetic algorithm is run for a number of generations and the best performer in the final generation is the winner, in the case of typing analysis it is hoped that the best individual of the final generation will identify those digraphs that are most characteristic of a test subject, potentially improving the neural networks performance.

7.3.3.2 Genetic Algorithm Software

The genetic algorithm software was written in C++, using Turbo C 3.0. A number of classes were developed so that they could be used in GA problems generally. The header files for the generated code can be found in Appendix C.

When the GA is training it is possible to watch the procedure using a the graphic display drivers developed for graphing in DOS. A screen shot of a GA training can be seen below.

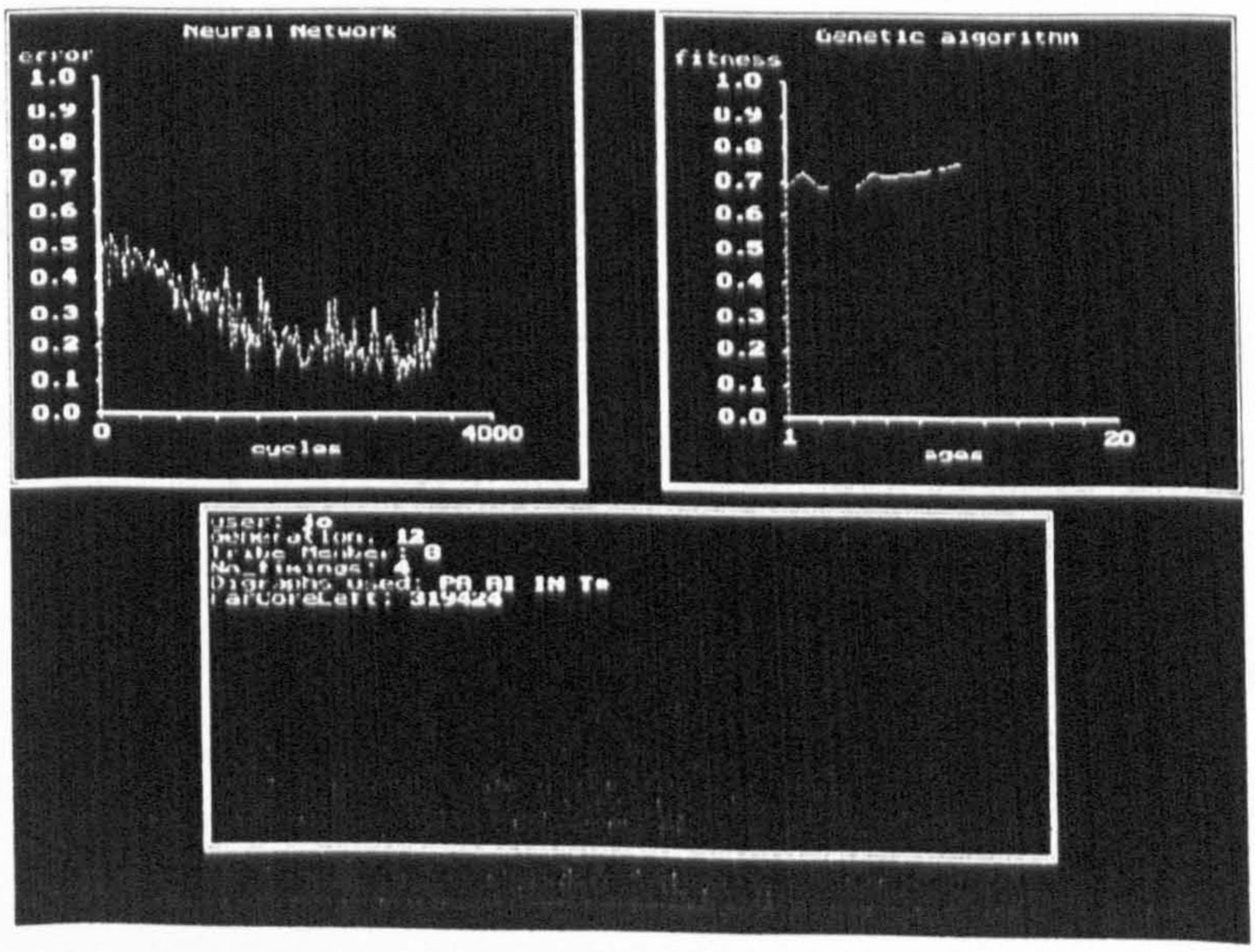


Figure 7-11 Screenshot of GA Training

7.3.3.3 Results of Optimisation through the use of GAs

The experiments into the optimisation of digraph selection was conducted on four subjects chosen at random, it was necessary to limit the number of subjects because of the length of time it took to calculate the fitness for each member of a generation. Although only four subjects were optimised the entire training set was used to train the neural nets and to evaluate the GA selection of digraphs.

The basic results; using every digraph and each reference with the basic neural net topology, a learning ratio of 1:1 and a training cycle of 4000 presentations are:

	0.7	0.8	0.9
REF 1	14	8	4
REF 2	7	6	5
REF 3	3	3	2
REF 4	3	3	2
Average	7	5	3

FAR given as %

	0.7	0.8	0.9
REF 1	31	42	59
REF 2	17	19	25
REF 3	40	49	55
REF 4	23	27	38
Average	28	34	44

FRR given as %

Table 7-5 GA Basic Results

One of the subjects chosen was the most inconsistent typists from the test subjects, it was their inconsistency that provided the high FRR for REF 3.

7.3.3.3.1 Experiment 1

In the first experiment a population of 12 was used and the number of generations that the genetic algorithm went on for was 20. The results of the optimisation can be seen below.

	0.7	0.8	0.9
REF 1	14	8	6
REF 2	7	7	5
REF 3	8	6	4
REF 4	2	2	1
Average	8	6	4

FAR

	0.7	0.8	0.9
REF 1	32	44	54
REF 2	12	15	30
REF 3	29	32	43
REF 4	16	20	29
Average	22	28	39

FRR

Table 7-6 GA Exp 1 Results

	Digraphs used	Max Digraphs
REF 1	4	5
REF 2	7	12
REF 3	10	16
REF 4	16	30

Table 7-7 GA Exp1 Digraphs

The results show an overall improvement in the performance of the neural net when using GA selected digraphs. However, in the case of REF1 there were not enough digraphs present to allow a proper selection by the genetic algorithm, leading to results almost equivalent to that attained through using all digraphs. The best performance by the genetic algorithm was in the selection of digraphs from REF 4, the FRR was reduced by 8% on average and the FAR by 1%. This improvement can be explained through the large choice of digraphs, allowing the best digraphs to be picked that minimised both the FRR and the FAR.

7.3.3.3.2 Experiment 2

In experiment two a population of 12 was used and the number of generations that the genetic algorithm went on for was 40. However, this time the genetic algorithms fitness was adjusted in respect to how few digraphs were used for identification.

	0.7	0.8	0.9
REF 1	20	9	5
REF 2	11	9	7
REF 3	8	7	4
REF 4	3	3	2
Average	11	7	5

FAR

	0.7	0.8	0.9
REF 1	31	53	63
REF 2	19	26	39
REF 3	29	32	41
REF 4	23	26	37
Average	26	34	45

FRR

Table 7-8 GA Exp2 Results

	Digraphs used	Max Digraphs
REF 1	2	5
REF 2	6	12
REF 3	7	16
REF 4	12	30

Table 7-10 GA Exp2 Digraphs

7.3.3.4 Optimisation Conclusion

As can be seen by the results the application of genetic algorithm can reduce the number of digraphs required to identify a user. The results showed a general improvement over the straight classification with no data optimisation.

7.4 Application of Keystroke Authentication within CISS

If used as an additional security mechanism at login the OPENA would be responsible for the collection of keystroke timings, as it is the only agent that directly interfaces to the OS.

The weights, thresholds and digraphs to be used by the authentication mechanism in construction of the multi-layer perceptron would be stored within the extended security segment of the SMIB indexed by the user's login name.

When an attempt is made to gain access to a user's account the UA or OPENA requests a secure access from the SSA. The SSA then requests that the UA prompt the user for their login name and password. The information gathered from the OPENA about the keystroke timings is passed to the SSA which passes them to the SMA. The SMA then interrogates the SMIB to check that the login name and password are valid and then for the information necessary to construct the multi-layer perceptron. The SMA then inputs the keystroke timings to the multi-layer perceptron and if the output is above the necessary threshold the user gains access.

7.5 Conclusion

The results from this investigation are encouraging in that they show considerable accuracy without too much refinement for a worst case scenario. The algorithm worked very well when multiple reference texts were used. Therefore, this authentication mechanism can be used within the login scenario with little more refinement.

However, the second static use proposed for it in section 7.2, at the entry of a volatile command requires more thought. If the suggested authentication mechanism was used the user would have to give multiple samples for each volatile command that would use this authentication mechanism. No user would want to provide the number of samples required, so alternative strategies must be thought of. Two examples are given below.

1. The user when using a volatile command is challenged for his password. This may sound as if it would become annoying. However, windows and UNIX already challenge as to the surety of their command, it would take little adaptation to adjust these OSs to request a password instead.
2. The user is allowed to initiate a volatile command as many times as it takes for the authentication mechanism to obtain enough samples for the teaching of its neural net. Thereafter the authentication mechanism will examine the keystroke timings of the volatile command entry.

Chapter 8

8. Monitoring Agent Specification

This chapter develops a conceptual model for a monitoring system that can be used to provide extended functionality within the monitoring agent of CISS. After a brief discussion of the threats against which modern security systems have consistently failed to protect. A basic model is defined with a definition of what a monitoring systems objectives should be. The basic model is then expanded and its application within the monitoring agent of CISS is described.

8.1 Introduction

Almost all security features placed on computer systems are preventative. Whether they are smart cards, passwords or biometric identification systems, they are barriers placed in the way of an attacker attempting to gain unauthorised access to a computer system. Chapter 7 described a new additive module which may be used to bolster security wherever it is possible to obtain keystroke timings. As useful as this technique is for access control once an attacker has by-passed it, it becomes redundant.

Conventional computer security systems have repeatedly failed to deal with three major security problems [103], [104], [105]:

1. illegal use of a system by an unauthorised user once access controls have been compromised;
2. abuse of privileges by an authorised user;
3. anomalous behaviour of system resources.

Most computer systems have no way of discovering that an attacker has obtained user rights on the system; the term for an attacker that assumes a user's rights through unauthorised use of their account is *masquerader*. The only tools that a security administrator has to aid him in the discovery of a masquerader are those provided through auditing facilities. Historically, the primary need for auditing in computer systems was for the calculation of charges that were incurred by user activity upon the system, therefore auditing facilities are not generally designed with the features necessary for good security. While many existing operating systems such as UNIX and VMS [106] offer limited auditing facilities, the amount of auditing information that can be generated is vast. For

instance, auditing facilities for VMS image termination data generates 20 Mbytes (and upwards) of information per a week for 100 users, a more detailed audit could result in a log thousands of megabytes long. Clearly the information cannot be processed by a single security administrator, and even if it was, discovery of a masquerader through the use of audit logs would have been made too late to prevent a masquerader from causing any damage.

Users of a system that abuse their privileges are another difficult problem for security administrators to deal with. Abuse can be present in many forms, from the simple act of using processing time for 'net surfing,' to actively attempting to by-pass security mechanisms for malicious purposes such as the tampering of files belonging to another user. Anderson [107], gives a system abuser the title of *misfeasor* and users attempting to by-pass security, *clandestine* users. The only way that this kind of activity can be detected is through the search of audit logs for indicative events that suggest abuse by a user, a search that could be through Mbytes of data. User abuse is potentially the most dangerous security breach as a regular user will have a good idea of where to direct attacks for the most damage, or where the most valuable information is kept upon a system.

The final problem that is difficult for conventional security systems to deal with is anomalous behaviour of system resources, particularly software. Anomalous behaviour in software is generally caused by one of two reasons:

1. bugs in badly written programs which can cause strange events, for instance the first release of Microsoft's Word 6 came complete with approximately three thousand bugs;

2. a malicious program [108] which has been introduced into the system; potentially causing data damage, denial of service, loss of confidentiality etc.

Malicious software can come in many forms.

- Viruses:

  These are self replicating programs containing a payload that can be delivered to cause malicious damage or inconvenience the user. An offshoot of the virus is the *worm*, which is a virus that causes damage through its own replication and subsequent tying up of system resources.

- Trojan horses:

  This is a program that disguises itself within an innocent program to gain entry to a system. When the program is run, so is the malicious section of code.

- Bombs:

  There are generally two types of bomb, *logic* and *time*. A logic bomb monitors a system until a certain logical condition is met, for example the removal of a person's name from the pay roll list. Once the condition is met the bomb goes off, damaging some part of the system. The time bomb waits for a certain time, maybe a 'Friday 13th' before it goes off.

Computer viruses are the most well known of all malicious software, though the distinction is not always as clear as the above categorisations might imply. Computer viruses have become much more complex as their writers have employed more sophisticated techniques in the creation of malicious software. One of the main tools a security administrator has in

the combat of computer viruses are virus scanners. Virus scanners operate by maintaining a list of known machine code patterns that appear within viruses and using them to search for matching patterns in files. If any of the patterns are found a suspected virus is reported.

However, a recent release by the 'Dark Avenger,' [109] is a virus pair that has 2 billion valid combinations for its source code. As with real viruses, its ability to rapidly produce dissimilar offspring make detection and elimination of it very difficult. Virus scanners, that operate through the matching of known byte combinations within viruses are unable to deal with the great magnitude of combinations. The most dangerous viruses are what are termed as 'stealth ' viruses. This group of viruses receive their name because of their method of infection; the virus hides itself within the code of an innocent looking program (host program) and when the host program is run copies of the virus are made. Viral actions are carried out, such as denial of service or data modification, and the system becomes infected. For example in DOS a host program may be any *.exe* file. With the increasing availability of programs via the Internet, stealth viruses and Trojan horses are becoming a source of major concern for security administrators.

One method of dealing with the security problems described above is through the continual surveillance of system activity. If the following two premises are true then the development of a system to deal with the security problems discussed is possible.

1. It is possible to learn the normal activity of a system, its resources, and the users upon it, so that its behaviour can be predicted.

2. A masquerader, privilege abuser, or virus, exhibit anomalous activity that can be detected as not being part of the normal system behaviour and the correct counter measures can be implemented to prevent/limit damage.

The reasoning behind the two premises is as follows.

- The behaviour of a system cannot be unpredictable for any length of time as that would provide no platform for any productive work to be done on it. User behaviour can be categorised due to two prevalent factors: humans form habits that are peculiar to them, for instance there are multiple ways to open a file and once a user uses one way they rarely change even though it may not be the most efficient way; job specialisation for example a secretary on the system will use different system resources than a programmer.

- Anomalous behaviour is any that falls outside of the norm for a specific entity. In the case of a masquerader the control behaviour pattern would be that of the user whose account is being used. It can be assumed that a masquerader will not perform the usual tasks that the user would be expected to since the masquerader is not there to perform as the legal user but instead there for exploration, or malicious damage. Programs should behave in a predictable manner. If they do not, for instance when infected by a stealth virus, they may exhibit file modifications or writes to memory that are uncharacteristic. A potential privilege abuser may show indicative signs by excessive browsing, the attainment of super user status, or the ability to assume another user's id.

The security problems discussed above can be partially solved through the strict application of a security policy, which may include regulations on:

1. use of the system and the user's privileges, 'Just because a file can be read does not mean that a user should read it;'
2. the introduction of non-authorised programs upon systems;
3. the use of isolated computers to test software.

However, policies and rules are notoriously ignored by many users and a firmer approach must be taken. This chapter describes a conceptual monitoring scheme within CISS and the 3-L architecture aimed at combating the previously described security problems. It is a specification for the CISS monitoring agent that has been overlooked in previous literature, see chapter 3. It offers great potential in local security and domain security.

A monitoring scheme has three stages.

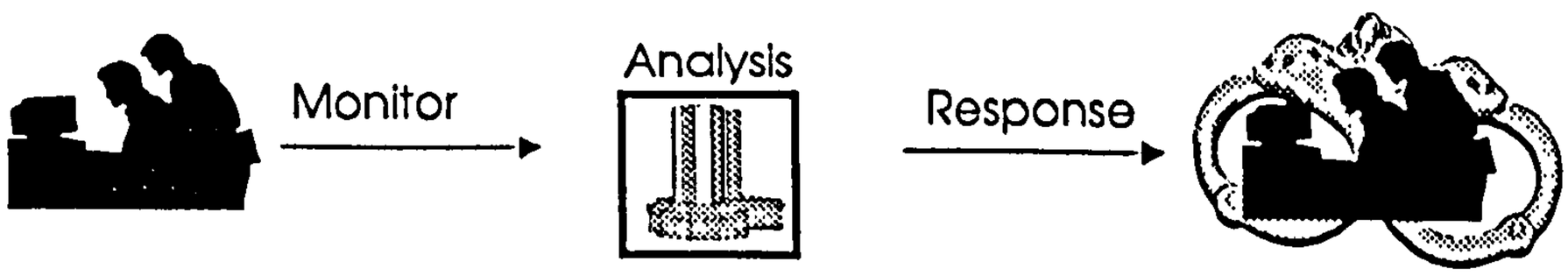


Figure 8-1 Stages of Intrusion Detection

- Monitor:

  system variables that are capable of being used to indicate anomalous behaviour must be monitored for real time analysis, and stored for historical analysis.

- Analysis:

  using the measurements of the system variables monitored the monitoring scheme must be capable of recognising anomalous behaviour upon the computer system.

- Response:

  depending on the conclusions reached by the analysis of indicators a response must be formed and implemented, limiting the damage to data upon the computer system and the protection of services to users.

8.2 The Basic Model

The suggested monitoring scheme can be divided into four sections as can be seen in Figure 8-2, below.

The general aims of a monitoring system should be:

1. the detection of intrusion;
2. the detection of abuse;
3. the detection of malicious software;
4. the implementation of counter measures.

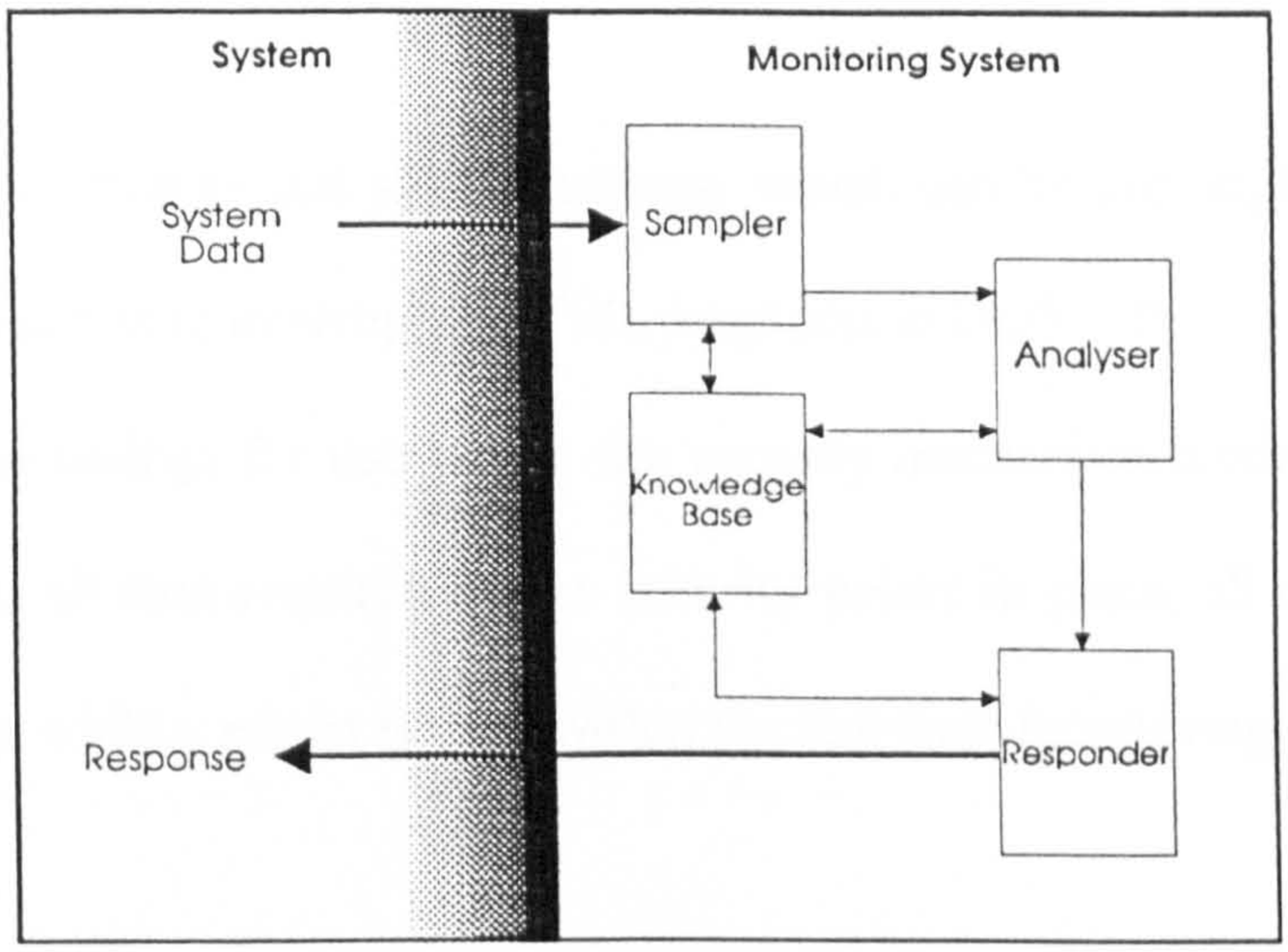


Figure 8-2 Monitoring Agent: Basic Model

The monitoring system should be capable of dealing with both real-time events and a detailed analysis of historic events. Real-time analysis will allow the system to immediately limit damage caused by masqueraders and abusers. However, because the system must run in real time, it is necessary to limit the amount of data processed. Therefore, it will be necessary for a subset of all data being collected to be used in the real-time section of the monitoring system, implying that some sort of filtering must take place to decide what data is to be used.

The detailed analysis of historic events can use complete audit logs and process the data looking for anomalous behaviour during off-peak hours, thereby reducing the impact on system performance.

8.2.1 Sampler

The sampler section must record system activity, which can be accomplished for example through the use of software interrupts or TSR programs in DOS. The sampler records such things as keystroke timings for use within the security mechanism described in chapter 7. The sampler audits all data required by the security policy in place, all of the data will be stored in audit logs while a subset is used within the real-time monitoring.

8.2.2 Knowledge Base

The knowledge base keeps the sum of knowledge that the monitoring system requires to operate. All auditing information for any session must be maintained by the knowledge base along with the run-time variables and the static conditions set by the implemented security policy. The basic information that the knowledge base holds is indicated below..

1. Security policy:

   this section would keep the security policy of the local domain so as to organise the monitoring within an overall security strategy.

2. Security log:

   a record of recent events that have taken place upon the system.

3. Profiles:

   for every user/resource a profile of system activity would be provided, see section 8.3.3.1.

4. Sampling:

   information for generation of security logs.

5. Analysis

   information that is required by the analysis techniques.

6. Response

   information that is required by the responder.

8.2.3 Analyser

The analyser takes information from the monitor and compares it with data stored within the knowledge base and from conclusions it makes determines whether a security violation has taken place. The three methods used in this task are given below.

1. Expert system:

   through the use of rules and the triggering of those rules an expert system is capable of analysing data with the same problem solving capability of a true expert.

2. Statistical modelling:

   through the use of mean and standard deviation models or of comparisons of recent events to known statistical models deviation from a previous pattern can be detected.

3. Pattern classifier:

   uses AI techniques for pattern recognition techniques to detect deviations from the norm.

8.2.4 Responder

Through information contained within the knowledge base and the output of the analyser the responder must decide and implement an action. It may be that the responder will

decide that no immediate action is necessary or that further events must be monitored or that a security violation has occurred and that a response is called for. The response that the responder may take will depend upon the security policy. For example, it may request further proof of identity from a suspected masquerader or simply logout the suspected masquerader from the system.

8.3 Conceptual Monitoring Agent

Taking the basic model described in 8.2 it will now be expanded to detail the various components, describing how they may be included within the CISS monitoring agent. Because it is necessary for the monitoring agent to keep records for its own functions an additional segment must be added to the SMIB - the monitor segment. This segment is divided into:

1. sampling;
2. analysis;
3. response.

As described in the basic model and to be further defined in section 8.3, the profiles of users/system resources will be stored within the existing extended security segment. The security policy is enforced through the set-up of CISS.

8.3.1 Sampler

As was mentioned in chapter 3, there are three points at which CISS and the outside world interface. Therefore, it is necessary that the task of sampling measurements is divided

amongst the three agents the: operational environment agent; user agent and security administration agent. In implementations where the SAA and UA functionality is divided, the MA must be capable of interacting directly with the UA. This requires a modification to the interaction of agents, and the subsequent addition of an MA-UA communication protocol. The measurements obtained through sampling will be used in the generation of log entries and the production of profiles, see section 8.3.3.1.

8.3.2 Security Logs

Historically the best method a security administrator had in the detection of malicious users/processes was through the analysis of security logs. Detailed audit logs are essential for automated monitoring, preferably they should be designed according to the domain's security needs, rather than its accounting requirements. The introduction of CISS to an existing system should allow the generation of specialised security logs through information gathered by the API of the OPENA.

Therefore, the monitoring agent's functionality must be expanded from its original specification in chapter 3, to provide full audit facilities for the local environment. The generation of a single log that represents the whole of the local environment's activity is more efficient than the generation of numerous logs for specific applications, a strategy that was implied by CISS's original specification. The generation of a single security log allows CISS to produce generic security logs for different computing platforms, making it capable to perform analysis within a heterogeneous network.

Through the use of security log entries a history of user activity can be built up. From this information a *profile* (see section 8.3.3.1) can be generated that describes the user's normal activity upon the system. This profile can then be used in the detection of anomalous behaviour during any future sessions using the user's account.

8.3.2.1 Logs Entries

The first step in the design of a monitoring scheme must be in the decision of what resources to monitor upon the system, and therefore how detailed the audit log should be. There have been studies [106] carried out on the effects of auditing and the levels to which it should be applied. The argument has always been that the lower the level of auditing the harder it is to circumvent. Therefore the difficulty for privilege abusers, specifically clandestine users, in performing malicious acts is increased. However, with low level auditing one is left with the fact that a lot of information will be generated, requiring large amounts of resources to be dedicated to sifting through the logs for security breaches. Higher level auditing, such as at the application level, produces less auditing information but can potentially be by-passed through the use of lower level commands. Therefore, a medium between the two is necessary.

The sampling section of the monitoring segment in the SMIB holds information on:

1. the level of auditing required by the security policy and security administrator;
2. specific sampling requests which are requested by the analysis techniques, see section 8.3.3.

To prevent unnecessary processing and storage of security log entries the monitoring agent will refine the information sent to it by the three agents involved in sampling and only archive entries that are specified by the entries within the sampling section of the SMIB.

The log entries are stored within the security log segment of the CISS's SMIB. Potentially the entries can be encrypted and only readable by the security administrator who would have the necessary key to decrypt them. However, the entries must be available to the monitoring agent for further use in the generation of profiles and automated audit analysis.

The security log should be subdivided into two sections:

- a low level audit log that stores the usage of system resources;
- a log of system activity that is used to store the higher level measurements potentially useful for security mechanisms such as non-repudiation. Activities are defined in section 8.3.2.1.

The logical section of the monitoring agent responsible for the formatting and storage of security log entries is the archiver.

The basic audit entry should be made up of five basic fields.

- Initiator:

  usually a user, though in certain circumstances it will be a process calling another process.

- System resource:

  can be a file, command, or hardware module.

- Action:

  is usually one of those described in table 8.2-1 below.

- Measurement:

  can be one of the four described in table 8.2-2, dependent upon the action and system resource.

- Time-stamp:

  is the time at which the action upon the system resource occurred.

Action	Description
Execute	Initiate a process
Terminate	Halt a process
Read	Read data from a system resource
Write	Write data to a system resource
Modify	Modify a system resource
Open	Open a system resource
Close	Close a system resource
Create	Create a system resource
Delete	Delete a system resource
Usage	Use of system resource

Table 8-1 Actions

Measurement	Description
Time	This is the time that an event took place e.g. login time or logout time
Duration	This is the length of time an event took place for, e.g. duration of a session
Cardinal	This is the number of times an event occurred or describes a quantity.
Binary	Whether an event occurred or not.

Table 8-2 Monitoring Measurement

For example a user Fred has an id 0001 uses the program copy.exe to copy the contents of a file 'first' to a second file 'second.'

[0001, copy.exe, execute, 1, 01:44:13:28:08:95]
[0001-copy.exe, first, open, 1, 01:44:13:28:08:95]
[0001-copy.exe, second, open, 1, 01:44:13:28:08:95]
[0001-copy.exe, first, read, 1040, 01:44:13:28:08:95]
[0001-copy.exe, second, write, 1040, 01:44:13:28:08:95]
[0001-copy.exe, first, close, 1, 01:44:13:28:08:95]
[0001-copy.exe, second, close, 1, 01:44:13:28:08:95]

System Resources	Description
CPU	CPU usage
I/O	Input/output devices used
Directories	Directories used within a session
Files	Files manipulated within a session
System Calls	System calls within a session

Table 8-3 Sample System Resources

The audit log is a low level record of potentially all system calls, the second type of record kept should be at a more abstract level, in the basic model this is called the activity log entry, e.g. login, logout.

- Initiator:
- as before.
- Activity:

  can be anything taken at a more abstract level than the audit logs, for example login, or logout.

- Measurement:

  is one of the four basic measures described within table 8.3-2.

- Audit-record-list:

  this is the list of low level audit entries that make up the activity.

- Time-stamp:

  as before.

Application	Activities
Editor	Editors used, files edited etc.
Window Command	Use of windows during a session
Web servers	Use of web servers within a session
Spreadsheets	Types of spreadsheets used
Mailer	Electronic mail, read, written etc.

Table 8-4 Sample Activities

8.3.2.2 The Sampling Agents

The UA interacts directly with the user and provides a way for the user to request security services directly from CISS. The type of requests that are made will be logged in the activity segment of the SMIB security log, the corresponding system usage measurements will be logged in the low level section of the security log. The SAA will be monitored mainly for a record of security administrator activity and not for the detection of malicious activity. The OPENA provides all of the low level sampling through its API which interfaces with the local operating system to obtain the necessary auditing information, the OPENA must also interface with other applications to obtain information on their usage.

Within the 3-L architecture it is expected that the majority of sampling will be accomplished within the LSU, where information will be gathered and transferred to the LSSs that are larger machines with the capability of processing and analysing the data.

8.3.3 Analysis

There are three methods of analysis that should be used within the monitoring scheme: statistical; rule-based, and AI. There are three types of analysis that should be provided through application of the methods:

1. real-time analysis - the detection of anomalous activity during a user's session;
2. historical analysis - the minutiae analysis of system activity performed off-line and periodically;
3. damage analysis - the assessment of damage caused by anomalous activity.

Analysis is conducted upon the information gathered through the monitoring scheme, as described within section 8.3. A recent approach to the monitoring of system activity is through the use of profiles generated for system resources.

8.3.3.1 Profiles

A profile is a set of measures kept upon an individual resource/entity that can be used in the identification of the individual. Prior research [110], [111], indicates that this approach, though not perfect, is a strong security mechanism for detecting intrusion upon a system.

Profiling can be divided into three sections.

1. Data filtering:

   through the correct data selection the most indicative measures are used while the rest are discarded to reduce processing. This is also a function of the analysis section of the monitoring agent.

2. Profile creation:

   from the filtered measures a control profile is created that indicates the boundaries of normal behaviour.

3. Profile Refinement:

   the profile is continuously refined so that it is up to date.

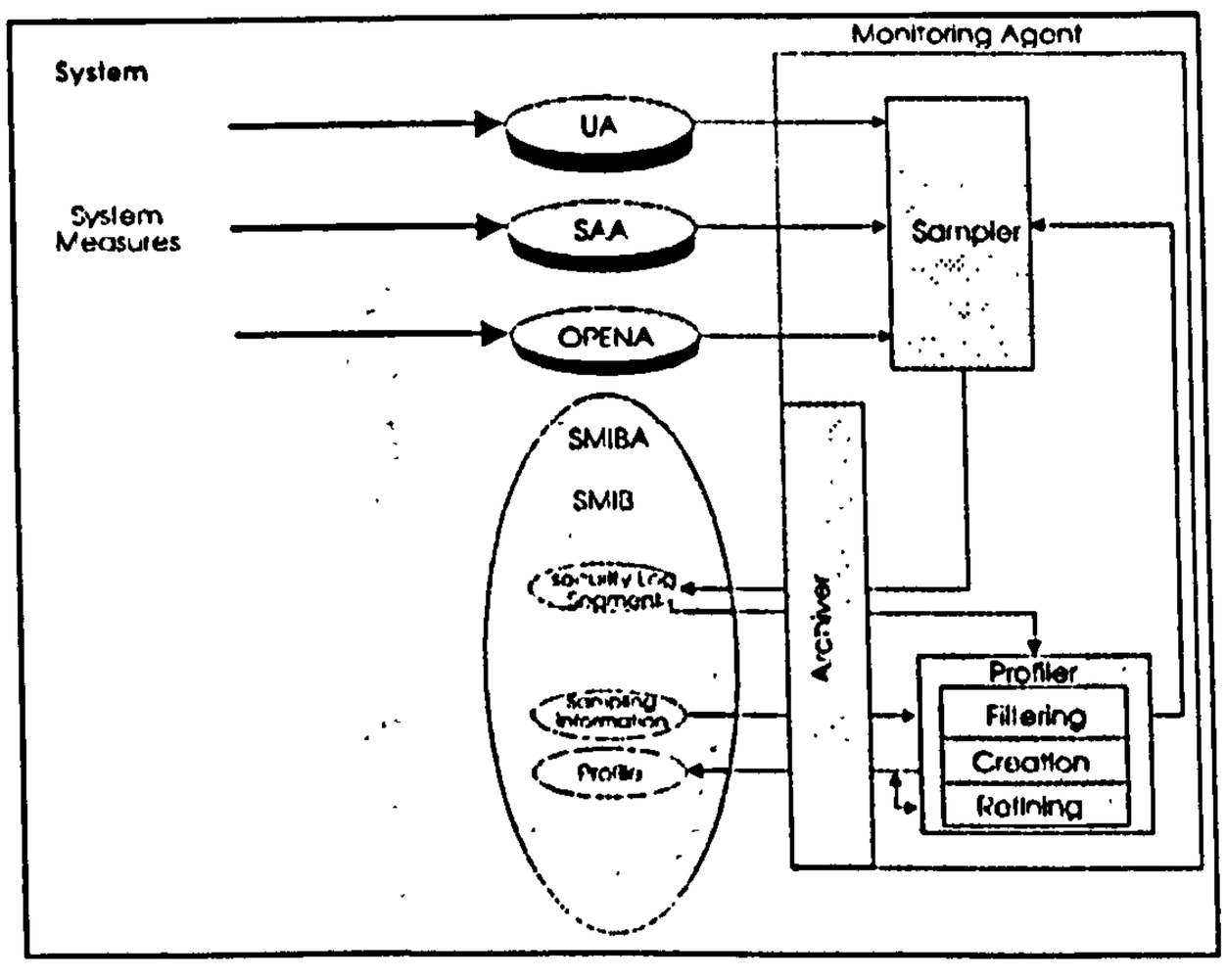


Figure 8-3 Profiler

Within the monitoring scheme there are two profiles used to describe an entity's normal behaviour, the activity profile and the behavioural profile.

The basic activity profile is shown below.

- Entity:

  is the user/process that the profile belongs to.

- Activity:

  is the activity that this profile describes.

- Analysis technique:

  is the technique chosen for the detection of anomalous behaviour in the activity defined earlier.

- List-of-audit records:

  is the list of low level audit entries that define the range of valid audit entries that make up the activity.

- Threshold:

  is the level at which the analysis technique output indicates an anomalous activity.

- Contributive value:

  is combined with other contributive values when the activity profile is used in conjunction with other profiles for behavioural profiles.

