CHAPTER 1

Some Introductory Words

1. Genesis and Goals of the Thesis

Cryptography, the study of secret writing, has a long history which until recently has been primarily associated with military and government needs. Cryptography has two major purposes. The first is the provision of *privacy* by concealing the contents of some message from *eavesdroppers* who are not intended to be able to read it (Fig 1.1). This has been the traditional application of cryptography.

Fig 1.1 - The Need for Privacy

However, with the increasing use of electronic media, there has been a developing need for an electronic equivalent of the written signature. This addresses the need for *authentication*, of proving who has sent a given message, such as legal contracts, letters, etc. in a manner that they cannot subsequently repudiate (Fig 1.2). Both of these aspects are important in modern commercial cryptography.

Fig 1.2 - The Need for Authentication

Traditional cryptographic schemes have relied on a series of *substitution* (where letters or words are replaced by others), and *transposition* or *permutation* (where the order of letters or words is changed) operations to conceal the message. The particular substitution or transposition used is controlled by a *key*. This key is used by both the sender and the recipient of the concealed mesage, and hence has to be kept secret in order to protect the secrecy of the message. Such schemes are called *Private-Key Cryptosystems* for this reason (Fig 1.3). The need to exchange keys via a trusted courier, and to keep them secret, was manageable whilst the communications channels were restricted to well defined military or diplomatic hierarchies.

However the explosion of electronic communications media, the rise of Electronics Funds Transfer (EFT), and the need for secure communications in business and the public in electronic mail schemes, have all lead to a need for some alternative scheme where this previous exchange of keys is not needed. Since the number of keys which would need to be exchanged grows as the square of the number of participants, this soon becomes infeasible. What is needed is the ability to be able to publish an encryption key (used either to conceal the message, or to verify a signature) in the equivalent of a telephone directory, without compromising the decryption key (used to restore the message or create a signature), and hence the security of the message. Such schemes are called *Public-Key Cryptosystems*, or *Two-Key Cryptosystems* (Fig 1.4). They were originally proposed in a pivotal paper by Diffie and Hellman [DiH76], who developed these concepts, and proposed a possible scheme; and also independently by Merkle [Mer78] who outlined the basic concepts, but did not propose a feasible system.

Fig 1.3 - Private-Key Cryptosystems

Ideally, public-key cryptosystems should be able to provide all the features desired in a secure communications system. However, as shown in Chapter 3, limitations in the implementation speed of such schemes at present restrict their usefulness to authentication and key exchange (important though these are). Consequently there is a need for hybrid schemes which use a public-key cryptosystem to exchange session keys and to sign messages, and a private-key cryptosystem to efficiently encrypt the message for privacy. This implies a need for good modern private keys schemes, preferably a number of different schemes which may be employed in applications of varying sensitivity in the commercial and government confidential areas. Current schemes in use are either propriety, or are felt to have key sizes which are too small. The remainder of this thesis details the development of a good new private key scheme, which is the major achievement of this thesis.

Fig 1.4 - Public-Key Cryptosystems

To achieve this, the Data Encryption Standard (DES) [NBS77] is analysed in detail as a source of insights to be used in the design of the new cipher. DES has achieved wide utilization, particularly in the banking and electronic funds transfer areas, where it is now an Australian standard [ASA85]. At the time of its introduction there was a considerable amount of controversy, primarily with the choice of a 56-bit key [DiH77], [Hel79a]. This was on the grounds that such a key length was small enough to be exhaustively searched on machines which could be constructed in the foreseeable future. With the march of computer technology, this risk appears to be rising. However, on all other accounts DES appears to be an excellent cryptosystem. The DES has withstood a concerted analysis by the open research community, since its adoption in 1977. The most successful attack to date, that of Biham and Shamir [BiS90] still performs worse than an exhaustive key-space search on the DES with the full 16 rounds.

With the current significant use of DES (especially in banking), there is interest in designing and building a DES-type cryptosystem with an extended key length, for several reasons. These include:

- Doubts about the ability of DES to withstand an attack based on exhaustive key-space searches on specialised hardware (cf. the decision of the US NBS to withdraw certification of DES for all but banking applications). This may be overcome by increasing the key length.
- The difficulty in obtaining DES systems outside the US, due to export restrictions.
- The desire to develop a scheme for which all the design criteria used are known, and hence open to criticism.

Given these reasons, and the confidence with which the DES is otherwise regarded, it was felt to be suitable for analysis to derive some design rules. Whilst the DES algorithm is public, its design criteria are classified. This thesis examines the design of the DES data encryption standard, and develops some possible design criteria for it in Chapters 4, 5 and 6. Using these criteria, along with insights based on generic substitution-permutation cipher construction, suggestions and support from the authors colleagues, and with some results on substitution box design by Pieprzyk and Seberry [PiF89], [Pie90], [Pie89], [PiS89], a new group of private key schemes, the LOKI family is designed. This development is detailed in Chapter 7. A preliminary analysis of one of the members of this family is then given. The appendices of this thesis present the detailed results from the analysis of one of the LOKI ciphers; the code used for the dependency analysis programs; and a sample LOKI cipher implementation.

2. Content of the Thesis

2.1. Main Thesis

The detailed contents of each chapter are as follows.

Chapter 1 is this overview of the thesis.

Chapter 2 surveys the field of cryptography, providing an overview of the modern private and public key cryptosystems used to provide privacy and authentication. It contrasts their strengths and weaknesses, and briefly describes a number of schemes either in use, or proposed for use. These algorithms form the basis against which further algorithm development will be compared.

Chapter 3 examines the RSA public key cryptosystem, which at present is generating considerable interest as a scheme with a wide range of potential uses. Indeed, it has already been adopted to provide authentication in EFT application in the UK. However, its practical use for providing secrecy has been limited because of the time taken to perform its essential encryption and decryption functions. Whilst the times achieved to date are sufficiently short for key exchange and authentication schemes, they are not fast enough for encrypting data to provide privacy. Thus in the near term, practical schemes will very likely use hybrid systems. These use a public key algorithm to exchange session keys, and to provide authentication and non-repudiation of messages, and a fast private key scheme to provide privacy. This chapter concludes that whilst public key schemes would seem to be the obvious realm of research into cryptographic algorithms because of their advantages in public networks, present limitations indicate that good private key schemes are still required. This chapter includes material previously presented to the "Secure Data Communications Workshop", organized by the IEEE Communications Section, Victorian Branch, held at the Town House, Melbourne, on 31st July 1987 [Bro87].

Chapter 4 examines the Data Encryption Standard DES. This is the most widely used private key encryption system at present, especially in the financial industry. Though there were some initial doubts about the design of DES, it has withstood considerable analysis by the open research community since its adoption in 1977. The design of the DES is regarded as sound, though the size of key is thought to be too small. Whilst DES is a standard, the design criteria used in its development have been classified by the US government. Because of this, and of increasing doubts about the ability of DES to withstand an attack based on an exhaustive key-space search using specialised hardware, there is considerable interest in developing an encryption scheme for which the design rules used are known, and hence open to analysis and criticism. This is the primary goal of this thesis. In order to achieve this, a detailed examination of the DES is done, since its design has withstood the test of time. This chapter reviews what is known about the design criteria for the S-boxes, P-boxes, and key scheduling in the current DES. It then presents some initial observations by the author on possible design rules for the permutations P, $PC2$, and the key rotation schedule KS in the DES. It then indicates how this information could be used to design new private key schemes with either full 64-bit keys (instead of the 56-bits used in the DES), or with double length keys of 128-bits. This chapter includes material previously presented to IFIP Sec'88, held on the Gold Coast, Australia, in May 1988, and subsequently published in [Bro89].

The next two chapters examine two key components of the DES structure in more detail: the permutations P and $PC2$ and the key schedule KS.

Chapter 5 develops some design criteria for the permutation P in a DES style cryptosystem. This permutation provides the diffusion component of a substitution-permutation network. Some empirical rules which seem to account for the derivation of the permutation used in the DES are first presented. The author then notes that these permutations may be regarded as latin-squares which link the outputs of S-boxes to their inputs at the next stage. A subset of these with a regular structure, and which perform well in a dependency analysis are then presented. The author then generalises the design rules for permutation P. These rules will be used later in Chapter 7 in the design of the LOKI algorithm. This chapter includes material previously presented to Eurocrypt '89 in Houthalen, Belgium, in April 1989, and published in [BrS90a].

Chapter 6 develops some design criteria for the key schedule in a DES style cryptosystem. The key schedule involves a Key Rotation component KS, and the permutation $PC2$. Together these provide for a diffusion of dependency of ciphertext bits on key bits. Some empirical rules which seem to account for the derivation of the key schedule used in the DES are first presented. A number of trials were run with various key schedules, and some further design rules were derived. An alternative form of key schedule was then tested. This used either a null $PC2$, or one in which permutations only occurred within the inputs to a given S-box, and a much larger rotation schedule than used in the DES. This was found to be as effective as the key schedule used in the current DES, and is proposed for use later in Chapter 7 in the design of the LOKI algorithm. This chapter includes material previously presented to Auscrypt '90, in Sydney, Australia, in January 1990, and published in [BrS90b].

Chapter 7 presents the major achievement of this thesis. It provides an overview of the LOKI encryption algorithm which may be used to encrypt and decrypt a 64-bit block of data using a 64-bit key. It was developed by the author and his colleagues Professor Jennifer Seberry and Dr. Josef Pieprzyk. The structure of the LOKI encryption algorithm was designed by the author based on the general design principles developed in the previous chapters. The design principles for selecting S-boxes developed by Pieprzyk and Seberry [PiF89], [Pie90], [Pie89], [PiS89] were used in the design of the S-boxes used in LOKI. This chapter completely details the proposed publicly standardised version of the

LOKI cipher. It also describes how customised versions of the cipher may be constructed which are believed to have the same degree of security as the standard version of the cipher. The author then summarises the encouraging results from the initial testing of the LOKI cipher, details of which are given in Appendix 1. The LOKI cipher may be used in any mode of operation currently defined for DES, with which it is interface compatible [AAA83]. In addition, the author has adapted two modes of operation of LOKI which compute a 64-bit or 128-bit Message Authentication Code (or hash values) respectively, from previous work by Davies and Price [DaP89], Winternitz [Win83], and Quisquater and Girault [QuG90]. These modes of operation may be used to provide authentication of a communications session, or of data files. Finally the author presents some observations about the likely performance of the LOKI cipher in both software and hardware implementations, which show that it compares extremely favorably against the alternatives. This chapter includes material previously presented to Auscrypt '90, in Sydney, Australia, in January 1990, and published in [BrS90b]. Versions of this material have also been supplied to the European RIPE consortium [BPS89], to Standards Australia, and to the Defence Signals Directorate, for evaluation.

Chapter 8 summarises the results and achievements of this thesis, as described in the preceeding chapters.

2.2. Appendices

The Appendices of this thesis comprise the detailed results from the analysis of the LOKI algorithm; the C program source for the DES dependency analysis programs; and the C program source for a sample implementation of the LOKI algorithm.

Appendix 1 presents the detailed results from the dependency analysis and preliminary statistical analysis of LOKI cipher.

Appendix 2 contains a listing of two programs **cp_dep** and **ck_dep**, which perform a dependency analysis of output bits on input and key bits for a DES like cipher. This analysis provides a measure of effectiveness of the permutations P, PC2, and the key schedule KS, in achieving the avalanche and completeness properties. This was used as a measure of effectiveness in evaluating variations of the permutations P, PC2, and the key schedule KS in DES like schemes, and in LOKI, in the development of the design criteria. In detail, the programs are:

cp_dep.c
: builds a dependency matrix for each ciphertext bit, based on plaintext bits, for a given P. It assumes that the S-boxes provide an adequate degree of confusion sufficient to ensure that all outputs are dependent on all inputs, and that the S-Box outputs are all equivalent. It takes as input a permutation P to use in the cipher structure being analysed.

ck_dep.c
: builds a dependency matrix for each ciphertext bit, based on keytext bits, for a given P, PC2 and key schedule. It assumes that the S-boxes provide an adequate degree of confusion sufficient to ensure that all outputs are dependent on all inputs, and that the S-Box outputs are all equivalent. It takes as input permutations P, and $PC2$, and

a key schedule KS to use in the cipher structure being analysed.

Appendix 3 contains a listing of a sample implementation of the LOKI cipher. These implement the first (small but slow) of the possible software implementations identified in Chapter 7.

3. Other Work by the Author

During the period when the work detailed in this thesis was being developed, the author was engaged in a couple of other related projects which may be of interest.

One discusses some alternative approaches to providing security in remote terminal sessions. Since remote terminal sessions mean that information is carried over a network that may not be trusted, there is a need for mechanisms to protect the security of information exchanged during the session. The protocols implemented will need to address the issues of authentication of the parties, negotiation of a suitable cipher to use, and the exchange of keys for that cipher. This paper examines each of these issues, describes a prototype implementation incorporating some of these features in the **seclog** program, and finishes with a description of work in progress on extending the widely used **telnet** protocol to incorporate some of these security features. This material was presented to AUUGC'90, World Trade Centre, Melbourne, Victoria, Australia 26th - 28th September, 1990 [Bro90b], [Bro90].