- Last-update:

  is the date at which the profile was last updated either through automatic update or manual update.

The basic behavioural profile is.

- Entity:

  as before.

- Activity list:

  is the list of activities that contribute to the behavioural profile.

- Analysis technique:

  is the technique used to detect anomalous behaviour.

- Threshold:

  is the value at which the analysis technique indicates anomalous behaviour is being exhibited, the threshold may not be a single value but in the case of a statistical technique it may be the standard deviation and mean of the activities.

- Last-update:

  as before.

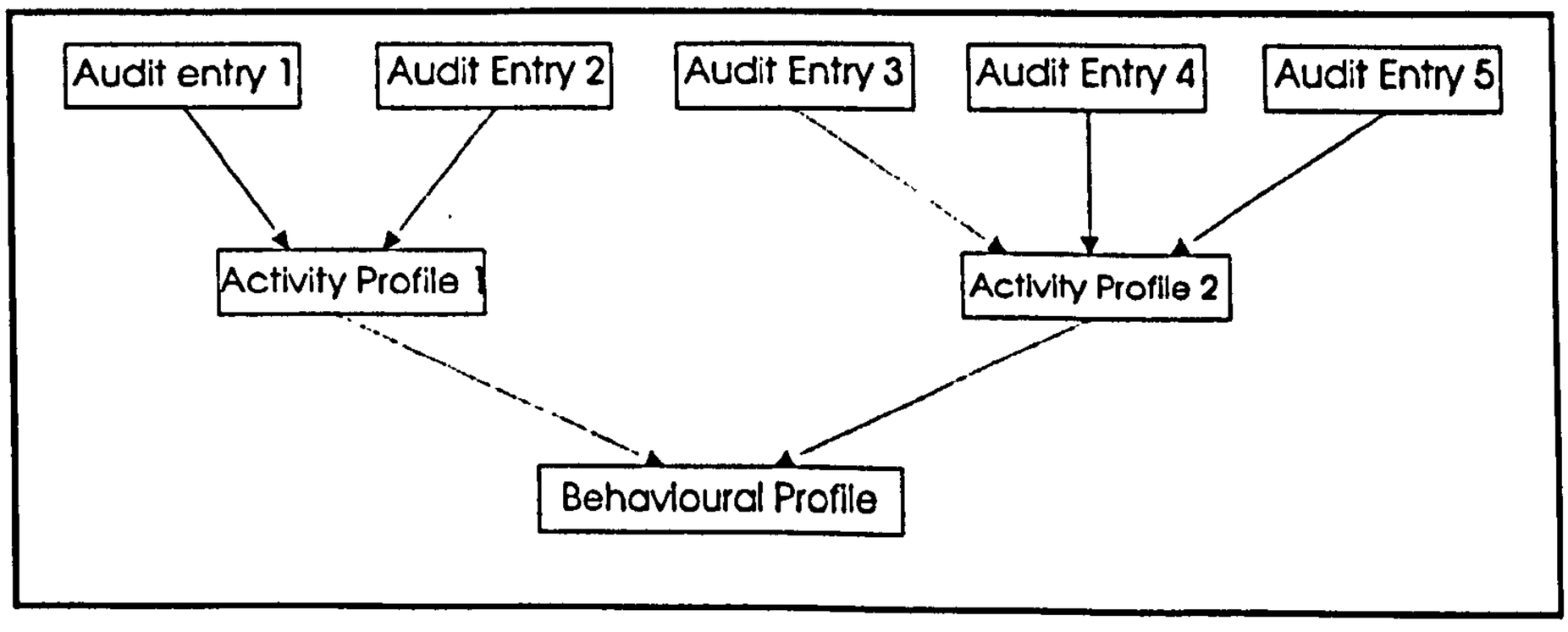


Figure 8-4 Profile Monitoring

Because profiles are meant to be representative of the users normal activity it is important to keep them up to date. This can be done through the acquisition of sample activity during

normal sessions. Automatic updating in this form can be accomplished daily or weekly, depending on the activity of the user. Updating a profile would only take place at the termination of a successful session. This reduces the chance of contamination of a profile by possible masquerader activity. The logical section within the monitoring agent that is responsible for profile refinement is the profile refiner.

The surreptitious updating of profiles can lead to the problem of slow change. If privilege abusers know that profiling is being used with automatic updates they may attempt to slowly modify their behaviour over time, consequently altering their profiles. Thus they may not exhibit anomalous behaviour even when abusing the system. For example, a privilege abuser can gradually introduce file browsing into his normal list of activities and then slowly increase his browsing activities.

The second way to update a profile is for the security administrator to force an update by requesting it through the SAA. This would be required at initial set-up of the system, at the addition of a new user, or at the addition of new software.

Here is an example in the use of behavioural profiles.

'Session Start-up.' This can be described as the first five minutes of a user's session. User Fred starts up in the usual way, he begins any login session with three activities. First, the compulsory login, secondly he checks his mail, and thirdly browses the news. It is not unusual for users to form habits such as this. The behavioural profile kept within the

knowledge base of a monitoring scheme would have been built up over a number of Fred's past sessions.

If Fred did not perform within the boundaries of his profile it would not necessarily follow that he is a masquerader. However, the monitoring scheme would be at a heightened level of suspicion. If enough anomalous activity was picked up over the course of one session by the analyser the monitoring system may decide that a challenge is necessary or that some other counter measure is called for.

The discussion of profiles has so far been limited to those of the human users. Using the profile scheme described above it is possible to create profiles for software, and pieces of hardware, such as computer terminals.

Software profiles will be limited to activities. By attributing a piece of software with an activity profile it should be possible to detect any anomalous writes or reads, that could be indications of a Trojan horse or virus. Such activity when detected by the monitoring scheme's analyser may demand the initiation of a virus scanner specifically targeting the suspect piece of software. If no virus is found the responder may tag the piece of software for the security administrators attention. The security administrator can then use the activity audit entries to examine exactly what the piece of software is doing.

Anomalous activity detection of software using profiling will only work if the knowledge base holds profiles prior to infection. Any infected software introduced into the system for the first time will not be detected as exhibiting anomalous behaviour when the stealth virus

delivers its payload. Since most organisations use a limited number of software packages it is possible that standard applications can be profiled and their profile introduced into the knowledge base prior to installation of the software upon the system these standard profiles can be generated either by the application suppliers or the CISS suppliers.

Computer terminals can be given behaviour profiles so that their actions upon a network can be monitored for anomalous activity. Activities that a terminal may be audited for, and for which a behaviour profile may be generated include; network connections, remote logins, traffic intensity, etc. This sort of profiling can help in the prevention of machines being used for activities such as hacking.

Profiles take time to be generated so the most vulnerable time for a system is when a new entity is placed upon it (user/software). Therefore, until enough information has been gathered for an initial profile standard security mechanisms must suffice.

8.3.3.2 Rule-Based Analysis

Rule based analysis can be used in one of three ways:

1. to automate the minutiae analysis of security logs;
2. to perform real-time analysis of system activity;
3. to form judgements about the overall state of security upon the system.

Expert systems work through the application of knowledge engineering. Experts are interviewed by knowledge engineers that attempt to obtain the knowledge built up by the experts through experience in their respective fields. The knowledge is then represented as

rules which work upon objects. An engine is then developed that can make judgements upon the results of applying the rules to objects. Previous work has been done on the application of expert systems to the automation of security log analysis [112], [113]. These applications have proved effective. The use of expert systems within a real-time scenario has also been looked at in recent research by Baur[114] and Lunt[115].

The production of a generic log as described in section 8.3.2 allows the application of generic rules within the expert system. This would allow the CISS monitoring agent to be updated easily within a heterogeneous computer system, with better rules being sent out from the CISS supplier as they become available.

The expert system requires rules upon several key facets of a system.

1. Normal user activity:

   these rules will contain the necessary information to detect normal activity upon the system.

2. Malicious user behaviour:

   these rules can be used to detect the patterns of behaviour that malicious users display, such as excessive browsing, attempts to gain another users privileges.

3. Malicious software behaviour:

   these rules can help detect infected software which may display behaviour such as modifications to files, attempted writes to boot sectors etc.

4. Current security policy:

   these rules will describe the current security policy within the domain and by their application, policy breakers will be detected.

A more specific set of rules may be included to provide detection of specific attempts to circumvent OS security. Along with the rules described above there should be *immediate* [116] rules, which are hard and fast, making no use of collected or expected user behaviour, such as three failed attempts to login successfully and the user account is suspended.

If the expert system is to be employed to perform comprehensive analysis of security logs then this can be done off-line allowing the rules the system uses to be more detailed. For a real-time application of expert systems to monitoring the rule sets should be limited so that the impact of system performance can be limited.

8.3.4 Statistical Based Analysis

This is where statistics are used to describe normal behaviour of a system and any behaviour that is detected and outside the normal bounds of the collected statistics is labelled anomalous and therefore suspect. The use of statistical methods for intrusion detection has been demonstrated by SRI International [117].

The sample data is maintained using profiles that are kept on individual users, which record frequency, mean and covariance of activity. When a user account is activated, information about the activity taking place is gathered through the sampler and compared against the recorded measurements within the SMIB indexed by the user accounts login name. The profile refiner limits the number of variables used to those that are indicative of normal user

behaviour. The thresholds at which user behaviour is no longer normal is determined by the profile refiner segment of the statistical analyser.

Statistical analysis can use models of distribution to detect anomalous behaviour. However, as these models may not be exact, errors in detection of anomalous activity [118] occur.

8.3.5 Neural Net Based Analysis

As was described in chapter 7, neural networks are capable of classifying patterns that are not apparent to humans. This ability gives them a great potential within the monitoring agent. However, because of their nature they require large amounts of learning data which may not always be readily available. Therefore, they should be used to analyse specific areas of activity, potentially the most characteristic areas of a user. By doing this the volume of necessary learning data is reduced making the application of a neural net more viable.

The advantages of using a neural net for analysis of system activity are that their application requires no assumptions about the nature of the data to be analysed. For instance, the statistical analysis relies on the assumption that a statistical model can describe activity. The expert system relies upon the assumptions of humans, the neural network forms its own assumptions through learning directly from available data.

8.3.6 Analysis Within CISS

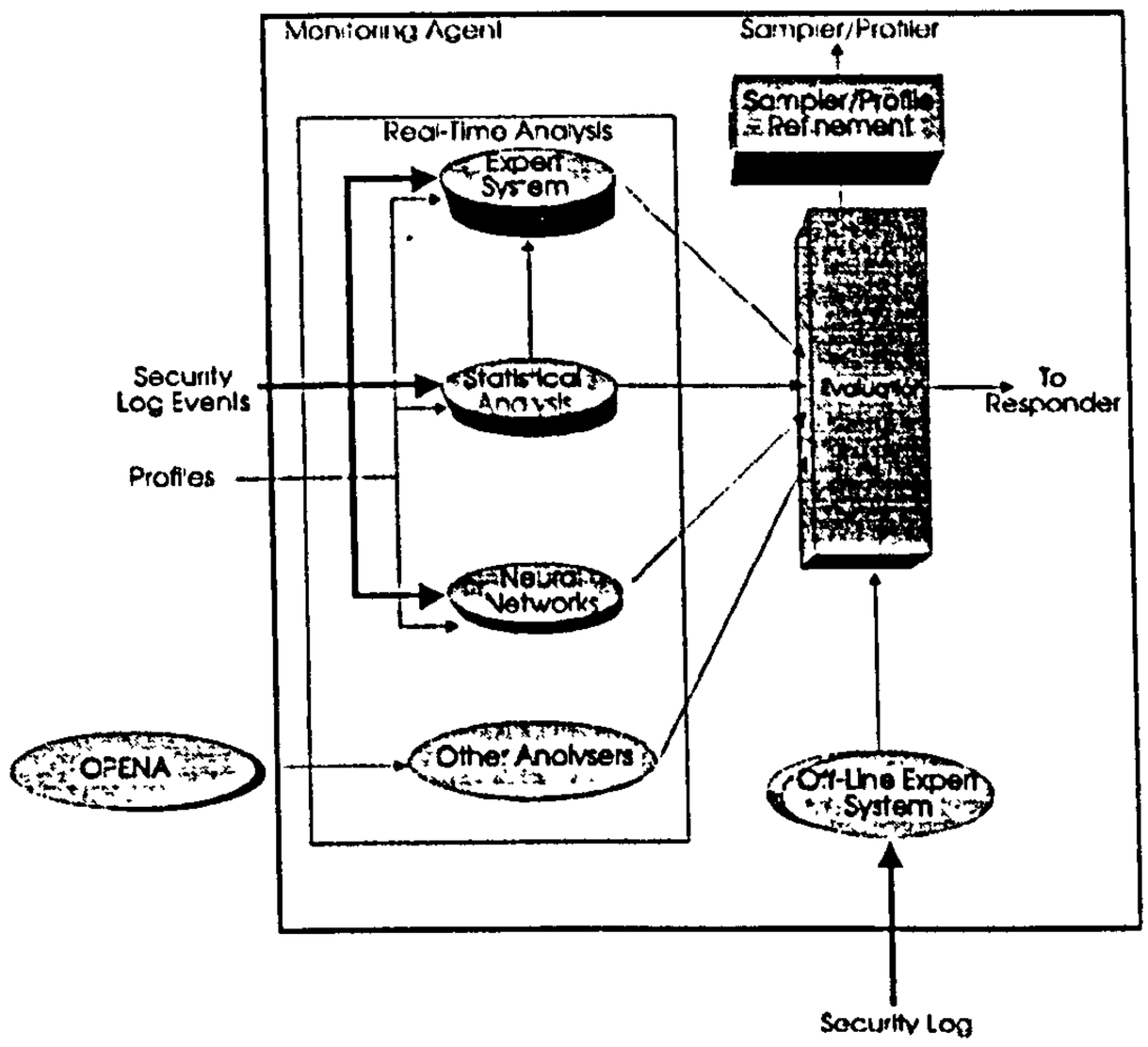


Figure 8-5 Analysis Within CISS

Within the monitoring agent there should be real-time analysis and off-line analysis. The real time analysis will be made up of the three specified analysers.

1. The task of the expert system will be to detect anomalous behaviour by applying the experience that security experts have imbued into it through the rules that they have generated for its inference engine. The expert system will use a subset of the rules as dictated by the same experts while monitoring in real-time.
2. The task of the statistical analyser is to detect anomalous behaviour of entities through activities outside of their norm as dictated by their profiles. This is more specific than that of the expert system, although the real-time expert system will use the results of statistical analyser in its own analysis.
3. The neural network should be used in very specific cases where the samples that are needed for its training can be collected and the problem is very specific.

As well as the anomaly detection through activity profiles there should be other real time supervisory mechanisms requiring analysis such as keystroke timings, network traffic analysis etc. There is also an off-line security log analyser, that is an expert system with a full rule base, that can conduct thorough analysis of the security log. Anomalies detected through these mechanisms are reported to the evaluation block of Figure 8-5.

The evaluation block can be either a simple sum threshold scheme with logic for further refinement of profiling and sampling, or it can be an expert system programmed to perform a decision on the current security of CISS through the data provided by the monitoring agents real-time analysers.

The evaluation segment of the monitoring agent will have an awareness rating so that the greater the awareness the more sensitive the analyser becomes. For instance, numerous suspicious activity may occur in isolation without necessarily tipping the monitoring agent that a masquerader is present. The frequency of suspicious activity contributes to the awareness value so that continued suspicious activity will eventual cause a response from the system.

The exact location of analysis processing will be implementation dependent, with the optimum choice being the use of a dedicated monitoring processer that will analyse all the information within an organisation/divisional unit.

8.3.7 Responder

The responder takes the result from the analysis section of the monitoring agent and forms a response dependent upon the current security policy. The response becomes more dramatic the worse the indications are from the analysis of system activity. For instance, if a user attains super-user status on a system and is not meant to have that ability then the user account may be frozen. A list of possible responses is given below, the further down the list the greater the security breach and the more extreme the result from the analyser.

1. No action necessary.
2. Raise awareness.
3. Take action, such as:
    - challenge user;
    - protect files;
    - deny services;
    - notify security administrator.
4. Shutdown LSU.
5. Shutdown LSS.
6. Shutdown CISS.

The responder is comprised of decision logic that may be in the form of an expert system. The actual response decided upon will be implemented via the SSA or the OPENA of the LSU, LSS, or DMC dependent upon the seriousness of the security breach. Shutdown of a specific level entity can be accomplished through revocation of the entity's public-key certificate and any SSids that are currently held by the entity. Without the means for authentication the entity will be effectively cut off from any services provided or protected

by CISS, this action may only be accomplished by the DMC and security administrator of a domain.

8.4 Conclusion

The monitoring agent described within this chapter uses three types of analysis techniques which when applied in isolation can be fallible. However, it is hoped that the analysis error can be reduced through the combination of the three techniques. A future implementation of the monitoring agent should minimise the error within the analysis section.

This chapter has extended and described the monitoring agent in more detail than has appeared in literature previously. The necessity for a strong monitoring scheme has been demonstrated through documented cases referenced throughout this chapter. To combat these a generic log can be created that is independent upon the computing platform that CISS secures. Making it possible to use generic profiles of CISS activity along side specific profiles for OS activity. The interactions of the extended monitoring agent within conventional CISS and the new architecture described within Chapter 4 has been described, providing a framework for intrusion detection through behavioural monitoring using CISS.

Chapter Nine

9. Conclusion

This chapter presents the author's concluding thoughts about the research that has been conducted during the course of this project. It begins with a discussion on the achievements of the work presented within this thesis, followed by a discussion on the limitations of the work. To conclude, some thoughts are given on where continuation of the research should be concentrated.

9.1 Achievements of the Research Programme

The research has accomplished the objectives as laid out in Chapter 1 in the following manner.

1. The CISS model has been further developed, in particular a management architecture has been developed to increase the security that CISS provides within a security domain. The management architecture allows heterogeneous computing platforms to be connected together while attaining the security requirements defined by a single policy.

2. A prototype board that can be used to primarily speed up the local security unit, but may be used by any of the three levels described within the new management structure, has been developed. It provides RSA and DES cryptographic algorithms in hardware. During the development of the board a generic SBus connector was designed that provides easy attachment to the SBus. Device drivers have been developed for the generic SBus connector.

3. Software has been produced to mirror the functionality of the prototype board. While producing these functions a mathematical module has been constructed for the provision of arithmetic components within CISS, so attaining a good platform for the implementation of alternative cryptographic techniques within the software.

4. A new access control mechanism has been investigated, applying neural networks and genetic algorithms to keystroke analysis. The experiment gave encouraging results, making further investigation warranted. The access control mechanism fits in well with

the philosophy of CISS, as it exploits an already common access control mechanism and may be easily added to most operating systems, while becoming invisible to users.

5. Finally, the monitoring agent of CISS has been defined through the development of a general monitoring scheme that exploits the use of profiles to detect anomalous behaviour by users, software, and hardware. The use of the monitoring agent within the 3-L architecture has been described and when implemented upon a LAN will reduce the need for multiple audit officers.

9.2 Limitations of the Research

One of the objectives of this research project was the optimisation of a software CISS prototype by implementation of functionality within hardware. The development of the CISS prototype was the domain of a second research project which unfortunately ran behind schedule and no prototype was produced during the time of active research within this research programme. Therefore, there could be no full analysis of the CISS system and consequently only limited implementation of functionality within hardware was accomplished. However, the hardware constructed during this research project provides the most intensive processes in dedicated hardware, which will greatly improve performance of the prototype.

The access control mechanism described in chapter seven produced results comparable, if not better, than other research on similar access control mechanisms conducted elsewhere. However, the application of neural networks and AI techniques never produce a hundred percent accuracy and it is inevitable that some errors will be present.

Although two models have been defined the 'proof of the pudding' is in the testing of the models through implementation. Unfortunately, it was not possible to test either of the models as no CISS implementation exists, and the full monitoring scheme could be considered three years work in its own right. Even so it is important that the models have been defined at this stage as they are necessary for further development of CISS beyond a prototype and into a real security service provider.

9.3 Suggestions and Scope for Future Work

The primary requirement for future work is the development of a prototype CISS, which can be analysed thoroughly and provides a foundation for optimisation. It was never within the scope of this project to create a prototype CISS.

The ENCDEC board can be further developed so that it may be ready for retail as a supplemental cryptographic board; not necessarily as part of a CISS product. As computer security becomes more important to businesses there will be a bigger demand for security products.

There could be development of the monitoring agent either as part of CISS or as an isolated monitoring security mechanism for computer systems. The application of genetic programming and genetic algorithms should be looked at in the context of security as both these techniques appear to be powerful tools as yet unused within the security field.

References

1 J.Schofield, "Data Takes to the Fast Lane," Guardian, 27th January, 1994.

2 "Gore Signals the Telecommunications Era as he Paves Way for Information Super Highway," Guardian, 21st December, 1993.

3 International Standards Organisation, "OSI RM Part 1: Basic Reference Model," ISO 7498-1, 1988.

4 "Hacked EC Computer," Guardian, 24th February, 1993.

5 "Freefone Lines Set Off Spate of Data Rape," Sunday Times, 14th November, 1993.

6 Audit Commission, "Survey of Computer Fraud & Abuse," 1990.

7 S.Muftic, A.Patel, P.Sanders, R.Colon, J.Heijnsdijk and U.Pulkkinen, "Security Architecture For Open Distributed Systems," Wiley, 1993

8 International Standards Organisation, "OSI RM Part 2: Security Architecture," ISO DIS 7498-2, 1988.

9 International Standard Organization, "Use of Encipherment Techniques in Communication Architecture," ISO/TC 97/SC 20/WG 3, 1986.

10 W.Vienna and D.Farmer, "What SATAN is," www.cs.ruu.nl/cert-uu/satan.html.

11 D.Kahn, "The Codebreakers, the Story of Secret Writing," Macmillan, 1967.

12 NIST FIPS 81, "DES Modes of Operation," 1980.

13 C.E.Shannon, "Communication Theory of Secrecy Systems," Bell System Technical Journal, Vol 28, pp 656-715, 1949.

14 B.Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C," Wiley, 1994.

15 NIST FIPS Publication 46, "DATA ENCRYPTION STANDARD," 1977.

16 J.L.Smith, "The Design of Lucifer: A Cryptographic Device for Data Communications," IBM Research report RC3326, 1971.

17 M.E.Hellman, Comments at 1993 RSA Data Security conference, 14-15 Jan, 1993

18 ANSI X9.23, "American National Standard for Financial Institution message Encryption,", 1988

19 E.F.Brickell, "Contemporary Cryptography: Cryptanalysis," IEEE Press, pp501-557, 1991.

20 D.E.Denning, "The US key Escrow Encryption Technology," Computer Communications, Vol 17, no 7, pp 453-457, 1994

21 E.F.Brickell, D.E.Denning, S.T.Kent, D.P.Maher and W.Tuchman, "SKIPJACK Review: Interim report," http://www.quadralay.com/www/crypt/clipper/skipjack-review, 1993.

22 B.P.Zajac, "US Encryption Policy - The Clipper Chip Controversy," Computer Law and Security Report, US Focus, pp138-139, May-Jun 1994.

23 Department of State, Office of Munitions Controls, "International Traffic in Arms Regulations," 22 CFR 120-130, 1989.

24 Department of State, Office of Defense Trade Controls, "International Traffic in Arms Regulations," 22 CFR 120-130, 1989.

25 J.L.Massey and X.Lai "Device For Converting a Digital Block and the Use Thereof," International Patent PCT/CH91/00117, 1991.

26 B.Schneier, "The IDEA Encryption Algorithm," Dr Dobb's Journal, pp 50-56, Dec 1993.

27 J.L.Massey and X.Lai, "Device for Converting a Digital Block and the use Thereof," International Patent PCT/CH91/00117, 1991

28 W.Meier, "On the Security of the IDEA Block Cipher," Proc EuroCrypt 93, pp371-385, 1993.

29 P.Zimmerman, "PGP User Guide," 1992.

30 W.Diffie and M.E.Hellman, "New Directions in Cryptography," IEEE Trans Informat. Theory, Vol IT-22, pp 644-654, 1976.

31 R.L.Rivest, A.Shamir and L.Adleman, "A Method For Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of The ACM, vol 21, no 2, pp 120-126, 1978.

32 M.R.Schroeder, "Number Theory In Science And Communication," Springer-Verlag, 1984.

33 D.M.Bressoud, "Factorization and Primality Testing," Springer-Verlag, 1990.

34 International Standards Organisation, "Information Technology Security Techniques Digital Signature Scheme Giving Message Recovery," ISO/IEC 9796, 1991

35 Échanges Télématiques Entre Les Banques et Leurs Clients, Standard ETEBAC 5, Comité Francais d'Orghanisation et de Normalisation Bancaires, 1989.

36 J.Linn, "Privacy Enhancement for Internet Electronic Mail: Part1 Message Encipherment and Authentication Procedures," RFC 989, 1987.

37 T.Elgamal, "A Public Cryptosystem and a Signature Scheme Based on Discrete Logarithms," IEEE Transactions On Information Theory, Vol 31, no 4, 1985

38 W.Diffie, M.E.Hellman and R.E.Merkle, "Cryptographic Apparatus and Method," U.S. Patent 4,200,582, 1980.

39 Federal Register, "Proposed Federal Information Processing Standard for Digital Signature Standard (DSS)," vol 56, no 169, 1991.

40 T.H.Cormen, C.E.Leiserson and R.L.Rivest, "Introduction to Algorithms," McGraw Hill, 1990.

41 C.Pomerance, "The Quadratic sieve Factoring Algorithm," Proc EUROCRYPT 84, pp169-182, 1984.

42 G.Carter, A.Clark, E.Dawson and L.Nielsen, "Analysis of DES Double Key Mode," Proc Information Security - the Next Decade, pp113-127, 95.

43 W.Tuchman, "Hellman Presents No Shortcut solutions to DES," IEEE Spectrum, Vol 16, pp40-41, 1979.

44 R.Rivest, "The MD5 Message Digest Algorithm," RFC 1321, 1991.

45 NIST FIPS YY, "Secure Hash Standard," 1992.

46 ANSI X9.9, "American National standard for Financial Institution Message Authentication," 1986.

47 M.S.Ganley, "Digital Signatures and their Uses," Computers and Security, Vol 13, pp385-391, 94.

48 S.Herda, "Non-Repudiation: Constituting Evidence and Proof in Digital Cooperation," Computer Standards and Interfaces, Vol 17, pp69-79, 95.

49 S.Herda, "Non-repudiation constituting Evidence and Proof in Digital Cooperation," Computer Standards and Interfaces, Vol 17, pp69-79, 1993.

50 S.J.Shepherd, P.W.Sanders, and A.Patel, "A Comprehensive Security System- the Concepts Agents and Protocols," Computers and Security, Vol 9, pp631-643, 1990.

51 S.Muftic, "Security Mechanisms for Computer Networks," Ellis-Horwood, 1991

52 F.M.Avolio, "Building Internet Firewalls," Business Communications Review, Vol 24, pp15-20, 1994.

53 "Ethernet Sniffer," Internal Report - University of Plymouth, 1995.

54 CCITT, "Data communication Networks Directory, X500-X521," 1989.

55 J.Kohl, "Distributed Open Systems: The Evolution of the Kerberos Authentication Service," Computer Society Press, 1994.

56 D.Barnes, "Designing More Secure LANs," Telecommunications, vol 27, pp64&98, 1993.

57 C.Stoll, "Stalking the Wily Hacker," Communications of the ACM, Vol 31, no 5, pp484-497, 1988.

58 A.Bonnet, J.P.Haton and J.M.Truong, "Expert Systems," Prentice Hall, 1988.

59 Microsoft Corporation, " Microsoft Visual C/C++: References ," 1991.

60 Microsoft Corporation, "Microsoft Visual Basic: References," 1991.

61 S.J.Shepherd, "A Distributed Security Architecture for Large Scale Systems," University of Plymouth, Ph.D. thesis, 1992.

62 International Telegraph and Telephone Consultative Committee V-24 Interface.

63 American National Standards Institute, Small Computer System Interface (SCSI) Specification, ANSI X3.131-1986.

64 E.H.Franck and J.Lyle, "SBus Specification B.0," Sun Microsystems Inc, 1990.

65 LSI Logic Corporation, "L64853A SBus DMA Controller Technical Manual," LSI Technical Publications, 1991.

66 "Open Boot PROM Toolkit User's Guide," Sun Microsystems Inc, 1990.

67 "Writing FCode Programs for SBus Cards," Sun Microsystems Inc, 1990.

68 E.F.Brickell, "Survey of Hardware Implementations of RSA," CRYPTO 89 Proc, Sproger-Verlag, 1990.

69 T.Beth, M.Frisch and G.J.Simmons, "Public Key Cryptography: State of The Art And Future Directions," Springer-Verlag, 1992.

70 P.Onions, "A High Speed Integrated Circuit for Applications to RSA Cryptography," University of Plymouth, Ph.D. Thesis, 1995.

71 "Technical Manual PQR512," LINTEL, Rev 1.0.

72 K.Kleiner, "Squemish Ossifarge Dents Electronic Armour," New Scientist, May 1994.

73 "CRY12C102 DES Chip," Technical Reference Manual, CRYPTECH.

74 S.R.Bourne, "The Unix System," Addison-wesley, 1983.

75 D.E.Knuth, "Seminumerical Algorithms: The Art of Computer Programming," Addison-Wesley, 1981.

76 "PC-CRYPTO TOOLKIT, User's and Technical Reference Manual Rev 2.0," Cryptech, 1989.

77 H.M.Woods, "The Use of Passwords for Controlled Access to Computer Resources," National Bureau of Standards Special Publication 500-9, 1977.

78 W.G.deRu and J.H.P.Eloff,"Improved Password Mechanisms Through Expert System Technology," Proc 9th Annual Computer Security Applications Conference 93, pp272-280, 1993.

79 B.L.Riddle, M.S.Miron and J.A.Semo, "Passwords in Use in a University Timesharing Enviroment," Computers and Security, Vol 8, no 7, pp 569-579, 1989.

80 D.Seeley, "A Tour of the Worm," Proc Winter Usenix Conference, pp287, 1989.

81 H.Johnstone, "BT Hacker Breaks into Security Files," The Times, 24th November, 1994.

82 R.Bright, "Smart Cards: Principles, Practice, Applications," Ellis Horwood, 1988.

83 A.P.Conn, J.H.Parodi and M.Taylor, "The Place of Biometrics in a User Authentication Taxonomy," Digital Equipment Corporation, pp 72-79, June 20, 1990.

84 N.M.Herbst and C.N.Liu, "Automatic Signature Verification Based on Accelerometry," IBM Journal of Research And Development, pp245-253, 1977.

85 C.Z.Bas, Y.Zhenli and Z.Lihe, "Automatic Speaker Verification Using the Neural Network and Combined LPC Parameters," Proc TENCON 93, pp345-347, 1993.

86 R.J.Spillane, "Keyboard Apparatus For Personal Identification," IBM Technical Disclosure Bulletin, 1975.

87 D.Umphress and G.Williams, "Identity Verification Through Keyboard Characteristics," Journal of Man-Machine Studies, Vol 23, pp263-273, 1985.

88 J.Leggett and G.Williams, "Verifying Identity via Keystroke Characterstics," journal of Man-Machine Studies, Vol 28, pp67-76, 1988.

89 M.Brown and S.J.Rogers, "User Identification via Keystroke Characteristics of Typed Names Using Neural Networks," Journal of Man-Machine Studies, Vol 39, pp999-1014, 1993.

90 Microsoft Corporation, "Microsoft MS-DOS6.22: User's Guide," 1994.

91 S.A.Bleha and M.S.Obaidat, "Computer Users Verification Using the Perceptron Algorithm," IEEE Transactions On Systems, Man, and Cybernetics, Vol 23, no3, pp900-902, 1993.

92 M.S.Obaidat, "A Comparative Performance Study of Neural Network Paradigms for Identifying Computer Users," Proc IEEE Computers and Communications, pp161-167, 1994.

93 R.Beale and T.Jackson, "Neural Computing an Introduction," Adam-Hilger, 1990.

94 M.H.DeGroot, "Probability and Statistics," Addision-Wesley, 1986.

95 P.K.Simpson, "Artificial Neural Systems," Pergamon Press, 1990.

96 M.Minsky and S.Papert, "Perceptrons," MIT Press, 1969.

97 R.R.Linas, "The Biology of the Brain: From Neurons to networks," W.H.Freeman and Company 1988.

98 F.Rosenblatt, "Principles of Neurodynamics," Spartan, 1962..

99 J.L.McClelland and D.E.Rumelhart, "Parallel Distributed Processing, Volume 1," MIT Bradforf Press, 1986.

100 T.Kohonen, "Correlation Matrix Memories," IEE Transactions on Computers, Vol 21, pp353-359, 1972.

101 A.N.Kolmogorov, "On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and Addition," American Mathematical Society Translation, Vol 28, pp55-59, 1963.

102 D.E.Goldberg, "Genetic Algorithms in Search, Optimisation. and Machine Learning," Addison-Wesley, 1989.

103 J.B.Madsen, "The Greatest Cracker-Case in Denmark: The Detecting, Tracing and Arresting of Two International Crackers," Proc USENIX, pp17-40, 1992..

104 M.Rentell and P.Jenner, "Breakdowns in Computer Security: Commentary and Analysis," Computer Weekly Publications and PA consulting Group, 1991.

105 Neil Fawcett, "Microsoft Gives Virus to 200 Top UK Developers," Computer Weekly, 23rd February, 1995..

106 C.Sandler, "User's guide to the Vax/VMS operating System," Scott, Foresman and Company, 1989.

107 J.P.Anderson, "Computer security Threat Monitoring and Surveillance," Technical Report, Anderson Co, 1980.

108 J.Hurska, "Computer Viruses and Anti-Virus Warfare," Ellis Horwood, 1992 .

109 B.Clough and P.Mungo, "Approaching Zero: Data Crime and the Computer Underworld," Faber and Faber, 1992.

110 D.E.Denning, "An Intrusion Detection-Model," Trans IEEE Software Engineering, Vol 13, no 2, pp222-232, 1987.

111 M.Denault, D.Gritzalis, D.Karagiannis and P.Spirakis, "Intrusion Detection: Approach and Performancs Issues of the SECURENET System," Computer Security, Vol 13, no 6, pp495-508, 1994.

112 C.Dowell and P.Ramstedt, "The COMPUTERWATCH Data Reductuction Tool," Proc Computer Security Conference Washington, pp98-108, 1990.

113 S.E.Smaha, "Haystack: An Intrusion Detection System," Proc IEEE 4th Aerospace Computer Security Applications Conference,1988.

114 D.S.Bauer and M.E.Koblentz, "NIDX- A Real-Time Intrusion Detection Expert System," Proc USENIX, pp261-273, 1988.

115 T.F.Lunt and R.Jagannathan, "A Prototype Real-Time Intrusion-Detection Expert System," Proc IEEE Security and Privacy, 1988.

116 T.F.Lunt, "Automated Audit Trail Analysis and Intrusion Detection: A Survey," Proc 11 National Computer Security Conference, 1988.

117 H.S.Javitz and A.Valdes, "The SRI IDES Statistical Anomaly Detector," Proc IEEE Security and Privacy, 1991.

118 T.F.Lunt, "A Survey of Intrusion Detection Techniques," Computers And security, Vol 12, pp405-418, 1993.

Appendix A

DES and RSA

A Description of the RSA Cryptosystem	261
A Description of the DES Cryptosystem	265
DES Tables	269

A Description of the RSA Cryptosystem

In 1976 Diffie and Hellman published "New Directions In Cryptography," [30] in which they laid the ground work for public-key cryptography, and in so doing gave two important definitions of 'one-way functions' and 'trapdoor one way functions.'

A 'one way function' is one in which $f(x)$ is easy to calculate.

$y = f(x)$ (easy for a given x, where the exact form of f is known),

but, almost impossible to calculate x given y

A 'trapdoor one way function' is a pair of functions, one of which acts as a one-way function while the other acts as a trapdoor, so that the calculation of the one-way function's inverse is possible.

$$y = f(e,x) \quad \textbf{Equation A-1}$$

$$x = f^{-1}(d,y) \quad \textbf{Equation A-2}$$

y in Equation A-1 is easily calculable for all x and e. However, given e and y it is infeasible to calculate x, this is the one-way function. Equation A-2 acts as the trapdoor, given d and y it is easy to calculate x. It should be impossible to calculate d given e and y, and without d it should be impossible to calculate x given y using Equation A-2.