The other project surveys the various alternatives available for the provision of a Pay-TV service in Australia. In particular, the technical alternatives available for the delivery, transmission, scrambling and key management components are presented. A selection of systems in, or proposed for use are then compared. This material was published in [Bro90a].

4. Definitions

Within this thesis, the following definitions apply:

Algorithm
: a clearly specified mathematical process for computation; a set of rules which, if followed, will give a prescribed result.

Autoclave
: a process of incorporating information from the message input to an encryption algorithm into the key scheduling of that algorithm. Thus different message inputs result in different keying choices in the various stages of the algorithm. This is usually acomplished by using some message bits as inputs to both the key selection as well as the message substitution parts of the algorithm.

Avalanche Property
: of a function implies that a change in an input bit should result in approximately half of the output bits changing [Fei73].

Bit Numbering
: bits within a 64-bit block b are numbered from 63 to 0, left to right, MSB to LSB, and denoted $[b_{63}b_{62}\cdots b_1b_0]$. The bits within a 32-bit block are numbered likewise

from 31 to 0, ie the string 10010 represents the number eighteen. When used to represent a polynomial in a Galois field, a block is interpreted as having the left most bit being the most significant. ie the string 100011011 represents the polynomial $x^8+x^4+x^3+x+1$.

Ciphertext
: clear text that has been encrypted.

Cleartext
: intelligible text or signals that have meaning and that can be read and used.

Completeness Property
: of a function implies that each output bit must be a complex function of all input bits [KaD79].

Concatenation
: of two blocks L and R of 32 bits, denoted L | R, will result in LR a 64-bit block consisting of the bits of L followed by the bits of R.

Confusion
: a term coined by Shannon [Sha49] to describe a function whose output is a complex function of its inputs (usually composed of a data block and a sub-key).

Diffusion
: a term coined by Shannon [Sha49] to describe a linear function whose output is a permutation (or wire-crossing) of its inputs, and which distributes input bits so as to remove any local clustering.

Data Encryption Algorithm (DEA)
: an algorithm designed to encrypt and decrypt blocks of data.

Decryption
: the transformation of ciphertext into cleartext.

Encryption
: the transformation of cleartext into ciphertext for the purpose of security or privacy.

Encryption algorithm
: a set of mathematically expressed rules for rendering information unintelligible, by effecting a series of transformations to the normal representation of the information, through the use of variable elements controlled by the application of a key.

Galois Field
: denoted $GF(p^n)$, is an extension of degree n of $GF(p)$. In particular, this thesis refers to binary fields, eg $GF(2^8)$.

Initialization Vector
: a binary vector used in the initial input block in cipher block chaining mode of operation.

Key
: a 64-bit quantity which is used for transformations between ciphertext and cleartext.

LOKI
: the name of the DEA being developed in this thesis.

Modulo 2 addition
: a mathematical operation equivalent to binary addition without carry. Sometimes referred to as an 'exclusive OR' operation.

Non-Linearity
: of a function f, belonging to the set F_n of all Boolean functions of n Boolean variables, is the minimum Hamming distance between f and the set of all linear Boolean functions L_n existing in F_n [Pie90], [Pie89].

Permutation
: a function that reorders a set of n input bits. It may be described by a table of size n. nb: this usage differs from that of Pieprzyk, for whom this term is a synonym for substitution.

ROL(B,n)
: the operation of rotating a block B n places to the left, with wraparound from the MSB to LSB. ie; b_0 becomes b_n, b_1 becomes b_{n+1} etc.

ROR(B,n)
: the operation of rotating a block B n places to the right, with wraparound from the LSB to MSB. ie; b_n becomes b_0, b_{n+1} becomes b_1 etc.

Substitution
: a function mapping a set of m bit input words onto a set of n bit output words. It may be described by a table of size 2^m.

CHAPTER 2

Data Encryption - an Introduction and Overview

1. Introduction

This chapter provides an overview of some modern encryption algorithms, both private and public key. It will describe the general framework within which these schemes have been developed, briefly detail a number of algorithms, and then describe in detail two particular schemes: DES and RSA. These have been chosen as the two algorithms generating the most interest in the private and public-key realms respectively.

The aim of this chapter is to establish the background against which the remainder of the thesis will be developed. In particular it intends to describe the theoretical framework, so far as is known, that has been used to design the DES and RSA algorithms. In the remainder of this thesis the author will be extending that analysis, ultimately using it to develop a new family of private key algorithms.

As described earlier, encryption algorithms may be catagorised into two broad families. Private key (one-key) algorithms in which both sender and recipient share a common secret key for both encryption and decryption; and public key (two-key) algorithms in which the sender uses a well known public encryption key, whilst the recipient uses a secret decryption key. These classes will be detailed separately.

2. Modern Private-Key Cryptography

2.1. Introduction

With the advent of computers, data encryption schemes underwent a revolutionary change. The use of computers enabled the cracking of hitherto secure schemes, but also enabled the implementation of schemes which until then were too complex for use by a manual cipherclerk. Based on the work of Shannon [Sha49], a new family of cryptosystems, known as *Substitution-Permutation networks* were developed. These led to the development of a system known as **Lucifer**, at the IBM Research Laboratories [Fei73], and subsequently to the **Data Encryption Standard (DES)** [NBS77]. DES has since achieved wide utilization, particularly in the banking and electronic funds transfer areas, where it is now a standard in a number of countries including the USA [NBS77], [Ins81], [AAA83], and Australia [ASA85].

2.2. Fundamental Concepts

In designing cryptographic algorithms, there are two basic cryptographic primitives: *substitution* and *transposition* (or *permutation*). In a substitution function, each character is replaced by some other character (see example instance in Fig 2.1).

Fig 2.1 - Substitution Operation

The particular substitution used is selected by some *key*. If characters are represented by a small number of bits, then such a scheme may be defeated by enumerating all possible cases to find the substitution used. If made sufficiently large and a suitably complex function is used, such a scheme is in practice unbreakable, since it would require the enumeration of all $n!$ cases (where n is the number of possible characters). Unfortunately it is also unworkable, since the number of permutations, and hence keys, would be equally large.

Traditionally, in a transposition function a set of characters were permuted (rearranged) into a new order, again with the particular rearrangement selected by some key. In modern implementations, this becomes a permutation of bits or a wire-crossing (see example instance in Fig 2.2).

Fig 2.2 - Permutation Operation

Since the complexity of such an operation is only linear in the number of bits used, it may be easily broken by testing each bit in turn to find its new location. It also implies that it is easy to build such a function for large numbers of bits.

Because of these weaknesses in both the basic primitives, the concept of *product ciphers* arose, whereby a series of substitution and transposition functions were combined to form a function which is much more secure than any of its components. Prior to the introduction of rotor machines, only simple product ciphers were possible. The development of rotor machines, and subsequently computers, changed this significantly.

Shannon [Sha49], in a key paper, described a particular class of product ciphers which used a number of small (and hence implementable) substitution boxes, termed S-boxes, connected by large permutation boxes, termed P-boxes (see Fig 2.3). He termed these *mixing transformations*, whereby the S-boxes provided *confusion* of the input, and the P-boxes provide *diffusion* of the S-box outputs across the inputs to the next stage. Such schemes are now termed *Substitution-Permutation (S-P) networks*.

Fig 2.3 - Sample S-P Network, with Avalanche Characteristic

Two key properties of such S-P networks are the *avalanche* property, identified by Feistel [Fei73]; and the *completeness* property, identified by Kam and Davida [KaD79]. The avalanche property states that changing one input bit should result in approximately half of the output bits changing. The completeness property states that each output bit must be a complex function of every input bit. These properties appear to be important criteria for providing cryptographic strength in the resultant system.

More formally, these are defined (see [WeT86]) as follows. A function f has a good *avalanche* effect if:

> for each bit i, $0 \leq i < m$, if the 2^m plaintext vectors are divided into 2^{m-1} pairs X and X_i with each pair differing only in bit i; and if the 2^{m-1} exclusive-or sums, termed avalanche vectors
>
> $$V_i = f(X) \oplus f(X_i) \qquad \text{(Eq 2.1)}$$
>
> are compared, then about half of these sums should be found to be 1.

A function f is *complete* if:

for each bit j, $0 \le j < m$ in the ciphertext output vector, there is at least one pair of plaintext vectors X and X_i, which differ only in bit i, and for which $f(X)$ and $f(X_i)$ differ in bit j

For a practical scheme however, it is obviously necessary for the recipient to be able to decipher the message. This implies that one must be able to compute the inverse of the enciphering function. For the simple S-P network shown above, it would be necessary to provide a separate decryption stage incorporating the inverses of the substitutions and permutations used. This is expensive in terms of both hardware and software. It is preferable to be able to use the same hardware for both encryption and decryption. This may be done by an elegant scheme which partitions the input and output blocks into two halves, $L(i-1)$ and $R(i-1)$, and uses only $R(i-1)$ at each round i (see Fig 2.4). This half $R(i-1)$, along with a key $K(i)$, is used as input to the encryption function g (which incorporates one stage of the S-P network), and $R(i-1)$ is also copied to $L(i)$ for the next round. The output from g is added modulo 2 to $L(i-1)$ to form $R(i)$ for the next round. Thus when decrypting, $L(i)$ is copied back into $R(i-1)$ where it is used as input to g in order to recover $L(i-1)$. Thus it can be used to recover the other half of the input upon decryption. Ciphers with this form are now refered to as *Feistel Ciphers* after Horst Feistel, a researcher from the IBM Thomas J. Watson Research Laboratory who did much of the early development of this type of cipher in the early 70s.

Fig 2.4 - A Round of a Feistel Cipher

2.3. Lucifer

The first scheme developed based on the concept of a substitution-permutation network was the Lucifer algorithm. This was developed at the IBM Thomas J Watson Research Laboratory in the early 1970s, by a team including Horst Feistel, William Notz and J. Lynn Smith. This development is detailed in a number of papers [Fei73], [FNS75], [SNO72], [Gir71]. The best current description of the Lucifer algorithm is that by Sorkin [Sor84], which also includes a Fortran implementation of the algorithm. Given the sparseness of detail in the earlier descriptions of Lucifer, this is a very useful aid to implementing it.

Fig 2.5 - Lucifer

Lucifer is a Feistel cipher. It has a block size of 128-bits, and a key of 128-bits. The input block is divided into two halves, the upper and lower halves, with the leftmost bit and byte being the lowest, and the rightmost bit and byte being the highest. In each round, the lower half of the input is modified by being summed modulo 2 with a function f of the upper half of the input, and the lower half of the key. The two input halves are then exchanged, and the key is rotated 7 bytes (56-bits) to the left. This is shown in Fig 2.5.

In more detail, function f consists of a substitution, addition of the key, and a permutation. This is shown in Fig 2.6. The substitution consists of eight identical substitution functions which map 8-bits of input and 1-bit of key to 8-bits of output. The substitution function in turn is composed of a pair of 4-bit to 4-bit substitution functions. Depending on the key-bit, these are composed as either $S0|S1$ or $S1|S0$. The eight bits of key used in this are taken from the leftmost byte of the partial key used in the round, with bit-7 used to select the function on message byte 0, bit-6 on message byte 1 etc. The output of the substitution functions are then summed modulo 2 with the partial key, a process termed *key interruption* in the descriptions of Lucifer. Finally, the result is permuted. This permutation is described as having two components. First a regular 8-bit to 8-bit *permutation* repeated for every byte of the result. Then an 8-bit to 64-bit *convolution* which maps those outputs a fixed displacement over the final function f output. From the perspective of modern analysis of this cipher, these two components may be regarded as a single 64-bit to 64-bit permutation with a very regular structure. For details of the actual functions used, and the key access schedule, see Sorkin [Sor84].

Fig 2.6 - Lucifer Function f

From the little discussion of Lucifer in the open literature, the consensus appears to be that it is a well designed and sound cipher. It was the predecessor of the DES. However, with the development of the DES, Lucifer has been eclipsed, apart from mainly theoretical interest. What little work has been published on the cryptanalysis of Lucifer and similar Feistel ciphers is given in [GrT78], [KaD79], [AnR82].

Fig 2.7 - DES Enciphering

2.4. DES

The Data Encryption Standard (DES) [NBS77] was developed out of the Lucifer cipher in the mid 70s. It has since achieved wide utilization, particularly in the banking and electronic funds transfer areas. It is an Australian standard [ASA85] for this purpose, and in the US has been certified until 1992, again for this purpose.

However, there was a considerable amount of controversy at the time of its introduction, primarily with the choice of a 56-bit key [DiH77], [Hel79a]. This was on the grounds that such a key length was small enough to be exhaustively searched on machines which could be constructed in the foreseeable future. With the march of computer technology, this risk appears to be rising. However, on all other accounts DES appears to be an excellent cryptosystem. The DES has withstood a concerted analysis by the open research community, since its adoption in 1977. The most successful attack to date, that of Biham and Shamir [BiS90] still performs worse than an exhaustive key-space search on the DES with the full 16 rounds.