Rivest, Shamir and Adleman's "A Method For Obtaining Digital Signatures And Public-Key Cryptography," [31] presented a one-way trapdoor function used in RSA.

There are three basic concepts of number theory that are required to understand the fundamentals of RSA.

1) One of the most important theorems in number theory and modern cryptography is Fermat's Little Theorem [32],

$$a^{n-1} = 1 \pmod{n}$$

where a and n are any positive integers.

Fermat's Little Theorem is true for all prime n.

An integer n that satisfies Fermat's Little Theorem is called a pseudo-prime and any n that fails is a composite number. A pseudo-prime may be proved composite when tested against another base a. If an integer n passes several tests against different bases it is very probably prime. Very rarely will a composite integer satisfy Fermat's Little Theorem for all bases. Integers that do this are known as Carmichael numbers [40], and to give an idea of how rare they are, there are only 255 of them below 100,000,000. An example of Carmichael numbers are 561, 1105 and 1729.

2) Euler's Totient Function [32] states that,

$x^{\phi(n)} = 1 \pmod{n}$,

where x and n are positive integers, $x < n$, and G.C.D(x, n) = 1.

$\phi(n)$ is Eulers totient function and is the number of integers less than n that are co-prime with n. eg,

$\phi(8) = 4$ since only 1, 3, 5 and 7 share a GCD = 1 with 8

If n is prime then,

$\phi(n) = n - 1$

eg,

if n = 7

then 1,2,3,4,5,6 share a GCD = 1 with 7

and $\phi(n) = 6$

If n is a composite number made up of two prime factors p and q

$\phi(pq) = (p-1)(q-1)$

eg,

if $p = 3$ and $q = 5$

then 1,2,4,7,8,11,13,14 share a GCD = 1 with 15

and $\phi(n) = 8$

3) Where $0 < e < m$ and GCD(m, e) = 1,

$de = 1 \pmod{m}$

d has a unique solution.

The trapdoor one-way function as suggested by Rivest, Shamir and Adleman is

$$f(x) = y = x^e \pmod{n}$$

The inverse function is

$$f^{-1}(y) = x = y^d \pmod{n},$$

where x is a non-negative integer, $0 \le x < n$.

For the RSA trapdoor one way function to work, x^{ed} must equal x.

If we make $n = pq$, where p and q are distinct large prime numbers,

$$de = \phi(n) + 1$$

and that is the same as

$$de = 1 \pmod{\phi(n)}$$

Therefore we can calculate a unique solution for d according to concept 3 above

Using the trapdoor one way function,

$$x^{ed} = x^{\phi(n)+1} \pmod{n}$$

$$= (x^{\phi(n)})x \pmod{n}$$

Using Euler's theorem it can be reduced to

$$x^{ed} = x \pmod{n}$$

A Description of the DES Cryptosystem

DES is a product cipher incorporating substitution, transposition and bit-by-bit modulo 2 addition. The DES algorithm is shown below in Figure A-3. All data such as the transposition and substitution tables can be found in DES tables section.

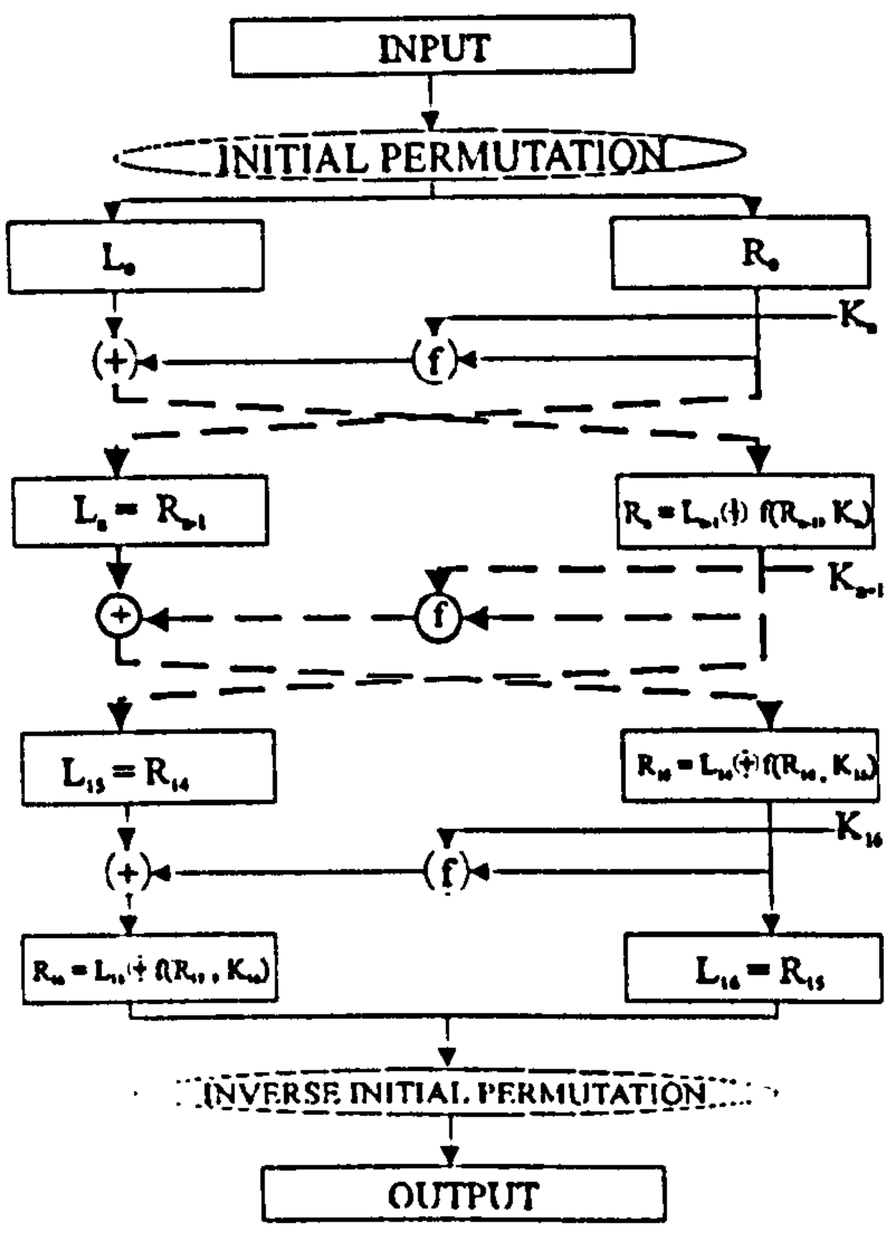


Figure A-1 DES Algorithm

DES requires an input block of 64 bits (*LR*), which undergoes an *initial permutation*, *IP*, that is not key dependent.

The input block is then split into two blocks of 32 bits, the left 32 bits (L_n) and the right 32 bits (R_n) . Blocks L_n and R_n undergo 16 complex key dependent computations before going through an inverse of the initial permutation *IP-1*. This is again not key dependent.

The complex key dependent calculations involve a block K of 48 bits derived from the key,

and the Blocks L and R.

$$L' = R$$
$$R' = L \oplus f(R, K)$$

Where $\oplus$ is bit by bit modulo 2 addition

We can use n to describe the nth iteration of the complex key dependent calculation,

$\geq n \leq 16$.

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} \oplus f(R_{n-1}, K_n) \quad \textbf{Equation A-3}$$

K_n is the 48 bits derived from the key by KS, the Key Schedule.

Key Schedule

$$K_n = KS(n, KEY)$$

Figure A-4 illustrates how Kn is calculated,

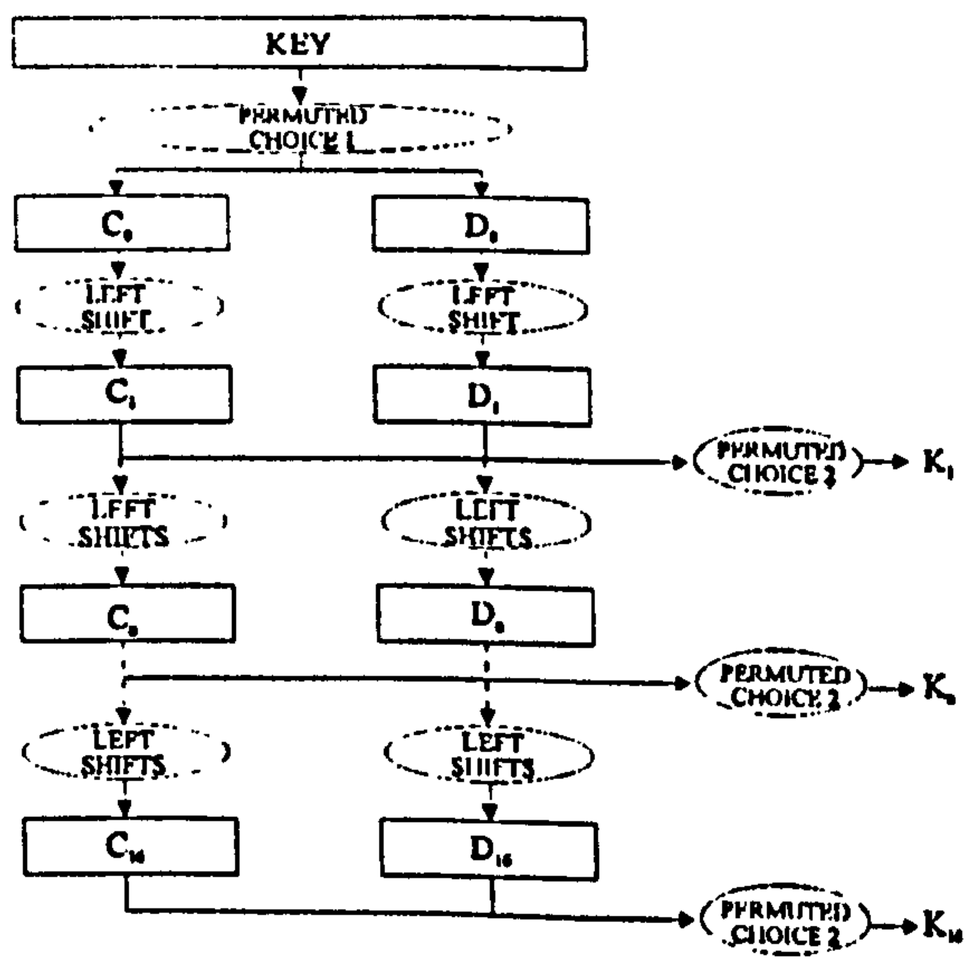


Figure A-2 Key Schedule

The 56 bits of the key (those bits that are not used for parity checking) are permuted using table *Permuted Choice 1* to form two 28 bit words *Cn* and *Dn*. These then undergo a left shift of 1 or 2 bits. *Kn* is formed from a transposition of the 56 bit binary number *CnDn*. However, only 48 bits are used in *Kn*.

Key Dependent Complex Calculation

The function of Equation A-3 is calculated using substitution boxes as in Figure A-5.

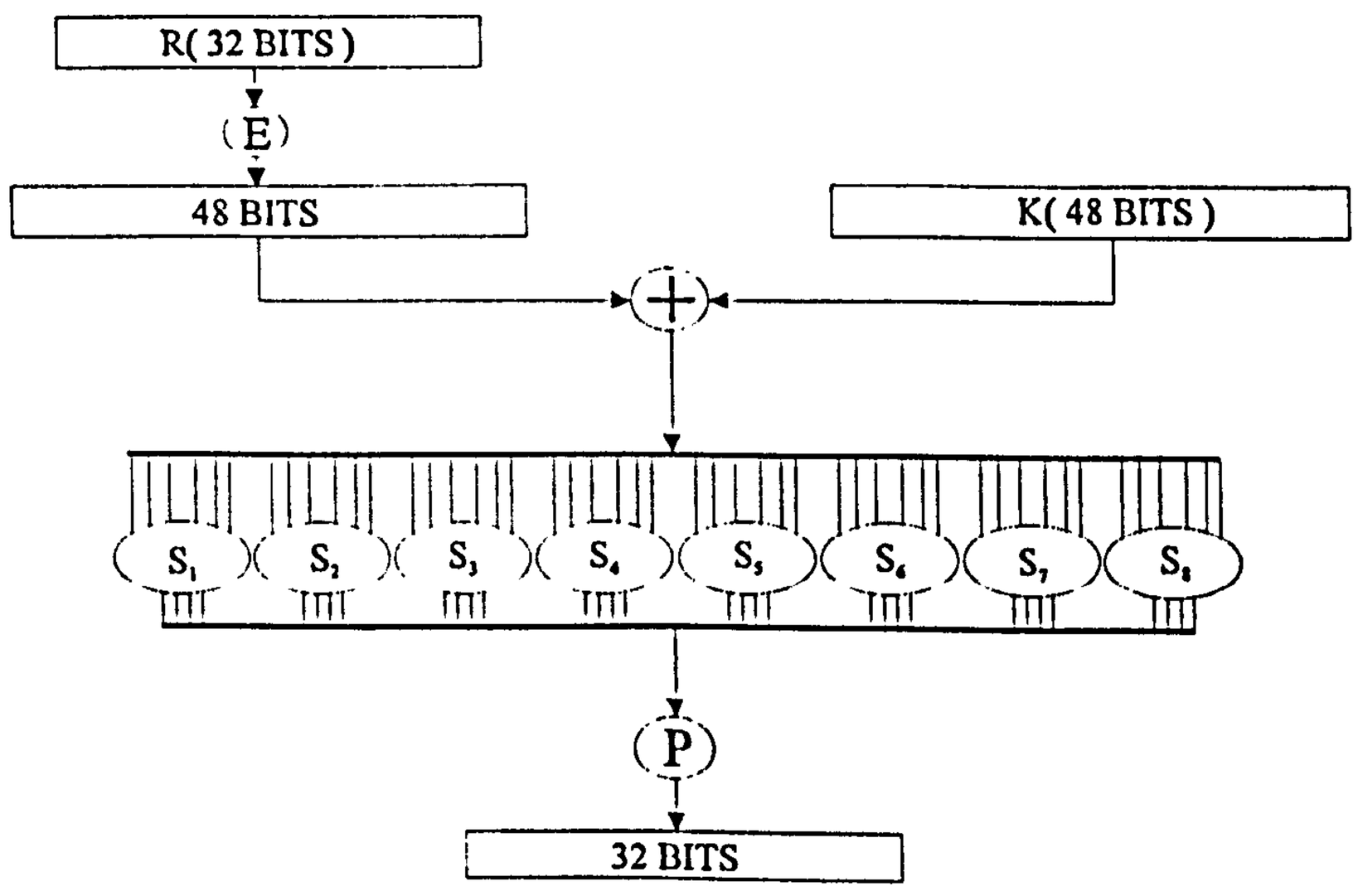


Figure A-3 Key Dependent Function

The 32 bit *Rn* block of data is permuted (using the permutation table E) into a 48 bit word and then modulo 2 addition is performed with *Kn*. The result of the modulo 2 addition is split into 8 6-bit words. Each 6-bit word undergoes a substitution according to the

Substitution tables and resulting in a 4-bit word. The S1 substitution table can be seen below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Table A-1 S1 Substitution Table

The 6-bit word's most significant bit and least significant bit are combined to give a number between 0 and 3 used to index a row in the substitution table, the 4 middle bits are used to index a column in the substitution table. The number found at the indexed location is a number between 0 and 15 that can be represented as a 4-bit word, eg if the 6 bit word was 001011 it would be substituted by 0010.

The combined result of the 8 substitution boxes give a 32-bit word that is then permuted according to permutation P. This final 32-bit word is added bit by bit modulo 2 addition to L_n.

The Key Dependent Complex Equation is performed 16 times. The resulting Rn and Ln are then combined to give a 64 bit output block which undergoes a final permutation.

DES TABLES

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	28	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Table A-1 Initial Permutation

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Table A-2 Inverse Initial Permutation

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Table A-3 S1 Substitution Table

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Table A-4 S2 Substitution Table

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Table A-5 S3 Substitution Table

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Table A-6 S4 Substitution Table

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Table A-7 S5 Substitution Table

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Table A-8 S6 Substitution Table

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Table A-9 S7 Substitution Table

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table A-10 S8 Substitution Table

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Table A-11 E Permutation Table

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Table A-12 P Permutation Table

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Table A-13 Permuted Choice 1

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Table A-14 Permuted Choice 2

ITERATION NUMBER	NUMBER OF LEFT SHIFTS
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Table A-15 Left Shifts Table

Appendix B

ENCDEC Board

SBus Timing Chart	**273**
Interface Schematic	**274**
Processing Board Schematic	**275**
RSA Schematic	**276**
DES Schematic	**277**
ENCEC Board Firmware	**278**

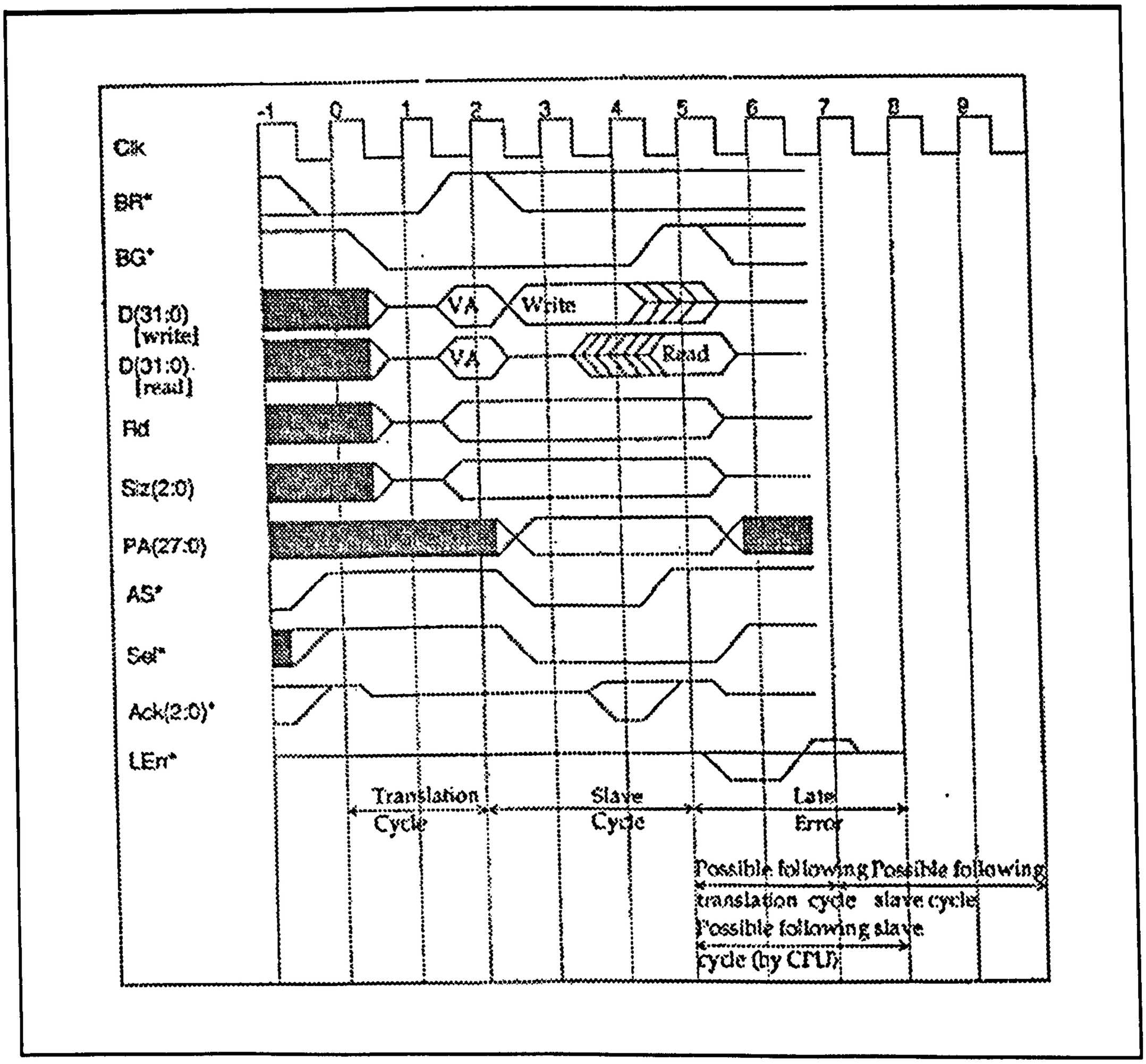


Figure B-1 SBus Timing Chart

Figure B-2 Interface Schematic

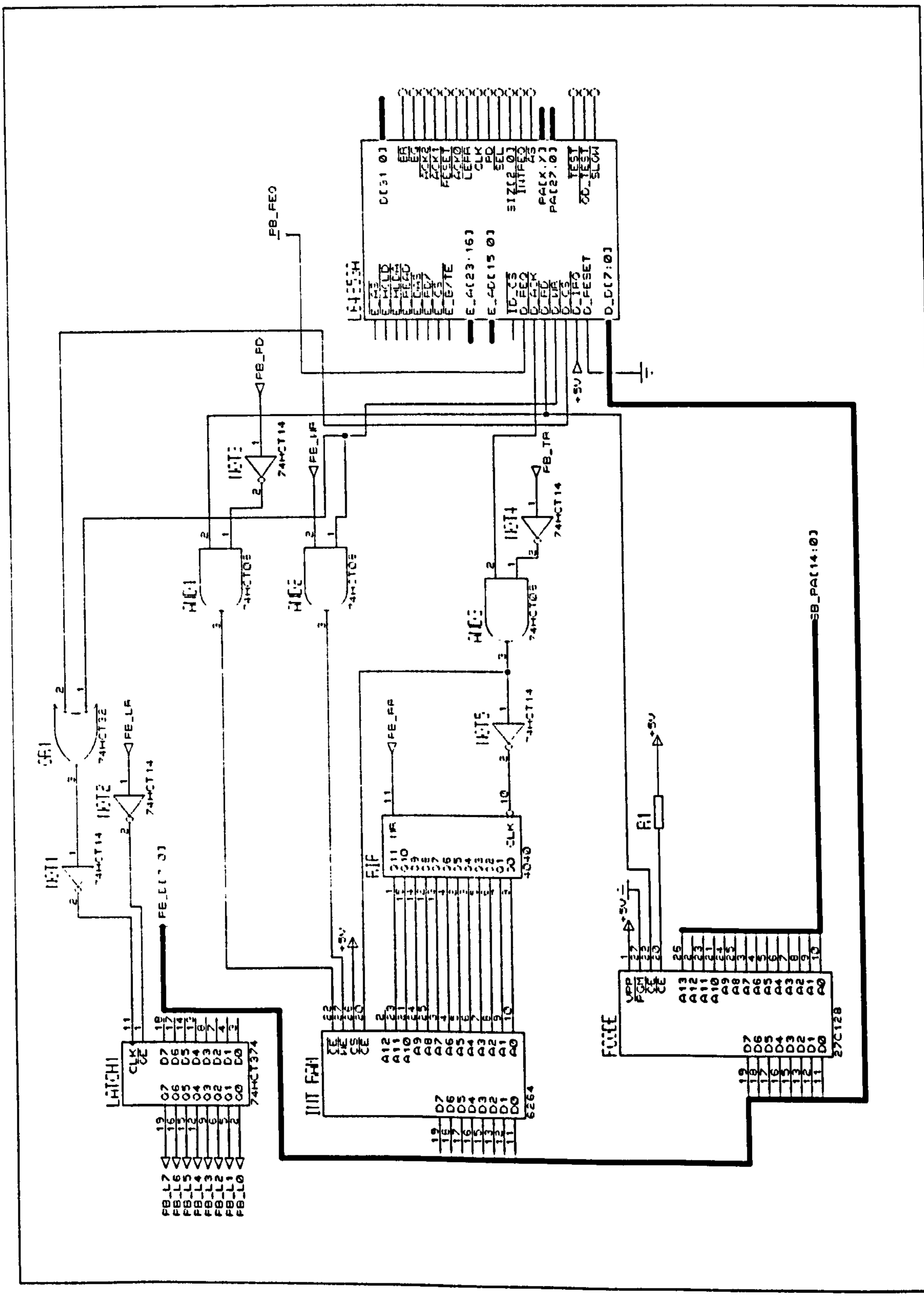


Figure B-3 Processing Board Schematic

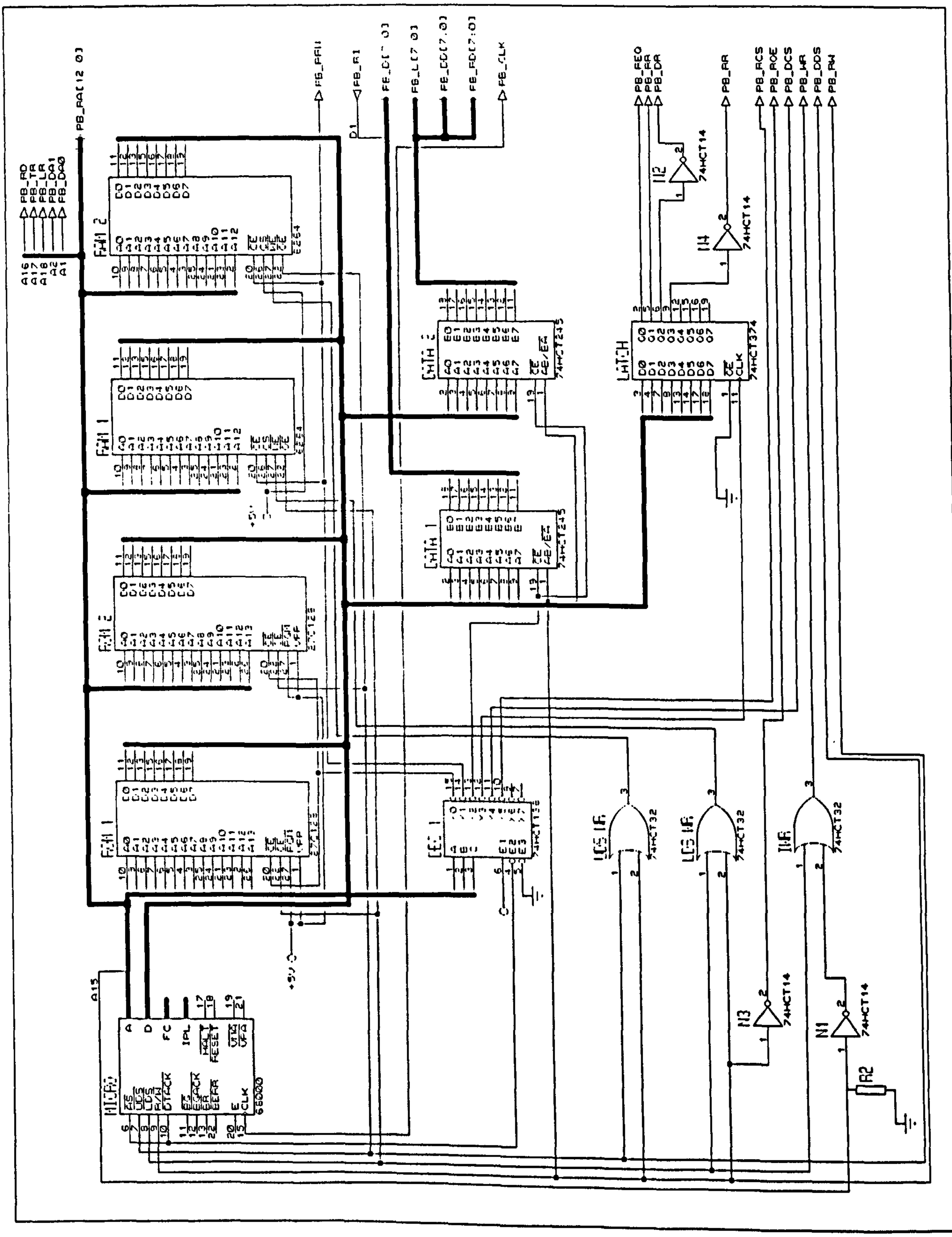


Figure B-4 RSA Schematic

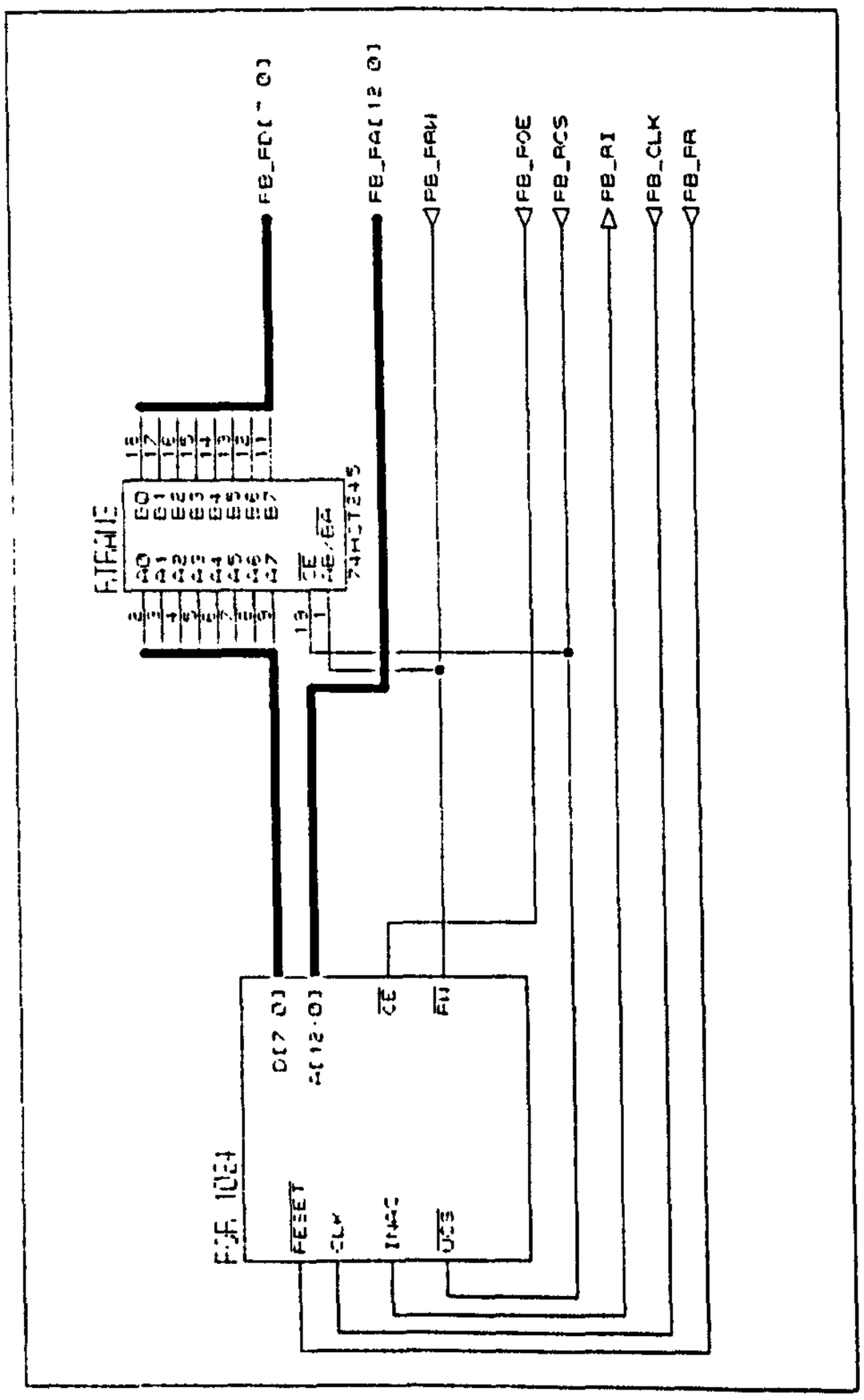
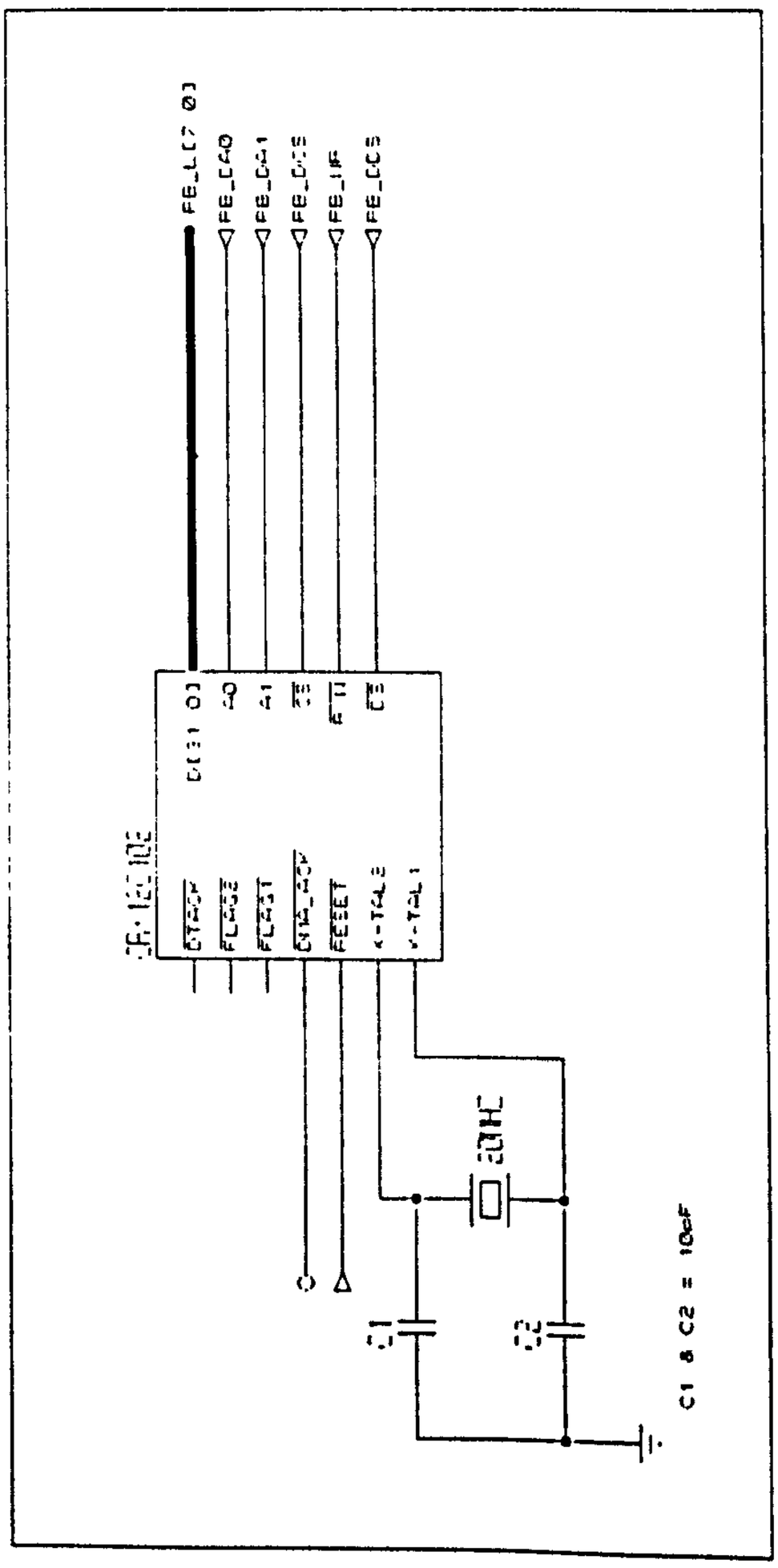


Figure B-5 DES Schematic



ENCDEC Board Firmware

```
*****************************************************************
*** File                Mult.S                                  ***
*** Language:          68000 Assembly                          ***
***                                                            ***
***                                                            ***
***        This is the firmware stored upon the Processing board ***
*****************************************************************

        dc.l    $203e80
        dc.l    $000400
        ds.l    $fe

***     PHYSICAL ADDRESS OF COMOPONENTS         ***

mem     equ     $200000
ramre   equ     $430001
ramwr   equ     $428001
regre   equ     $440000
latch   equ     $600001
desch   equ     $800000
dc_st2  equ     $800004  ; write/read dc=des command register
dcm_st3 equ     $800006  ; write/read dcm=des command mod register
den_st1 equ     $800002  ; write/read den=des config register
rsach   equ     $a00000

***     INITIALISE BOARD ***

        move.b  #$06,latch ;reset ram counter
        move.b  #$91,latch ; and deschip


***     TEST TO SEE IF DATAREADY IS SET     ***

loop:   move.b  regre,d0
        andi.b  #$01,d0
        cmpi.b  #$01,d0
        bne     loop

***     MOVE CON_PK TO 68K MEMORY       ***

str:    move.l  #$200000,a1
        move.b  #$02,latch
        move.b  #$00,latch

*****************************************************************
***CONTROL PACKETS                                          ***
***                                                         ***
***                                                         ***
***     RSA_PACKET              DES_PACKET          ***
***      Function               Function            ***
***      Blank                  Blank               ***
***      Mod_length             Util                ***
***      Exp_length             Header              ***
***      Mess_length            Mess_length         ***
***                                                         ***
*****************************************************************


        move.b  ramre,(a1)+             ;load con_pk
        move.b  ramre,(a1)+
        move.b  ramre,(a1)+
        move.b  ramre,(a1)+
        move.b  ramre,(a1)+
        move.b  ramre,(a1)+

***     CHECK WHAT FUNCTION IS REQUESTED  ***

        move.b  $200000,d0
        moveq.b #$07,d1
        btst    d1,d0
        beq     chf
        bsr     rsa
```

```
        andi.b    #$7f,$200000
        move.b    $200000,d0
chf:    subq.b    #$01,d1
        btst      d1,d0
        beq       chf1
        bsr       dess
        andi.b    #$bf,$200000
        move.b    $200000,d0
chf1:   subq.b    #$02,d1
        btst      d1,d0
        beq       chf2
        bsr       test
        andi.b    #$ef,$200000
        move.b    $200000,d0
chf2:   move.b    $200000,d0
        andi.b    #$10,d0
        beq       chf3
        move.w    #$0000,d0
        bsr       error

***     MOVE DATA TO INTRAM     ***

chf3:   move.w    $200004,d0
        move.b    #$02,latch
        move.b    #$00,latch
        move.l    #$200000,a0

loop6:  move.b    (a0)+,ramwr       ;load intram with
        move.b    (a0)+,ramwr       ;new con_pk
        move.b    (a0)+,ramwr
        move.b    (a0)+,ramwr
        move.b    (a0)+,ramwr
        move.b    (a0)+,ramwr

loop7:  move.b    (a0)+,ramwr       ;load intram with
        move.b    (a0)+,ramwr
        move.b    (a0)+,ramwr
        move.b    (a0)+,ramwr
        move.b    (a0)+,ramwr
        move.b    (a0)+,ramwr
        move.b    (a0)+,ramwr
        move.b    (a0)+,ramwr
        sub.w     #$0008,d0 ;enc mess
        bhi       loop7

skip:   move.b    #$02,latch
        move.b    #$00,latch
        move.b    #$01,latch

***     SETUP DMA ACCESS        ***

loop8:  move.b    regre,d0    ;wait for end of read signal
        andi.b    #$01,d0
        cmpi.b    #$01,d0
        beq       loop8

        move.b    #$03,latch
        move.b    #$91,latch

        jmp       loop

**********************************************************
***                                                    ***
***        What Follows Are The Various Subroutines    ***
***                                                    ***
**********************************************************


**********************************************************
***                                                    ***
*** This section is used to load new keys into the     ***
*** the relevant line in the memory table. At the      ***
*** moment each line is 24 bytes long, enough to hold  ***
*** 1 DES key, an IV, and the current working IV for   ***
*** on going encryption/decryption.                    ***
***                                                    ***
```

```
**********************************************************

keyld:    move.w    sr,-(a7)
          movem.l   d0-d7/a0-a6,-(a7)

          move.b    #$00,latch

          clr       d1
          move.l    #$200006,a0
          move.l    #$203000,a1
          move.b    #$18,d0          ;This indicates space available
          move.b    $200003,d1
          mulu      #$0018,d1
          adda.l    d1,a1
          move.b    d0,d2
          move.l    a1,a2
          move.l    #$200006,a3
keyl:     move.b    ramre,(a1)+
          sub.b     #$01,d0
          bne       keyl
keyl2:    move.b    (a2)+,(a3)+
          sub.b     #$01,d2
          bne       keyl2
          movem.l   (a7)+,d0-d7/a0-a6
          rtr

test:     move.w    sr,-(a7)
          movem.l   d0-d7/a0-a6,-(a7)

          move.w    $200004,d0
          move.l    #$200006,a0
tlp:      move.b    ramre,(a0)+
          subq.w    #$01,d0
          bne       tlp
          move.w    $200000,d0
          move.l    #$200006,a0
          move.l    #$200006,a1
tlp1:     move.b    (a0)+,d1
          move.b    d1,(a1)+
          subq.w    #$01,d0
          bne       tlp1
          movem.l   (a7)+,d0-d7/a0-a6
          rtr

*****************************************
***       The cry12C102 commands      ***

lek1      equ       $ee       ;Load External key
lek2      equ       $ed
lek3      equ       $ee
lek4      equ       $ef
slk1      equ       $e8       ;Select Key
slk2      equ       $e9
slk3      equ       $ea
slk4      equ       $eb
nop       equ       $89
liv       equ       $c0
rfv       equ       $c2
des       equ       $a8       ;DES on work register 1
mre_ecb   equ       $02
mrd_ecb   equ       $03
ce_fb8    equ       $00
cd_fb8    equ       $01

**********************************************************
***                                                    ***
*** The following sections deal with DES               ***
*** Encryption/Decryption                              ***
***                                                    ***
**********************************************************

dess:     move.w    sr,-(a7)
          movem.l   d0-d7/a0-a6,-(a7)

          move.b    $200000,d0
          and.b     #$20,d0
```

```
        beq     dlp2
        bsr     keyld
        and.b   #$df,$200000
        jmp     desex

dlp2:   move.w  $200004,d0
        move.l  #$200006,a0
dlp1:   move.b  ramre,(a0)+
        subq.w  #$0001,d0
        bne     dlp1

        move.b  $200000,d0
        andi.b  #$01,d0
        beq     dlp

        move.b  $200000,d0
        andi.b  #$08,d0
        bne     dlp

        clr     d0
        move.w  $200004,d0
        divu    #$0008,d0
        and.l   #$ffff0000,d0
        ror.l   #$08,d0
        ror.l   #$08,d0
        move.l  #$08,d3
        sub.w   d0,d3
        move.l  #$200006,d2
        clr     d1
        move.w  $200004,d1
        add.l   d1,d2
        add.l   d3,d2
        move.l  d2,a0
        move.b  d3,-(a0)
        add.w   d3,$200004

dlp:    move.b  #$04,latch
        move.b  #$00,$201fa4
        move.b  #$00,$201fa5
        move.b  #$00,latch

        move.b  #$00,dcm_st3
        move.b  #nop,dc_st2
        move.b  #$80,dcn_st1        ;This sets the cry12e bus size
        move.b  #$10,dcn_st1        ;This clears the reserve bit in flag2

        clr     d0
        move.w  $200004,d0
        divu    #$0008,d0
        and.l   #$0000ffff,d0
        move.l  #$200006,a0
        move.l  #$200006,a1

        move.b  $200002,d2
        move.b  #$04,d1
        btst    d1,d2
        bne     pecb
        subq.b  #$01,d1
        btst    d1,d2
        bne     ecb
        subq.b  #$01,d1
        btst    d1,d2
        beq     dnx
        move.b  #$09,d7
        jmp     cbc
dnx:    subq.b  #$01,d1
;       btst    d1,d2
;       beq     dnx1
;       move.b  #$0a,d7
;       jmp     fbc
;dnx1:  subq.b  #$01,d1
;       btst    d1,d2
;       beq     dnx2
;       move.b  #$0b,d7
;       jmp     fbc
dnx2:   jmp     uterr
```

```
uterr:      move.b    #$f0,latch
            jmp       uterr

************************************
***         Electronic codebook    ***
***         non pipelined          ***

ecb:        jsr       lkey
            move.b    $200000,d2
            andi.b    #$08,d2
            bne       ecbdec

***         ecb encryption         ***

            move.b    #$00,dcm_st3
            move.b    #mre_ecb,dc_st2

            jsr       enc
            jmp       desex

***         ecb decryption ***

ecbdec:     move.b    #$00,dcm_st3
            move.b    #mrd_ecb,dc_st2

            move.b    #$ff,$201fa4
            jsr       dec
            jmp       desex

************************************
***         ecb pipelined          ***

pecb:       jsr       lkey

            move.b    $200000,d2
            andi.b    #$08,d2
            bne       pecbde

***         ecb encryption         ***

pecbl:      move.b    #$00,dcm_st3
            move.b    #ee_fb8,dc_st2

            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            subq.w    #$0001,d0

            jsr       enc

            move.b    #$ff,$201fa5
            move.w    #$0000,d0
            jsr       enc
            jmp       desex

***         pecb decryption        ***

pecbde:     move.b    #$00,dcm_st3
            move.b    #ed_fb8,dc_st2

            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            subq.w    #$0001,d0

            jsr       dec
```

```
pecbl2:   move.b    #$ff,$201fa5
          move.b    #$ff,$201fa4
          move.w    #$0001,d0
          jsr       dec
          jmp       desex

**********************************
***       Cipher block coding   ***

cbc:      jsr       lkey

          clr       d5
          move.b    $200000,d4
          andi.b    #$02,d4
          beq       cbcnx
          move.w    #$0008,d5
          jmp       cbcnx1
cbcnx:    move.w    #$0010,d5
cbcnx1:   move.l    #$203000,a3
          move.l    a3,a2
          clr       d4
          move.b    $200003,d4
          mulu      #$0018,d4
          adda.l    d4,a3
          adda.l    d4,a2
          adda.l    #$0010,a2
          adda.l    d5,a3

          move.b    #liv,dc_st2
          move.b    (a3)+,desch
          move.b    (a3)+,desch
          move.b    (a3)+,desch
          move.b    (a3)+,desch
          move.b    (a3)+,desch
          move.b    (a3)+,desch
          move.b    (a3)+,desch
          move.b    (a3)+,desch
          move.b    (a3)+,desch
          move.b    $200000,d2        ;work out whether encryption
                                      ; or decryption is required
          andi.b    #$08,d2
          bne       cbcdec

***       cbc encryption        ***

cbcnx2:   move.b    d7,dcm_st3
          move.b    #ce_fb8,dc_st2

          move.b    (a1)+,desch       ; Fills the pipe
          move.b    (a1)+,desch
          move.b    (a1)+,desch
          move.b    (a1)+,desch
          move.b    (a1)+,desch
          move.b    (a1)+,desch
          move.b    (a1)+,desch
          move.b    (a1)+,desch
          subq.w    #$0001,d0

          jsr       enc
          move.b    #$ff,$201fa5      ; Initialise the memory location for
          move.w    #$0000,d0 ;  the last read
          jsr       enc

          move.b    #rfv,dc_st2

          move.b    desch,(a2)+
          move.b    desch,(a2)+
          move.b    desch,(a2)+
          move.b    desch,(a2)+
          move.b    desch,(a2)+
          move.b    desch,(a2)+
          move.b    desch,(a2)+
          move.b    desch,(a2)+
          move.b    desch,(a2)+
          jmp       desex
```

```
***         cbc decryption          ***

cbcdec:     move.b      d7,den_st3
            move.b      #ed_fb8,de_st2

            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
            subq.w      #$0001,d0

            jsr         dec

cbcnx3:     move.b      #$ff,$201fa5
            move.b      #$ff,$201fa4
            move.w      #$0001,d0
            jsr         dec

            move.b      #rfv,de_st2

            move.b      desch,(a2)+
            move.b      desch,(a2)+
            move.b      desch,(a2)+
            move.b      desch,(a2)+
            move.b      desch,(a2)+
            move.b      desch,(a2)+
            move.b      desch,(a2)+
            move.b      desch,(a2)+
            move.b      desch,(a2)+
            jmp         desex

temp:       move.b      #$00,latch
            jmp temp

desex:      move.b      #$04,latch
            move.b      #$00,$201fa4
            move.b      #$00,$201fa5
            move.b      #$00,latch

            movem.l     (a7)+,d0-d7/a0-a6
            rtr

*********************************************
***                                       ***
***         Standard encryption routine   ***
***                                       ***
*********************************************

enc:        move.w      sr,-(a7)
            movem.l     d1-d7/a2-a6,-(a7)

            move.b      $201fa5,d5
            move.b      #$00,$201fa5
            andi.b      #$ff,d5
            bne         encl2

encl:       sub.w       #$0001,d0 ;test must be here for messages
            beq         encl6                   ; smaller than 8 bytes
            bls         enex

encl6:      move.b      den_st1,d3
            and.b       #$08,d3
            bne         encl6

            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
            move.b      (a1)+,desch
```

```
            move.b    #$00,latch
encl1:      move.b    den_st1,d3
            and.b     #$80,d3
            bne       uterr
encl2:      move.b    den_st1,d3
            and.b     #$20,d3
            beq       encl2

encl3:      move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+

            jmp       encl

encx:       move.b    #nop,dc_st2
            movem.l   (a7)+,d1-d7/a2-a6
            rtr

*********************************************
***                                       ***
***       Standard decryption routine     ***
***                                       ***
*********************************************

dec:        move.w    sr,-(a7)
            movem.l   d1-d7/a2-a6,-(a7)

            move.b    $201fa5,d5
            move.b    #$00,$201fa5
            andi.b    #$ff,d5
            bne       decl2

decl:       move.b    den_st1,d3
            and.b     #$08,d3
            bne       decl

            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch
            move.b    (a1)+,desch

            move.b    #$00,latch
decl1:      move.b    den_st1,d3
            and.b     #$80,d3
            bne       uterr
decl2:      move.b    den_st1,d3
            and.b     #$20,d3
            beq       decl2

decl3:      move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+
            move.b    desch,(a0)+

            sub.w     #$0001,d0 ;test must be here for messages
            bne       decl                    ; smaller than 8 bytes

decl4:      move.b    $200000,d0
            and.b     #$01,d0
            beq       decx
```

```
            move.b    $201fa4,d0
            andi.b    #$01,d0
            beq       deex

            clr       d0
            move.b    -(a0),d0
            sub.w     d0,$200004

deex:       move.b    #nop,dc_st2
            movem.l   (a7)+,d1-d7/a2-a6
            rtr

*********************************************************
***                                                   ***
***         This routine loads the des key into the   ***
***         chip                                      ***
***                                                   ***
*********************************************************

lkey:       move.w    sr,-(a7)
            movem.l   d0-d7/a0-a6,-(a7)

            clr       d1
            move.l    #$203000,a0
            move.b    $200003,d1
            mulu      #$0018,d1
            adda.l    d1,a0
            move.b    #lck1,dc_st2

            move.b    (a0)+,desch
            move.b    (a0)+,desch
            move.b    (a0)+,desch
            move.b    (a0)+,desch
            move.b    (a0)+,desch
            move.b    (a0)+,desch
            move.b    (a0)+,desch
            move.b    (a0)+,desch
            move.b    (a0)+,desch

            move.b    #slk1,dc_st2

            move.b    #$d0,latch
krec:       move.b    #$a0,d4
kel:        move.b    dcm_st3,d3
            and.b     #$40,d3
            bne       kex
            subq.b    #$01,d4
            bne       kel

kex:        movem.l   (a7)+,d0-d7/a0-a6
            rtr

*********************************************
***                                       ***
***         RSA address commands          ***
***                                       ***
*********************************************

abort       equ       $a0200e
plend       equ       $a02004
inten       equ       $a02006
intdi       equ       $a0200e
start       equ       $a02008
plore       equ       $a02000

*********************************************
***                                       ***
***         RSA program commands          ***
***                                       ***
*********************************************

endr        equ       $00
initmod     equ       $10
ldou0       equ       $20
ldou1       equ       $21
ldou5       equ       $25
ldou6       equ       $26
```

```
ldin7     equ     $37
ldin5     equ     $35
ldin6     equ     $36
ldin0     equ     $30
multpl0   equ     $60
multpl6   equ     $66
square    equ     $c0
shrigh    equ     $90
shleft    equ     $80
expon     equ     $70
preset    equ     $f0

rsa:      move.w  sr,-(a7)
          movem.l d0-d7/a0-a6,-(a7)

cn:       move.b  $200000,d0
          andi.b  #$20,d0
          bne     keyin
          jmp     rdata

*** Turn modulus into 2's complement ***

keyin:    andi.b  #$df,$200000
          move.w  $200004,d0

          move.l  #$200006,a0       ; load key
rlp:      move.b  ramre,(a0)+
          subq.w  #$0001,d0
          bne     rlp


          move.l  #$202134,a0       ; store the modulus
          move.l  #$200006,a1
          move.b  $200002,d0
rlp1:     move.b  (a1)+,(a0)+
          subq.b  #$01,d0
          bne     rlp1

          clr     d0
          move.l  #$200006,a0
          move.b  $200002,d0
          move.b  d0,d1

lmod:     eori.b  #$ff,(a0)+
          subq.b  #$01,d0
          bne     lmod

lmod1:    addi.b  #$01,-(a0)
          bcc     erprog
          subq.b  #$01,d1
          bne     lmod1
          move.w  #$0001,d0
          bsr     error
          jmp     rex

erprog:   move.b  #$08,latch
          move.b  #$00,latch
          move.b  #$00,inten
          move.b  #$00,plore

rep:      move.b  regre,d0
          move.b  #$30,latch
          and.b   #$02,d0
          beq     rep
          move.b  #$00,latch

          move.l  #$a00000,a0

          move.b  $200000,d0
          andi.b  #$40,d0
          bne     muld

*** Load Modulus ***

          move.b  #$f4,(a0)+ ; preset 4
          addq.l  #$01,a0
```

```
                move.b    #$f0,(a0)+ ; preset 0
                addq.l    #$01,a0
                move.b    #$fe,(a0)+ ; preset e
                addq.l    #$01,a0
                move.b    #ldin7,(a0)+          ; loadin7
                addq.l    #$01,a0
                move.b    #shrigh,(a0)+
                addq.l    #$01,a0
                move.b    #shrigh,(a0)+
                addq.l    #$01,a0

                move.b    $200001,d4
                cmp.b     #$00,d4
                beq       lm1
lm:             move.b    #shleft,(a0)+
                addq.l    #$01,a0
                addq.b    #$01,d4
                cmp.b     #$08,d4
                bne       lm

lm1:            move.b    #initmod,(a0)+        ; initmod
                addq.l    #$01,a0

*** load message ***

                move.b    #$f4,(a0)+
                addq.l    #$01,a0
                move.b    #$f0,(a0)+
                addq.l    #$01,a0
                move.b    #$fe,(a0)+
                addq.l    #$01,a0
                move.b    #ldin0,(a0)+
                addq.l    #$01,a0

                move.b    $200001,d5
                cmp.b     #$00,d5
                beq       one
                cmp.b     #$07,d5
                beq       cpl2
                move.b    #$07,d5
                sub.b     $200001,d5
cpl:            move.b    #shleft,(a0)+
                addq.l    #$01,a0
                subq.b    #$01,d5
                bne       cpl
                jmp       cpl2
one:            move.b    #shrigh,(a0)+
                addq.l    #$01,a0

cpl2:           move.b    #ldou0,(a0)+
                addq.l    #$01,a0
                move.b    #ldin0,(a0)+
                addq.l    #$01,a0

*** calculation ***

                clr       d0
                move.b    $200002,d0
                move.b    $200001,d2
                cmp.b     #$00,d2
                bne       mbit
                mulu      #$0008,d0
                jmp       nxcpl
mbit:           sub.b     #$01,d0
                mulu      #$0008,d0
                clr       d2
                move.b    $200001,d2
                add.w     d2,d0

nxcpl:          addq      #$02,d0
                move.w    d0,d1
                move.w    d0,d2
                ror.w     #$08,d0               ; set the presets
                ori.b     #preset,d0
                move.b    d0,(a0)+
                addq.l    #$01,a0
```

```
        ror.w   #$04,d1
        ori.b   #preset.d1
        move.b  d1,(a0)+
        addq.l  #$01,a0
        ori.b   #preset.d2
        move.b  d2,(a0)+
        addq.l  #$01,a0

        clr     d0
        move.b  $200002,d0
        add.l   #$200006,d0
        move.l  d0,a1
        move.b  $200003,d0
        move.b  (a1)+,d1
        move.b  d1,d2
        moveq.b #$08,d3         ; bit counter
        moveq.b #$04,d4         ; 1st nibble counter
        and.b   #$f0,d2         ; check to see if 1 in first
        bne     lepr            ; nibble
        subq.b  #$04,d3
lepr:   subq.b  #$01,d3
        btst    d3,d1
        bne     lepr2
        subq.b  #$01,d4
        bne     lepr            ; if no 1 in first nibble
        move.w  #$0002,d0 ;  then signal error
        bsr     error
        jmp     rex

lepr2:  subq.b  #$01,d4         ; decrement first nibble counter
        beq     lepr5
lepr3:  subq.b  #$01,d3
        btst    d3,d1
        bne     lepr4
        move.b  #square,(a0)+
        addq.l  #$01,a0
        subq.b  #$01,d4
        beq     lepr5
        jmp     lepr3

lepr4:  move.b  #square,(a0)+
        addq.l  #01,a0
        move.b  #multpl0,(a0)+
        addq.l  #$01,a0
        subq.b  #$01,d4
        beq     lepr5
        jmp     lepr3

lepr5:  cmp.b   #$00,d3         ; start using expon
        bne     lepr7           ;  command
        jmp     lepr8

lepr6:  move.b  (a1)+,d1
        move.b  d1,d2
        ror.b   #$04,d2
        andi.b  #$0f,d2
        ori.b   #expon,d2
        move.b  d2,(a0)+
        addq.l  #$01,a0

lepr7:  andi.b  #$0f,d1         ; store low nibble
        ori.b   #expon,d1
        move.b  d1,(a0)+
        addq.l  #$01,a0

lepr8:  subq    #$01,d0
        bne     lepr6

        clr     d0
        move.b  #$f4,(a0)+
        addq.l  #$01,a0
        move.b  #$f7,(a0)+
        addq.l  #$01,a0
        move.b  #$f0,(a0)+
        addq.l  #$01,a0

        move.b  #$08,d5
```

```
        move.b  $200001,d0
        add.b   #$01,d0
        sub.b   d0,d5
        beq     lepre
righ:   move.b  #shrigh,(a0)+
        addq.l  #$01,a0
        subq.b  #$01,d5
        bne     righ

lepre:  move.b  #ldou1,(a0)+
        addq.l  #$01,a0
        move.b  #endr,(a0)+
        addq.l  #$01,a0

jo:     move.b  #$00,plend; end key memory access

        move.b  $200002,d1          ; store modulus for further
        move.l  #$200006,a0         ; use
        move.l  #$202328,a1
        move.l  #$a01c00,a2         ; data-mem7 on rsach
lepr9:  move.b  (a0),(a1)+
        move.b  (a0)+,(a2)+
        addq.l  #$01,a2
        subq.b  #01,d1
        bne     lepr9

        move.b  #$82,d1
        sub.b   $200002,d1
lepra:  move.b  #$00,(a2)+
        addq.l  #$01,a2
        subq.b  #$01,d1
        bne     lepra
        jmp     rex

*** Multiply cryptech program ***

*** Load Modulus ***

muld:   andi.b  #$bf,$200000
        move.b  #$f4,(a0)+ ; preset 4
        addq.l  #$01,a0
        move.b  #$f0,(a0)+ ; preset 0
        addq.l  #$01,a0
        move.b  #$fe,(a0)+ ; preset e
        addq.l  #$01,a0
        move.b  #ldin7,(a0)+        ; loadin7
        addq.l  #$01,a0
        move.b  #shrigh,(a0)+
        addq.l  #$01,a0
        move.b  #shrigh,(a0)+
        addq.l  #$01,a0

        move.b  $200001,d4
        cmp.b   #$00,d4
        beq     mul2
mul1:   move.b  #shleft,(a0)+
        addq.l  #$01,a0
        addq.b  #$01,d4
        cmp.b   #$08,d4
        bne     mul1

mul2:   move.b  #initmod,(a0)+      ; initmod
        addq l  #$01,a0

*** load mult A ***

        move.b  #$f4,(a0)+
        addq.l  #$01,a0
        move.b  #$f0,(a0)+
        addq.l  #$01,a0
        move.b  #$fe,(a0)+
        addq.l  #$01,a0
        move.b  #ldin0,(a0)+
        addq.l  #$01,a0
```

```
            move.b    $200001,d5
            cmp.b     #$00,d5
            beq       mone
            cmp.b     #$07,d5
            beq       mul4
            move.b    #$07,d5
            sub.b     $200001,d5
mul3:       move.b    #shleft,(a0)+
            addq.l    #$01,a0
            subq.b    #$01,d5
            bne       mul3
            jmp       mul4
mone:       move.b    #shrigh,(a0)+
            addq.l    #$01,a0
mul4:       move.b    #ldou0,(a0)+
            addq.l    #$01,a0
```

*** load mult B ***

```
            move.b    #$f4,(a0)+
            addq.l    #$01,a0
            move.b    #$f0,(a0)+
            addq.l    #$01,a0
            move.b    #$fe,(a0)+
            addq.l    #$01,a0
            move.b    #ldin6,(a0)+
            addq.l    #$01,a0

            move.b    $200001,d5
            cmp.b     #$00,d5
            beq       mone2
            cmp.b     #$07,d5
            beq       mul6
            move.b    #$07,d5
            sub.b     $200001,d5
mul5:       move.b    #shleft,(a0)+
            addq.l    #$01,a0
            subq.b    #$01,d5
            bne       mul5
            jmp       mul6
mone2:      move.b    #shrigh,(a0)+
            addq.l    #$01,a0
mul6:       move.b    #ldou6,(a0)+
            addq.l    #$01,a0
            move.b    #ldin6,(a0)+
            addq.l    #$01,a0
```

*** calculation ***

```
            clr       d0
            move.b    $200002,d0
            move.b    $200001,d2
            cmp.b     #$00,d2
            bne       mulb
            mulu      #$0008,d0
            jmp       nxmul
mulb:       sub.b     #$01,d0
            mulu      #$0008,d0
            clr       d2
            move.b    $200001,d2
            add.w     d2,d0

nxmul:      addq      #$02,d0
            move.w    d0,d1
            move.w    d0,d2
            ror.w     #$08,d0     ; set the presets
            ori.b     #preset,d0
            move.b    d0,(a0)+
            addq.l    #$01,a0
            ror.w     #$04,d1
            ori.b     #preset,d1
            move.b    d1,(a0)+
            addq.l    #$01,a0
            ori.b     #preset,d2
            move.b    d2,(a0)+
            addq.l    #$01,a0
```

```
                move.b      #multpl0,(a0)+
                addq.l      #$01,a0

*** Loadout the result ***
                clr         d0
                move.b      #$f4,(a0)+
                addq.l      #$01,a0
                move.b      #$f1,(a0)+
                addq.l      #$01,a0
                move.b      #$f0,(a0)+
                addq.l      #$01,a0

                move.b      #$08,d5
                move.b      $200001,d0
                add.b       #$01,d0
                sub.b       d0,d5
                beq         mulo
rmul:           move.b      #shrigh,(a0)+
                addq.l      #$01,a0
                subq.b      #$01,d5
                bne         rmul

mulo:           move.b      #ldou1,(a0)+
                addq.l      #$01,a0
                move.b      #endr,(a0)+
                addq.l      #$01,a0

                jmp         jo

*** Encrypt data ***

rdata:          move.b      $200000,d0
                and.b       #$10,d0
                bne         mulpl
                jmp         rec

mulpl:          clr         d0
                move.w      $200004,d0
                move.l      #$200006,a0
mlp:            move.b      ramre,(a0)+
                subq.w      #$0001,d0
                bne         mlp

                move.l      #$a01800,a2
                move.l      #$200006,a0
                move.b      $200002,d0
mlpr:           move.b      (a0)+,(a2)+
                addq.l      #$01,a2
                subq.b      #$01,d0
                bne         mlpr

                move.w      #$200,d0
                sub.b       $200002,d0
mlpr2:          move.b      #$00,(a2)+
                addq.l      #$01,a2
                subq.w      #$01,d0
                bne         mlpr2
                move.l      #$200006,a1
                move.b      $200002,d0

mlp2:           move.b      $200002,d1      ; modulus length counter
                bsr         rsaed

                andi.b      #$ef,$200000
                clr         d1
                move.b      $200002,d1
                move.w      d1,$200004
                jmp         rex

rec:            clr         d0
                move.w      $200004,d0
                move.l      #$200006,a0
lp:             move.b      ramre,(a0)+
                subq.w      #$0001,d0
                bne         lp
```

```
        move.l  #$200006,a0
        move.l  #$200006,a1
        move.w  $200004,d0

rdlp2:  move.b  $200002,d1      ; modulus length counter

        bsr     rsaed

        clr     d4
        move.b  $200002,d4
        sub.w   d4,d0
        bhi     rdlp2
        jmp     rex

rex:    movem.l (a7)+,d0-d7/a0-a6
        rtr

*********************************************************
***     RSA ENCRYPTION/DECRYPTION               ***
***                                             ***
***     a0 = read counter address               ***
***     a1 = write counter address              ***
***     d0 = message length remaining           ***

rsaed:  move.w  sr,-(a7)

        move.l  #$a00000,a2
rsa2:   move.b  (a0)+,(a2)+
        addq.l  #$01,a2
        subq.b  #$01,d1
        bne     rsa2
        clr     d2

        move.b  #$82,d2
        sub.b   $200002,d2
rsa9:   move.b  #$00,(a2)+
        addq.l  #$01,a2
        subq.b  #$01,d2
        bne     rsa9
        move.b  #$00,start

        move.b  #$60,latch
rsa3:   move.b  regre,d2
        and.b   #$02,d2
        beq     rsa3

        move.b  $200002,d1
        move.l  #$a00400,a2

        move.b  $200001,d2
        cmp.b   #$00,d2
        beq     extra
        move.b  #$00,$202580
        move.l  #$202581,a3
        jmp     rsa4
extra:  move.l  #$202580,a3

rsa4:   move.b  (a2)+,(a3)+
        addq.l  #$01,a2
        subq.b  #$01,d1
        bge     rsa4

        clr     d1
        clr     d3
        move.b  $200002,d2
        move.l  #$202580,a3     ; most sig byte of result
        move.l  #$202134,a4     ; most sig byte of modulus
        move.b  (a3)+,d1
        and.b   #$01,d1         ; Test extra result byte
        bne     large           ; jump to routine which deals with
                                ; a greater result than modulus
```

```
rsa5:     move.b    (a4)+,d1
          move.b    (a3)+,d3
          cmp.b     d1,d3
          bhi       large
          cmp.b     d1,d3
          bne       less
          subq.b    #$01,d2
          bne       rsa5
          jmp       large

less:     move.l    #$202581,a3       ; routine for result less than
          move.b    $200002,d1        ;  modulus
lelp1:    move.b    (a3)+,(a1)+
          subq.b    #$01,d1
          bne       lelp1
          jmp       rsax

large:    move.l    #$202581,d1       ; routine for result greater
          clr       d3
          move.b    $200002,d3        ;  than modulus
          add.l     d3,d1
          move.l    #$202328,d2
          add.l     d3,d2
          move.l    d1,a2
          move.l    d2,a3
          move.b    $200002,d1
          clr       d2
          move.l    a1,a4
          clr       d6
          move.b    $200002,d6
          add.l     d6,a4
          add.l     d6,a1
lalp2:    add.b     -(a3),d2
          add.b     -(a2),d2
          bcs       lalp
          move.b    d2,-(a4)
          clr       d2
          jmp       lalp1
lalp:     move.b    d2,-(a4)
          move.b    #$01,d2
lalp1:    subq.b    #$01,d1
          bne       lalp2

rsax:     rtr

error:    move.w    sr,-(a7)
          movem.l   (a7)+,d1-d7/a0-a6

          move.b    #$0f,$200001
          move.w    #$00000,$200006
          move.w    d0,$200004

          movem.l   (a7)+,d1-d7/a0-a6
          rtr
end
```

Appendix C

Software

Encdec Header Files	**296**
Genetic Algorithm Header Files	**302**
Neural Network Header Files	**307**
Software RSA Functions	**311**
Software DES Functions	**318**
The Four Basic Arithmetic Algorithms	**320**