The basic DES enciphering chain is shown in Fig 2.7. In brief, the input block is permuted by an initial permutation IP, is then subjected to a complex key-dependent computation, and is finally permuted by IP^{-1}, the inverse of the initial permutation. On deciphering, the same process is used, except that the complex component is run backwards, using keys in reverse order, via the method described previously (see Fig 2.4).

In more detail, the input block X of 64 bits is first permuted by the initial permutation IP, and then divided into two halves $L(0)$ and $R(0)$. The complex key-dependent function consists of sixteen rounds in which the left component $L(i-1)$ is added modulo 2 to the output of a non-linear function g whose input is both the right component $R(i-1)$ and part of the key $K(i)$. The halves are then interchanged to form the input for the next round (with the exception of the final round). Finally the output is permuted by IP^{-1} (the inverse of the input permutation IP) to yield the output ciphertext Y. For details of these permutations, and all the other functions in the DES, see [ASA85], or any of the modern textbooks on cryptography (eg [SeP89], [Den82]).

The cryptographic security of DES rests, in the main, on the complex function g. It is shown in more detail in Fig 2.8. Its purpose is to ensure that the final DES output vector Y (see Fig 2.7), is a complex nonlinear function of the input, such that there is no easy way of deriving the input knowing only the output, short of enumerating all possible keys. In detail, in round i, $1 \le i \le 16$, the right component of the message input $R(i-1)$ is first expanded from 32 to 48 bits via an expansion function $E(R(i-1))$. These are then XORd with a 48-bit component extracted from the key $K(i)$ to form vector A. This vector is partitioned into eight 6-bit blocks which are used as input to eight substitution boxes (S-boxes). Each S-box produces a 4-bit output, all of which are concatenated to form the vector B. This vector is permuted by P and then XORd with the left component $L(i-1)$ of the message. In more detail, each S-box may be regarded as being composed of four substitution functions (row functions or rows). The two (outer or selector) bits of the 6-bit input selects one of these four functions, which then operate on the four (inner or substitution) bits to give a 4-bit output. This output is then permuted by P. Together these form one stage of an S-P network. The repetition of this procedure sixteen times forms the full S-P network.

Fig 2.8 - DES Function g

The other critical component in the DES algorithm is the *key scheduling* stage. It is responsible for forming the sixteen 48-bit sub-keys used in the rounds of the encryption procedure, as shown in Fig 2.9. This function is important since if the same key is used on successive rounds, it can weaken the resulting algorithm. This trap is mostly avoided with the specified key scheduling algorithm, except for a small number of keys known as *weak* or *semi-weak* keys (see [GrT78], [MeM82], [MoS87], [MoS86], and [ASA85]).

Fig 2.9 - DES Key Scheduling

In detail, the 64-bit key is permuted by $PC1$. This permutation performs two functions. First it strips the eight parity bits out. Then it distributes the remaining 56 bits over two 28-bit halves $C(0)$ and $D(0)$. Subsequently, these form two distinct paths. At each round, each 28-bit register is rotated left either one or two places according to the schedule shown in Table 2.1.

Table 2.1 - Key Schedule for DES

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Shift	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1
Total	1	2	4	6	8	10	12	14	15	17	19	21	23	25	27	28

After the shift, the resultant 28-bit vectors are permuted by $PC2$ (which in fact consists of two 28-bit permutations, not one 56-bit permutation), and are concatenated together to form the sub-key for that round. Note that the total number of shifts is 28; hence $C(16) = C(0)$, and $D(16) = D(0)$. When decrypting, it is necessary to generate the keys in reverse order. Because the final vector is the same as the initial vector, this can be done by rotating right, reading the table in reverse order (and with no shift before generating the first key). Thus, with minor changes, the same hardware is used for both encryption and decryption.

2.5. DES Variants

Given that it is generally accepted that the DES cipher is well formed, and that there are no known shortcuts to cryptanalysing it, then its main shortcoming appears to be its 56-bit key size, which is widely regarded as being too small. Several variants of DES have been developed. Some of these use the same S-box and permutation P structure, but alter the key schedule used. They include the KSX scheme described in [Out86], and the extended KS128X scheme [Out89]. In these schemes, Outerbridge has designed a key schedule inspired by that used in Lucifer. KSX uses a 64-bit key partitioned into 16 nybbles (of 4-bits each). The nybbles are selected according to a fixed permutation and are presented to the S-box inputs, and then the key register is rotated 1 nybble before the next round. In KS128X a 128-bit key is used, partitioned into bytes, the lower 6 being selected as the partial key. Again the whole register is rotated, this time by 5 bytes to the left. It is very similar to the schedule used in Lucifer.

The author believes this approach has advantages, particularly in the use of a much cleaner key schedule. However it still retains the original DES S-P function. Recent developments, referred to later in this thesis have indicated how improvements can be made in this area which will result in a more secure algorithm.

Other schemes retain the same overall structure, but also amend the function f. These include the family of subtractive encryptors [Mor83], and the layered approach [MoT86]. These schemes have not been investigated further.

2.6. FEAL

The FEAL algorithm, developed by researchers at NTT in Japan is one of the strongest contenders amongst the new set of private-key algorithms. It too is a Feistel cipher, able to encrypt 64-bit data blocks with a 64-bit key. It has been optimised for implementation on eight-bit processors, and in hardware, and for this reason a very simple function S is used to replace S-boxes in each round of the cipher. This function is used four times per round and is

$$T = X_1 + X_2 + \delta \bmod 256, \ \delta=0,1 \qquad \text{(Eq 2.2)}$$

$$S(x, y, \delta) = \mathrm{ROT2}(T) \qquad \text{(Eq 2.3)}$$

where ROT2 refers to a rotation of 2 bits to the left of an 8-bit operand. It is also used in the key schedule. FEAL was originally described as a 4-round version in [ShM88]. This version was subsequently broken [Boe89]. The FEAL cipher was then strengthened to an 8-round version, and a challenge was issued to anyone who could break it [Miy90]. This version has also been implemeted in hardware, as described in [Miy88]. An n-round version of FEAL was also submitted to the European RIPE consortium for evaluation as a possible authentication primitive for use in the RACE European high-speed network.

The author's main reservation about FEAL is in the choice of the function S (Eq 2.2). This function is near-linear, in contrast to the conventional wisdom that states that a good cryptographic substitution function should be as non-linear as possible. Also, the absence of a permutation means that the dispersion of bit dependencies of output bits on input and key occur only via the rotation and action of function S, significantly slowing its dispersion. Biham and Shamir [BiS90] in their description of the differential cryptanalysis technique, have stated that FEAL-8 may be broken using a known plaintext attack. It appears clear that more than 8 rounds should be used for this scheme.

2.7. Khufu, Khafre and Snefru

Khufu, Khafre and Snefru are a family of ciphers developed by Ralph Merkle from Xerox Parc. They are described in an internal report which was subsequently published (without permission) on USEnet. A corrected version of this paper was presented to Crypto '89 by Merkle. Khufre is an encryption function designed for fast bulk encryption of data. It uses precomputed tables for speed. Khafre is also an encryption function, but is designed for encrypting only small amounts of data. It uses a fixed set of tables, at the cost of a more complex scrambling function in each round. Snefru is a one-way hash function which is used to reduce a message of arbitary length into a small residue (of 64, 128, or 256 bits).

The basic design philosophy that Merkle has used is that of a Feistel cipher, in common with most of the other modern private-key schemes. However, he has emphasised ease of implementation in software. In doing so he has decided that since table lookups will be used to implement a SP function, rather than doing 4 or 8 in parallel, and ORing the results together, as would be done when implementing most Feistel ciphers, he would incorporate the results from each lookup into the next stage. Effectively this results in having a much larger number of rounds, each of which only maps a 8-bits of input onto 32-bits of output, which is then added modulo 2 to the other data half. The input half is then rotated according to a rotation schedule to bring a new byte into position for the next round. Overall, the same number of table lookups may well result, depending on how many rounds are done. Merkle claims that this process greatly improves the spread of dependencies of output bits on input and key.

Rather than a standard key schedule component, Merkle has incorporated the key into the schemes via either the S-boxes used, or the rotation schedule. In Khufu, the key is used, along with an initial set of S-boxes, to dynamically calculate a pseudo-random set of S-boxes for use in encrypting the message. The key is also expanded and summed modulo 2 to the data at the start and end of the data process. In Khafre, a standard set of S-boxes are used, and the key is incorporated into the rotation schedule. In Snefru, the standard S-boxes are also used in the hashing process.

For more details on these algorithms see Merkle [Mer89]

Merkle has subsequently announced that a 2 round version of Snefru has been broken by Eli Biham from Israel, in an article to the USEnet news network on 19 April 1990. He has subsequently recommended that at least a 4 round, and probably 8 round version should be used. A $1000 prize is offered to anyone who can break the 4 round version, in the same article.

The author believes that these ciphers incorporate some interesting concepts. However, the author has major reservations about the use of random S-boxes in the encryption function in each round. All the research to date indicates that the role of these functions in providing confusion of the input are critical to the security of the cipher. If the encryption functions should be linear or near linear, a significant advantage is given to the cryptanalyst. A further problem with these schemes is that whilst the idea of running the table lookups serially is fine, in practice Merkle has recommended a number of rounds that is far too small. Since Bihar and Shamir [BiS90] report that 8 round versions of traditional Feistel ciphers can be broken, this would seem to indicate that 32 to 64 rounds of Merkle's ciphers would be required for equivalent security, rather than the 4 or 8 he nominates.

2.8. LOKI

The LOKI scheme is the latest of the new private key block algorithms. It was developed by the author and his colleagues Professor Jennifer Seberry and Dr. Josef Pieprzyk. It is a family of block ciphers, with Feistel structure similar to the DES, but with a much simplified key schedule, and different permutation and substitution functions. It is the major achievement of this thesis, and is documented in Chapter 7.

3. Public-Key Cryptography

3.1. Introduction

Public-Key Cryptography (PKC), or *Two-Key Cryptography* was originally proposed in a pivotal paper by Diffie and Hellman [DiH76], who developed the concept, and proposed a possible scheme. It was also developed independently by Merkle [Mer78] who outlined the basic concepts, but did not propose a feasible system. Such schemes provide the ability to publish an encryption key (used to conceal the message) in the equivalent of a telephone directory, without compromising the decryption key (used to restore the message), and hence the secrecy of the message. This distinction from traditional private-key schemes was shown in Figs 1.1 and 1.2.

More formally, in a PKC the enciphering (concealing) algorithm E_{K_1}, and deciphering (restoring) algorithm D_{K_2}, are controlled by two distinct pieces of information, an *encryption key* K_1, and a *decryption key* K_2, with the property that it is computationally infeasible to compute the decryption key D_{K_2} from the encryption key K_1. Computationally infeasible implies that with the speed of computers currently available, the time taken to break the scheme is unreasonably large (usually at least of order years, if not orders of magnitude greater). This contrasts with the keys used in traditional Private Key Cryptosystems where the encryption and decryption keys are either identical, or can easily be computed from each other. Before a secure communications channel can be established using such a scheme, this key must be exchanged over some other secure channel, traditionally by courier. If there are n participants, then up to $\frac{n(n-1)}{2}$ keys would have to be exchanged in advance, a practical impossibility for any reasonably sized n. Thus traditionally cryptography has had applications limited to use in hierarchical organizations where the number of links can be minimised and predicted in advance. In contrast a PKC encryption key K_1 can be placed in a public directory, for use by anyone who wishes to communicate securely with the holder, without compromising the decryption key K_2, and without them having to have anticipated the need to communicate with each other. Consequently it enables the possibility of users on an electronic network to communicate securely with each other with little more difficulty than that required to establish a link between them.

A relative of a PKC is a *Public-Key Distribution Scheme* (PKDS). In a PKDS, the two participants exchange messages until a common key is known by each, which it is computationally infeasible to compute by any eavesdropper from the messages exchanged. This common key is then used in a Private Key Cryptosystem. Obviously any PKC can be used as a PKDS. The main reason for using a PKDS rather than a PKC is that public key functions tend to be difficult and slow to implement, thus in a PKDS it need only be done once per session, rather than continuously. Also, some PKDS are easier to implement than a fully fledged PKC. The original scheme proposed by Diffie and Hellman was such a PKDS. Many current "public-key" implementations are in fact PKDS, where a Private-Key scheme is used to carry the bulk of the traffic for a session.