ENCDEC Header Files

```
/*************************************************************

Name:           Joseph Morrissey
File:           Crypt.h
Language: C

This header file defines the functions used for direct access to
the ENCDEC board.

*************************************************************/

#ifndef     CRYPT_H
#define     CRYPT_H


/* Include Files */
#include    <sys/types.h>
#include    <stdio.h>
#include    <fcntl.h>

/* Function */
#define     C_RSA               0x80
#define     C_DES               0x40
#define     C_KEYLD             0x20
#define     C_DECRYPT           0x08
#define     C_MULTLOAD          0x40
#define     C_MULT              0x10
#define     C_TEST              0x10
#define     C_POWER             0x04
#define     C_BEGIN             0x02
#define     C_END               0x01

/* Utilities */
#define     C_PECB    0x10
#define     C_ECB     0x08
#define     C_CBC     0x04
#define     C_OFB     0x02
#define     C_CFB     0x01

#define     C_BLANK             0x00
#define     C_MAX               4000
#define     C_HEAD              6

/* Definition of the generic PB-packet */
typedef struct  {
            u_char      function;
            u_char      er1;
            u_char      er2;
            u_char      er3;
            u_short     message_length;

            u_char      message[C_MAX];
}BOARD_DATA;

            /* Print PB-packet to specified file */
            void        crypt_print(BOARD_DATA *bd, FILE *out);

            /* Read and write PB-packet to board */
            void        send_to_board(BOARD_DATA *bd, FILE *board_fd);
            void        read_from_board(BOARD_DATA *bd, FILE *board_fd);

            /* segment a string for storage within the message
            segment of the PB-packet */
            int         mess_write(BOARD_DATA *bd, u_char *ptr, u_int length, u_int offset);
            int         mess_read(BOARD_DATA *bd, u_char *ptr, u_int length, u_int offset);

#endif
```

```
/*************************************************************

Name:           Joseph Morrissey
File:           harded.h
Language:       C

This file contains all the declarations for the manipulation of files
and data for DES or RSA calculations using the ENCDEC board.

*************************************************************/

#ifndef     H_HARDED
#define     H_HARDED

/* Include files */
#include    <stdio.h>
#include    <sys/types.h>
#include    "long.h"
#include    "crypt.h"

/* Definitions */
#define     P_TEST              10              /*Number of tests for primality*/
#define     BUF_LENGTH          8000

/* CRYPT_FLAG */
#define     H_ENCRYPT           1
#define     H_DECRYPT           0

/* MODE_FLAG */
#define     H_SINGLE_CRYP       1
#define     H_DOUBLE_CRYP       0

/* PROFILE_FLAG */
#define     H_PROFILE1          0x04
#define     H_PROFILE2          0x0c
#define     H_PROFILE3          0x1c
#define     H_PROFILE4          0x3c

/* FRAME HEADER DEFAULT SIZE */
#define     H_P1        2
#define     H_P2        3
#define     H_P3        7
#define     H_P4        11

#define     H_BEGIN 1
#define     H_END               2
#define     H_MIDDLE            3
#define     H_SINGLE            4

/* FHEADER CONTENTS */
#define     HF_INFO_OPT             0x20
#define     HF_INFO_UMN             0x10
#define     HF_INFO_TIME            0x08
#define     HF_INFO_LENGTH          0x04
#define     HF_INFO_STATUS          0x02
#define     HF_INFO_COUNTER         0x01
#define     HF_DEF                  0x00

/* INFO&STATUS FLAG */
#define     HST_BEG0x10
#define     HST_END0x20
#define     HST_SIN  0x30

#define     H_ERROR             0
#define     H_COMPLETE          1
#define     H_CONTINUE          2

/* BOOLEAN VALUES */
#define     H_TRUE   1
#define     H_FALSE 0

/* ERRORS */
#define     HE_OTHER            -1
#define     HE_LENGTH           -2
#define     HE_NIBBLE           -4
#define     HE_FILE_OPEN        -6
```

```
#define     HE_PARAM        -15
#define     HE_BOARD        -16
#define     HE_FRAME        -17

/* DES modes */
#define     HE_ECB          0x01
#define     HE_CBC          0x02
#define     HE_OFB          0x04
#define     HE_CFB          0x08
#define     HD_ECB          0x11
#define     HD_CBC          0x12
#define     HD_OFB          0x14
#define     HD_CFB          0x18

#define     BOARD_NAME      "/dev/encdec0"

/* RSA key structure */
typedef struct {
            LONG_TYPE       exp;
            LONG_TYPE       mod;
}H_RSA_KEY;

/* DES key structure */
typedef     struct {
            LONG_TYPE       key;
            LONG_TYPE       iv;
}H_DES_KEY;

/* Data type structure */
typedef struct{
            char        *ptd;
            u_int       dlg;
}H_DATA_TYPE;

/* Frame header structure */
typedef struct {
            u_char      F_header[2];
            u_char      F_info_opt;
            u_char      F_info_umn[4];
            u_char      F_info_time[4];
            u_char      F_info_length;
            u_char      F_info_status;
            u_char      F_info_counter;
}HF_HEAD;

/*****************************************************************************
            Key Generation Routines
*****************************************************************************/
int         ED_pz_verify(LONG_TYPE *l, u_int nmbtest);
int         ED_pz_gen(LONG_TYPE *l, u_int nmbtest, char nibble);
int         ED_pz_pair_gen(LONG_TYPE *pr1, LONG_TYPE *pr2, u_int length, char ch_mod);
int         ED_pz_key_step(LONG_TYPE *k1, LONG_TYPE *k2, LONG_TYPE *nmb1,
                           LONG_TYPE *nmb2, char expon);
int         ED_key_load(LONG_TYPE       *exp, LONG_TYPE *mod);
int         ED_multiply_load(LONG_TYPE *l);
int         ED_multiply(LONG_TYPE *a, LONG_TYPE *b, LONG_TYPE *mod, LONG_TYPE *res);
int         ED_GCD(LONG_TYPE *x, LONG_TYPE *y, LONG_TYPE *res);

/*****************************************************************************
            Key Storage Routines
*****************************************************************************/
int         ED_keyfile_trafo(char *filename, H_RSA_KEY *key);
int         ED_keystruct_trafo(char *filename, H_RSA_KEY *key, char overwrite);

/*****************************************************************************
            Encryption/Decryption Routines
*****************************************************************************/
int         ED_file_crypt(char crypt_flag, char mode_flag, char stuff_flag, H_RSA_KEY *key1,
                    H_RSA_KEY *key2, char profile1, char profile2, char *datfile_in,
                    char *datfile_out, char overwrite);
int         ED_data_crypt(char crypt_flag, char mode_flag, char stuff_flag,
                    H_RSA_KEY *key1, H_RSA_KEY *key2, char profile1, char profile2,
                    H_DATA_TYPE *data_in, H_DATA_TYPE *data_out);
int         ED_file_decrypt(FILE *data_in, FILE *data_out, H_RSA_KEY *key,
                            char profile);
int         ED_data_decrypt(H_DATA_TYPE *data_in, H_DATA_TYPE *data_out, H_RSA_KEY *key,
```

```
                    char profile);
int         ED_file_encrypt(FILE *data_in, FILE *data_out, H_RSA_KEY *key,
                    char profile);
int         ED_data_encrypt(H_DATA_TYPE *data_in, H_DATA_TYPE *data_out, H_RSA_KEY *key,
                    char profile);

/*****************************************************************************
        Framing Routines
*****************************************************************************/
int         ED_unpeel_frametype2(char *frame, char *data_buf, u_int frame_length,
                        u_int *data_length, char first_frame);
int         ED_unpeel_frametype1(char *frame, char *data_buf, u_int frame_length,
                        u_int *data_length);
int         ED_f_construct_frametype2(FILE *data_in, char *frame_flag, char *frame,
                        u_int frame_length, char head, char first_frame);
int         ED_d_construct_frametype2(H_DATA_TYPE *data_in, H_DATA_TYPE *data_out,
                        u_int frame_length, char head);
int         ED_f_construct_frametype1(FILE *data_in, char *frame_flag, char *frame,
                        u_int frame_length, char head);
int         ED_d_construct_frametype1(H_DATA_TYPE *data_in, H_DATA_TYPE *data_out,
                        u_int frame_length, char head);
int         ED_unpeel_frame(char *frame, char *data_buf, u_int *data_length,
                        u_int frame_length, HF_HEAD *fh);
int         ED_f_create_frame(FILE *dat_in, char *frame_flag, char *frame,
                        u_int frame_length, HF_HEAD *fh, u_int psize);
int         ED_d_create_frame(H_DATA_TYPE *data_in, H_DATA_TYPE *data_out, u_int frame_counter,
                        u_int data_counter, u_int frame_length, HF_HEAD *fh,
                        u_int psize, char *frame_flag, u_int nbytes);
int         ED_remove_frameheader(HF_HEAD *fh, char *frame, char *data_buf, u_int *data_length,
                            u_int frame_length);


/*****************************************************************************
        DES Functions
*****************************************************************************/
int         ED_des_key_write_to_file(H_DES_KEY *des, char *filename, int overwrite);
int         ED_des_key_read_from_file(H_DES_KEY *des, char *filename);

int         ED_des_sendkey(H_DES_KEY *des, char position);
int         ED_generate_des_key(H_DES_KEY *des);
int         ED_des_file(int crypt_mode, int stuff_flag, H_DES_KEY *des, char *datfile_in,
                            char *datafile_out, int overwrite);

#endif      H_ENCDEC
```

```
/*************************************************************

Name:     Joseph Morrissey
File:     long.h
language: C

This file contains the function definitions for long integer
manipulation.

*************************************************************/

#ifndef   LONG_H
#define   LONG_H

/* Include Files */
#include  <stdio.h>
#include  <sys/types.h>

/* Long type structure definition */
typedef struct{
          u_int     bln;
          u_char    negative;
          u_char    *num;
}LONG_TYPE;

/* LONG_TYPE management */
void      default_LT(LONG_TYPE *ltp);
void      copy_LT(LONG_TYPE *ltp1, LONG_TYPE *ltp2);
void      free_LT_mem(LONG_TYPE *ltp);
void      print_LT(LONG_TYPE *ltp, FILE *ptr);

/* Type conversion functions */
void      string_to_LT(LONG_TYPE *ltp, u_char *ptr, u_int bitlen, u_char lock);
void      LT_to_string(LONG_TYPE *ltp, u_char *string, u_int str_length, char fill);
void      uint_to_LT(LONG_TYPE *ltp, u_int i);
void      int_to_LT(LONG_TYPE *ltp, int i);
void      uchar_to_LT(LONG_TYPE *ltp, u_char c);
void      char_to_LT(LONG_TYPE *ltp, char c);

/* Basic mathematical functions */
int       add_LT(LONG_TYPE *ltp1, LONG_TYPE *ltp2, LONG_TYPE *res);
int       subtract_LT(LONG_TYPE *ltp1,LONG_TYPE *ltp2, LONG_TYPE *res);
int       multiply_LT(LONG_TYPE *ltp1, LONG_TYPE *ltp2, LONG_TYPE *res);
int       divide_LT(LONG_TYPE *ltp1, LONG_TYPE *ltp2, LONG_TYPE *res);
int       modulus_LT(LONG_TYPE *ltp1, LONG_TYPE *ltp2, LONG_TYPE *res);
int       inc_LT(LONG_TYPE *ltp);
void      gen_long_number(LONG_TYPE *ltp, u_int ln);

/* Bit manipulation */
int       not_LT(LONG_TYPE *ltp, LONG_TYPE *res);
int       or_LT(LONG_TYPE *ltp1, LONG_TYPE *ltp2, LONG_TYPE *res);
int       and_LT(LONG_TYPE *ltp1, LONG_TYPE *ltp2, LONG_TYPE *res);
int       rshift_LT(LONG_TYPE *ltp, int l, LONG_TYPE *res);
int       lshift_LT(LONG_TYPE *ltp, int l, LONG_TYPE *res);

/* Logical tests */
u_int     greater_than_LT(LONG_TYPE *ltp1, LONG_TYPE *ltp2);
u_int     less_than_LT(LONG_TYPE *ltp1, LONG_TYPE *ltp2);
u_int     equal_LT(LONG_TYPE *ltp1, LONG_TYPE *ltp2);
u_int     not_equal_LT(LONG_TYPE *ltp1, LONG_TYPE *ltp2);

/* Advanced mathematical functions */
int       mod_power_LT(LONG_TYPE *ltp, LONG_TYPE *l, LONG_TYPE *mod, LONG_TYPE *res);
int       GCD(LONG_TYPE *x, LONG_TYPE *y, LONG_TYPE *res);
int       INVERSE(LONG_TYPE *x, LONG_TYPE *n, LONG_TYPE *res);
void      update(LONG_TYPE *y,LONG_TYPE *a, LONG_TYPE *b);

/* String manipulation */
u_int     in_bytes(u_int bln);
u_int     find_bitlen(u_char *ptr, u_int length);
u_char    strind(u_char *str, u_int pos);

#endif
```

Genetic Algorithm Header files

```
//*************************************************
//
//          Name:               Joseph Morrissey
//          File:               organism.h
//          Language:           C++
//
//          This header file contains declarations for
//          the class organism that describes a genetic
//          problem solver.
//
//*************************************************

#ifndef     H_ORGANISM
#define     H_ORGANISM

// Include Files
#include              <fstream.h>
#include              "c:\tc\gen_cpp\chr.h"

typedef     unsigned int          u_int;
typedef     unsigned char         u_char;

class ORGANISM {
public:
            CHROMOSOME      *genotype;
            u_int                       no_of_chromosomes;

// CONSTRUCTORS
            inline      ORGANISM(void);
                        ORGANISM(ORGANISM &o);
                        ORGANISM(CHROMOSOME *g, u_int no_of_chr);
// DESTRUCTOR
            ~ORGANISM(void);
// GET FUNCTIONS
            inline u_int          GetNoOfChromosomes(void);
            inline u_int          GetNoOfGenes(u_int chr);
            inline double         GetPmutation(u_int chr, u_int gen);
            inline double         GetPcrossover(u_int chr, u_int gen);
            inline double         GetFitness(u_int chr, u_int gen);
            inline BINARY         GetGene(u_int chr, u_int gen);
// SET FUNCTIONS
            inline void           SetPmutation(u_int chr, u_int gen, double pm);
            inline void           SetPcrossover(u_int chr, u_int gen, double Pc);
            inline void           SetFitness(u_int chr, u_int gen, double fit);
            inline void           SetGene(u_int chr, u_int gen, BINARY &bin);
            void                        SetRandomGeneValues(void);
// PRINT FUNCTION
            void        print(ostream& out = cout);
// OPERATOR
            ORGANISM&           operator=(ORGANISM &o);
// FRIEND FUNCTION
            friend      ostream&  operator << (ostream& o, ORGANISM& org);
            friend      istream&  operator >> (istream& i, ORGANISM& org);
};

#endif
```

```
//*****************************************
//
//          Name:     Joseph Morrissey
//          Files:    chromosome
//          Language: C++
//
//          This header file contains declarations
//          a genetic organisms chromosomes.
//
//*****************************************

#ifndef   H_CHR
#define   H_CHR

// Include Files
#include  "c:\tc\gen_cpp\binary.h"

typedef   unsigned int          u_int;
typedef   unsigned char u_char;

class     CHROMOSOME {
          BINARY  *gene;
          u_int   no_of_genes;
          double  *pmutation;
          double  *perossover;
          double  *fitness;
public:
// CONSTRUCTORS
          inline  CHROMOSOME(void);
                  CHROMOSOME(CHROMOSOME &c);
                  CHROMOSOME(BINARY *b, u_int ng, double *pm, double *pc, double *fit);
// DESTRUCTOR
          ~CHROMOSOME(void);
// GET FUNCTIONS
          inline u_int      GetNoOfGenes(void);
          inline double     GetPmutation(u_int gen);
          inline double     GetPerossover(u_int gen);
          inline double     GetFitness(u_int gen);
          inline BINARY     GetGene(u_int gen);
// SET FUNCTIONS
          inline void SetPmutation(u_int gen, double pm);
          inline void SetPerossover(u_int gen, double pc);
          inline void SetFitness(u_int gen, double fit);
          inline void SetGene(u_int gen, BINARY &bin);
          inline void SetRandomGeneValue(u_int gn);
// PRINT FUNCTION
          void      print(ostream& out = cout);
// OPERATOR
          CHROMOSOME& operator=(CHROMOSOME &c);
// GENETIC FUNCTIONS
          u_int     AlleleMutation(void);

// FRIEND FUNCTION
          friend    ostream&  operator << (ostream& o, CHROMOSOME& c);
          friend    istream&  operator >> (istream& i, CHROMOSOME& c);
};
#endif
```

```
//*****************************************************
//
//          Name:     Joseph Morrissey
//          File:     binary.h
//          Language: C++
//
//          This File contains the declarations for a class
//          of long binary bit strings
//
//*****************************************************

#ifndef   H_BINARY
#define   H_BINARY

#include  <stdio.h>
#include  <iostream.h>

typedef   unsigned char       u_char;
typedef   unsigned int        u_int;

class     BINARY {
public:
          u_char    *bits;
          u_int               no_bits;
// Constructors
          inline    BINARY(void);
          BINARY(BINARY &bit);
          BINARY(u_char     *bit, u_int no_bits);
// Destructors
          inline    ~BINARY(void);
// Bit Manipulation Functions
          void      SetBit(u_int set, u_int bit_no);
          u_int     GetBit(u_int bit_no);
          void      RandomGenerator(u_int ln);
// Operator Overloads
          BINARY&             operator=(BINARY& b);
// Other Functions
          void      print(ostream& out = cout);
//Friendly Functions
          friend    istream&  operator >> (istream& i, BINARY& b);
          friend    ostream&  operator << (ostream& o, BINARY& b);

// Inline Functions
          inline int  GetNoBits(void);
};

u_int     GetAlignmentConstant(u_int bitlength);

#endif


//*****************************************
//
//          Name:     Joseph Morrissey
//          File:     random.h
//          Language: C++
//
//          This header file contains the declarations
//          for the random number generation function.
//
//*****************************************
#ifndef   H_RANDOM
#define   H_RANDOM

#define   RANMAX 20000

void              init_rand(void);
double            random_double(void);
unsigned int      flip(double probability);
unsigned int      rnd(unsigned int lower, unsigned int upper);

#endif
```

```
//***********************************************
//
//          Name:     Joseph Morrissey
//          File:     scale.h
//          Language: C++
//
//          This header file contains the declarations
//          necessary for scaling functions
//
//***********************************************

#ifndef   H_SCALE
#define   H_SCALE

// Include files
#include  "c:\tc\gen_cpp\org.h"

// Scaling constants
const     double    CMULT=2.0;
const     double    FACTOR1=1.0;
const     double    FACTOR2=0.0;

struct  fitness_stat {
          double    max_fitness, min_fitness, avg_fitness, sumfitness;
};

// Scaling functions
void                LinearPrescale(fitness_stat &fstat, double &factor1, double &factor2);
double              LinearScale(double factor1, double factor2, double fitness);
fitness_stat        LinearScalePopulation(ORGANISM *org_ptr, unsigned int population,
                          unsigned int chromosome_no, unsigned int gene_no);
void                FillFitnessStat(ORGANISM *org_ptr, unsigned int population,
                          unsigned int chr_no, unsigned int gene_no, fitness_stat& fstat);

#endif



//*******************************************
//
//          Name:     Joseph Morrissey
//          File:     utility.h
//          Language: C++
//
//          This header file contains miscellaneous
//          declarations for useful functions
//
//*******************************************

#ifndef   H_UTILITY
#define   H_UTILITY

// Include files
#include  <stdio.h>
#include  <fstream.h>

// Typedef declarations
typedef   unsigned int          u_int;
typedef   unsigned char         u_char;

// Functions
          u_int     in_bytes(u_int bln);
          u_int     find_bitlen(u_char *ptr, u_int length);
          u_char    strind(u_char str, u_int pos);
          int       upchar(char &ptr);
          void      to_upper(char *str_ptr);
          void      print_binary(ofstream &outfile, unsigned int number);

#endif
```

Neural Network Header Files

```
//*************************************************
//
//          Name:     Joseph Morrissey
//          File:     net.h
//          Language: C++
//
//          This header file contains the declarations
//          for net.cpp.  Which contains the functions
//          for using the node class to generate neural
//          networks
//
//*************************************************

#ifndef   H_NET
#define   H_NET

#include  <iostream.h>
#include  "c:\tc\neu_cpp\node.h"

typedef   unsigned int          u_int;
typedef   unsigned char         u_char;

class net {
private:
          node      *node_list;
          u_int     no_nodes;
          u_int     no_stimulators;
          u_int     no_thinkers;
          u_int     no_responders;
public:
//Constructors
          inline net(void);
          net(net &n);
          net(node *list, u_int no, u_int no_st, u_int no_th, u_int no_res);
//Destructor
          ~net(void);
//Useful creation functions
          void      ConnectionList(istream &i);
          void      ConnectionList(int nodes[3], int **list, int *no_ip);
          net       &operator=(net &n);
//Print Functions
          void      print(ostream& out = cout);
//Neural Functions
          inline u_int          GetNoResponders(void);
          inline u_int          GetNoThinkers(void);
          inline u_int          GetNoStimulators(void);
          void                  SetRandom(void);
          void                  UpdatePathConnections(void);
          void                  CalculateNetOutput(void);
          void                  SetStimulus(double *stim);
          void                  GetOutput(double *response);
          virtual void          learn(double *expected_output);
//Other Functions
          friend    ostream&    operator << (ostream& s, net& n);
          friend    istream&    operator >> (istream& i, net& n);
};

#endif
```

```
//*****************************************
//
//          Name:     Joseph Morrissey
//          File:     node.h
//          Language: C++
//
//          This header file contains the declarations
//          for node.cpp.  Which make up the node
//          functionality for neural networks.
//
//*****************************************

#ifndef   H_NODE
#define   H_NODE

// Include files
#include  "c:\tc\neu_cpp\path.h"

typedef   unsigned int        u_int;
typedef   unsigned char       u_char;

class node {
public:
          u_int               node_id;
          double              threshold;
          double              output;
          path                *input;
          unsigned int        no_inputs;
//CONSTRUCTORS
          inline node(void);
          node(node &n);
          node(u_int id, double t, path *i, u_int no_of_inputs);
//DESTRUCTOR
          ~node(void);
//PRINT FUNCTIONS
          void        print(ostream& out = cout);
//OPERATOR FUNCTIONS
          node        &operator=(node &n);
//NEURAL FUNCTIONS
          void                SetThreshold(double value = 0.0, u_int set=0);
          void                UpdateInputs(node *node_list);
          double              SumInputs(void);
          void                NodeOutput(void);
          virtual double      ThresholdFunction(double num);
};

#include  <iostream.h
ostream&  operator << (ostream& s, node& n);
istream&  operator >> (istream& i, node& n);

#endif
```

```
//*********************************************
//
//          Name:     Joseph Morrissey
//          File:     path.h
//          Language: C++
//
//          This header file contains declarations for
//          path.cpp.  Which provides the information
//          for connecting nodes.
//
//*********************************************

#ifndef   H_PATH
#define   H_PATH

//Include files
#include  <fstream.h>
#include  "c:\tc\neu_cpp\node.h"

typedef   unsigned int          u_int;
typedef   unsigned char         u_char;

class     path {
public:
          u_int     node_id;
          double    *ip;
          double    weight;
          double    var1;
//CONSTRUCTORS
          inline    path(void);
          inline    path(path &p);
          inline    path(u_int node_id, double w);
//DESTRUCTORS
          ~path(void);
//PRINT FUNCTIONS
          void      print(ostream& out = cout);
//GET FUNCTIONS
          inline    u_int       get_root(void) {return (node_id);}
//OPERATORS
          path& operator=(path &p);
//NEURAL FUNCTIONS
          void                  SetWeight(double value=0.0, u_int set=0);
          inline    void        ConnectPath(double* ptr);
          inline                double PathOutput(void);
};

#include  < iostream.h >
ostream&  operator << (ostream& s, path& p);
istream&  operator > (istream& i, path& p);

#endif
```

Software RSA Functions

```
//*********************************************
//
//          Name:     Joseph Morrissey
//          File:     encdec.h
//          Language: C
//
// This header file contains the declarations
//          for encdec.c. which contains the functions
//          that provide the same functionality as the
//          encdec board provides
//
//*********************************************


#ifndef   H_ENCDEC
#define   H_ENCDEC

#include  <stdio.h>
#include  "long.h"

#define   EC_P_TEST                10              /*Number of tests for primality*/
#define   EC_BUF_LENGTH  8000

/*CRYPT_FLAG*/
#define   EC_ENCRYPT       1
#define   EC_DECRYPT       0

/*MODE_FLAG*/
#define   EC_SINGLE_CRYP  1
#define   EC_DOUBLE_CRYP 0

/*PROFILE_FLAG*/
#define   EC_PROFILE1      0x04
#define   EC_PROFILE2      0x0c
#define   EC_PROFILE3      0x1c
#define   EC_PROFILE4      0x3c

/*FRAME HEADER DEFAULT SIZE*/
#define   EC_P1    2
#define   EC_P2    3
#define   EC_P3    7
#define   EC_P4    11

#define   EC_BEGIN         1
#define   EC_END           2
#define   EC_MIDDLE        3
#define   EC_SINGLE        4

/*FHEADER CONTENTS*/
#define   ECF_INFO_OPT             0x20
#define   ECF_INFO_UMN             0x10
#define   ECF_INFO_TIME            0x08
#define   ECF_INFO_LENGTH          0x04
#define   ECF_INFO_STATUS          0x02
#define   ECF_INFO_COUNTER         0x01

/*INFO&STATUS FLAG*/
#define   ECST_BEG         0x10
#define   ECST_END         0x20
#define   ECST_SIN         0x30

#define   EC_ERROR         0
#define   EC_COMPLETE      1
#define   EC_CONTINUE      2

/*BOOLEAN VALUES*/
#define   EC_TRUE          1
#define   EC_FALSE         0

/*ERRORS*/
#define   ECE_OTHER        -1
#define   ECE_LENGTH       -2
#define   ECE_NIBBLE       -4
#define   ECE_FILE_OPEN    -6
#define   ECE_PARAM        -15
#define   ECE_FRAME        -16
```

```
typedef struct {
        LONG_TYPE       exp;
        LONG_TYPE       mod;
}EC_RSA_KEY;

typedef struct{
        char    *ptd;
        u_int   dlg;
}EC_DATA_TYPE;

typedef struct {
        u_char  F_header[2];
        u_char  F_info_opt;
        u_char  F_info_umm[4];
        u_char  F_info_time[4];
        u_char  F_info_length;
        u_char  F_info_status;
        u_char  F_info_counter;
}ECF_HEAD;

int     EC_pz_verify(LONG_TYPE *l, u_int nmbtest);
int     EC_pz_gen(LONG_TYPE *l, u_int nmbtest, char nibble);
int     EC_pz_pair_gen(LONG_TYPE *pr1, LONG_TYPE *pr2, u_int length, char ch_mod);
int     EC_pz_key_step(LONG_TYPE *k1, LONG_TYPE *k2, LONG_TYPE *nmb1,
                        LONG_TYPE *nmb2, char expon);
int     EC_keyfile_trafo(char *filename, EC_RSA_KEY *key);
int     EC_keystruct_trafo(char *filename, EC_RSA_KEY *key, char overwrite);
int     EC_file_crypt(char crypt_flag, char mode_flag, char stuff_flag, EC_RSA_KEY *key1,
                        EC_RSA_KEY *key2, char profile1, char profile2, char *datfile_in,
                        char *datfile_out, char overwrite);
int     EC_data_crypt(char crypt_flag, char mode_flag, char stuff_flag,
                        EC_RSA_KEY *key1, EC_RSA_KEY *key2, char profile1, char profile2,
                        EC_DATA_TYPE *data_in, EC_DATA_TYPE *data_out);
int     EC_file_decrypt(FILE *data_in, FILE *data_out, EC_RSA_KEY *key,
                        char profile);
int     EC_data_decrypt(EC_DATA_TYPE *data_in, EC_DATA_TYPE *data_out, EC_RSA_KEY *key,
                        char profile);
int     EC_file_encrypt(FILE *data_in, FILE *data_out, EC_RSA_KEY *key,
                        char profile);
int     EC_data_encrypt(EC_DATA_TYPE *data_in, EC_DATA_TYPE *data_out, EC_RSA_KEY *key,
                        char profile);
int     EC_unpeel_frametype2(char *frame, char *data_buf, u_int frame_length,
                        u_int *data_length, char first_frame);
int     EC_unpeel_frametype1(char *frame, char *data_buf, u_int frame_length,
                        u_int *data_length);
int     EC_f_construct_frametype2(FILE *data_in, char *frame_flag, char *frame,
                        u_int frame_length, char head, char first_frame);
int     EC_d_construct_frametype2(EC_DATA_TYPE *data_in, EC_DATA_TYPE *data_out,
                        u_int frame_length, char head);
int     EC_f_construct_frametype1(FILE *data_in, char *frame_flag, char *frame,
                        u_int frame_length, char head);
int     EC_d_construct_frametype1(EC_DATA_TYPE *data_in, EC_DATA_TYPE *data_out,
                        u_int frame_length, char head);
int     EC_unpeel_frame(char *frame, char *data_buf, u_int *data_length,
                        u_int frame_length, ECF_HEAD *fh);
int     EC_f_create_frame(FILE *dat_in, char *frame_flag, char *frame,
                        u_int frame_length, ECF_HEAD *fh, u_int psize);
int     EC_d_create_frame(EC_DATA_TYPE *data_in, EC_DATA_TYPE *data_out, u_int frame_counter,
                        u_int data_counter, u_int frame_length, ECF_HEAD *fh,
                        u_int psize, char *frame_flag, u_int nbytes);
int     EC_remove_frameheader(ECF_HEAD *fh, char *frame, char *data_buf, u_int *data_length,
                        u_int frame_length);
int     EC_fill_frame(ECF_HEAD *fh, char *frame, char *data_buf, u_int frame_length);

#endif  H_ENCDEC
```

```
//*********************************************
//
//          Name:     Joseph Morrissey
//          File:     encdec.h
//          Language: C++
//
// This header file contains the declarations
//          for encdec.cpp.  which contains the functions
//          that provide the same functionality as the
//          encdec board provides
//
//*********************************************

#ifndef   H_ENCDEC
#define   H_ENCDEC

#include  <fstream.h>
#include  "Long.h"

typedef   unsigned int        u_int;
typedef   unsigned char       u_char;

#define   P_TEST   10                 //Number of tests for primality
#define   BUF_LENGTH      8000

//CRYPT_FLAG
#define   ENCRYPT         1
#define   DECRYPT         0

//MODE_FLAG
#define   SINGLE_CRYP     1
#define   DOUBLE_CRYP     0

//PROFILE_FLAG
#define   PROFILE1        0x04
#define   PROFILE2        0x0c
#define   PROFILE3        0x1c
#define   PROFILE4        0x3c

//FRAME HEADER DEFAULT SIZE
#define   P1        2
#define   P2        3
#define   P3        7
#define   P4        11

#define   BEGIN     1
#define   END       2
#define   MIDDLE    3
#define   SINGLE    4

//FHEADER CONTENTS
#define   F_INFO_OPT        0x20
#define   F_INFO_UMN        0x10
#define   F_INFO_TIME       0x08
#define   F_INFO_LENGTH     0x04
#define   F_INFO_STATUS     0x02
#define   F_INFO_COUNTER    0x01

//INFO&STATUS FLAG
#define   ST_BEG    0x10
#define   ST_END    0x20
#define   ST_SIN    0x30

#define   ERROR             0
#define   COMPLETE          1
#define   CONTINUE          2

//BOOLEAN VALUES
#define   TRUE      1
#define   FALSE     0

//ERRORS
#define   E_OTHER           -1
#define   E_LENGTH          -2
#define   E_NIBBLE          -4
#define   E_FILE_OPEN       -6
```

```
#define   E_PARAM          -15
#define   E_FRAME          -16

struct    RSA_KEY {
          LONG_TYPE        exp;
          LONG_TYPE        mod;
};

struct    DATA_TYPE{
          char       *ptd;
          u_int      dlg;
};

struct    F_HEAD {
          u_char     F_header[2];
          u_char     F_info_opt;
          u_char     F_info_umm[4];
          u_char     F_info_time[4];
          u_char     F_info_length;
          u_char     F_info_status;
          u_char     F_info_counter;
          };

// Key Generation Routines
int       EC_pz_verify(LONG_TYPE &l, u_int nmbtest);
int       EC_pz_gen(LONG_TYPE &l, u_int nmbtest, char nibble);
int       EC_pz_pair_gen(LONG_TYPE &pr1, LONG_TYPE &pr2, u_int length, char ch_mod);
int       EC_pz_key_step(LONG_TYPE &k1, LONG_TYPE &k2, LONG_TYPE &nmb1,
                    LONG_TYPE &nmb2, char expon);

// Key Storage Routines
int       EC_keyfile_trafo(char *filename, RSA_KEY &key);
int       EC_keystruct_trafo(char *filename, RSA_KEY &key, char overwrite);

// Encryption/Decryption Routines
int       EC_file_crypt(char crypt_flag, char mode_flag, char stuff_flag, RSA_KEY &key1,
                    RSA_KEY &key2, char profile1, char profile2, char *datfile_in,
                    char *datfile_out, char overwrite);
int       EC_data_crypt(char crypt_flag, char mode_flag, char stuff_flag,
                    RSA_KEY &key1, RSA_KEY &key2, char profile1, char profile2,
                    DATA_TYPE &data_in, DATA_TYPE &data_out);
int       EC_file_decrypt(fstream &data_in, fstream &data_out, RSA_KEY &key,
                    char profile);
int       EC_data_decrypt(DATA_TYPE &data_in, DATA_TYPE &data_out, RSA_KEY &key,
                    char profile);
int       EC_file_encrypt(fstream &data_in, fstream &data_out, RSA_KEY &key,
                    char profile);
int       EC_data_encrypt(DATA_TYPE &data_in, DATA_TYPE &data_out, RSA_KEY &key,
                    char profile);

// Framing Routines
int       EC_unpeel_frametype2(char *frame, char *data_buf, u_int frame_length,
                    u_int *data_length, char first_frame);
int       EC_unpeel_frametype1(char *frame, char *data_buf, u_int frame_length,
                    u_int *data_length);
int       EC_construct_frametype2(fstream &data_in, char *frame_flag, char *frame,
                    u_int frame_length, char head);
int       EC_construct_frametype2(DATA_TYPE &data_in, DATA_TYPE &data_out,
                    u_int frame_length, char head);
int       EC_construct_frametype1(fstream &data_in, char *frame_flag, char *frame,
                    u_int frame_length, char head);
int       EC_construct_frametype1(DATA_TYPE &data_in, DATA_TYPE &data_out,
                    u_int frame_length, char head);
int       EC_unpeel_frame(char *frame, char *data_buf, u_int *data_length,
                    u_int frame_length, F_HEAD &fh);
int       EC_create_frame(fstream &dat_in, char *frame_flag, char *frame,
                    u_int frame_length, char head, u_int psize);
int       EC_create_frame(DATA_TYPE &data_in, DATA_TYPE &data_out, u_int frame_counter,
                    u_int data_counter, u_int frame_length, F_HEAD &fh,
                    u_int psize, char *frame_flag, u_int nbytes);
int       EC_remove_frameheader(F_HEAD &fh, char *frame, char *data_buf, u_int *data_length,
                    u_int frame_length);
int       EC_fill_frame(F_HEAD &fh, char *frame, char *data_buf, u_int frame_length);

#endif    H_ENCDEC
```

```
//*********************************************
//
//          Name:     Joseph Morrissey
//          File:     long.h
//          Language: C++
//
//          This header file contains the declarations
//          for long.cpp.  Which contains the functions
//          for long integer calculations using the
//          LONG_TYPE class.
//
//*********************************************

#ifndef   LONG_H
#define   LONG_H

#include  <iostream.h>

typedef   unsigned int        u_int;
typedef   unsigned char       u_char;

class LONG_TYPE {
public:
          u_int       bln;
          u_char      negative;
          u_char      *num;
//Default constructor
          LONG_TYPE(void);

// lock = 1 prevents elimination of zeros before MSig 1
          LONG_TYPE(u_char *ptr, u_int length, u_char lock = 0);

//Clone - copy constructor
          LONG_TYPE(const LONG_TYPE& lt);

//Destructor
          ~LONG_TYPE(void);

//Conversion Constructor
          LONG_TYPE(u_char c);
          LONG_TYPE(u_int i);
          LONG_TYPE(char c);
          LONG_TYPE(int i);

//Operators
          LONG_TYPE       &operator=(LONG_TYPE& l);
          u_char          operator[](u_int i);
          LONG_TYPE       operator--(void);
          LONG_TYPE       operator+(LONG_TYPE& l);
          LONG_TYPE       operator-(LONG_TYPE& l);
          LONG_TYPE       operator*(LONG_TYPE& l);
          LONG_TYPE       operator/(LONG_TYPE& l);
          LONG_TYPE       operator%(LONG_TYPE& l);
          LONG_TYPE       operator|(LONG_TYPE& l);
          LONG_TYPE       operator&(LONG_TYPE& l);
          LONG_TYPE       operator ^ (LONG_TYPE& l);
          LONG_TYPE       operator ~ (LONG_TYPE& l);
          LONG_TYPE       &operator++(int);
          LONG_TYPE       &operator/=(LONG_TYPE& l);
          LONG_TYPE       &operator*=(LONG_TYPE& l);
          LONG_TYPE       &operator|=(LONG_TYPE& l);
          LONG_TYPE       &operator&=(LONG_TYPE& l);

//Logical Operators
          u_int     operator==(LONG_TYPE& l);
          u_int     operator!=(LONG_TYPE& l);
          u_int     operator>(LONG_TYPE& b);
          u_int     operator<(LONG_TYPE& b);

          void            to_string(u_char *string, u_int str_length, char fill=0);
          LONG_TYPE       mod_power(LONG_TYPE &l, LONG_TYPE &mod);
          void            print(ostream &out = cout);
          void            gen_long_number(u_int ln);

//Get Characteristic
          inline    u_int     get_length(void);
```

```
        inline      u_char*     get_number(void);

//Set Characteristic
        inline      u_char      get_sign(void);
        inline      void        set_length(u_int ln);

};

LONG_TYPE       GCD(LONG_TYPE &x, LONG_TYPE &y);
LONG_TYPE       INVERSE(LONG_TYPE &a, LONG_TYPE &b);

//Misc Functions
void        update(LONG_TYPE &y, LONG_TYPE &a, LONG_TYPE &b);
u_int       in_bytes(u_int bln);
u_int       find_bitlen(u_char *ptr, u_int length);
u_char      strind(u_char str, u_int pos);

#endif
```

Software DES Functions

```
//*********************************************
//
//          Name:     Joseph Morrissey
//          File:     des.h
//          Language: C
//
//          This header file contains the declarations
//          for long.cpp.  Which contains the functions
//          for DES calculations.
//
//*********************************************

#ifndef   DES_H
#define   DES_H

#define   SD_ECB            0x01
#define   SD_CBC            0x02
#define   SD_OFB            0x04
#define   SD_CFB            0x08

#define   SDES_ENC          0x01
#define   SDES_DEC          0x00

#define   SDES_OTHER        -1
#define   SDES_FILE_OPEN    -2

typedef struct{
          unsigned char     key[8];
          unsigned char     IV[8];
} SDES_KEY;

typedef struct {
          unsigned char     *data;
          unsigned int      lng;
} SDES_DATA;

int       EC_data_decrypt(SDES_DATA *data_out, SDES_DATA *data_in, SDES_KEY *deskey,
                          unsigned char crypt_mode, unsigned char crypt_flag);
int       EC_electronic_codebook(SDES_DATA *data_out, SDES_DATA *data_in, SDES_KEY *deskey,
                          unsigned char crypt_flag);
int       EC_cipher_block_chaining(SDES_DATA *data_out, SDES_DATA *data_in, SDES_KEY *deskey,
                          unsigned char crypt_flag);
int       EC_output_feedback(SDES_DATA *data_out, SDES_DATA *data_in, SDES_KEY *deskey,
                          unsigned char crypt_flag);
int       EC_cipher_feedback(SDES_DATA *data_out, SDES_DATA *data_in, SDES_KEY *deskey,
                          unsigned char crypt_flag);
int       EC_file(char crypt_mode, char crypt_flag, char stuff_flag, SDES_KEY *deskey,
                          char *datfile_in, char *datfile_out, char overwrite);
int       EC_f_electronic_codebook(FILE *data_out, FILE *data_in, SDES_KEY *deskey,
                          unsigned char crypt_flag);
int       EC_f_cipher_block_chaining(FILE *data_out, FILE *data_in, SDES_KEY *deskey,
                          unsigned char crypt_flag);
int       EC_f_output_feedback(FILE *data_out, FILE *data_in, SDES_KEY *deskey,
                          unsigned char crypt_flag);
int       EC_f_cipher_feedback(FILE *data_out, FILE *data_in, DSES_KEY *deskey,
                          unsigned char crypt_flag);
void      EC_error(char error);
int       EC_GenerateKey(SDES_KEY *deskey);
int       EC_KeyWrite(SDES_KEY *deskey, char *filename, int overwrite);
int       EC_KeyRead(SDES_KEY *deskey, char *filename);

int       EC_des(unsigned char *data_out, unsigned char *data_in, unsigned char k[16][48]);
int       EC_ckdf(unsigned char out[32], unsigned char R[32], unsigned char k[48]);
int       EC_expand_data(unsigned char *array1, unsigned char *array2, int no_bytes);
int       EC_reduce_data(unsigned char *array1, unsigned char *array2, int no_bytes);
int       EC_key_schedule(unsigned char k[16][48], unsigned char key[64], int enc);

#endif
```

The Four Basic Arithmetic Algorithms

Addition

This algorithm is used for the addition of nonnegative integers of radix b and with n-places. Given two nonnegative integers $(A_1, A_2, A_3.....A_n)_b$ and $(B_1, B_2, B_3,.....B_n)_b$ we wish to form their radix b sum, $(S_0, S_1, S_2,.....S_n)_b$.

j keeps track of the digit position, **k** keeps track of the carry.