Given the obvious advantages of public-key schemes in modern applications, the obvious question is why they have not yet been extensively used. The answer lies partly with the inate conservatism of some of these fields, who prefer to wait until the security of such schemes has been repeatedly confirmed (a valid stance, given that one proposed class of schemes, low-density knapsack PKC such as the Merkle-Hellman scheme have all proved to be insecure), but mostly from the computational complexity involved in implementing many of these schemes. That is, they involve using operations such as exponentiation on very large numbers, which, as we shall see later, take a considerable amount of time. Consequently, they are simply not able to compete with the speed of Private-Key schemes such as DES. Although a considerable amount of effort is being expended to improve their performance, and indeed some progress is being made, nonetheless it appears that we need to wait for computational speeds to increase further, before these schemes become widely used. Meanwhile the major use at present and for the short term future, of public key schemes is in hybrid systems, in which a public key algorithm is used to exchange a session key, and to provide authentication and non-repudiation, whilst a fast private key scheme is used to provide privacy.

One of the major advantages of PKC's is their relatively straightforward and simple mathematic description, which contrasts strongly with the opaqueness of such private key systems as DES. Public key schemes are based on a class of functions known as one-way trapdoor functions, derived from a class of computationally difficult problems termed NP (non-deterministic polynomial) problems (see Williams [Wil82]). Private key schemes, in contrast, rely on a series of substitution and transposition operations which are very complex to analyse mathematically. Consequently there is a chance of providing a mathematical proof of the security of PKC's, which is not possible with private key schemes. Although no conclusive proofs have yet been obtained, Brassard [Bra79] has shown that for both the RSA and the Diffie-Hellman key distribution schemes if a successful cryptanalytic attack is possible in polynomial time then it would imply that $NP = CoNP$, a result that is regarded as improbable at present, and is thus indicative of the probable security of these schemes.

An excellent introduction to number theory and its applications is the book by Schroeder [Sch84], which introduces the various axioms and theorems in number theory, and illustrates their applications in many diverse fields, including cryptography. A synopsis of the material covered in the book is given in his paper [Sch85]. An overview of the area's of mathematics used in cryptology, including in PKC's, along with a number of simple examples is given by Simmons [Sim79b].

An overview of computational complexity is provided in the Turing Award lecture by Cook, reprinted in [Coo83]. Yao [Yao82] presents an introduction to extensions to Information Theory made possible by using the concepts developed in modern computational complexity theory. He provides a precise definition of trapdoor functions and describes their applications in cryptography, pseudo-random number generation and abstract complexity theory.

3.2. Public-Key Distribution Systems (PKDS)

3.2.1. Description

The first type of system developed to use the concept of publicly available cryptographic keys was the public key distribution system (PKDS). A PKDS permits two users to exchange a private key over an insecure channel. Once this private key is exchanged, it is used in a conventional private key cipher. This approach to public key systems is of considerable interest since the "expensive" calculations needed for the public key component need only be done infrequently on relatively small blocks of data (namely the key), and then fast hardware implementing a private-key scheme may be used to exchange the actual messages needed in that session. This form of hybrid system has already been implemented by a number of people. The feature distinguishing a PKDS from a PKC is that the private key exchanged is determined solely from the publically distributed keys, and cannot be any arbitrary value.

In their original paper Diffie and Hellman [DiH76], proposed a PKDS, whose security rests on the difficulty of computing discrete logarithms over a finite field $GF(q)$ where q is prime. Let

$$Y = \alpha X \ (mod\ q), \qquad \text{for } 1 \le X \le q-1, \qquad \text{(Eq 2.4)}$$

where α is a fixed primitive element of $GF(q)$; then X is referred to as the logarithm of Y to base $\alpha (mod\ q)$. Calculation of Y from X is relatively easy, taking at most $2 \log q$ multiplications (Knuth [Knu81]), and with more recent algorithms and special hardware, possibly much faster. However computing X from Y is still regarded as being very difficult, though the modulus of the field in use needs to be chosen carefully, (see Odlyzko [Odl84]).

In the Diffie-Hellman scheme each user generates a random number X_i, chosen from the set [1,q-1], and keeps it secret whilst publishing Y_i s.t.

$$Y_i = \alpha X_i \ mod\ q, \qquad \text{(Eq 2.5)}$$

in a public file along with their name and electronic address. When two users i and j wish to communicate, they use as a key K_{ij} s.t.

$$K_{ij} = \alpha^{X_i X_j} \; mod \; q \qquad \text{(Eq 2.6)}$$

$$= Y_j^{X_i} \; mod \; q \; (which \; user \; i \; can \; compute)$$

$$= Y_i^{X_j} \; mod \; q \; (which \; user \; j \; can \; compute)$$

Both users i and j can compute K_{ij} efficiently. However any other user would have to compute K_{ij} from Y_i and Y_j, which would require computing the discrete logarithm of one of these; ie

$$K_{ij} = Y_i^{\log_\alpha Y_j} \; mod \; q. \qquad \text{(Eq 2.7)}$$

Hence the security of this scheme rests on the difficulty of computing these logarithms. This is known to be a hard problem, currently orders of magnitude more difficult than the effort needed to exchange keys.

3.2.2. Implementation

Several hybrid public-key, private-key systems have been implemented using the Diffie-Hellman PKDS to exchange a session key subsequently used by a DES encryptor chip. Mitre Corporation have developed such a system for use on their LAN. It uses a Z-80 microprocessor to perform the public key exchange, and a DES chip for the main data encryption operations; Berkovits, Kowalchuk and Schanning [BKS79], and Hershey [Her80]. A hardware implementation of the public key part of this system was done by Hewlett Packard, Yiu and Peterson [YiP82], which increased the key encryption rate from 25 bits/sec to 3 Kb/sec. Subsequently Mitre designed a VLSI parallel/pipeline processor to perform these modular exponentiations, Fam [Fam82]. When designing the implementation used, the field $GF(2^{127})$ was chosen since it has a number of computational advantages, which make it easy to implement efficiently in both hardware and software. However Gait [Gai79], Herlestam [HeJ81], Mullin [MNW84], and Odlyzko [Odl84] show that because of these advantages the field is insecure with respect to computing discrete logarithms, and hence cryptosystems based on it must be regarded as having being compromised for all but the lowest level of security.

Additional PKDS schemes are defined in Matsumoto, Takashima and Imai [MTI86], which presents a classification of the proposed schemes, determines the most computationally efficient, and shows that it is equivalent in security to the original Diffie-Hellman scheme.

El Gamal [ElG85] describes a true PKC related to the Diffie-Hellman PKDS. This scheme also relies on the difficulty of computing discrete logarithms for its security. It is regarded as a serious alternative to RSA for authentication and key exchange uses, but like RSA, requires efficient modular exponentiation.

3.3. The Rivest-Shamir-Adleman (RSA) PKC

Rivest, Shamir and Adleman [RSA78] outlined the first practical and fully-fledged public key cryptosystem (RSA), the security of which is based on the difficulty of factoring large integers. This system is still the most highly regarded, and is the one most commonly suggested for use in public key systems. It has been placed as an appendix in a number of standards related to key exchange in Electronic Funds Transfer (EFT) schemes, in the authentication of certificates issued by an Electronic Directory Service, and in the authentication of participants in wide-area networks. Zimmerman [Zim86] outlines in detail, a set of protocols which could form the basis of an industry standard for RSA usage.

The major advantages of RSA are that it does not increase the size of the message and that it may be used to provide both privacy and authentication (by providing a digital signature) over communications links. Its main disadvantage is its reliance on modular exponentiation, an operation which is still moderately time consuming, though much effort is being expended to improve the efficiency of this operation.

3.3.1. An Overview of RSA

In the RSA system, each user wishing to receive encrypted messages, performs the following steps:

- selects two large primes p and q at random, each about 100 digits long (for key lengths believed secure at present),
- calculates the modulus of their system $R = pq$,
- selects at random a number e with $e < R$, $GCD(e, \phi(R)) = 1$, where $\phi(R) = (p-1)(q-1)$, is the Euler totient function of R, (and e is usually small, for example 3, and may be fixed for all users [QuC82], [Wil86], [KBS86b], and [KBS86a]).
- solves for the unique d in the congruence $de \equiv 1 \ (mod \ \phi(R))$, $\quad 0 \le d < R$,
- broadcasts the public key $E = <e, R>$
- and keeps secure the private key $D = <d>$.

A sender can encrypt a message M (expressed as some large integer of a size less than the length of the modulus used in the system, and created by concatenating characters together to form a bit-string of the appropriate length) to form the ciphertext C using the public encryption key $<e, R>$ by calculating:

$$C = M^e \ (mod \ R), \qquad 0 \le C < R \qquad \text{(Eq 2.8)}$$

The receiver can then decipher this using his private decryption key $<d>$ by calculating:

$$M = C^d \ (mod \ R) \qquad \text{(Eq 2.9)}$$

The system relies on the following well known number theoretic identity (Euler's generalization of Fermat's theorem):

$$\text{If } GCD(X, R) = 1, \qquad \text{(Eq 2.10)}$$

$$\text{then } X^{\phi(R)} \equiv 1 \ (mod \ R) \quad \text{hence}$$

$$C^d = M^{ed} \equiv M^{1+n\,\phi(R)} \equiv$$

$$M^1 M^{n\phi(R)} \equiv M\,1 \equiv M \ (mod\ R).$$

RSA can also be used as a signature scheme since the encryption and decryption operations are commutative. That is, a sender can create a signature S for a message M using their private key $<d>$ (which only they have access to), by calculating:

$$S = M^d \ (mod\ R) \qquad \text{(Eq 2.11)}$$

which may be verified by anyone using the sender's public key $<e, R>$ (obtained from the public directory), by calculating:

$$M = S^e \ (mod\ R), \qquad \text{(Eq 2.12)}$$

The signature can then be encrypted using the recipient's public key, in order to privately transfer it to the recipient.

3.3.2. RSA Weaknesses and Related Schemes

There is a minor problem known as the signature reblocking problem, which must be overcome when the same scheme is used for both secrecy and authentication. It occurs because each user has modulii R of different sizes, and hence a block of ciphertext in one user's field may not be a valid block in the other users field. This becomes a problem when user A signs a message for B, and then wishes to protect its privacy by encrypting with B's public key. If user A's field is larger than B's, and the signed message also turns out to be too large (ie $R_B < S_{AB} < R_A$), then it cannot be directly encrypted (ie $S_{AB}^{e_B} \ (mod\ R_B)$ cannot be decrypted correctly). Several strategies are suggested to overcome this problem. These include using separate modulii for secrecy and for authentication, guaranteeing that the signature modulii is always smaller than any secrecy modulii [RSA78], and reversing the order of signature and secrecy encryptions, since these functions are commutative [Koh78]. Alternatively the problem may be avoided by using a hashing function to compress the message into a single block, which is then signed with the private key only. Because a one-way function is used to perform the hashing operation, no significant information is conveyed about the content of the message. Hence no secrecy step is needed. [Dav83], [Den83], [Den84].

Another problem caused by RSA being an authentication scheme as well as a privacy scheme, is that it is exposed to attack when a cryptanalyst is permitted to obtain signatures on arbitrary messages of his choice, and thus over several messages build up enough information to break some other message. Denning [Den84] proposes a method of forming signatures by performing a transformation on the original message, which both overcomes this weakness, and reduces the computation involved in creating the signature.

A number of other attacks have been suggested on the RSA system. The first was by Simmons and Norris [SiN77], and was rebutted by Rivest in [Riv78]. Then Herlestam [Her78] suggested a generalised form of attack, based on iterated encipherment of the ciphertext, and dependent on the number of fixed points in the encryption function. This was refuted by Rivest [Riv79], who indicated errors in the analysis. Williams and Schmid [WiS79] analysed this attack and several others, and indicated some conditions on the primes p and q which minimised the possibility of such attacks. Gordon [Gor84] in a key paper collected these various requirements, analysed all of the proposed attacks on the RSA system, and outlined an algorithm for generating primes p and q which satisfied all the conditions necessary to minimise the chance of success of any of these attacks, and thus construct a secure system. This algorithm has been suggested as the basis of an appendix on generating p and q in several standards suggesting the use of RSA.

The security of the RSA system is known to be at most as difficult as factoring the modulus $R{=}pq$. It is not known whether it is as difficult as that. Rabin [Rab79] has proposed a PKC scheme, related to RSA, for which it may be proved that cracking the scheme is equivalent to factorizing the modulus. Williams [Wil80] and [Wil84] has also derived such an encryption scheme for which this is true. However because of the constructive nature of this proof, the Williams systems are open to chosen ciphertext attack under certain conditions. Hence these schemes are not as highly regarded as RSA, and are of interest mostly because of the proof of difficulty of breaking them being equivalent to factoring provides further indications of the difficulty of cracking the similar RSA scheme.

Another related alternative scheme is proposed in Kravitz and Reed [KrR82a], [KrR82b]. The security of this system is based on the difficulty of determining the degrees of the irreducible factors of a high-degree polynomial. It is not as secure as the RSA scheme, however it is interesting in that it presents a generalization of the RSA scheme into a polynomial ring message space.

A recent comparison of the relative security of RSA and the ElGamal schemes is given in [Oor91].

3.3.3. Implementation

Muller [Mul83] describes the development of a cryptoprocessor, a microprocessor-based board added to a workstation, which handles all the tasks necessary to support secure communications links between workstations. This system uses an 8086 microprocessor to perform the housekeeping tasks and the public key (RSA) key encryption; and a WD2001 DES chip to perform high-speed private key encryption. The whole system is contained on a Multibus board which may be plugged into a workstation. It is assumed that there is a secure host on the system which can act as a name and public key server. For key sizes of 288 bits it took about 4 minutes to perform the two encryptions needed for a secret and signed letter. They conclude that this performance could undoubtably be improved with better algorithms or special hardware.

Serpell, Brookson and Clark [SBC84] suggest the use of a stand-alone unit which may be connected between an RS232 terminal and its host or modem to perform the necessary link encryption. Each users key certificate would be stored on a magnetic key inserted into this device, and would be created by a secure central key certifying authority whose public key is well known. Users would exchange key certificates over the unsecure link, from which the other's public key may be obtained. The unit developed was based on an 8085 microprocessor performing the housekeeping, a hardware exponentiator built using TTL logic, and a British Telecom B-Crypt private key chip. It could exchange 512-bit session keys in 3 seconds.

Jackson, McEnvoy and Newman [JMN84] describe the Project Universe experiment, in which local area, and wide area networks are linked, with optional transport layer encryption. They used RSA as a PKDS to exchange DES master keys, which in turn were used to exchange session keys. Performance measurements indicate that the use of encryption added only a constant overhead, which was not of major significance, especially on the wide-area links.

Brickell provides a brief survey of hardware implementations of RSA in [Bri90].

4. Further Reading

A survey of modern cryptographic techniques may be found in Diffie and Hellman [DiH79], who provides an introduction to cryptographic fundamentals, a taxonomy of cryptographic systems both private and public-key, a description of some of the practical problems encountered, and a review of their various applications. More specialised coverage may be found in Computing Surveys 11(4) 1979 which contains three articles ([Lem79], [Sim79a], and [PoK79]) surveying the status of cryptology at that time. There is a particular emphasis on the emerging PKC, and on the evolution of the relevance of cryptology from being strictly a government and military concern, to being of major interest to business (especially banking), and eventually to the public as well. This trend continues as electronic messaging and funds transfer becomes more prevalent. Other popular introductions to, and surveys of public key systems are given by Simmons [Sim79b] who presents a popular summary of these recent developments, Hellman [Hel79b] who presents an overview of the mathematics used in public key schemes, and Beker [Bek84] who presents a brief survey of currently used encryption techniques. Simmons [Sim82] collects all the original key papers on public key systems, along with some commentary into one book. Denning [Den82] provides an overview of encryption techniques and access control methods, as used in current computing and data communications systems. Seberry and Pieprzyk in their book [SeP89] provide the latest survey of the field, and is an excellent introductory text.

5. Conclusions

This chapter provides the background information on the design of modern private and public key schemes which was used as the basis from which the research detailed later in this thesis was developed.

CHAPTER 3

Techniques for Implementing the RSA Public Key Cryptosystem

1. Introduction

From the preceeding survey of cryptographic algorithms, public-key cryptosystems are shown to embody a number of desirable properties. In particular, when used in a large network, with participants previously unacquainted, the ability to exchange public-keys is highly desirable in providing efficient communications.

For practical use though, these schemes need to be fast and efficient. This chapter will examine in detail the implementation of the most common public-key cryptosystem, RSA. It will show that the essential exponentiation component of the RSA scheme is inherently time-consuming, and that the best generally available implementations, whilst being suitable for key-exchange and authentication, are not fast enough for providing secrecy of all traffic exchanged. Further, whilst dealing specifically with RSA, most of these comments are equally applicable to the other exponentiation based public-key schemes including Diffie-Hellman [DiH76], and ElGamal [ElG85].

It will conclude that a complete security package will need to incorporate a fast, efficient and secure private key cryptosystem in a hybrid scheme, in order to provide a total solution. The remainder of this thesis will detail the development of suitable private key schemes for this use.

2. RSA Encryption and Decryption

The RSA public-key cryptosystem has been described in the preceeding chapter. Fundamental to the RSA algorithm is the modular exponentiation of very large integers, of up to 200 digits (or 600 bits) in length. These very large numbers are necessary to ensure the security of the scheme, as the effort needed to break the scheme (by factoring the modulus R), is of $O(e^{f(R)})$, an exponential of a function f on the modulus R. The best currently known algorithm for factorizing a number R with a large prime factor p is a variation on Lenstra's elliptic curve algorithm proposed by Brent [Bre86], which runs in time

$$O\left(e^{\sqrt{\ln p \ln \ln p}}\right) \qquad \text{(Eq 3.1)}$$

and which leads to the choice of R around 200 digits (or 600 bits) in length [DHS84], [Bre86]

Table 3.1 provides a comparison of the effort required for encryption and factorisation based on various key lengths[1].

Table 3.1 - Complexity of Factorization and Exponentiation Operations Modulo R		
Digits in R	Operations in Factorization	Operations in Exponentiation
20	7.2e+03	8.0e+03
40	3.1e+06	6.4e+04
60	4.6e+08	2.1e+05
80	3.7e+10	5.1e+05
100	1.9e+12	1.0e+06
120	7.6e+13	1.7e+06
140	2.3e+15	2.7e+06
160	5.9e+16	4.0e+06
180	1.2e+18	5.8e+06
200	2.3e+19	8.0e+06

Since no conventional computers support integer calculations of this size, the problem can be approached from two basic directions. One is to break the numbers into words of a size supported by the hardware on current machines, and carry out each step of the calculation in a number of stages using multi-precision arithmetic. The other technique is to design and build specialised hardware which supports the large wordsize required. Both of these methods have been attempted, and both can utilise some of the techniques described below.

[1] currently 1e+16 operations is regarded as a limit for computational feasibility (ie 1 year on a 1000 MIP machine). Times may be derived by assuming one instruction per microsecond on a 1 MIP machine, and knowing that there are 3e+13 microseconds in a year.

Table 3.2 - Example of Long-hand Multiplication

452		452	
* 125		* 125	
______		______	
2260	(* 5*1)	452	(* 1)
		______	
		4520	(pp *10)
+ 9040	(* 2*10)	+ 904	(* 2)
______		______	
=11300	(pp step 1)	= 5424	(pp step 2)
		______	
		54240	(pp *10)
+45200	(* 1*100)	+ 2264	(* 5)
______		______	
=56500	(Result)	=56500	(Result)
______		______	
Right-To-Left Evaluation		**Left-To_Right Evaluation**	

In multi-precision arithmetic the calculation is done on each digit of the number in succession, with a carry or borrow being propagated into the next word of the number. The size of the digit used depends on the complexity of the algorithm, and on the word size of the machine (or specialised hardware) being used. The algorithms needed to do this are well known, and may be found in Knuth [Knu81], and also in Schroeder's excellent introductory text to number theory applications [Sch84]. The basic multi-precision multiplication algorithm they describe, runs in $O(n^2)$ time on a number n bits in length. This corresponds to performing long-hand multiplication of decimal integers (Table 3.2); which can be done either right-to-left, or left-to-right. The former is more usual, though the latter is often more convenient for computer implementation.

Since in RSA, both encryption and decryption involve modular multiplication, a division stage must be performed after the multiplication to extract the remainder (mod R). Several techniques have been suggested to minimise the time taken by this process.

One is to precompute a multiplication table for one of the operands, and use it to perform the multiplication with a series of additions and table lookup steps, as described by Henry [Hen81] and Quisquater et al. [QuC82].

Another is to perform the modulo reduction in parallel with each stage of the multiplication calculation, by removing the most significant digit, and adding its remainder (mod R) to the partial sum. This process achieves a saving by a factor of about 6, and the author strongly recommends that it be used. It is based on a result presented in the paper by Chivers [Chi84], which states that

$$\left\{ \sum_{i=0}^{n} a_i \, b_i \right\} (mod\ m) \equiv \qquad \text{(Eq 3.2)}$$

$$\left\{ \sum_{i=0}^{n-2} a_i b_i + a_{n-1} b_{n-1} (mod\ jm) \right\} (mod\ m), \text{ for all } j$$

The technique is described by Chivers [Chi84], Blakley [Bla83], and Willoner et al. [WiC81] and produces a result that it is never larger than $n + \ln n$ bits long. In more detail, it involves removing the most significant digit (MSD) of the result, and adding its remainder mod m to the number represented by the remaining digits. To use it, a table is constructed of these remainders for all possible digits. The size of the table obviously depends on the size of the base used. Typically bases of 1, 4, or 8 bits are used, leading to table sizes of 1, 16, or 256 entries. See Tables 3.3 and 3.4 for examples (which are presented in base 10). This process can be generalised to an algorithm for reducing any number larger than the modulus down until it has no more digits than the modulus, by iterating over all digits larger than the modulus length, and scaling up the remainders appropriately.

Table 3.3 - Number and Remainder (mod 517)

n	n (mod 517)
1000	483
2000	449
3000	415
4000	381
5000	347
6000	313
7000	279
8000	245
9000	211

Table 3.4 - Example of Modulo Reduction

4120	(mod 517)
———	
0120	(remove MSD)
+ 381	(add remainder of 4000 mod 517)
———	
= 501	(result mod 517)

Some, or all of the above techniques could be used by an implementor in conjunction with a conventional multi-precision arithmetic package. Alternatively one of several fast multiplication algorithms, followed by a fast division algorithm designed for this application, could be used. This technique is outlined by Mohan et al. [MoA85]. Asymptotically, the fastest integer multiplication algorithm known, is the Schonhage-Strassen Algorithm [AHU74]. It is one of a family of algorithms which break the number into blocks, and treat each block as the coefficients of a polynomial. The polynomials are evaluated at suitable points, the products calculated, and the product polynomial found by interpolation. Any one of a number of interpolation schemes can be used, the fastest uses the Discrete Fourier Transform and the Convolution Theorem, and can multiply two integers of length n in $O(n \log n)$ steps. An iterative process is then used to perform the division step, optimised by careful choice of the modulus R. This technique seems particularly well suited for implementation on current digital signal processing chips.

The use of addition chains, rather than conventional binary arithmetic, provide another alternative for hardware implementations. This technique for computing the multiplications in the modular exponentiation has efficiencies approaching that of FFT methods, but with a much easier implementation. Yacobi [Yac91] describes such a method.

A further alternative, where special hardware is used, is to have special carry-save multiplication hardware, where the addition of carries between digits in the partial result is delayed until the end of the multiplication step by using two registers for each number. An extension of this technique which incorporates modulo reduction similar to that described above, is presented by Brickell [Bri82] and performs modular multiplication in time $O(n+7)$ with $O(n)$ gates. This is the preferred approach when special hardware is to be designed. It is possible to multiply two numbers in $O(\log n)$ time, using $O(n^2)$ gates. Currently for the size of numbers proposed with RSA, this appears to be unachievable. However, as circuit densities increase, it will undoubtably lead to faster implementations.

Modular exponentiation is performed using a series of modular multiplications, in the well known square and multiply algorithm (see example below) [Knu81].

$$80^{53} \equiv ((((((80^2 . 80)^2)^2) . 80)^2)^2 . 80) \qquad \text{(Eq 3.3)}$$

This appears to be an essentially sequential process whose time efficiency cannot be generally improved upon beyond the improvements in the underlying multiplication stages.

In the specific case of RSA some optimizations can be made. Encryption can be performed using a small exponent e (possibly fixed), which limits the number of steps needed significantly (since if e only has say 8 significant bits, then at most 8 square and multiply steps need to be done). The decryption key however, must be long to ensure security [Wei90]. However in the decryption stage the calculations can be done independently (mod p) and (mod q), and the results combined using the Chinese Remainder Theorem. Since p and q are each about 1/2 the size of R, this results in a saving of 1/2 to 3/4 of the time, depending on the multiplication algorithm used. In more detail, the Chinese Remainder Theorem states that there can only be one solution to a set of congruences, given that the modulii are relatively prime [Knu81]. In this specific case, form:

$$M_1 = M \ (mod\ p) = (C\ (mod\ p))^{d\ (mod\ (p-1))} \qquad \text{(Eq 3.4)}$$

$$M_2 = M \ (mod\ q) = (C\ (mod\ q))^{d\ (mod\ (q-1))} \qquad \text{(Eq 3.5)}$$

Then the system of equations:

$$M = M_1\ (mod\ p), \qquad M = M_2\ (mod\ q) \qquad \text{(Eq 3.6)}$$

will have a single solution (being the required message M by construction) of the form:

$$M = [((M_2 + q - M_1)u)(mod\ q)]p + M_1, \qquad \text{(Eq 3.7)}$$

where $pu\ (mod\ q) \equiv 1$

The multiplicative inverse u is calculated at key generation time, and is kept along with the secret decryption key. This method is described by Quisquater et al. [QuC82].

These various choices (outlined above), which can be made in implementing RSA, and the tradeoff's between them are summarised in a significant overview paper by Rivest [Riv84]. They are:

Integer Multiplication of two n-bit integers requires time:

- $O(n^2)$ on a sequential computer using a standard algorithm
- $O(n)$ on special purpose serial/parallel multiplication hardware, with $O(n)$ gates,
- $O(\log n)$ on special purpose parallel/parallel multiplication hardware with $O(n^2)$ gates.