1. $\mathbf{j} = n$, $\mathbf{k}=0$
2. $S_j = (A_j + B_j + k) \bmod b$
3. $\mathbf{k} = (A_j + B_j + k) / b$

   k may only equal 1 or 0

4. $\mathbf{j} = \mathbf{j} - 1$
5. If $\mathbf{j} > 0$ goto 2
6. $S_0 = \mathbf{k}$
7. End

Subtraction

This algorithm is used for the subtraction of two nonnegative integers of radix b and with n-places. Given two nonnegative integers $(A_1, A_2, A_3,.....A_n)_b$ and $(B_1, B_2, B_3,.....B_n)_b$, with the relationship $(A_1, A_2, A_3,.....A_n)_b \geq (B_1, B_2, B_3,.....B_n)_b$, we wish to form their nonnegative radix-b difference, $(D_1, D_2, D_3,.....D_n)_b$.

j keeps track of digit position, **k** keeps track of borrow.

1. $\mathbf{j} = n$, $\mathbf{k} = 0$
2. $D_j = (A_j - B_j + k) \bmod b$
3. $\mathbf{k} = (A_j - B_j + k) / b$

   k may only take the value 0 or **-1**

4. $\mathbf{j} = \mathbf{j} - 1$
5. If $\mathbf{j} > 0$ goto 2
6. If $\mathbf{k} = -1$ then the input to the algorithm did not conform to the assumptions made.
7. End

Multiplication

This algorithm is used for the multiplication of two nonnegative integers of radix b, one with n-places, the other with m-places. Given two nonnegative integers $(A_1, A_2, A_3,.....A_n)_b$ and $(B_1, B_2, B_3,.....B_m)_b$, we wish to form their radix-b product $(P_1, P_2, P_3,.....P_{m+n})_b$.

1. $(P_1, P_2, P_3,.....Pn)b = (0, 0, 0,.....0)_b$, **j** = m
2. If $B_j = 0$ set $P_j = 0$ and goto 10
3. **i** = n, **k** = 0
4. $t = A_i \times B_j + P_{i+j} + k$
5. $P_{i+j} = t \bmod b$
6. **k** = t / b
7. **i** = **i** - 1
8. If **i** > 0 goto 4
9. $P_j = k$
10. **j** = **j** - 1
11. If **j** > 0 goto 2
12. End

Division

This algorithm is used for the division of a nonnegative integer by another nonnegative integer. Given $A = (A_1, A_2, A_3,.....A_{m+n})_b$ and $B = (B_1, B_2, B_3,.....B_n)_b$ where $B_1 <> 0$ and n > 1, we wish to form the radix-b quotient $A / B = (Q_0, Q_1, Q_2,.....Q_m)_b$ and the remainder A mod $b = (R_1, R_2, R_3,.....R_n)_b$.

1. **d** = b / $(B_1 + 1)$, $(A_0, A_1, A_2,.....A_{m+n})_b$ = (A1, A2, A3,.....Am+n)b,
   $(B_1, B_2, B_3,.....B_n)_b = (B_1, B_2, B_3,.....B_n)_b \times d$
2. **j** = 0
3. If $A_j = B_1$, set q = b-1 ELSE set $q = (A_jb+A_j+1) / B_1$
4. If $B_2q > (A_jb+A_j+1-qB_1)b + A_{j+2}$ then q = q -1 and goto 4
5. $(A_1, A_2, A_3,.....A_{j+n})_b = (A_1, A_2, A_3,.....A_{j+n})_b - q(B_1, B_2, B_3,.....B_n)_b$
6. $Q_j = q$
7. If the result of step 5 was positive goto 9

8. $Q_j = Q_j - 1$, $(A1, A2, A3,.....An)_b = (A_1, A_2, A_3,.....A_{j+n})_b + (0, B_1, B_2,.....B_n)_b$

9. $j = j + 1$

10. If $j \leq m$ goto 3

11. $(Q_0, Q_1, Q_2,.....Q_m)_b$ is the desired quotient, the remainder = $(A_{m+1}....A_{m+n})_b$ / d

Appendix D

Papers

Applications of Keystroke Analysis For Improved Login Security and Continuous User Authentication **325**

Increased Domain Security Through Application of Local Security and Monitoring **341**

User Identification by Static Keystroke Analysis **351**

Applications of keystroke analysis for improved login security and continuous user authentication

S.M.Furnell, J.P.Morrissey, P.W.Sanders and C.T.Stockel
Network Research Group, Faculty of Technology, University of Plymouth, Plymouth, United Kingdom. E-mail : stevef@soc.plym.ac.uk

Abstract

This paper examines the use of keystroke analysis as a means of improving authentication in modern information systems, based upon the biometric measurement of user typing characteristics. The discussion identifies that the concept may be implemented in two ways, providing the basis for both an enhanced authentication front-end (in combination with the entry of a standard password) as well as for continuous, transparent supervision throughout the session.

Two practical systems have been implemented and evaluated, based upon static and dynamic verification techniques. The static verifier module uses a neural network approach, whilst the dynamic verifier involves statistical analysis methods.

The effectiveness of each module is examined using experimental test subject groups (15 typists in the static system and 30 in the dynamic study). The results observed allow both the authentication and analysis strategies to be contrasted, along with a general assessment of the protection that the combination of techniques would afford.

The paper also discusses how the techniques could be integrated within a more comprehensive intrusion detection framework that would be capable of identifying various other classes of system abuse.

Keywords : Computer Security, Keystroke Analysis, Intrusion Detection, Authentication, Biometrics, Passwords, Neural Networks.

1. Introduction

The issues of user identification and authentication are of paramount importance in the provision of secure information systems. If a user is not identified, it is impossible to grant any specific access rights or ensure individual accountability for activities. Without authentication, user identities could be used by unauthorised parties for illegitimate purposes. Whilst obtaining a (claimed) identity is extremely straightforward, the subsequent verification is problematic and various possible approaches exist. The potential foundations for authentication are commonly categorised as follows (Wood 1977) :

- something the user *knows* (e.g. a password);
- something the user *has* (e.g. a token such as a card or key);
- something the user *is* (e.g. a biometric such as fingerprint or voice pattern).

The first of these methods is currently the most widely used, with passwords having the advantages of both conceptual simplicity and easy implementation. However, it has been identified that passwords provide an unreliable basis for user authentication (Jobusch and Oldehoeft 1989) and that stronger methods are necessary, using techniques that are more difficult, if not impossible, to forge. In general, the use of a biometric appears to be the strongest option, as it may not be easily guessed, stolen or transferred to other people in the same way as secret knowledge or a token. However, the cost of the technology required to successfully implement most biometric methods largely precludes its uptake in many cases. What is, therefore, required is a biometric measurement that can be obtained without requiring any form of additional hardware. Fortunately, such a characteristic can potentially be identified in the form of users typing style (or keyboard rhythm).

The basic premise of the approach is that typing characteristics will be reasonably unique to each user, revealing an individual "signature". The concept of using keystrokes to assess identity was originally proposed by Spillane (1975) and can be claimed to provide a "behavioural" biometric measurement, in that the act of typing represents how a user does something as opposed to being a physiological characteristic.

2. An overview of keystroke analysis concepts

Before considering a monitoring strategy, it was necessary to identify suitable typing characteristics that could be used as a basis for analysis (and hence authentication). Several factors were considered, as listed below :

- inter-keystroke times;
- keystroke duration times;
- typing error frequency;
- force of keystrokes;
- keystrokes / words per minute.

Of these, the inter-keystroke time was considered likely to be the most characteristic and was, therefore, adopted as the basis for the practical investigations to be described later.

Users are likely to differ dramatically in terms of typing styles and abilities, depending upon factors such as their familiarity with the keyboard, experience and any formal tuition. Indeed, previous research (Card et al. 1980) has determined six general skill classifications based upon the average inter-keystroke time of the subject. In theory it should be relatively easy to differentiate between users from different categories, whereas discrimination within a category may be more problematic.

The typing characteristics of each user must be assessed to create an appropriate *profile* that may then be used for subsequent authentication. Any significant departures from the profile will cause impostor alerts.

Keystroke analysis may be incorporated into an authentication system in two ways - referred to as *static* and *dynamic* verification strategies.

- *Static verification*
  In this scenario authentication is based upon the entry of a static text string. There are two points at which such analysis could be used; in the initial authentication of a user (i.e. at login) and whenever a volatile command is entered.

  - *Login*
    There are usually two points during the login process that are ideal for static keystroke analysis, at the entry of *username* and *password*. Passwords are currently the most widely used form of authentication, but are often compromised by users failing to follow simple procedures, recognised in a variety of previous studies (such as, avoiding *dictionary* words, or incorporating *numbers* into the password sequence). Using keystroke analysis, the username and password would be entered as usual, but rather than just authenticating the user from this alone the system would also analyse the way in which it was typed, providing a further level of authentication.

  - *Volatile commands*
    These are defined as any commands that may delete, modify or copy information stored by the system, such as the '*deltree*' or '*xcopy*' commands in MS-DOS (Microsoft 1994). These could be used by malicious intruders to vandalise information, inconveniencing legitimate users and potentially causing serious damage. There would, however, be a practical limit to the number of commands that could be monitored in this way as, for each one to be analysed, a specific profile would need to be obtained from the legitimate user.

  Static verification appears to be the most common approach, having been the basis for a number of previous studies (Bleha et al. 1990; Joyce and Gupta 1990).

- *Dynamic verification*
  Using this approach authentication is based upon any arbitrary text input, allowing greater scope for supervision in real-time during user sessions. Monitoring could occur continuously, providing a transparent (i.e. non-intrusive) means of authentication that is not currently possible with most other methods (even those based on biometric features).

  In this scenario authentication is no longer reliant on a single judgement but becomes continuous throughout the session. One advantage here is that it should serve to prevent "logical piggybacking", whereby an intruder attempts to utilise an unattended terminal that is already logged into another account.

As with other biometric systems, the effectiveness of keystroke analysis may be judged on the basis of two factors : the False Acceptance Rate (FAR) and the False Rejection Rate (FRR). The FAR relates to errors where impostors are falsely believed to be legitimate users. Conversely, the FRR refers to errors where the system falsely identifies the legitimate user as an impostor. These rates share a mutually exclusive relationship, such that configuring the tolerances of a system to give good results for one will generally cause a degradation of the other (and, as such, it is not possible to attain optimum levels for both measures). An "equal error" scenario is not really an appropriate compromise and a decision must, therefore, be made as to which rate should receive priority.

In actual fact, the priorities will vary depending upon whether a static or dynamic authentication system is used. In the static scenario, minimisation of the FAR is considered to be the priority, as any successful impostor could potentially go unchecked for a whole session. However, in the dynamic scenario, with continuous assessment, a greater window for impostor detection is available and so minimising the FRR becomes the most important consideration (as rejections *during* a session could irritate and disrupt a legitimate user more significantly than occasional false login failures). A further important consideration in the dynamic scenario is the speed with which the identity assessment can be provided by the system (i.e. how many keystrokes could an impostor enter before being noticed ?).

3. PRACTICAL STUDIES OF KEYSTROKE ANALYSIS

The research team has conducted two practical studies to help evaluate the effectiveness of the keystroke analysis concept. Both of these investigations utilised PC-based experimental systems and related to the static and dynamic verification approaches respectively.

The systems share common core functionality, using PC hardware interrupts to detect keyboard actions and collect inter-keystroke timings (with one millisecond accuracy), but differ significantly in terms of the analysis strategies employed. In the static analyser, authentication is based upon neural network techniques, whilst the dynamic system uses statistical methods.

In both cases, analysis was restricted to character pairs (or *digraphs*) involving alphabetic and "space" keystrokes, as these were considered the most likely to reveal any characteristic rhythm and were also found to produce the best results in a previous study which conducted a comprehensive investigation of this aspect (Leggett and Williams 1988). Both systems also attempted to ensure that the typing profiles were representative of users normal styles by ignoring erroneous inputs. The profiler for the static analyser achieved this by totally disregarding all inaccurate samples, whereas the version for the dynamic system (where samples were longer and, hence, abandonment would have been impractical) filtered out and ignored any deleted keystrokes.

A significant number of test subjects were involved in both studies, with abilities ranging from experienced typists to comparative novices.

Specific details of the two systems and the results obtained are described in the sections the follow.

4. STATIC KEYSTROKE ANALYSIS

The static keystroke analyser was implemented using a neural network for pattern recognition, based upon a multi-layer perceptron network. The perceptron is one of the best studied single layer neural networks (McClelland and Rumelhart 1986) and is made up of several simple biological neuron models. The simple model is constructed from input lines, a thresholding unit and an output line. If the inputs to the thresholding unit are greater than a threshold value the perceptron "fires", as shown in figure 1.

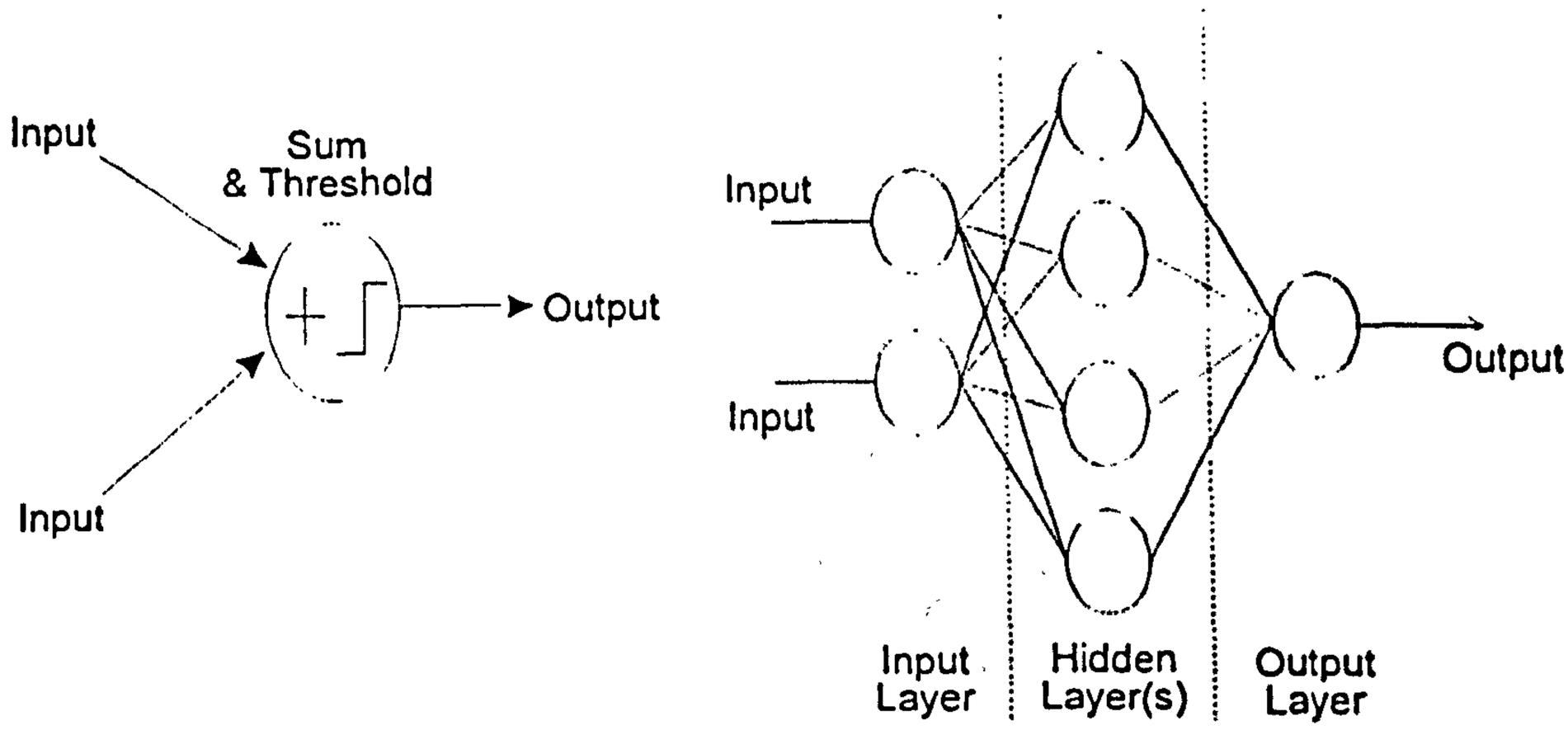


Fig. 1 : Single and multi-layer perceptrons

However, the single layer perceptron is only able to classify linearly separable problems and the variability of user typing styles suggests that keystroke analysis does not fit into this category. As a result, a multi-layer perceptron was used, which is capable of solving non-linear problems. These networks are made up of several neuron models, with at least one layer which is not directly connected to the input or output of the network. The version used in the experiment employed the back-propagation learning algorithm (McClelland and Rumelhart 1986), which attempts to minimise the error of the network by comparing the actual output to a target output for specific input vectors and adjusting the weights of input lines accordingly.

The static verification system is actually the more recent of the two implementations and, as such, the test subject base to date has been rather small. A *sampler* program was used to collect samples from 15 experimental test subjects. Each subject typed four reference phrases 35 times each. The four phrases were as follows:

REF 1 : *"PRINT"*;
REF 2 : *"TRANSFERENCE"*;
REF 3 : *"RED SKY AT NIGHT"*;
REF 4 : *"RUGBY PLAYERS ODD SHAPED BALLS"*.

These phrases were chosen to allow sufficient overall representation of left-hand, right-hand and crossover digraphs, and differing phrase lengths were used to allow an examination of the effect that this had on attempts to classify the user's samples.

The inter-keystroke times obtained from sampling the test subjects typing were saved to a file for later use by the pattern recognition module.

Set phrases were used to make the classification as difficult as possible, giving a worst case scenario where all users have the same password, but with a real life scenario for the volatile commands. The use of individually distinct phrases such as usernames will be investigated in future experiments but, because everyone is familiar with typing their own names, it is reasonable to assume that individual typing techniques will be more prominent and, therefore, more easily identifiable. Genetic algorithms will also be used in later experiments to identify the specific digraphs that the neural network uses to identify users.

In the experiment every user had their own multi-layer perceptron for each phrase, to identify the specific user from the others sampled. Twenty-five samples of the user to be identified were placed into the training set of the multi-layer perceptron, along with twenty-five samples from each of the other users to provide false samples. The remaining 10 samples from each user were used to evaluate the success that each network attained in identifying the legitimate user and rejecting impostors. This gave a total of 150 legitimate user attempts and 2100 impostor attempts as the basis for evaluation.

Each neural network was presented with 4000 inter-keystroke samples for supervised training. The samples were chosen at random from the training set. The resulting neural network was then evaluated, with the output being a value between 0 and 1 (with 1 indicating a positive identification of the user). In actual fact, three decision boundaries (i.e. the points at which the neural networks output would be taken as a positive identification) were evaluated, namely 0.7, 0.8 and 0.9.

Table 1 presents the results of the initial experiment. In this case the neural network was trained on a ratio of one true sample to one false sample, again the actual samples were chosen randomly.

	0.7	0.8	0.9
REF 1	12	8	4
REF 2	4	4	3
REF 3	5	4	2
REF 4	3	2	2
Average	6	5	3

FAR given as %

	0.7	0.8	0.9
REF 1	30	42	58
REF 2	22	25	34
REF 3	24	27	34
REF 4	53	38	43
Average	32	33	42

FRR given as %

Table 1 : Effectiveness with 1:1 training ratio

It was found that the neural network provided a lower FRR if the training was biased towards using more samples of the user to be identified. A ratio of four true samples to one false sample gave better results, as shown in table 2.

	0.7	0.8	0.9
REF 1	24	21	15
REF 2	8	6	5
REF 3	8	6	5
REF 4	4	3	2
Average	11	9	6

FAR given as %

	0.7	0.8	0.9
REF 1	10	15	26
REF 2	15	19	25
REF 3	21	23	27
REF 4	28	32	40
Average	19	22	30

FRR given as %

Table 2 : Effectiveness with 4:1 training ratio

However, given that the static analysis at login could be based upon the entry of two phrases (i.e. username and password), it was decided to evaluate the effectiveness in this context. This was achieved using a combination of the REF 2 and REF 3 phrases, giving the results in table 3. The success of the authentication was determined using three confidence classifications:

- rejected (R), where a sample pair is rejected by both neural nets;
- low (L), where a sample pair is rejected by one neural net;
- high (H), where a sample pair is accepted by both neural nets.

	R	L	H
0.7	87	13	1
0.8	89	11	0
0.9	92	8	0

Impostor attempts (%)

	R	L	H
0.7	4	38	58
0.8	5	10	55
0.9	7	50	43

User attempts (%)

Table 3 : Results for two phrase combination

It is important to reduce the FRR so that users are not overly inconvenienced through multiple authentication requests. However, in a static verification system it is more important (especially where the mechanism may be used at the login point of a session) for the FAR to be minimised, to prevent malicious attackers gaining initial entry to the system. If the reference phrases used are short, then a relatively high FRR of around 10% may be acceptable.

The background and findings of this study are described in more detail in Morrissey (1995).

5. DYNAMIC KEYSTROKE ANALYSIS

The other experimental study concerned the evaluation of a *dynamic* verification approach. As previously mentioned, this system was actually the earlier of the two implementations and analysis in this case was based upon statistical methods rather than neural networks, with the profiles storing mean and standard deviation values for each digraph.

A total of 30 test subjects were involved in this study, with each being required to submit a profile sample and two additional test samples. The profiling procedure required users to enter two samples of a 2200 character *reference text*. A more significant length was necessary in this scenario to ensure that each users "natural" typing style emerged and that sufficient samples of each digraph were obtained to enable appropriate mean and standard deviation values to be established. The two further text samples (of 574 and 389 characters) were used to represent impostor attempts by comparing them against all profiles not belonging to the same user. As such, the results in this phase were derived from more than 1700 impostor attempts.

The monitoring system compared the inter-keystroke times from the test samples against user profiles, with incompatible times being judged invalid. These judgements were then analysed in two ways to detect impostors :

1. monitoring the percentage of invalid keystrokes during the 100 most recently typed;

2. monitoring the number of consecutive invalid keystrokes.

However, it was recognised that even legitimate users would generate some degree of invalid keystrokes and, as a consequence, each profile held associated *authentication thresholds* for these factors, with intrusion alerts being generated if either was exceeded during monitoring.

Given that the dynamic analyser would be used for continuous monitoring, the minimisation of false rejections was viewed as a priority. It was considered that if the FRR could be totally eliminated, then what would then be observed would effectively be a "worst-case" FAR. To this end, the authentication thresholds in the profiles were determined for each subject on an individual basis (by observing the peak values from the comparison of the two additional typing samples against their profile). This ensured that thresholds were set such that the legitimate user would always pass. The aims of the study were, therefore, to determine the FAR and the speed of successful impostor detection.

In terms of overall impostor detection effectiveness, the experimental system exhibited an overall FAR of 15% (based upon 13% for the first sample and 18% for the second). This figure would be of less significance when considered along with the initial authentication provided by the static analyser and the combination of the two methods would almost certainly defeat most impostors.

In the cases where detection was achieved, the other important consideration was how many keystrokes the impostor was able to enter before being detected. The experimental findings here are presented in figure 2, which shows the percentage of impostors detected within five distinct keystroke ranges (with cumulative values also indicated).

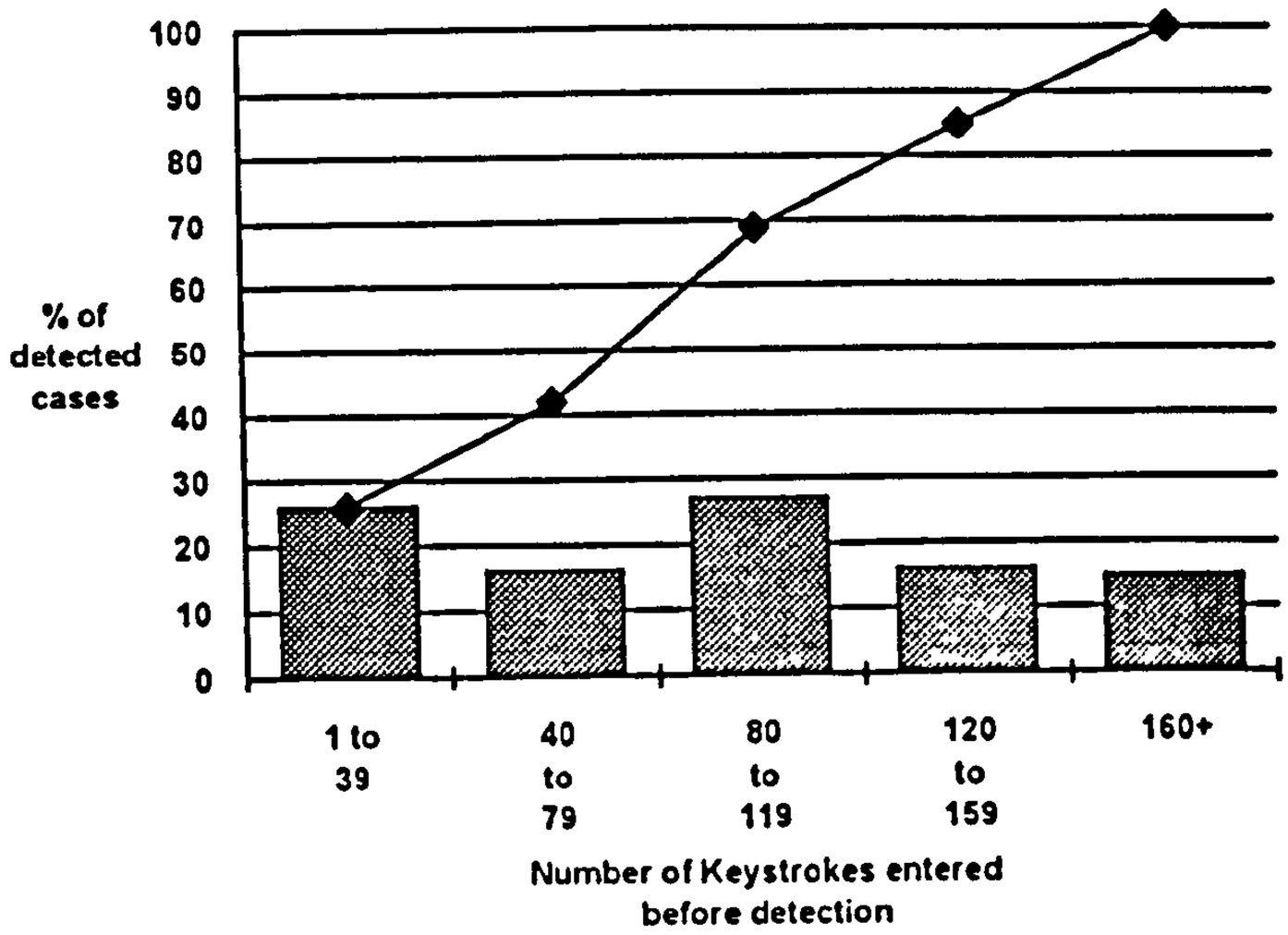


Fig. 2 : Keystrokes before impostor detection

These results indicate that the vast majority of impostors would be detected within 160 keystrokes (the equivalent of two standard lines of text), with detection in under 40 keystrokes in 26% of cases. Whilst this may not combat the most destructive scenarios (e.g. the immediate entry of "delete *.*" would very likely be unchallenged, unless specifically monitored as a *volatile command* by the static analyser), it should be sufficient to identify the more common types of intruder who generally require sustained access in order to effect a serious breach. It should be noted that detection is likely to be quicker in cases where an impostor takes the place of a legitimate user, as a certain percentage of invalid keystrokes would already have been registered (by the legitimate user), causing the rejection threshold to be reached more easily.

A more detailed discussion of the dynamic analyser in terms of both the system implementation and the results observed is presented in Furnell (1995).

6. AN ASSESSMENT OF THE PRACTICAL STUDIES

An overall comparison of the experimental results is presented in table 4.

	Static Analyser	Dynamic Analyser
No. test subjects	15	30
FAR (%)	8*	15
FRR (%)	7*	0**

* based upon two phrase combination
** achieved via authentication thresholds

Table 4 : Comparison of experimental study results

At this stage it has only been possible to evaluate the two systems independently (as only a small number of test subjects were common to both studies), with the consequence that an overall FAR for the combined strategies could not be obtained. However, it is certainly true to say that in some cases where a false acceptance would occur within one analyser, the other system could compensate by successfully trapping the impostor.

With regard to the performance of the individual systems, it can be observed that the neural network techniques used by the static verifier do not appear to yield such a significant improvement over the statistical approach as one might expect. However, it is expected that the performance could be improved further with the incorporation of a genetic algorithm to determine the most characteristic user digraphs within each phrase. It is also recognised that as the analysers operate in different contexts, such a comparison is not exactly comparing like with like.

The single phrase results for the static analyser were rather disappointing in terms of the levels of false rejection observed. Whilst it is considered that these would not pose a significant problem in the login context (and could be largely overcome by adopting the two phrase analysis), the issue would be more problematic if monitoring volatile commands (which would normally be single phrases). The results indicate that this could lead to repeated false rejections within a session (the factor that the dynamic analyser strives to avoid), making supervision somewhat less than transparent. For this reason the static system is not considered suitable for the second purpose unless it can be strengthened. In addition, the increased FRR with longer text strings is worthy of further examination, although observation during the study revealed that test subjects generally made more mistakes with the longer samples, necessitating repeated re-entry, which potentially led to uncharacteristic typing.

With regard to the dynamic analyser, the main point that must be recognised is that the FRR of 0% was obtained artificially. However, it is still envisaged that false rejections would not be frequent enough to significantly worry legitimate users in practice if authentication thresholds were set correctly.

Having established the effectiveness of the two techniques, it is also interesting to consider how they could be integrated within a more comprehensive monitoring framework. This issue is discussed in the next section which presents the conceptual design of a system called IMS (for Intrusion Monitoring System).

7. A wider intrusion detection framework

It must be recognised that the use of keystroke analysis alone does not provide a comprehensive basis for intrusion detection, as the technique is only suitable as a user authentication mechanism. It is actually possible to identify several distinct classes of computer system abuser (Anderson 1980) :

- external penetrators (i.e. unauthorised users of the system);

- masqueraders (i.e. authorised users who operate under the identity of another user);
- clandestine users (i.e. users who evade access controls and auditing);
- misfeasors (i.e. legitimate users who abuse their privileges).

It should be evident that keystroke analysis would only be suitable for detecting the first and second of these classes and would be ineffective against the other two (given that these are legitimate users operating under their own identities). In addition, no defence would be provided against malicious processes (i.e. viruses, worms and Trojan Horse programs).

Therefore, in order to provide more comprehensive intrusion detection it becomes necessary to monitor other aspects of user behaviour, along with general events that may be indicative of intrusions.

With regard to enhancing behaviour profiles, suggested strategies include the monitoring of :

- time and location of system access;
- operating system command usage;
- application usage;
- data access.

Such characteristics, in combination with the existing information on user typing styles, would allow significantly more comprehensive profiles to be constructed.

In addition to profiles, the system could specifically monitor for a variety of events that might form part of a known intrusion scenario. The occurrence of such events would be regarded as suspicious, especially in aggregation, and would be used to increase the alert status for the affected user or process. Some illustrative examples of such events are given below, along with the type of intruder that the occurrence of each would indicate :

- out of hours access (*penetrator* or *misfeasor*);
- use of dormant accounts (*external penetrator* or *masquerader*);
- excessive use of system "help" facilities (*external penetrator*);
- modification of executable file (*malicious process*).

The use of profiles and rules in this way is based upon a similar premise to the approach used by the IDES intrusion monitor (Lunt 1990) and various other detection systems (Mukherjee et al. 1994).

At a high level, the proposed IMS architecture is based upon the concept of a centralised *Host* handling the monitoring and supervision of one or more *Clients* running on local workstations. The purpose of the Clients is to collect relevant data on user and process activity and respond to any suspected intrusions detected by the Host. All behaviour profiles, generic rules and such like would be maintained securely at the Host, which also handles all of the analysis and the main bulk of other processing associated with the supervision. By contrast, the Client involves no local data storage and acts almost exclusively as an agent of the Host.

At a lower level, the system would be comprised of a number of functional modules, as shown in figure 3 and outlined below.