Modular Multiplication of two n-bit integers modulo a third n-bit integer requires the same order of time, but with a constant speedup factor if special algorithms are used as described previously.

Modular Exponentiation requires $O(n)$ multiplications. A constant improvement in time efficiency may be obtained by using either:

- a left-to-right exponentiation algorithm and precomputing powers of M, or alternatively
- a right-to-left exponentiation algorithm and performing the two modular multiplications in parallel.

Implementation tricks appropriate to performing RSA encryption and decryption include:

- using a fast clock rate
- using a short encryption exponent e (however the decryption exponent d must be long)
- using the Chinese Remainder Theorem to operate modulo p and q separately.

3. RSA Key Generation

The first stage of the key generation process for RSA involves finding two large primes p and q, to form the modulus of the system $R = pq$. The choice of these primes is critical to the success of RSA as a secure cryptosystem. Several proposed attacks on RSA are based on the possibility of a bad choice in these primes. The necessary conditions imposed on the primes are summarised in a significant paper by Gordon [Gor84], which is recommended as a basis on which the primality selection routines should be based. He designates as *strong primes* those numbers which satisfy the following conditions:

- p is a large prime

- p is chosen at random from a random seed
- p has a specified number of bits
- $p-1$ has a large prime factor r
- $p+1$ has a large prime factor s
- $r-1$ has a large prime factor t

These conditions imply the following:

$$p = 2jr + 1 \qquad \text{(Eq 3.8)}$$

$$p = 2ks - 1 \qquad \text{(Eq 3.9)}$$

$$r = 2Lt + 1 \qquad \text{(Eq 3.10)}$$

for some j, k, L, where r, s, t are primes. The algorithm consists of finding primes s, t, and then constructing r and p from them. The details needed to do this are given in Gordon's paper [Gor84].

One of the crucial steps though, is to first identify a prime number. Much recent work has been done on primality testing algorithms. Near polynomial-time deterministic algorithms have been developed, as well as polynomial-time probabilistic algorithms which trade-off speed against the probability of error). It thus appears at this time, that primality testing is significantly easier than factorization of large integers (and hence that RSA is both reasonably secure and efficient). A survey and overview of the recent developments in primality testing is given by Pomerance [Pom81], who outlines the various algorithms in use. Nearly all of these are probabilistic, and make use of strong pseudo-prime tests, which if failed prove that the integer is definitely composite. An analysis of a couple of pseudo-prime tests in practical use is given in Bond [Bon84]. The original test is derived from Fermat's Little Theorem, which states that: If n is prime, then for all $a : 0 < a < n$

$$a^{n-1} = 1 \; mod \; n. \qquad \text{(Eq 3.11)}$$

All prime numbers will satisfy this test. Some composite numbers, termed *pseudo-primes*, will also satisfy it. However, if the test is repeated a large (say 100) number of times, then the chance of a composite being accepted is very small. Nonetheless, there are some numbers, called *strong pseudo-primes*, which pass this test for all n. Thus other tests have been suggested, which reduce further the chance of such numbers being accepted as primes. These include the **Solovay Strassen Test**:

Choose a at random from $1 < a < n-1$
Accept n if:

$GCD(a,n) = 1$,

and $\left(\frac{a}{n}\right) \pmod{n} = a^{\frac{(n-1)}{2}} \pmod{n}$

where $\left(\frac{a}{n}\right) \pmod{n}$ is the Jacobi symbol

Otherwise reject.

and the **Rabin Test**:

Let $n = 2^r d + 1$, where d is odd.
Choose a at random from $1 < a < n-1$
Accept n if either:

$a^d = 1 \pmod{n}$

or

$a^{2^j d} = -1 \pmod{n}$,

for some $j : 0 \leq j < r$

Otherwise reject.

These tests may be iterated by repeating them for different, randomly chosen values of a.

The major algorithms for primality testing are described in several recent papers:

- Solovay et al. [SoS77] present the fast Solovay-Strassen probabilistic algorithm for primality testing.
- Couvreur et al. [CoQ82] survey all the techniques known in 1982, summarizing their various advantages and disadvantages (a good overview paper).
- Adams et al. [AdS82] present a new set of pseudo-prime tests which result in fewer inaccurate results.
- Adleman et al. [APR83] present a very fast deterministic algorithm for determining whether a number is prime or not. It uses a series of pseudo-prime tests to narrow the range of possible divisors of a number, which is then exhaustively searched to determine whether there are any factors. It has an upper bound on the running time for a number p of:

  $(\log p)^{c \log\log\log p}$, for some constant c

  An improved version of the algorithm is presented by Cohen et al. [CoL84], along with results from a practical implementation.

Having found the primes p and q, and obtained $R = pq$, and selected an encryption exponent e (which may well be constant for all users, but otherwise will usually be some small prime, see [QuC82], [Wil86], [KBS86b], and [KBS86a]), the final stage of the key generation process to find a number d such that

$$de \equiv 1 \ (mod\ \phi(R)), \qquad 0 \le d < R. \qquad \text{(Eq 3.12)}$$

The usual technique suggested for performing this calculation is to use an extended Euclid's algorithm for finding the $GCD(\phi(R),e)$, as described in Knuth [Knu81]. Alternatives to this include using a binary algorithm, also described in Knuth [Knu81], and extended by Purdy [Pur83] who describes a variant using carry-save hardware which can perform the GCD calculation in time $O(n)$ with $O(n)$ gates; or a systolic algorithm developed by Brent and modified by Bojanczyk [BoB86], which provides a similar level of performance. Some constructive techniques for generating strong RSA primes are described by Demytko [Dem89] and Maurer [Mau90].

4. RSA Implementation in Practice

The better known RSA implementations (either in full, or of modular exponentiation implementations), may be considered in two groups, software implementations on existing computers, and specialised hardware implementations:

4.1. Software Implementations of RSA

A number of software implementations of RSA, typically encrypting at 1-10 bits/second, have been attempted. These include:

- Schaller et al. [ScA83] describe an implementation for a Z-80 microprocessor of a system where RSA is used to distribute keys for a private-key cryptosystem (DES), along with the associated name and key servers, and message flow control, as an experiment in the use of such a system; Similar schemes are described by Jackson et al. [JMN84] and Muller-Schloer [Mul83].
- Coyne [Coy85] describes a full RSA system, used to provide secure electronic mail, which has been implemented at the Basser Dept. of Computer Science at the University of Sydney.
- Aruliah [Aru85] presents an ISO Standard Pascal implementation which has been fully tested on a number of systems, and which it is suggested may be used as a comparison suite for other implementations.
- Burton [Bur84] gives code in Ratfor which implements a full RSA system.
- Dusse [Dus91] describes an implementation of RSA and other cryptographic algorithms on the Motorola 56000 DSP. The RSA implmentation is capable of encrypting at 11 kbits/sec. a full RSA system.

4.2. Hardware Implementations of RSA

A number of hardware implementations of RSA have been attempted. These include:

- Rivest [Riv80]. This is a VLSI implementation using a 512 bit ALU providing modular arithmetic including exponentiation, prime number generation, GCD calculation, extraction of a small common divisor if one exists, and RSA key generation. It could achieve an encryption rate of 1200 bits/second.

- Rieden et al. [RSW82] report on the work done by a team at Sandia National Labs in the US, designing a radiation hardened LSI chip set, which provided exponentiation of 336-bit messages at a rate of 430 bits/sec, using two chips.
- British Telecom have released advanced information on a 256-bit RSA exponentiator chip [BBB86], known as the METEOR VLSI Public Key Device, which will be capable of encrypting at 9600 bits/second.
- Serpell et al. [SBC84], describe a hybrid cryptographic system which uses RSA to distribute keys which are then used in a private-key (B-Crypt scheme). The RSA calculations are done on the British Telecom hardware exponentiation chip.
- Orton et al. [ORS87] present some results from the implementation of a modular RSA chip which may be stacked to obtain longer bit-lengths.
- Janiak [Jan87] describes a chip set, based on the TMS320C10 DSP processor, which can exponentiate 640 bit numbers in 1.7 secs. It is designed to be used as an auxiliary processor in PC's and Workstations.
- Hoornaert et al [HDV89] describe a chip being developed by Cryptech, capable of encrypting at 17 kbits/sec.

These and other hardware RSA implementations are compared in a recent survey by Brickell [Bri90]. The fastest implementation that he notes is that by Cryptech. Chips by British Telecom, Plessey and Sandia are quoted to encrypt at 10 kbits/sec. Brickell also notes that recent DSP implementations have achieved speeds comparable to this.

5. Conclusion

In this chapter, various alternative methods for implementing the RSA cryptosystem, being the most widely regarded public-key cryptosystem, have been outlined. Methods surveyed include conventional multi-precision arithmetic with modulo reduction performed either in software on conventional machines or specially designed long bit-length hardware; specialised multiplication algorithms which may be optimised for use on high-speed Digital Signal Processing (DSP) chips; and the use of carry-save multiplication techniques on specially designed chips. None of these methods seem to offer much hope of implementing exponentiation, and hence the RSA scheme sufficiently fast enough to be able to encrypt all data being exchanged on modern high-speed data links. Hence there is a need for good private-key algorithms for use in a hybrid scheme.

CHAPTER 4

Design Rules for the DES

1. Introduction

The Data Encryption Standard (DES) [NBS77] was developed out of the Lucifer cipher. At the time of its introduction, there was a considerable amount of controversy over the choice of a 56-bit key [DiH77], [Hel79a]. This was on the grounds that such a key length was small enough to be exhaustively searched on machines which could be constructed in the foreseeable future. With the march of computer technology, this risk appears to be rising. However, on all other accounts DES appears to be an excellent cryptosystem. The DES has withstood a concerted analysis by the open research community, since its adoption in 1977. The most successful attack to date, that of Biham and Shamir [BiS90] still performs worse than an exhaustive key-space search on the DES with the full 16 rounds.

Given this basically favorably assessment, the DES will be closely examined as a basis for developing design rules for new cryptosystems. This chapter will discuss some possible design criteria which could have been used in the DES, derived both from those reported in the literature, and from some initial observations by the author. These show that the DES was very carefully designed, and that its various components are far from random. It will then illustrate how these criteria could be applied in the design of new Feistel ciphers with either 64-bit or 128-bit keys and data.

2. Description of the DES

The basic DES enciphering chain was shown in Fig 2.7. In brief, the input block is permuted by an initial permutation IP, is then subjected to a complex key-dependent computation (Fig 2.8), and is finally permuted by IP^{-1}, the inverse of the initial permutation. On deciphering, the same process is used, except that the complex component is run backwards, using keys in reverse order, via the method shown above.

In more detail, the input block X of 64 bits is first permuted by the initial permutation IP, and then divided into two halves $L(0)$ and $R(0)$. The complex key-dependent function consists of sixteen rounds in which the left component $L(i-1)$ is exclusive-or'd (XOR'd) with the output of a non-linear function g whose input is both the right component $R(i-1)$ and part of the key $K(i)$ (Fig 2.9). The halves are then interchanged to form the input for the next round (with the exception of the final round). Finally the output is permuted by IP^{-1} (the inverse of the input permutation IP) to yield the output ciphertext Y.

3. Known DES Design Criteria

All the functions used in the DES are specified in tabular form. No description is provided on why they are chosen. In fact it is stated that the design criteria are classified.

Nonetheless, some of the criteria have been released, and others may be deduced by examining the tables as given. The search for the design criteria is motivated by two reasons, first to aid in the cryptanalysis of the DES, and second in designing new and extended Feistel ciphers. Whilst the primary concern in this thesis is with the latter purpose, the results should also be of interest to those interested in cryptanalysis of such ciphers. The structure of the DES will be examined in terms of the various permutation, substitution and key scheduling components, with design rules being devised for each.

3.1. S-box Design Criteria

The S-boxes in DES are the prime source of complexity in the algorithm, and are largely responsible for the security of the scheme. Consequently, considerable time has been spent on analysing them, and hence more is known about them than the P-boxes. The actual criteria used are classified secret. It has been stated[1] that there were 12 (possibly 13) criteria used, which resulted in about 1000 suitable S-boxes, of which the implementors chose 8. Some of the criteria used were reported to an NBS workshop on cryptography [BGK77], and are:

- S1 Each row function in an S-box is a permutation of the integers 0 to 15
- S2 No S-box is a linear or affine function of the input.
- S3 Changing 1 input bit to an S-box results in changing at least 2 output bits
- S4 $S(x)$ and $S(x \oplus 001100_2)$ must differ in at least 2 bits (where $S(x)$ is the output of an S-box given input vector x).

The following rules were labelled (see [BGK77]) as resulting from design criteria:

- S5 $S(x) \neq S(x \oplus 11ef00_2)$ for any choice of e or f.
- S6 The S-boxes were chosen to minimise the difference between the number of 1's and 0's in any S-box output when any single input bit is held constant.

These rules were analysed by Brickell et al. [BMP87]. Rule S1 states, in effect, that each row of each S-box is a 1-1 function. Rule S2 ensures that the DES algorithm as a whole is not linear or affine. If it were, it would substantially reduce the security of the scheme, and enable reasonably easy recovery of a key with a set of plaintext/ciphertext pairs. The next two rules S3 and S4, seem to incorporate the concept of the avalanche effect, in that it ensures that small changes to one or two bits of input result in large changes in the output. Rule S5, reduces the probability that two different inputs would be mapped onto the same output (nb: function g is neither 1-1 nor *onto*, it must only generate the same output from the same input and key); and that the image space (ie: the range of possible values generated by g) is as large as possible. Rules S1, S3, and S4 seem to imply rule S6 (which accords with the original statement that it is a consequence of the design criteria), and is related to the S-boxes being permutation functions.

[1] by Alan G. Konheim at a Seminar series on Cryptography in Amsterdam, 1986 [Kon86]

Another consequence of design criteria was noted by Meyer and Matyas in their book [MeM82], namely that:

- S7 the S-boxes chosen required significantly more logical minterms to implement than a random choice would require. A minterm is a logical AND (boolean product) of input bits (and their negations), which form a necessary (but not sufficient) condition for a particular output bit to be asserted. An output bit may have its value completely described by the logical OR (boolean sum) of all its minterms. The number of minterms in such an expression (after simplification) is a measure of its complexity. In the worst case, for n input bits, 2^n minterms may be needed to describe each output bit.

A random choice of S-box would lead to a distribution with a mean around 32. However Meyer and Matyas stated that as they added more design rules, the average number of minterms rose from around 46 in the early stages, to around 56 in the final design (with the distribution mostly ranging from 52 to 59). The final choice of 8 boxes was taken from the low end of this range (52 and 53) in order to make hardware implementation easier. This rule would indicate that the S-boxes are considerably more complex than a random choice would provide. It also implies that cryptanalysis of DES via formal coding techniques is extremely difficult, as was illustrated by Schaumuller-Bichl [Sch82].

3.2. P-box Design Criteria

There are a number of permutation networks in the DES design, IP, IP^{-1}, E, P, $PC1$, and $PC2$. They appear to be performing various functions. No design criteria for them have been released. However by examination of the various networks, certain features can be deduced. In general these all appear to be motivated by the avalanche and completeness criteria, in that there is a wide dispersion of outputs across the inputs to the next stage. We will examine these networks in turn.

3.2.1. IP and IP^{-1}

These two permutations form the initial and final steps respectively in the enciphering chain for DES. It has been argued [DDF83], [Des84] that these permutations have no cryptographic significance, since exhaustive plaintext searches can be conducted on messages after these transformations have been applied. This argument probably does not hold when DES is used in one of its chaining modes. Other reasons put forward for their inclusion are that hardware implementations would benefit, and that it might help thwart other forms of attack. Further evidence for this may be found in Schaumuller-Bichl [Sch82] who shows that the complexity of DES depends mainly on the S-boxes, key scheduling and number of rounds, rather than the permutations. It was also noted that IP is extremely regular. As is shown in [HGD84], if the input block is broken into 8-bit bytes, then it may be implemented as eight iterations of an 8-bit permutation {2,4,6,8,1,3,5,7} into eight shift registers: a much more compact hardware realization, than a true 64-bit permutation would imply. Its functionality seems to be to distribute bits from each input byte over all eight bytes in the input block. Also it sends all even numbered bits to the left hand vector, and odd numbered bits to the right hand vector.

3.2.2. $PC1$

This transformation performs a very similar function to IP, and additionally strips off the parity bits to reduce the 64-bit key value entered down to the 56-bit key actually used. It is nearly as regular as IP, except for an asymmetry around the two 28-bit halves. As with IP, it may be implemented with eight iterations of a 7-bit permutation {0,1,2,3,4,5,6,7} (the parity bit 8 is discarded). However now the permuted value is read out by shifting two 28-bit halves toward the centre, rather than to one end, as in IP (see [HGD84]).

3.2.3. P and E

It has been noted by Davies [Dav82], Davio et al [DDF83], and Desmedt [Des84], that P and E need to be analysed together as the functional composition $P.E$. In fact $S.P.E$ should be analysed thus, being one round in an S-P network, since

$$R(i) = L(i-1) \oplus P(S(E(R(i-1)) \oplus K(i))). \tag{Eq 4.1}$$

The expansion function E is very regular. If the 32-bit input block is broken into eight 4-bit blocks, then the 6-bit output block is formed by concatenating the adjacent bit from the neighbouring blocks. If the 6 input bits to an S-box are labelled $abcdef$, then

- E1 bits ab and ef are replicated in the neighbouring blocks and include the S-box selector bits for both the current (bits a and f), previous (bit b), and next (bit e) S-boxes,
- E2 bits cd are substituted in the current S-box only.

Thus E serves to provide some diffusion by spreading the input bits at each round into adjacent S-boxes.

Now consider the composition $P.E$. The input of P is divided into 4-bit outputs from the eight S-boxes, and the output from E is divided into 6-bit inputs to the eight S-boxes at the next stage. Provided the S-boxes fulfil their design requirements, it may be assumed that all S-box outputs are equivalent, and that the inputs may be considered in the three pairs indicated above. Instead of expressing this permutation in terms of bit positions, it may be written in terms of which S-box output is connected by $P.E$ to each input bit at the next stage (see Table 4.1).

Table 4.1 - P.E Permutation

S-box	Takes Inputs from S-boxes						Excluded S-box
	a	b	c	d	e	f	
1	7	4	**2**	5	6	**8**	3
2	6	8	**3**	7	5	**1**	4
3	5	1	**4**	6	7	**2**	8
4	7	2	**5**	8	**3**	1	6
5	3	1	2	**6**	**4**	8	7
6	4	8	**7**	1	3	**5**	2
7	3	5	4	**8**	2	**6**	1
8	2	6	3	**1**	**7**	4	5

This form highlights regularities not previously noted in the literature. From it the author observes that:

- P1 each S-box input bit comes from the output of a different S-box
- P2 no input bit to a given S-box comes from the output of the same S-box
- P3 an output from S(i-1) goes to one of the ef input bits of S(i), and hence via E an output from S(i-2) goes to one of the ab input bits.
- P4 an output from S(i+1) goes to one of the cd input bits of S(i).
- P5 for each S-box output, two bits go to ab or ef input bits, the other two go to cd input bits. This was noted in Davies [Dav82].

These rules all appear consistent with the implementation of the avalanche and completeness effects. They ensure that every output bit becomes a function of each input bit in as few rounds as possible. Meyer [Mey78] (also in [MeM82]) has shown that with the networks specified in DES, then after 5 rounds, every output bit is a function of all input bits. This analysis is repeated an extended in the next section. This result would appear to be an important consequence of the design criteria.

Permutations satisfying these criteria were generated. First rules P1 and P2 imply that each S-box takes inputs from the outputs of 6 other distinct S-boxes. Hence one S-box apart from itself must have been missed in this selection of inputs. There are 264 ways possible to do this, which shall be termed *exclusion sets*. Then rules P3, P4 and P5 were used to generate classes of possible permutations[2]. A total of 178 such classes were found from 96 of the 264 exclusion sets (the others generated no valid permutations, due to the missing S-box being one of those required by rules P3 or P4). Each of these classes could generate a number of possible permutations by swapping the order of bits in each of the pairs ab, cd, and ef; or by rearranging the order of the four output bits from each S-box.

It is worth noting that both Davies [Dav82] and Davio et al [DDF83] have commented that the actual order of output bits from each S-box is irrelevant. This is because a functionly equivalent $S.P.E$ combination can be formed by rearranging the order of bits in each S-box (by rearranging columns), and then altering permutation P to compensate. As Hoornaert et al [HGD84] show, this is desirable in order to design an efficient hardware implementation which minimises the number of wire crossings (and hence area) in P. It certainly illustrates that any analysis of these components needs to be performed on them as a whole.

3.2.4. $PC2$

The permutation $PC2$ is responsible for selecting 48 of the 56 key bits available on each round. The output of this permutation can be analysed in terms of the eight 6-bit blocks

[2] the mandatory wirings required by rules P3 and P4 were arbitrarily assigned to bits c and e/a respectively.

which become S-box inputs after being XOR'd with the expanded message block. As with the other components in the key scheduling stage, it can be regarded as two distinct 28 to 24-bit selections and permutations, with no interaction between the two halves. Again, like the permutation P, $PC2$ needs to be analysed in conjunction with the other components of function f, and the key scheduling function KS. We have

$$R(i) = L(i-1) \oplus P(S(E(R(i-1)) \oplus PC2(KS(U, i)))) \qquad \text{(Eq 4.2)}$$

$$where \;\; U = PC1(K)$$

The author has noted that in each 6-bit output block corresponding to an S-box input, if the bits are sorted into ascending order of their input bits, then:

- C1 bits permuted to the same S-box input are no closer than 3 bits apart
- C2 bits permuted to an S-box input must have a span from lowest to highest input bit number of at least 22 of the 28 bits in each key half (alternatively, the average spacing must be at least 3 2/3)
- C3 bits permuted to the selector bits a, f on a given S-box must not be adjacent in the sorted list of input bits
- C4 bits not selected by PC2 must be at least 3 places apart

Rule C1 implies that the same bit of the key will not be used as input to the same S-box on successive rounds (since in the key schedule, bits are shifted only 1 or 2 places). Rule C2 results from application of the avalanche and completeness effects to ensure that the output depends on all key bits in as few rounds as possible, again as confirmed by Meyer [Mey78] (also in [MeM82]). Rule C3 ensures that the same bit is not used as a selector bit on successive permutations to an S-box. Rule C4 ensures that the same bit will not be missed on successive rounds (ie: that all keys bits are used as soon as possible).

3.3. Key Scheduling

Aside from the S-boxes and P-boxes, the other component critical to the security of DES is the *key scheduling* algorithm (see Schaumuller-Bichl [Sch82]). Several authors [Ber82], [Dav82], [Des84], [QDD86] have noted that DES in fact uses a 768-bit key (formed from a 48-bit key to each of 16 stages), which is generated from the 56-bit key supplied. If either the key scheduling is poor [GrT78], or a bad choice of key is made [Dav82], then the same subkey might be repeated for more than one round, leading to insecurities in the scheme. No design rules have been published for the scheme currently in use. The design of the key scheduling function is obviously related to the design of $PC2$ by rules C1 and C4 given above. The author notes that the key scheduling function ensures that:

- K1 each bit is used as input to each S-box
- K2 no bit is used as input to the same S-box on successive rounds
- K3 the total number of bits rotated is 56 (which implies that $K(0) = K(16)$, enabling the decryption operation to use right shifts in reverse order).

Suggestions for alternate forms of key scheduling have been made in Berson [Ber82] and Quisquater et al [QDD86]. Berson suggests using a key of 768 bits, bypassing the key scheduling stage of the DES and used as sixteen 48-bit words. However Biham and Shamir have shown that this is ineffectual [BiS90], and offers little additional security. Quisquater et al suggest alternative key scheduling strategies to expand the original key to form the sub-keys. However these again provide little additional security to Biham and Shamirs attack, which effectively recovers the sub-keys, rather than the original key. Since it is the sub-keys which are used to perform the decryption, schemes obscuring the original key provide no additional security. It seems that key and data block size should be comparable for optimum security. Extending one but not the other provides little additional security.

4. Ciphertext Dependency Analysis

One technique to provide a measure of the ability of a given S-P design to fulfil the completeness effect, is to analysis how rapidly the output bits become dependent on every input and key bit. This analysis was first performed by Meyer [Mey78] (also in [MeM82]). A basic assumption in this analysis is that each S-boxes ensures that every output bit was a complex function of all its input bits.

4.1. Ciphertext Dependence of Plaintext Bits

To provide a measure of this dependency, a $64*64$ array $G_{a,b}$ is formed. Each element $G_{a,b}(i,j)$ of which specifies a dependency of output bit $X(j)$ on input bit $X(i)$, between rounds a and b. The number of marked elements in $G_{0,r}$ indicates the degree to which complete dependence was achieved by round r. Details of the derivation of this matrix, and the means by which entries are propagated, may be found in [MeM82]. The author repeated this analysis for the current DES, and extended it by analysing each of the 178 candidates for permutation P, as well as for a worst case P (see Table 4.2).

Table 4.2 - Worst Case DES P

1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8

The results are summarised in Table 4.3, and lead to the following conclusions:

- the current DES P results in complete ciphertext-plaintext dependence after 5 rounds.
- all 178 candidate permutations P result in complete dependence after 5 rounds.
- the worst case permutation P results in complete dependence after 7 rounds. Here E alone is relied upon to provide diffusion.