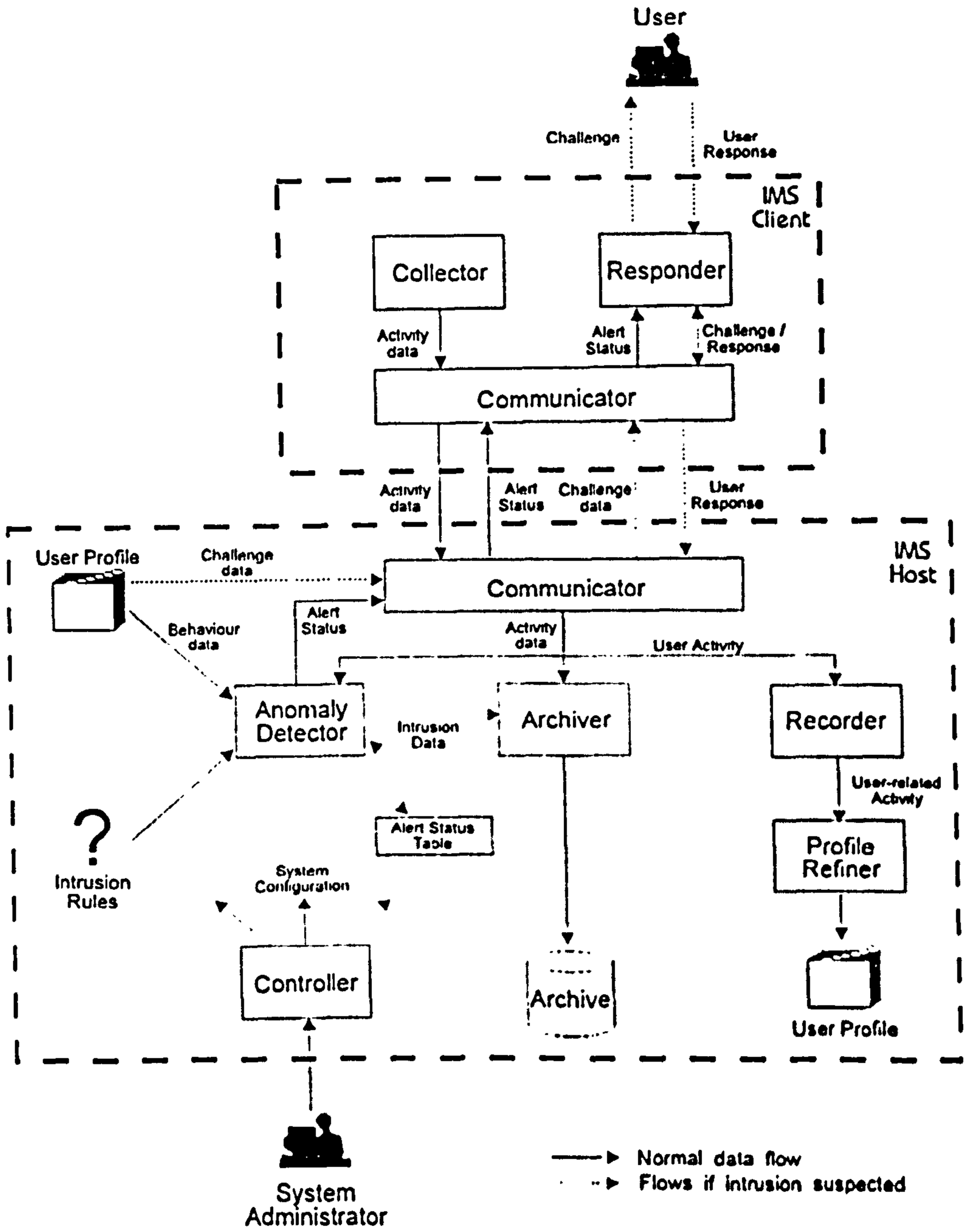


Fig. 3 : IMS Architecture

Anomaly Detector

The purpose of the *Anomaly Detector* will be to analyse user and process activity for signs of suspected intrusion, comparing it against the behaviour profiles that apply to the current users (claimed)' identity as well as against the generic intrusion rules. It is envisaged that this will be comprised of a number of further sub-modules, each handling a specific aspect of anomaly detection (e.g. keystroke analysis).

Profile Refiner

The IMS would also utilise activity data as the basis for updating behaviour profiles. This recognises the possibility that user behaviour may legitimately change over time and the *Profile Refiner* would provide an automatic means for profiles to be updated to reflect this.

It would be most appropriate for the *Profile Refiner* to be based upon the neural network approach, given that the inherent ability to analyse and recognise patterns could allow behavioural characteristics to be identified that might not be apparent to a human observer. In this way, the effectiveness of the system would have the potential to improve over time, in that it could gradually learn more patterns of legitimate activity for each user and determine which of the profiled characteristics provide the most reliable discriminators.

It would be undesirable for the *Profile Refiner* to utilise data that is later found to be anomalous and refinement would only take place after the termination of user sessions (provided, of course, that no intrusions were proven during this time).

Recorder

The *Recorder* handles the short-term storage of system activity data during the period of a user session. Upon termination the information will be picked up and used by the *Profile Refiner*, provided that the session was not considered anomalous.

Archiver

The *Archiver* will collect data relating to all security relevant events (including login failures, intrusion alerts, authentication challenges, suspended sessions and the like) and store it in a long-term archive (in the same manner as a traditional audit trail), providing a more permanent record of activities and suspected anomalies.

Collector

The *Collector* represents the interface between the IMS and the underlying applications, with the responsibility for obtaining information on all relevant user and system activities. These would be monitored with two objectives :

- to collect data on those events which pertain to monitored behaviour characteristics;
- to identify those events which may affect the security of the system (for comparison against generic intrusion indicators).

The resolution of data collection would be determined at the Host by the System Administrator.

Responder

This module resides in the Client and handles the task of responding to anomalies detected by the Host. The operation of the *Responder* would centre around the continuous monitoring of the alert status transmitted by the Host, with increases in the level triggering

appropriate actions. The nature of response might include issue of a user authentication challenge, suspension of a session or cancellation of a process.

Communicator

The *Communicator* provides the communications interface between the Host and the local Client systems and, as such, the functionality of this module is duplicated on both sides of the link. The principal functions would include transmitting user and process information to the Host and then subsequently keeping the Client(s) informed of the current alert status. If implemented in a heterogeneous environment, the Client side would be responsible for resolving any operating system differences that exist within the monitoring domain so that information could be presented to the Host in a consistent, standardised format.

Controller

This module allows the System Administrator to configure the operation of the IMS system. On the Host side, this would apply to the *Anomaly Detector* (e.g. behaviour characteristics to consider / prioritise, generic rules in operation), the *Profile Refiner* (e.g. frequency of refinement) and the *Archiver* (e.g. level of detail required). On the Client side, configuration would affect the operation of the *Collector* (e.g. the level of data collection) and the *Responder* (e.g. the level of response at each alert level). These settings would be controlled and recorded through the Host system, with any Client settings being established at the time of session initiation.

The operation of these modules would combine to provide a system architecture offering improved security, whilst at the same time retaining the advantages of end-user convenience and financial viability.

8. Conclusions

The overall results from the experimental studies are considered to be encouraging and, although both systems exhibited some degree of error, it must be remembered that the approaches are intended to supplement or strengthen security as opposed to providing a total solution.

Both techniques have considerable potential for easy integration into existing systems, with the static analyser enhancing existing password mechanisms and the dynamic system providing the basis for continuous supervision at subsequent times. This has already been achieved in the context of a demonstrator system developed since the experimental study, which allows transparent monitoring of user activity (in real-time) on a client workstation.

Planned future development of the systems include the enhancement of the static analyser to incorporate a genetic algorithm, along with subsequent modification of the dynamic system to utilise these techniques if the results are suitably encouraging.

The incorporation of keystroke analysis into an overall IMS-type framework is viewed as advantageous to provide more comprehensive supervision. Whilst the concept is not envisaged as a replacement for conventional authentication and access control methods, it will provide a way to strengthen existing systems and complement any security already provided.

References

Anderson, J.P. 1980. "Computer Security Threat Monitoring and Surveillance", James P. Anderson Co., Fort Washington, PA (Apr.).

Bleha S; Slivinsky, C.; and Hussien, B. 1990. "Computer-Access Security Systems Using Keystroke Dynamics", *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12, no.12: 1217-1222.

Card, S.K.; Moran, T.P.; and Newell, A. 1980. "The keystroke level model for user performance time with interactive systems", *Communications of the ACM* 23: 396-410.

Furnell, S.M. 1995. "Data Security in European Healthcare Information Systems", PhD Thesis. School of Electronic, Communication and Electrical Engineering, University of Plymouth, UK.

Jobusch, D.L. and Oldehoeft, A.E. 1989. "A Survey of Password Mechanisms : Part 1", *Computers & Security* 8, no. 7: 587-604.

Joyce, R and Gupta, G. 1990. "Identity Authentication Based on Keystroke Latencies", *Communications of the ACM* 33, no.2: 168-176.

Legget, J. and Williams, G. 1988. "Verifying identity via keystroke characteristics", *International Journal of Man-Machine Studies* 28: 67-76.

Lunt, T.F. 1990. "IDES: An Intelligent System for Detecting Intruders", In *Proceedings of the Symposium : Computer Security, Threat and Countermeasures* (Rome, Italy, Nov. 1990).

McClelland, J.L. and Rumelhart, D.E. 1986. *Parallel Distributed Processing, Volume 1*. MIT Bradford Press.

Microsoft. 1994. *MS-DOS 6.21: User's Guide*. Microsoft Corporation.

Morrissey, J.P. 1995. "A hardware implementation of a Local Security Unit for CISS", PhD Thesis. School of Electronic, Communication and Electrical Engineering, University of Plymouth, UK.

Mukherjee, B.; Heberlein, L.T.; Levitt, K.N. 1994. "Network Intrusion Detection", *IEEE Networks 8*, no.3: 26-41.

Spillane, R. 1975. "Keyboard apparatus for personal identification", IBM Technical Disclosure Bulletin, 17, 3346.

Wood, H.M. 1977. *The use of passwords for controlled access to computer resources*, National Bureau of Standards Special Publication 500-9, U.S Dept. of Commerce / NBS, (May).

Increased Domain Security Through Application of Local Security and Monitoring

J.P.Morrissey, P.W.Sanders and C.T.Stockel
Network Research Group, Faculty of Technology,
University of Plymouth, Plymouth.
E-mail: josephm@soc.plym.ac.uk

Abstract

This paper presents a model for increasing security within a security domain through the use of localised security services and continuous monitoring. The model divides security services between three logical structures Local Security Units, Local Security Servers and Domain Management Centres. The localisation of security allows the functional divisions within organisations to implement modified security dependent upon their individual needs.

Keywords : Computer Security, Monitoring, Local Security, CISS.

1. Introduction

As networks and the consequential connection of computers with their stored data becomes increasingly common, the security of that data becomes ever more important. Current techniques in security have proved themselves to be ineffective when confronted by credible hackers [1], [2], [3]. As more and more people gain access to networks and the computers connected to them, as current policy within the EC and USA indicates they will, there will be more people tempted to pursue illegal activities. Many may wish to instigate malicious damage while others may only inadvertently cause damage. Whatever the reason, the action is illegal and security upon the computer systems must be capable of dealing with these attempts.

To combat the increased security risks that placing sensitive information upon open computer networks with ever increasing numbers of users introduces, new methods for security must be found. This paper uses CISS (Comprehensive Integrated Security System)[4] as its basic security provider. The International Standards Organisation (ISO) have been setting standards for open interconnection of systems. The CISS model attempts to provide the required security as indicated by the ISO within relevant publications [5], [6] and it also attempts to provide the security necessary for open distributed environments. CISS is an adaptable security system capable of being added to existing computer systems to bring their current level of security up to a level acceptable by their owners.

2. Comprehensive Integrated Security System

The CISS architecture was developed to protect a computer system within an open distributed environment against the threats detailed in Table 1.

Threat	Description
Masqueraders	When a legal entity (user/program) impersonates another.
Illegal Associations	Where an illegal entity forms an association with a legal entity that violates the authentication and authorisation policies in place.
Non-authorised access	Where an intruder/user gains access to resources that violates the access control policy.
Denial of Service	Where a legal entity is denied access to resources that are required for proper operation.
Repudiation	Prevent the false denial of a legal entity that has provided a service or resource.
Leakage of Information	Loss of confidentiality, anonymity, and misappropriation.
Traffic Analysis	Deduction of information from characteristics of data transfer.
Invalid Message sequencing	Prevention of re-submission of data beyond its valid lifetime, replay attacks.
Data Modification	Malicious or accidental modification of data.
Deduction of Information	Collection of data from summaries in a distributed database
Illegal Modification of Programs	Malicious or accidental modification of software.

Table 1 Threats

To combat these threats CISS makes available services that can be used to provide the necessary security measures. The security services are provided through the correct sequencing of the security mechanisms shown in Table 2. The services can be requested by users, or forced upon them by the security administrator's configuration of CISS according to the security policy in place.

	Security Mechanism	Description
En	Encipherment	Application of cryptographic algorithms for confidentiality.
DS	Digital Signature	Use of cryptographic techniques to provide proof of origin.
AC	Access Control	Controlling the access to resources upon a network.
DI	Message Authentication Codes	Algorithms to check the integrity of communicated data.
AE	Authentication Exchange	Techniques by which the identity of an entity can be confirmed.
TP	Traffic Padding	Provide protection against traffic analysis
RC	Routing Control	Control the path along which data is communicated.
NT	Notarisation	Provision of proofs.

Table 2 Security Mechanisms

X indicates a mechanism is present and O indicates its absence.

Service	En	DS	AC	DI	AE	TP	RC	Nt
Peer Entity Authentication	X	X	O	O	X	O	O	O
Data origin Authentication	X	X	O	O	O	O	O	O
Access Control Service	O	O	X	O	O	O	O	O
Data Confidentiality	X	O	O	O	O	O	O	O
Non-repudiation Origin	O	X	O	X	O	O	O	X
Non-repudiation delivery	O	X	O	X	O	O	O	X

Table 3 Sample Services

Because a security system is a complex and potentially large software construction problem. CISS has been divided into ten functional elements known as agents. The agents are responsible for the co-ordination, supply and control of security services to users/entities.

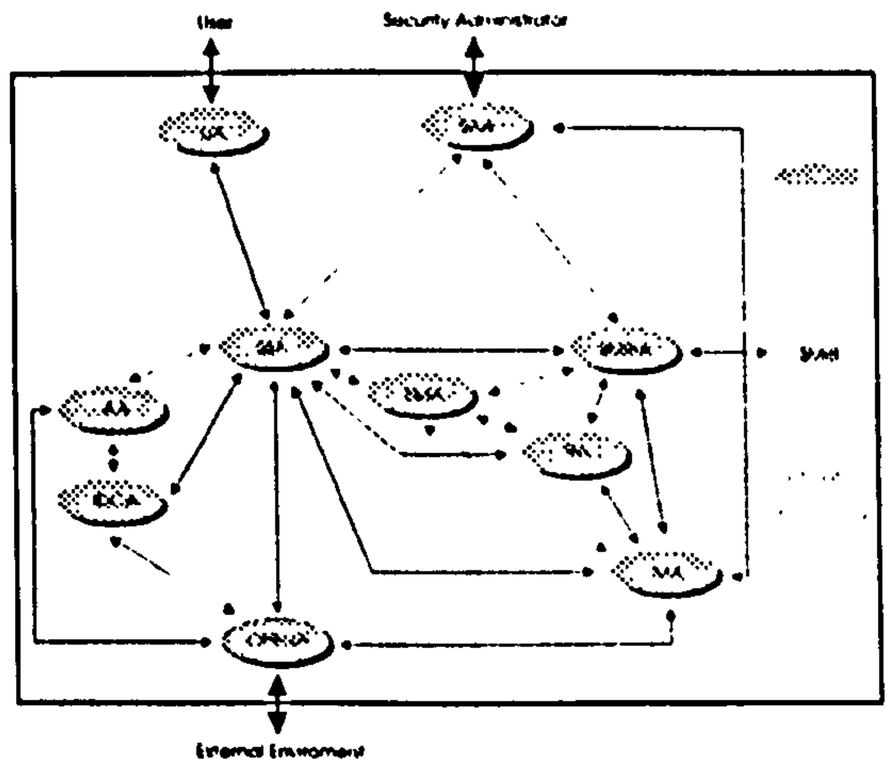


Figure 1 10 Agent Interaction

The Security Management Information Base
The security management information base (SMIB) is the CISS knowledge store. It contains information on the configuration of CISS through the selection of mechanisms, services, protocols and the subsequent limitations upon the variables in their use as specified by a security policy. The SMIB contains information on authorised users, authentication data, user entity capabilities and privileges.

The 10 CISS agents are:

1. **User Agent (UA)**
   This is the interface of CISS through which the users can access its security mechanisms and services directly via a request. Dependent upon the security policy some security services will be provided irrespective of the user's wishes.
2. **Security Administration Agent (SAA)**
   This is the second interface to CISS and is solely used by the security administrator. The division of functionality between the UA and the SAA has come about due to the necessity of making the UA a multi-user agent producing complex code, the separation produces simpler code for the SAA.
3. **Operational Environment Agent (OPENA)**
   This is the third and final interface through which CISS communicates with the outside world. Its main function is interfacing the operating system (OS), applications etc, to the SSA for the provision of security services. The OPENA is made up in part by the application programming interface (API). The interface allows applications to access CISS services giving CISS greater flexibility for implementation within existing systems. The OPENA interacts with the AA and IDCA for communication with entities external to the local security domain.
4. **Security Services Agent (SSA)**
   The SSA is the central agent through which all services are requested. The SSA has contact with all other agents. The SSA selects the security mechanisms that a requested security service is made up from and with assistance from the security mechanism agent implements the service.
5. **Security Mechanisms Agent (SMA)**
   The SMA interfaces the security mechanisms to the SSA for the provision of security services.
6. **Security Management Information Base Agent (SMIBA)**
   The SMIBA is responsible for interfacing the SMIB to the other agents within CISS. It is the only agent with direct access to the SMIB.

7. **Association Agent (AA)**
   The AA is responsible for the associations between agents within the local security domain.
8. **Inter Domain Communication Agent (IDCA)**
   The IDCA is responsible for communication with entities external to the local security domain. It must negotiates the required parameters for a secure connection with any external entity. The IDCA interacts with the AA and the OPENA for the provision of secure inter domain communications.
9. **Monitoring Agent (MA)**
   The monitoring agent monitors the activity of CISS, primarily the actions of the SSA, and logs the events within the SMIB. The historical events that are stored within the SMIB are only accessible by the security administrator for the preparation of audit logs.
10. **Recovery Agent (RA)**
    The recovery agent is responsible for the recovery of CISS when faults either at component level or procedural level occur.

3. Local Security

The 3-L architecture is comprised of a domain management centre (DMC), multiple local security servers (LSSs) and a local security unit (LSU) at each user terminal. Multiple LSUs are hosted upon a single LSS, the number of LSSs within an organisation would be dependent upon its size and its internal policy for computer asset organisation. The three levels would interact within the boundaries of the security domain to enable the requirements of the organisation's security policy to be met both within the security domain and in communication with the external domain environment.

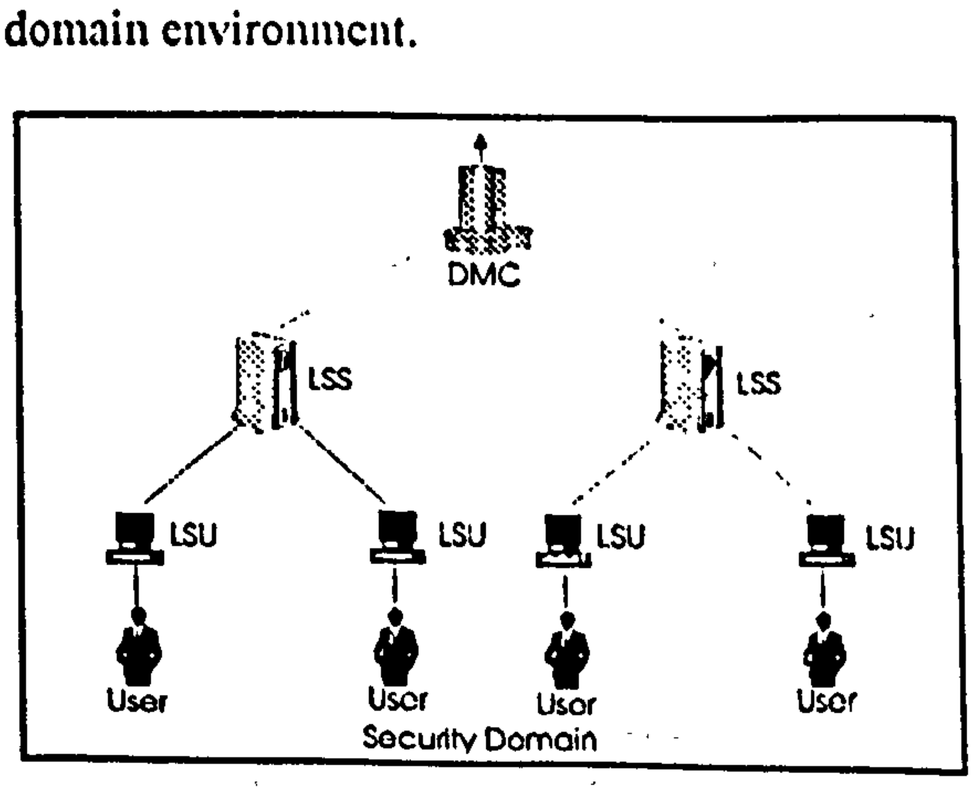


Figure 2 Three Level Architecture

Each higher level within the 3-L architecture is aware of the capabilities of lower level units that are connected to it, information concerning the hosted security modules would be kept within the SMIBs of the LSSs and DMCs. This allows the upper levels to determine whether services requested by lower level units meet with the required security laid down by the domain security policy and therefore whether the operation can go ahead, or if the operation requires more security than is available the operation is prevented by the higher level entity and the reason is logged with the requesting unit.

The objective of the three level architecture (3-L architecture) is to provide management of security services within the boundaries of a security domain in a distributed environment by selective deployment of the CISS agents. Some of the security services that must be co-ordinated over the entire domain are described below.

- Logging and Auditing Security Relevant Events:
  Audit information must be analysed for potential privilege abuse and intrusion detection.
- Inter-domain Communications:
  When a user wishes to perform some function manipulating an entity within a different security domain the management structure of each domain must negotiate what services are required for secure communication over the public unsecured network.
- Generation, Storage and Distribution of Cryptographic Keys:
  To maintain confidentiality within the domain it is necessary to generate keys to a set policy and then use secure protocols for their storage and distribution.
- Domain Notary Service
  The security management must provide *trusted third party* (TTP) services for users. This means that the TTP[7] functions of registration, notarisation, certification and public key verification along with public-key distribution must be provided.
- Trusted entry point
  Firewalls [8], are used for access control to local area networks through the internet. This method of access control should be implemented in CISS via a single point of entry and exit to the security domain, allowing

the implementation for efficient security logging and access control.

- Domain Communications:

  If a user wishes to perform a remote operation upon a machine that is within the same security domain, the security management structure must make sure that all security requirements are met according to the policy laid out within the domain.

- Access Control

  Through the use of the security domains SMIB the access rights and privileges of individuals may be determined so that unauthorised manipulation of domain entities can be secured against.

- Authentication

  Every person that wishes to use CISS functionality must first be authenticated through information stored within the security domains SMIB.

The CISS agents used within each level are listed in the table below.
X-Present
O-Absent

	DMC	LSS	LSU
UA	O	O	X
SAA	X	X	X
SSA	X	X	X
SMA	X	X	X
SMIBA	X	X	O
OPENA	X	X	X
AA	X	X	O
IDCA	X	O	O
MA	X	XO	XO
RA	X	XO	O
ADDITIONAL	X	X	X

Table 4 Agents Used within 3-L

Some of the co-ordination duties are common amongst the three levels and some are specific to a certain level, the table below sums up the division of labour for the 3-L architecture.
X-Provided
O-Not Provided

Security Service	DMC	LSS	LSU
Logging and Auditing Security Relevant Events	X	X	X
Generation, Storage and Distribution of Cryptographic Keys	X	X	X
Domain Notary Service	O	X	O
Inter-Domain Communications	X	O	O
Trusted Entry Point	X	O	O
Domain Communications	O	X	O
Access Control	O	X	O
Authentication	O	X	X

Table 5 3-L Division Of Labour

The 3-L architecture relies heavily upon the use of public-key[9] and secret-key cryptosystems[10]; the public-key cryptographic techniques are used for the authentication/integrity and confidentiality requirements while the secret-key cryptosystems are used to accomplish very fast bulk encryption solely for the purpose of confidentiality. Every user, LSU, LSS and DMC will have their own public cryptographic key pair used for authentication and certified by an LSS, DMC or external TTP. The use of the LSU, LSS and DMC key pairs are for security service management only, while the user's public-key pair is used to attain personal liability and accountability.

Domain Management Centre

The domain management centre is the highest authority within the three layer architecture and is comprised of most agents within the basic CISS model, see table 4. For all intents and purposes it acts as a central entry point to entities requesting associations on the opposite side of a security boundary. The DMC is the only level entity within a security domain that has the direct use of the IDCA. Within the DMC's SMIB each LSS is listed along with its inter domain communication properties. The property list may simply contain an attribute that refers the DMC to the LSS of interest for instructions, or it may be a detailed list of external entities that may form an association with an entity inside the security domain.

For example, a business wishes to place the machines it uses for accounting upon a LAN so that its accountants may access them from remote sites across the domain. However, the administration is worried that the placement of the machines upon the LAN will put them at risk from attackers outside of the organisations security domain. Therefore, a rule is entered into the DMC SMIB that no external associations are allowed with the accounting machines, thus securing them from potential external hackers just as an Internet firewall screens attempted associations.

The DMC is responsible for the implementation of the security policy laid out for a domain. To accomplish this the DMC is capable of modifying the SAA or rather the SMIB entries that determine the security mechanism used by the various protocols within the 3-L architecture by each of the levels, such as digital signature systems or cryptosystems used for confidentiality.

The co-ordination and synchronisation of management keys within the security domain is the primary responsibility of the DMC. The DMC ensures synchronisation by regular distribution of a list of LSS public-key certificates to all LSSs. The certificate list is distributed in off-peak hours so that system performance is not adversely affected. The DMC may demand the generation of new keys from its hosted LSSs according to the security policy of the domain. The time for which keys may be valid will be convenient intervals dependent on the organisation. For example, weekly distributions of the certificate list may be accomplished on Sunday nights, which in all likelihood is the quietest night in most organisations.

Local Security Server

The local security server provides the framework for secure communications across the LAN and has a part in all of the LSU's activities. The LSS can be thought of as a file server and any references to files will be examined by the LSS. It enforces the protocols to be used as indicated by the DMC and subsequently entered into the LSS's SMIB. The DMC provides the LSS with notary services and trusted key certificates. It is envisaged that the DMC should be a physically secured machine to which access and the obtaining of system privileges is restricted to minimal personnel. The DMC should be solely controlled by the security section of the organisation.

The LSS provides secure services such as secure file transfer, notary services, etc. to the LSUs that it hosts.

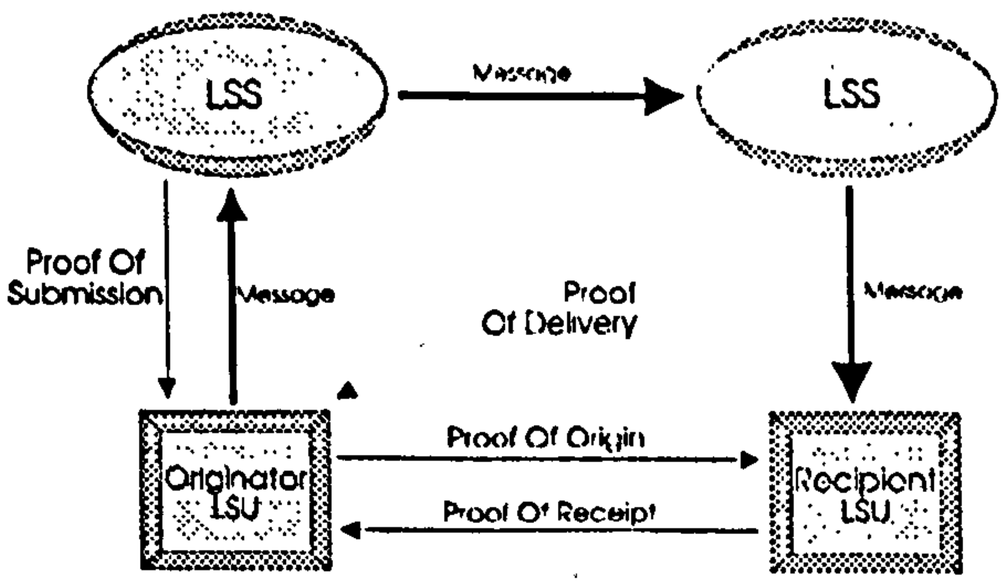


Figure 3 Non-repudiation Routes

For secure communication outside of the security domain the LSU contacts the LSS which then contacts the DMC on the LSU's behalf.

Local Security Unit

The local security unit is situated at the user terminal and is concerned primarily with the confidentiality of data between it and the LSS, and the authentication of users. If dumb terminals are used in networks then due to their lack of processing power and therefore their inability to maintain confidential communications between themselves and their host, they provide a small security risk. In the case of dumb terminals it must be assumed that the physical links between the terminal and the host is secured, either through cryptodevices [11] at either end of the connection, or physical impediments, for example concrete encasement of the line.

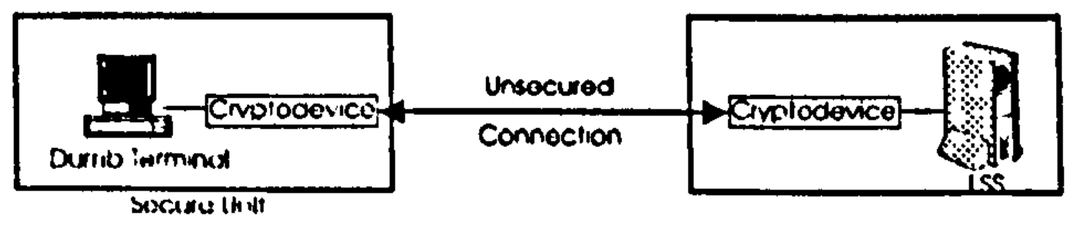


Figure 4 Secure Dumb Terminal

In the event of dumb terminals being used within the 3L architecture the LSU will reside on the host and the security methods for confidentiality mentioned above must be relied upon for secrecy. However, with the downward spiral in the cost of processing power, and the movement towards distributed processing it can be assumed that most user terminals within LANs recently set-up will not be dumb terminals, with the most common approach being the use of PCs. How much processing power these terminals have is extremely variable and dependent upon such things as the computer purchase policy of the company, therefore the architecture of a security system must be flexible enough to cope with different levels in computing technology.

The local security unit provides the processing power for most of the work in the provision of security under CISS. Within the 3-L architecture there are many LSUs within a domain, this division of processing power is a more efficient way of meeting the needs of users than through the use of a large central processing element. Therefore, it is important that the LSU works as quickly and as efficiently as possible.

The bulk of processing done by the LSU will be taken up through the cryptosystems used within the security protocols. Public-key and Secret key systems can be very processor intensive and should therefore be provided by dedicated hardware wherever possible, Fast RSA hardware and cryptographic boards have been designed by the Network Research Group[12], [13] for this very purpose.

The LSU is aware of its LSS's and DMC's public-key as it will have cached a copy of the DMC's signed certificate attesting to the validity of the LSSs public-key.

4. Expansion of the Monitoring Agent

Almost all security features placed on computer systems are preventative. Whether they are smart cards, passwords or biometric identification systems, they are barriers placed in the way of an attacker attempting to gain unauthorised access to a computer system. As useful as these techniques are for access control once an attacker has by-passed them, it becomes redundant. As a consequence conventional computer security systems have repeatedly failed to deal with three major security problems [14]. [15]. [16]:

1. illegal use of a system by an unauthorised user once access controls have been compromised;
2. abuse of privileges by an authorised user. User abuse is potentially the most dangerous security breach as regular users will have a good idea of where to direct attacks for the most damage, or where the most valuable information is kept upon a system;
3. anomalous behaviour of system resources, through presence of logic bombs, bugs, viruses and Trojan horses.

The traditional method for detecting cases one and two above is through the use of audit logs. Many existing operating systems such as VMS [17] offer limited auditing facilities that can be used to detect masqueraders. However, the amount of auditing information that can be generated is vast. Therefore, the information cannot be processed by a single security administrator, and even if it was, discovery of a masquerader would have been made too late to prevent a masquerader from causing any damage.

Computer viruses are the most well known of all malicious software and have become much more complex as their writers have employed more sophisticated techniques in the creation of malicious software.

Perhaps the best method of dealing with the security problems described above is through the continual surveillance of system activity. If the following two premises are true then the development of a system to deal with the security problems discussed is possible.

1. It is possible to learn the normal activity of a system, its resources ,and the users upon it, so that its behaviour can be predicted.
2. A masquerader, privilege abuser, or virus, exhibit anomalous activity that can be detected as not being part of the normal system behaviour and the correct counter measures can be implemented to prevent/limit damage.

The reasoning behind the two premises is as follows.

- The behaviour of a system cannot be unpredictable for any length of time as that would provide no platform for any productive work to be done on it. Therefore, it is possible to predict a systems behaviour

- User behaviour can be categorised due to two prevalent factors: humans form habits that are peculiar to them, for instance there are multiple ways to open a file and once a user uses one way they rarely change even though it may not be the most efficient way; job specialisation, for example a secretary on the system will probably use different system resources than a programmer.

- Anomalous behaviour is any that falls outside of the norm for a specific entity. In the case of a masquerader the control behaviour pattern would be that of the user whose account is being used. It can be assumed that a masquerader will not perform the usual tasks that the user would be expected to since the masquerader is not there to perform as the legal user but instead there for exploration, or malicious damage. Programs should behave in a predictable manner. If they do not, for instance when infected by a stealth virus, they may exhibit file modifications or writes to memory that are uncharacteristic. A potential privilege abuser may show indicative signs by excessive browsing, the attempted/actual attainment of super user status, or the ability to assume another user's id.

A monitoring scheme has three stages.

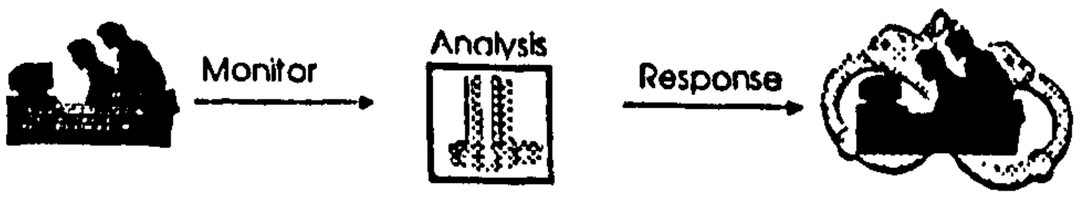


Figure 5 Stages of Intrusion Detection

- Monitor:

  system variables that are capable of being used to indicate anomalous behaviour

must be monitored and stored for historical analysis.

- Analysis:
  using the measurements of the system variables monitored the monitoring scheme must be capable of recognising anomalous behaviour upon the computer system.
- Response:
  depending on the conclusions reached by the analysis of indicators a response must be formed and implemented, limiting the damage to data upon the computer system and the protection of services to users.

The extended monitoring agent allows the generation of a single security log that can be comprised of generic CISS events and specific events for the computing platform upon which CISS exists, i.e. the Operating System.

Through the use of security log entries a history of user activity can be built up. From this information a *profile* can be generated that describes the user/entity's normal activity upon the system. This profile can then be used in the detection of anomalous behaviour during any future user/entity activity.

Profiling can be divided into three sections.

1. Data filtering:
   through the correct data selection the most indicative measures are used while the rest are discarded to reduce processing. This is also a function of the analysis section of the monitoring agent.
2. Profile creation:
   from the filtered measures a control profile is created that indicates the boundaries of normal behaviour.
3. Profile Refinement:
   the profile is continuously refined so that it is up to date.

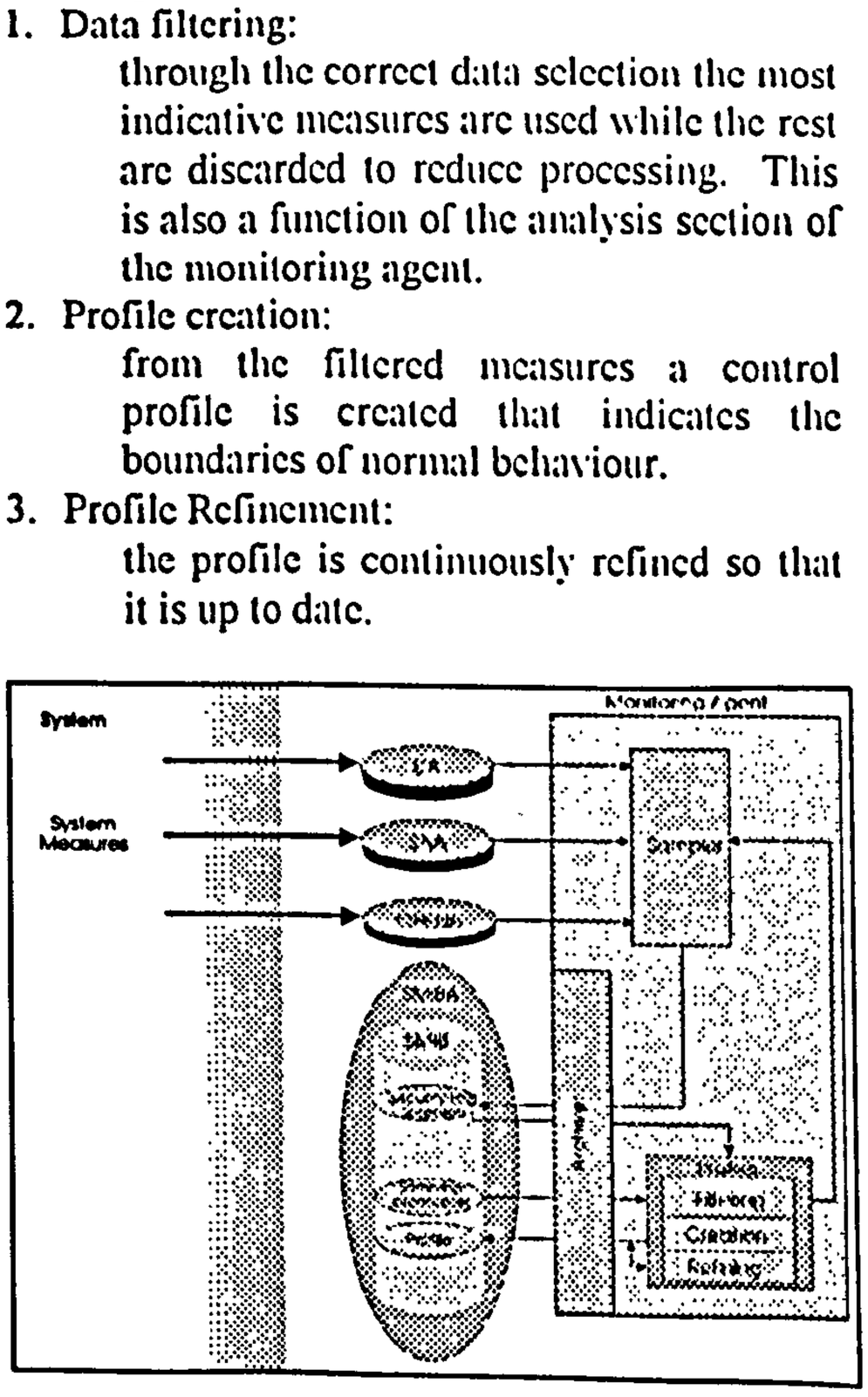


Figure 6 Profiler

The basic CISS *security log* will be made up of two entries the *basic log entry* and the *activity entry*. The basic log entry is made up of fields that identify the user/entity that caused an event, the time at which the event took place, the system resource acted upon, etc. An *activity entry* is a list of basic log entries linked to a single user/entity action such as use of mail tools, text editors etc.

The two security log entries are used by the monitoring system to create profiles- *activity profiles* and *behavioural profiles*. Activity profiles describe a user's/entity's normal activity entries. Behavioural profiles related activities such as session 'start-up' which may be the first activities a user does at the start of each session; login, reading mail, reading subscribed newsgroups.

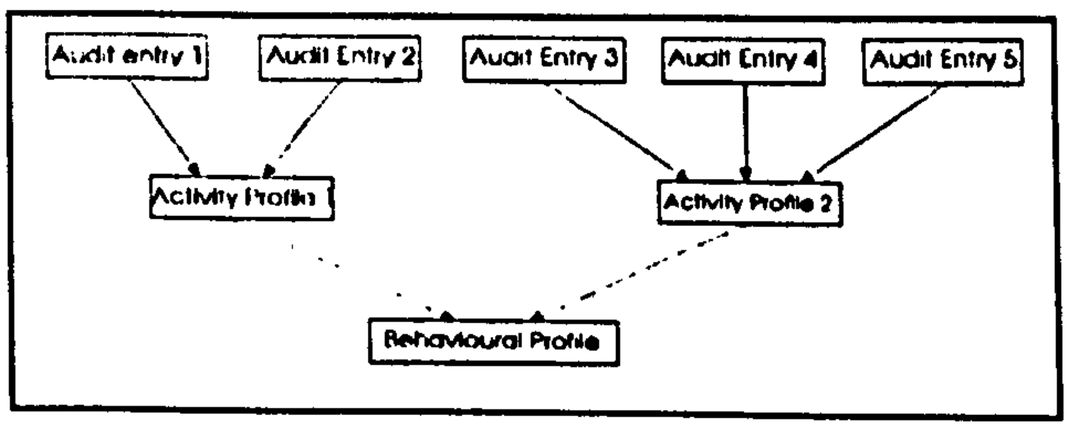


Figure 7 Profile Monitoring

Because profiles are meant to be representative of the users normal activity it is important to keep them up to date. This can be done through the acquisition of sample activity during normal sessions. Automatic updating in this form can be accomplished daily or weekly, depending on the activity of the user. Updating a profile would only take place at the termination of a successful session. This reduces the chance of contamination of a profile by possible masquerader activity. The logical section within the monitoring agent that is responsible for profile refinement is the profile refiner.

The surreptitious updating of profiles can lead to the problem of slow change. If privilege abusers know that profiling is being used with automatic updates they may attempt to slowly modify their behaviour over time, consequently altering their profiles. Thus they may not exhibit anomalous behaviour even when abusing the system. For example, a privilege abuser can gradually introduce file browsing into his normal list of activities and then slowly increase his browsing activities.

The second way to update a profile is for the security administrator to force an update by requesting it through the SAA. This would be required at initial set-up of the system, at the

addition of a new user, or at the addition of new software.

Software profiles will be limited to activities. By attributing a piece of software with an activity profile it should be possible to detect any anomalous writes or reads, that could be indications of a Trojan horse or virus. Such activity when detected by the monitoring scheme's analyser may demand the initiation of a virus scanner specifically targeting the suspect piece of software. If no virus is found the responder may tag the piece of software for the security administrators attention. The security administrator can then use the activity audit entries to examine exactly what the piece of software is doing.

Anomalous activity detection of software using profiling will only work if the knowledge base holds profiles prior to infection. Any infected software introduced into the system for the first time will not be detected as exhibiting anomalous behaviour when the stealth virus delivers its payload. Since most organisations use a limited number of software packages it is possible that standard applications can be profiled and their profile introduced into the knowledge base prior to installation of the software upon the system. These standard profiles can be generated either by the application suppliers or the CISS suppliers.

Computer terminals can be given behaviour profiles so that their actions upon a network can be monitored for anomalous activity. Activities that a terminal may be audited for, and for which a behaviour profile may be generated, include network connections, remote logins, traffic intensity, etc. This sort of profiling can help in the prevention of machines being used for activities such as hacking.

Profiles take time to be generated so the most vulnerable time for a system is when a new entity is placed upon it (user/software). Therefore, until enough information has been gathered for an initial profile standard security mechanisms must suffice.

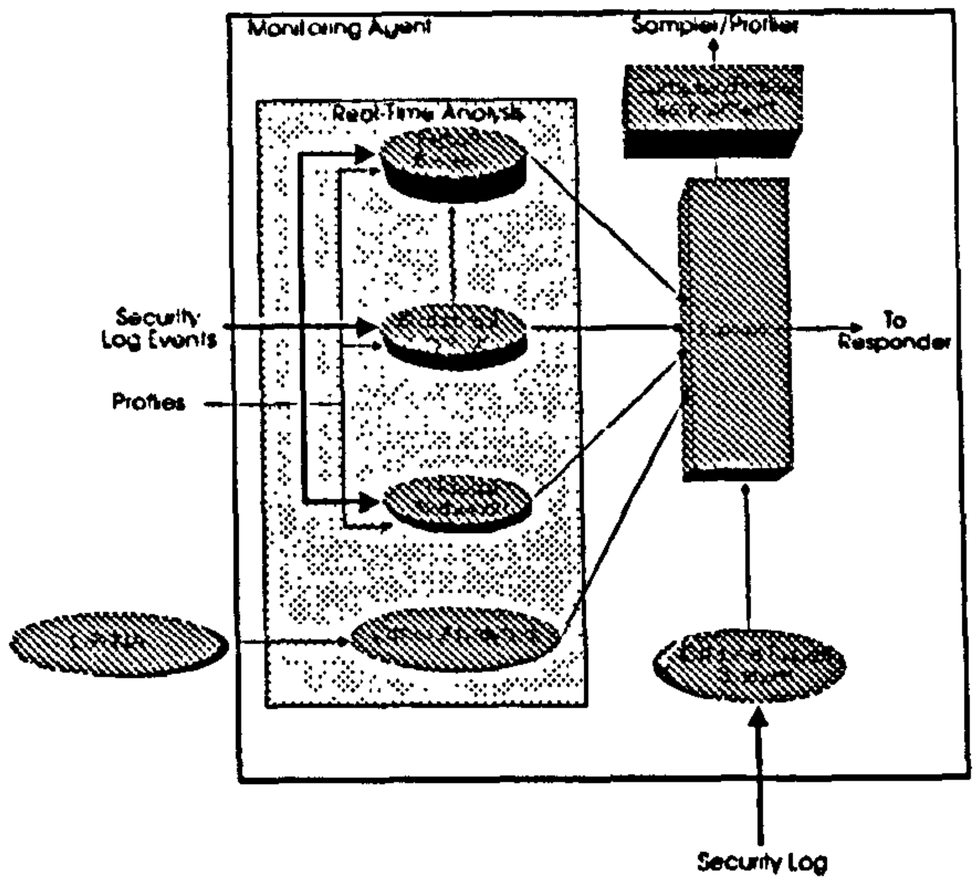


Figure 8 Analysis Within CISS

Within the monitoring agent there should be real-time analysis and off-line analysis. The real time analysis will be made up of the three specified analysers.

1. The task of the expert system will be to detect anomalous behaviour by applying the experience that security experts have imbued into it through the rules that they have generated for its inference engine. The expert system will use the essential rules as dictated by the same experts monitoring in real-time.
2. The task of the statistical analyser is to detect anomalous behaviour of entities through activities outside of their norm as dictated by their profiles. This is more specific than that of the expert system, although the real-time expert system will use the results of statistical analyser in its own analysis.
3. The neural network should be used in very specific cases where the samples that are needed for its training can be collected and the problem is very specific.

As well as the anomaly detection through activity profiles there should be other real time supervisory mechanisms requiring analysis, such as keystroke timings, network traffic analysis, etc. There is also an off-line security log analyser, that is an expert system with a full rule base, that can conduct thorough analysis of the security log. Anomalies detected through these mechanisms are reported to the evaluation block of Figure 8.

The evaluation block can be either a simple sum threshold scheme with logic for further refinement of profiling and sampling, or it can be an expert system programmed to perform a decision on the current security of CISS through the data provided by the monitoring agents real-time analysers.

The evaluation segment of the monitoring agent will have an awareness rating so that the greater the awareness the more sensitive the analyser becomes. For instance, numerous suspicious activity may occur in isolation without necessarily tipping the monitoring agent that a masquerader is present. The frequency of suspicious activity contributes to the awareness value so that continued suspicious activity will eventual cause a response from the system.

5. Conclusion

The division of the CISS agents into separate functional units as described within the 3-L architecture allow CISS to be deployed upon computing platforms of different capacity, thereby reducing the impact in system performance by tailoring the security activities to the capabilities of machines within a heterogeneous computing environment. When each of the level units within the 3-L architecture are equipped with hardware cryptographic capability it is envisaged that the security system will provide quick response and low system impact. A prototype is in development at the Network Research group within the University of Plymouth.

The extended monitoring agent can provide increased security through monitoring system activity. It is important that new techniques be found in the detection of anomalous behaviour upon a system. The increasing frequency of reports in the world news of computer security breaches only lends greater urgency to the development of innovative and effective ways of securing computer resources.

1 "Hacked EC Computer," Guardian, 24th February, 1993.

2 "Freefone Lines Set Off Spate of Data Rape," Sunday Times, 14th November, 1993.

3 Audit Commission, "Survey of Computer Fraud & Abuse," 1990.

4 S.Muftic, A.Patel, P.Sanders, R.Colon, J.Heijnsdijk and U.Pulkkinen, "Security Architecture For Open Distributed Systems," Wiley, 1993

5 International Standards Organisation, "OSI RM Part 2: Security Architecture," ISO DIS 7498-2, 1988.

6 International Standard Organization, "Use of Encipherment Techniques in Communication Architecture," ISO/TC 97/SC 20/WG 3, 1986.

7 CCITT, "Data communication Networks Directory, X500-X521," 1989.

8 F.M.Avolio, "Building Internet Firewalls," Business Communications Review, Vol 24, pp15-20, 1994.

9 J.Neehvatal, "Contemporary Cryptography: Public Key Cryptography," IEEE Press, 1991.

10 B.Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C," Wiley, 1994.

11 D.Barnes, "Designing More Secure LANs," Telecommunications, vol 27, pp64&98, 1993.

12 P.Onions, "A High Speed Integrated Circuit for Applications to RSA Cryptography," University of Plymouth, Ph.D. Thesis, 1995.

13 J.P.Morrissey, "A hardware implementation of the CISS Concept," University of Plymouth, Ph.D. Thesis, 1995.

14 J.B.Madsen, "The Greatest Cracker-Case in Denmark: The Detecting, Tracing and Arresting of Two International Crackers," Proc USENIX, pp17-40, 1992..

15 B.Clough and P.Mungo, "Approaching Zero: Data Crime and the Computer Underworld," Faber and Faber, 1992.

16 Neil Fawcett, "Microsoft Gives Virus to 200 Top UK Developers," Computer Weekly, 23rd February, 1995..

17 C.Sandler, "User's guide to the Vax/VMS operating System," Scott, Foresman and Company, 1989.

User Identification by Static Keystroke Analysis

J.P.Morrissey, P.W.Sanders and C.T.Stockel
Network Research Group, Faculty of Technology,
University of Plymouth, Plymouth.
E-mail: josephm@soc.plym.ac.uk

Abstract

This paper gives an introduction into computer system user identification and presents the results of an investigation into the application of a multi-layer perceptron and a genetic algorithm to user identification. The multi-layer perceptron was trained to recognise the keystroke timings of 4 static text strings from 15 users. The genetic algorithm was applied to optimising the data used to train the multi-layer perceptron and identification of the user. The security mechanism investigated can be easily added to any computer system to increase 'password' security.

Keywords : Perceptron, Genetic Algorithm, Keystroke Analysis, Authentication.

1.Introduction

Secured computer systems aim to protect the data processing services they provide and the data they store within memory. The first line of defence that any computer system has against malicious attacks is to deny access to potential attackers. The only way that this can be done is to identify the person that is attempting to log onto the system, and thereby determine whether they have a legitimate right to services.

2.User Authentication

An individual possesses three qualities that may be used to identify them [1]:

- what the individual knows;
- what the individual owns physically;
- what the individual is.

Any or all of these qualities can be used in a user authentication mechanism, with the strongest authentication mechanisms using more than one of these qualities.

What They Know

Authentication systems exploiting this quality generally require the individual to know a secret password which must be presented when challenged. Other systems may use a database that contains detailed personal knowledge about its users, whenever someone attempts to login they are asked random questions, such as, birthdates of offspring, names of relatives, favourite item of music, and the answer given is compared to those in the database.

The password is the most common authentication mechanism in use at present. When a user logs into a computer system he is usually challenged for a password, this password is kept in a database indexed by the user's login name. If users choose their passwords according to common security policy [2], then the password mechanism is very secure, especially when augmented with a limited amount of unsuccessful logon attempts (i.e. a "three tries and you are out" policy).

However, it has been shown in various studies [3], that no matter how theoretically secure a password mechanism is, its overall security is compromised very easily by:

- users picking easily guessable words such as, "password," "drowssap," or using words derived from their login names, or found in dictionaries. Dictionary words are particularly dangerous because programs exist to crack user accounts [4], by simply using an exhaustive search technique to obtain passwords;
- because some users communicate their passwords to other users, or write them down near their terminals, for example the BT [5] case.

Even with its faults, the password mechanism is still in wide use as the sole form of user authentication on many computer systems. The low cost of implementing a password mechanism and the familiarity users have with the mechanism makes certain that for the foreseeable future it will continue to be the most popular form of authentication.

What They Own

In this situation the legitimate user will have a token that must be presented when challenged. The token is usually in the form of a magnetic stripe card or increasingly a smart card [6]. The difference between the two cards is a matter of intelligence, the smart card, as its name implies, has some intelligence in the form of on-board processing capabilities. The present smart card technology is split into two main areas: active cards and second generation cards. Active cards are meant to be used in a stand alone mode where power and processing is done on-board, where as the second generation cards are passive and

require a card reader. However, most cards in use fall into the area between the two.

The use of tokens has increased in everyday life. with almost everybody carrying one (e.g. Automatic Teller Machines, A.T.M. cards). Tokens used in an authentication mechanism require the purchase of specialised equipment. For example in the case of magnetic stripe cards, not only must a magnetic stripe card reader be purchased for all physical locations where user authentication is deemed necessary, but so must cards for all users on the system. This initial outlay of capital on specialised equipment can make a token scheme prohibitive in most environments except those that require high security, such as government installations. banks. etc.

With the continuing reduction of processor size, smart cards are being used in 'see through' authentication mechanisms [7]. that do not necessarily require a specialised reader but require the purchasing of smart cards for all users.

Tokens used in an authentication mechanism can provide good security. but users are fallible and may lose or forget cards. The token may also be stolen, seriously compromising security.

What They Are

When a biological characteristic can be used to identify an individual it is called a *biometric*. Biometrics can be further divided into two groups, *active* and *passive*. A passive biometric is a physical characteristic that may be scanned with no effort by the subject other than the exhibition of said characteristic, e.g. finger-prints, or facial features. An active biometric could be hand writing [8]. or voice patterns [9].

In a biometric authentication system an attacker can only hope to imitate the characteristic that the mechanism uses to identify legitimate users. Some of the biometrics that have been researched are human faces, speech, finger prints, body odour and iris patterns. Due to the technologies employed in biometric measurement the field is mainly experimental and has not found many applications in existing authentication mechanisms.

However, biometrics offer potentially the best form of user authentication since they measure actual characteristics of what a user is. As with any authentication mechanism they are not fool proof. In passive biometric authentication mechanisms the user must present a part of the body to identify themselves. It is of course possible for an attacker to obtain the necessary body part. However, the effort and will on the part of the attacker must be far greater than that necessary, for example to steal a slip of paper with a user's password on it. It is also possible for two people to have similar active characteristics and therefore fool an active biometric authentication mechanism, but, if an attacker happens to have a dissimilar active biometric they must make an extreme effort to mimic the user.

Biometrics used in an authentication mechanism offer several advantages:

- requires very little training for users to use;
- there is no risk of losing, or having a token stolen as there is in a token mechanism;
- no risk of having an item of knowledge (password) becoming known.

However. biometrics also have several disadvantages:

- all passive systems require processing equipment either audio, visual or chemical that can be very expensive;
- the processing of biometric data can be very processer intensive;
- the information to deceive a system may be acquired from the user through sampling;
- they are never perfect, recognition of a subject is generally given as a degree of confidence.

Biometric authentication systems utilise the only measurement that actually tests the identity of a person. It can be argued that the first two qualities merely identify the knowledge of a password and the token itself, not the person themselves.

This paper investigates and discusses a cheap solution for an active biometric security mechanism, that can easily be introduced into most existing systems using the most common authentication mechanism - the password.

3.Keystroke Analysis

In 1975 Spillane [10], proposed that just like handwriting, typing styles were an active biometric measurement. When analysing typing, various factors can be measured, for example:

- inter-keystroke timing (the timing between consecutive key depressions);
- depression time (the length of time a key is depressed);
- pressure of key depression.

Further research [11], [12], [13] into the area of typing analysis has confirmed Spillane's proposal. An advantage of typing analysis over other biometric systems is that it minimises some of the general disadvantages present in them:

1. it potentially requires no extra equipment;
2. most of the data processing can be done in an initial learning phase.

Typing analysis can work in one of two ways.

Static analysis is conducted upon text strings that do not change. There are two points at which such analysis could significantly augment the security of a computer system; in the initial authentication of a user (i.e. at login) and whenever a volatile command is used. At *login* there are usually two points that are ideal for static typing analysis, at the entry of *user name* and the corresponding *password*. A volatile command is one which when used can be potentially harmful. They are commands that generally delete or modify data stored within computer memory, such as the 'deltree' or 'del' commands in MS-DOS [14].

Dynamic analysis is conducted upon previously unknown text strings that are entered by a user. The entry characteristics can be stored and analysed later or they may be analysed in real-time, allowing continuous supervision of a session.

The effectiveness of a user authentication mechanism can be measured using two figures; the *false rejection rate* (FRR) and the *false acceptance rate* (FAR). The FRR represents the percentage of legitimate users actually classified by the user authentication mechanism as attackers. The FAR is the percentage of attackers classified by the mechanism as legitimate users.

The investigation conducted into typing analysis can be split into 3 phases:

1. sampling;
2. classification of typing samples;
3. data optimisation.

4.Sampling

The typing samples collected for the investigation were taken using an IBM PC with a 386-33MHz DX processor using a standard UK configuration Qwerty keyboard. Through examination of other research in the area of typing analysis [11], [12], [13], [15], [16], it was decided that the inter-keystroke timing was the best discriminator of people's typing styles, and one of the few useful typing characteristics for static analysis. The sampling program was written in C, utilising the DOS keyboard interrupts to time the intervals between key depressions.

Every subject that took part in the experiment entered four set reference text strings. The four strings were as follows:

REF1 : "*PRINT*";
REF2 : "*TRANSFERENCE*";
REF3 : "*RED SKY AT NIGHT*";
REF4 : "*RUGBY PLAYERS ODD SHAPED BALLS*";

At the end of each text entry the subjects had to depress the *return* key.

The text strings were chosen to make good use of the keyboard, such as cross digraphs that use both halves of the keyboard, REF1 was chosen for familiarity to the test subjects, as it is one of the most common words that a computer literate person will type, all four reference texts use the three main rows of keys on a Qwerty keyboard and REF4 is unusual in the combination of words and digraphs.

The subjects were forced to use static text strings to make classification of their samples as difficult as possible; common sense dictates that typing styles are exaggerated when users type in familiar text strings such as their names. The choice of set strings of text represents a worst case scenario for typing analysis of passwords (where all users have the same password) and the real situation of volatile commands that are set by the operating system being used and common to all users. The four text strings were also graduated in length to evaluate what effect it had on classification of samples.

Before the test subjects began typing they were told:

1. speed was a factor;
2. any mistakes made during entry would cause the entry to be invalid and they would have to enter the whole text string again.

Fifteen subjects took part in the experiment, who were all reasonably familiar with typing, coming from a computing background. Every subject entered each text string 35 times, making all entries at one 10-15 minute sitting. The inter-keystroke times were recorded by the sampling program and stored within files.

5.Classification

The reason for choosing the multi-layer perceptron for pattern recognition lies in the fact

that it is one of the few classification techniques that can solve problems involving non-linear separability, and the potentially infinite variety of individual typing styles suggests that typing analysis is a non linear problem.

In 1943 McCulloch and Pitts proposed a model of a biological neuron. It was not till 1962 that simple collections of the basic neuron model became known as *Perceptrons* [17]. A multi-layer perceptron has an input layer, an output, and at least one hidden layer which is not directly connected to the outside world. The topology of a multi-layer perceptron can be seen in Figure 1.

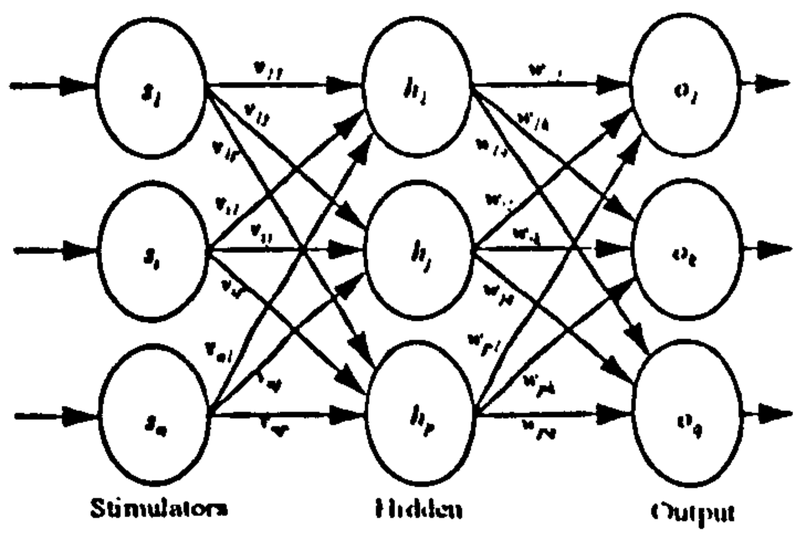


Figure 1 Multi-Layer Perceptron Model

The learning algorithm employed for the multi-layer perceptron used in the experiment was the 'back-propagation' algorithm [18]. The back propagation algorithm tries to minimise the error of each node's output. The algorithm starts calculating errors from the output layer and then propagates the error back to the hidden layer till the errors for all nodes is known.

A multi-layer perceptron network was created for each REF text for every test subject. Each test subject's networks were taught to distinguish their keystroke timings from the other test subjects. The neural net, if working perfectly, would give an output of 1 if the specific user's timings were the input vector, and a 0 if the input vector was one of the other 14 test subjects. However, multi-layer perceptrons rarely give perfect outputs so a threshold scheme had to be set. It was decided to evaluate the efficacy of three threshold values 0.7, 0.8, and 0.9, any value output by the network above these thresholds was taken as recognition of a specific test subject.

The basic topology of the multi-layer perceptron was:

- input nodes equal to the number of digraphs in the REF text;
- one hidden layer, with a number of nodes equal to twice the input nodes;
- a single output node.

The neural net was trained with 4000 presentations of training samples. The training vector used for each cycle was chosen randomly from the training set. The networks were evaluated with the final 10 typing samples of each subject, giving for test purposes 150 legitimate attempts and 2100 attempts of illegitimate access.

	0.7	0.8	0.9
REF 1	24	21	15
REF 2	8	6	5
REF 3	8	6	5
REF 4	4	3	2
Average	11	9	6

FAR given as %

	0.7	0.8	0.9
REF 1	10	15	26
REF2	15	19	25
REF 3	21	23	27
REF 4	28	32	40
Average	19	22	30

FRR given as %

Table 1 Results

The low FAR makes the network appear more suitable for login security, as the FRR is still a little high for use in a volatile command security mechanism.

Since the neural network analysis seemed more suited to a login security mechanism it was decided to investigate the use of two reference texts, thus emulating the login process where a username and password are used. This is a worse case scenario where all users have the same name and password.

Two neural nets were employed in this two reference strategy, one for the login name and the other for the password. Every attempt of authentication to the system is placed into one of three categories depending on the combined result of the neural nets.

- (R)ejected - both samples were rejected by their respective neural networks.
- (L)ow - one of the two samples was rejected.
- (H)igh - Both samples were accepted

REF 1 and REF 2

	R	L	H
0.7	72	26	2
0.8	76	22	2
0.9	82	17	1

Impostor attempts (%)

	R	L	II
0.7	3	29	68
0.8	4	33	63
0.9	9	39	52

User Attempts (%)

Table 2 Two Reference Results (REF1 and REF2)

REF 2 and REF 3

	R	L	II
0.7	87	13	1
0.8	89	11	0
0.9	92	8	0

Impostor Attempts (%)

	R	L	II
0.7	4	38	58
0.8	5	40	55
0.9	7	50	43

User Attempts (%)

Table 3 Two Reference Results (REF2 and REF3)

As can be seen, it is possible to increase the security significantly, using a threshold value of 0.9 and a requirement of at least an L category result, the FRR can be brought down to 7% and the FAR to 8%.

6.Data Optimisation

The next part of the investigation was an attempt to improve the results of the basic multi-layer perceptron through optimisation of the input vectors. Starting from an initial conjecture that:

"each individual has specific digraphs that are particularly characteristic of their typing style,"

the possibility of identifying characteristic digraphs was researched.

As with the perceptron the genetic algorithm (GA), came about through research into an aspect of the real world: evolution, and more specifically the mechanism behind it, that provides the necessary small increments in performance for natural selection to work, genetics. Evolution is a natural sorting mechanism that seeks to optimise performance of living entities, it relies upon the fact that the more successful individuals of any group will pass on their genetic information to their children. In the process of breeding there is a chance that the genetic material passed onto the next generation will be mutated, introducing a characteristic that was not previously present in the groups genetic pool. If this mutation is a contributing factor to the success of an individual then it is likely to be passed onto the next generation.

Genetic algorithms are slightly different from other algorithms that perform search and optimisation procedures in several ways:

- GAs conduct their search from several points and not from a single point;
- GAs use probabilistic rules and not deterministic ones;
- GAs need only know the objective function and no other auxiliary information;
- GAs need only a coding of the parameter set and not the parameters themselves.

The experiments into the optimisation of digraph selection was conducted on four subjects chosen at random. It was necessary to limit the number of subjects because of the length of time it took to calculate the fitness (how well that member performed i.e. its ability to correctly classify a typing sample) for each member of a generation. Although only four subjects were optimised the entire training set was used to train the neural nets and to evaluate the GA selection of digraphs.

The basic results: using every digraph and each reference with the basic neural net topology and a training cycle of 4000 presentations are:

	0.7	0.8	0.9
REF 1	14	8	4
REF 2	7	6	5
REF 3	3	3	2
REF 4	3	3	2
Average	7	5	3

FAR given as %

	0.7	0.8	0.9
REF 1	31	42	59
REF 2	17	19	25
REF 3	40	49	55
REF 4	23	27	38
Average	28	34	44

FRR given as %

Table 4 GA Basic Results

One of the subjects chosen was the most inconsistent typists from the test subjects, it was their inconsistency that provided the high FAR for REF 3.

In the first experiment a population of 12 was used and the number of generations that the genetic algorithm went on for was 20. The results of the optimisation can be seen below

	0.7	0.8	0.9
REF 1	14	8	6
REF 2	7	7	5
REF 3	8	6	4
REF 4	2	2	1
Average	8	6	4

FAR

	0.7	0.8	0.9
REF 1	32	44	54
REF 2	12	15	30
REF 3	29	32	43
REF 4	16	20	29
Average	22	28	39

FRR

Table 5 GA Exp 1 Results

	Digraphs used	Max Digraphs
REF 1	4	5
REF 2	7	12
REF 3	10	16
REF 4	16	30

Table 6 GA Exp1 Digraphs

The results show an overall improvement in the performance of the neural net when using GA selected digraphs. However, in the case of REF1 there were not enough digraphs present to allow a proper selection by the genetic algorithm, leading to results almost equivalent to that attained through using all digraphs. The best performance by the genetic algorithm was in the selection of digraphs from REF 4, the FRR was reduced by 8% on average and the FAR by 1%. This improvement can be explained through the large choice of digraphs, allowing the best digraphs to be picked that minimised both the FRR and the FAR.

In experiment two a population of 12 was used and the number of generations that the genetic algorithm went on for was 40. However, this time the genetic algorithms fitness was adjusted in respect to how few digraphs were used for identification.

	0.7	0.8	0.9
REF 1	20	9	5
REF 2	11	9	7
REF 3	8	7	4
REF 4	3	3	2
Average	11	7	5

FAR

	0.7	0.8	0.9
REF 1	31	53	63
REF 2	19	26	39
REF 3	29	32	41
REF 4	23	26	37
Average	26	34	45

FRR

Table 7 GA Exp2 Results

	Digraphs used	Max Digraphs
REF 1	2	5
REF 2	6	12
REF 3	7	16
REF 4	12	30

Table 9 GA Exp2 Digraphs

7.Conclusion

The results from using just the neural network with no data optimisation are very encouraging. It must be remembered that this method of user authentication is not suggested as a total solution but is meant as a way of increasing the level of security on a password mechanism cheaply and relatively painlessly in terms of the complexity of adding it to the system. The experiment was implemented in such a way as to make life as difficult as possible for the classifier, and from the results obtained the multi-layer perceptron has coped very well.

Two conclusions that can be drawn from the results given by the neural net are:

1. that the optimum text length for identification purposes involving keystroke analysis is greater than 5 and less than 31, optimally around 13-20 letters long;
2. the combination of two texts in a login phase would provide adequate security for almost any login mechanism.

As can be seen by the results of the data optimisation the application of a genetic algorithm can reduce the number of digraphs required to identify a user. The results showed a general improvement over the straight classification with no data optimisation.

The results from this investigation are encouraging in that they show considerable accuracy without too much refinement for a worst case scenario. The algorithm worked very well when multiple reference texts were used. Therefore, this authentication mechanism can be used within the login scenario with little more refinement.

However, the second static use proposed (the entry of a volatile command) requires more thought. If the suggested authentication mechanism was used the user would have to give multiple samples for each volatile command that would use this authentication mechanism. No user would want to provide the number of samples required, so alternative strategies must be thought of, two examples are given below.

1. The user when using a volatile command is challenged for his password. This may sound as if it would become annoying. However, windows and UNIX already challenge as to the surety of their command, it would take little adaptation to adjust these OSs to request a password instead.

2. The user is allowed to initiate a volatile command as many times as it takes for the authentication mechanism to obtain enough samples for the teaching of its neural net. Thereafter the authentication mechanism will examine the keystroke timings of the volatile command entry.

1 H.M.Woods, "The Use of Passwords for Controlled Access to Computer Resources," National Bureau of Standards Special Publication 500-9, 1977.

2 W.G.deRu and J.H.P.Eloff,"Improved Password Mechanisms Through Expert System Technology," Proc 9th Annual Computer Security Applications Conference 93, pp272-280, 1993.

3 B.L.Riddle, M.S.Miron and J.A.Semo, "Passwords in Use in a University Timesharing Environment," Computers and Security, Vol 8, no 7, pp 569-579, 1989.

4 D.Seeley, "A Tour of the Worm," Proc Winter Usenix Conference, pp287, 1989.

5 H.Johnstone, "BT Hacker Breaks into Security Files," The Times, 24th November, 1994.

6 R.Bright, "Smart Cards: Principles, Practice, Applications," Ellis Horwood, 1988.

7 A.P.Conn, J.H.Parodi and M.Taylor, "The Place of Biometrics in a User Authentication Taxonomy," Digital Equipment Corporation, pp 72-79, June 20, 1990.

8 N.M.Herbst and C.N.Liu, "Automatic Signature Verification Based on Accelerometry," IBM Journal of Research And Development, pp245-253, 1977.

9 C.Z.Bas, Y.Zhenli and Z.Lihe, "Automatic Speaker Verification Using the Neural Network and Combined LPC Parameters," Proc TENCON 93, pp345-347, 1993.

10 R.J.Spillane, "Keyboard Apparatus For Personal Identification," IBM Technical Disclosure Bulletin, 1975.

11 D.Umphress and G.Williams, "Identity Verification Through Keyboard Characteristics," Journal of Man-Machine Studies, Vol 23, pp263-273, 1985.

12 J.Leggett and G.Williams, "Verifying Identity via Keystroke Characterstics," journal of Man-Machine Studies, Vol 28, pp67-76, 1988.

13 M.Brown and S.J.Rogers, "User Identification via Keystroke Characteristics of

Typed Names Using Neural Networks." Journal of Man-Machine Studies, Vol 39, pp999-1014, 1993.

14 Microsoft Corporation, "Microsoft MS-DOS6.22: User's Guide," 1994.

15 S.A.Bleha and M.S.Obaidat, "Computer Users Verification Using the Perceptron Algorithm," IEEE Transactions On Systems, Man, and Cybernetics, Vol 23, no3, pp900-902, 1993.

16 M.S.Obaidat, "A Comparative Performance Study of Neural Network Paradigms for Identifying Computer Users." Proc IEEE Computers and Communications, pp161-167, 1994.

17 F.Rosenblatt, "Principles of Neurodynamics," Spartan, 1962..

18 J.L.McClelland and D.E.Rumelhart, "Parallel Distributed Processing, Volume 1," MIT Bradforf Press, 1986.