Table 4.3 - Dependency of Ciphertext bits on Plaintext bits in DES			
Round	Std P	Candidate P	Worst P
1	6.25	6.25	6.25
2	32.06	32.03-32.10	21.09
3	73.49	73.44-73.58	43.75
4	96.90	96.87-96.95	68.75
5	100.00	100.00	89.06
6	100.00	100.00	98.44
7	100.00	100.00	100.00
8	100.00	100.00	100.00

The P-box used in the current DES has a propagation profile that falls fairly close to the median of the 178 classes identified. It thus appears to be a fairly representative example of them. In conjunction with the substantially inferior profile of the worst case P-box, this also provides a strong indication that the design rules identified are comprehensive, at least as far as this aspect of the P-box design is concerned.

4.2. Ciphertext Dependence on Key Bits

This analysis is substantially more complex, since it is dependent on the choices of permutations P and PC2 as well as the KS the key schedule. This dependency has been analysed for each of the P permutations used above, with the current key schedule, and with the current, worst case, and a sample regular PC2. To quantify this dependency, a $64*56$ array $F_{a,b}$ is formed, each element of which specifies a dependency of output bit $X(j)$ on key bit $U(i)$. The vector U is that formed after PC1 is applied, ie $U = PC1(K)$. The number of marked elements in $G_{0,r}$ will be examined to provide a profile of the degree of dependence achieved by round r. Details of the derivation of this matrix, and the means by which entries are propagated, as before, may be found in [MeM82]. The author repeated this analysis for the current DES, and extended it as indicated before. The results are summarised in Table 4.4 for the current PC2, Table 4.5 for a worst case PC2, and Table 4.6 for a sample PC2. The sample PC2 was designed to satisfy the current rules known, and to use a regular wiring. It is listed in Table 4.7.

These results have led to the following conclusions:

- there are two errors in Meyers analysis. Firstly in his key schedule implementation he appears to have shifted, rather than rotated key bits. This alters the "type" of dependence, but not the number of dependent bits. Secondly, he appears to have miscalculated the dependence percentage. This same error was made in his cipher-plaintext dependence analysis in his original paper, but was corrected in his book. This correction did not occur with the cipher-key dependence analysis, and it is suspected that the same systematic error is at fault.
- the current DES PC2 results in complete ciphertext-key dependence after 5 rounds for current candidate permutations P, and in 7 rounds for the worst case permutation P.

- the worst case DES PC2 results in complete ciphertext-key dependence after 7 rounds for current candidate permutations P, and in 9 rounds for the worst case permutation P. Here E along with the key schedule are relied upon to provide diffusion.
- the sample DES PC2 results in complete ciphertext-key dependence after 5 rounds for current candidate permutations P, and in 6 rounds for the worst case permutation P. Interestingly, the profile is slightly worse for the current and candidate permutations P, but significantly better for the worst case.

Table 4.4 - Dependency of Ciphertext bits on Key bits in DES with Current PC2

Round	Std P	Candidate P	Worst P
1	5.36	5.36	5.36
2	39.17	38.39-39.29	24.67
3	82.25	81.25-82.48	53.12
4	98.44	98.21-98.55	79.69
5	100.00	100.00	95.65
6	100.00	100.00	99.78
7	100.00	100.00	100.00
8	100.00	100.00	100.00

Table 4.5 - Dependency of Ciphertext bits on Key bits in DES with Worst Case PC2

Round	Std P	Candidate P	Worst P
1	5.36	5.36	5.36
2	42.19	42.19	21.65
3	81.47	81.47	43.97
4	91.29	91.29	67.41
5	96.21	96.21	86.61
6	99.55	99.55	95.31
7	100.00	100.00	97.99
8	100.00	100.00	99.55

Table 4.6 - Dependency of Ciphertext bits on Key bits in DES with Sample PC2			
Round	Std P	Candidate P	Worst P
1	5.36	5.36	5.36
2	38.62	38.39-38.84	22.77
3	81.47	81.25-81.70	48.66
4	98.21	98.21	75.45
5	100.00	100.00	94.20
6	100.00	100.00	100.00
7	100.00	100.00	100.00
8	100.00	100.00	100.00

Table 4.7 - Sample DES PC2											
1	5	10	15	20	25	2	6	11	16	21	26
3	7	12	17	22	27	4	8	13	18	23	28
29	33	38	43	48	53	30	34	39	44	49	54
31	35	40	45	50	55	32	36	41	46	51	56

5. Design of new DES style Ciphers

Having completed a preliminary analysis of the DES based on the results documanted in the literature, and on some initial observations on the design of some of the permutations, the author will now make some preliminary comments about possible design directions for a new Feistel cipher with a DES-like structure. These will be based on extending the work of Morris-Yates [Mor86], who used a subset of the criteria detailed previously. It involves widening all the data paths to twice the current width, and extending or redesigning all the S-boxes, P-boxes, and key scheduling. The initial motivation for doing this was to overcome the doubts about the ability of DES to withstand an attack based on exhaustive key-space searches by doubling the key length. This raises the effort needed from 2^{55} trials to 2^{111} trials[3], an impossibility in the foreseeable future. It will also result in a scheme for which all the design criteria used are known, and hence open to criticism; and which is not bound by foreign government restrictions. These preliminary design comments will help identify the areas where further research is needed in formulating suitable design rules.

5.1. S-box Redesign

Since sixteen S-boxes will be required in the extended scheme, an additional eight boxes will be required. One possibility is simply to reuse the existing eight boxes, either by using them twice, or by designing an additional eight only. These options were rejected

[3] there is a saving of half resulting from the well known symmetry in DES, namely that $DES(\bar{X}) = \overline{DES(X)}$

both for copyright reasons, and because of doubts about the cryptological security of the current S-box 4 [BMP87], [Sha86]. Hence it was decided that sixteen new S-boxes will be generated. Ideally all possible 6*4 bit functions would be generated and evaluated against Both the known design rules, and additional criteria based on non-linearity criteria suggested by Pieprzyk [PiF89] and Carroll[4]. A subset will then be selected from those which satisfy these criteria. Because of the time this would take, it is envisioned that a random selection of boxes based on the non-linearity requirement will be generated and tested against the remaining criteria. This will be done both to generate suitable boxes for the proposed design, and to extend work (see [BMP87]) on the analysis of the design rules, identifying both redundant rules and possibly new rules.

5.2. P-box Redesign

The use of double width data paths implies a complete redesign of all the permutations. IP, IP^{-1}, and $PC1$ may be easily extended because of their regular design, if they are included at all. Morris-Yates [Mor86] has argued that because of doubts about their cryptological usefulness, they need not be provided at all (with the exception of the parity stripping component of $PC1$), saving a significant portion of silicon real-estate. If they are provided then the efficient implementation suggested by Hoornaert et al [HGD84] should be used. These functions will not be used in the extended scheme.

The expansion function E may also be easily extended due to its regular design. This leaves the important permutations P and $PC2$. Because these are central components of the S-P network, and of the key schedule, they are critical to the security of the new scheme. As with the S-boxes, a sample will be randomly generated and tested against the known rules in an effort to isolate any further design criteria, and to identify $S.P$ combinations which are functionally identical. It is here that this work differs most from Morris-Yates, who chose purely random permutations. Initially these will be specified as follows:

- P will use the existing design rules, but will have a regular wiring array. eg one proposal has inputs for $S(i)$ from $S(i-2), S(i+1), S(i+8), S(i+5)$ (see Table 4.8). Such a regular array leads to efficient implementation in both hardware and software.

Table 4.8 - Sample Extended DES P

15	2	9	6	16	3	10	7	1	4	11	8	2	5	12	9
3	6	13	10	4	7	14	11	5	8	15	12	6	9	16	13
7	10	1	14	8	11	2	15	9	12	3	16	10	13	4	1
11	14	5	2	12	15	6	3	13	16	7	4	14	1	8	5

- PC2
  will use the existing rules modified to suit the larger shifts of 3 or 4 places needed in the Key Schedule. This implies that the constant 3 in rules R1 and R4 will become 5

[4] in personal communications.

in the initial design. Again as in the case of P, a regular wiring will be used, though it is complicated by the need to select a subset of the bits. One example distributes a bit to each S-box in turn, skipping every 7th bit (see Table 4.9).

Table 4.9 - Sample Extended DES PC2

1	10	20	29	38	48	2	12	21	30	40	49
3	13	22	31	41	50	5	14	23	33	42	51
6	15	24	34	43	52	7	16	26	35	44	54
8	17	27	36	45	55	9	19	28	37	47	56
57	66	76	85	94	104	58	68	77	86	96	105
59	69	78	87	97	106	61	70	79	89	98	107
62	71	80	90	99	108	63	72	82	91	100	110
64	73	83	92	101	111	65	75	84	93	103	112

5.3. Key Scheduling Extension

In the extended algorithm, the same style of shift and select key scheduling will be used. A new shift schedule will be designed to satisfy the known design criteria, which will involve some interaction with the design of $PC2$, as indicated before. Since sixteen rounds are planned in the extended design (provided analyses of the dependence of output bits on input and key prove satisfactory), this implies rotations greater than 2 bits in order to have a total rotation of 112 bits in sixteen rounds. This impacts on the minimum spacing requirement between bits selected in $PC2$. A sample schedule is given in Table 4.10.

Table 4.10 - Key Schedule for an Extended DES

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Shift	3	3	3	4	4	4	4	3	3	3	4	4	4	4	3	3
Total	3	6	9	13	17	21	25	28	31	34	38	42	46	50	53	56

5.4. Dependency Analysis of An Extended DES

To provide an indication of the suitability of the extended rules, Meyer's dependency analysis was applied to the sample permutations developed, as well as to the worst case permutations P and PC2 (given in Tables 4.14 and 4.15). The results are given in Table 4.11 for Ciphertext-Plaintext dependence, Table 4.12 for Ciphertext-Key dependence with a worst case PC2, and in Table 4.13 for Ciphertext-Key dependence with the sample PC2. These results show that with the sample permutations, the dependency profile is only one round worse than in the current case (consistent with a doubling of the data lengths). With the worst case permutations however, the profile worsens significantly.

6. Conclusion

In this chapter, all the known design criteria for the S-boxes, P-boxes, and key scheduling in the Data Encryption Standard have been identified. These all appear to be related to the avalanche and completeness effects. This chapter concludes with a discussion on how these criteria could be used to design an extended DES, as well as aiding in identifying further design criteria. The following chapters will extend the analysis of the overall structure, in particular that of permutation P and the key schedule.

Table 4.11 - Dependency of Ciphertext bits on Plaintext bits in Extended (128-bit) DES with a Sample PC2

Round	Sample XDES P	Worst XDES P
1	3.125	3.125
2	17.68	10.55
3	50.98	21.88
4	84.47	34.38
5	98.44	46.88
6	100.00	59.38
7	100.00	71.88
8	100.00	84.38
9	100.00	94.53
10	100.00	99.21
11	100.00	100.00
12	100.00	100.00

Table 4.12 - Dependency of Ciphertext bits on Key bits in Extended (128-bit) DES with a Worst Case PC2

Round	Sample XDES P	Worst XDES P
1	2.68	2.68
2	19.08	11.05
3	57.37	22.94
4	89.51	35.71
5	98.44	49.33
6	99.89	63.28
7	100.00	76.12
8	100.00	87.44
9	100.00	95.76
10	100.00	99.22
11	100.00	99.89
12	100.00	100.00

Table 4.13 - Dependency of Ciphertext bits on Key bits in Extended (128-bit) DES with a Sample Regular PC2		
Round	Sample XDES P	Worst XDES P
1	2.68	2.68
2	20.54	13.39
3	61.16	31.36
4	92.86	48.77
5	99.55	62.95
6	100.00	76.00
7	100.00	87.61
8	100.00	95.65
9	100.00	99.22
10	100.00	100.00
11	100.00	100.00
12	100.00	100.00

Table 4.14 - Worst Case Extended DES P

1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8
9	9	9	9	10	10	10	10	11	11	11	11	12	12	12	12
13	13	13	13	14	14	14	14	15	15	15	15	16	16	16	16

Table 4.15 - Worst Case Extended DES PC2

1	2	3	4	5	6 7	8	9	10	11	12	
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
57	58	59	60	61	62	63	64	65	66	67	68
69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92
93	94	95	96	97	98	99	100	101	102	103	104

CHAPTER 5

On the Design of Permutation P in Feistel Ciphers

1. Introduction

In the previous chapter, some possible design criteria used in the DES, both those reported in the literature, and those noted by the author, were discussed. This chapter further considers the design of the permutation box P, which provides the diffusion component of a *substitution-permutation* network in Feistel ciphers. Two key properties of such S-P networks are the *avalanche* property, and the *completeness* property, which ensure that every output bit becomes a function of each input bit in as few rounds as possible. Meyer [Mey78] (also in [MeM82]) has quantified this property for the DES, by showing that after 5 rounds, every output bit is a function of all input bits. In the previous chapter, this form of analysis was used as a measure of effectiveness in the design of permutation P. This analysis is to be used further in this chapter to extend the understanding of the role of permutation P in Feistel ciphers, and to identify some more general design rules that those used in the DES, which will be of use in designing new ciphers.

2. Empirical P-box Design Criteria

The central component of the DES cryptosystem is the function g. This is implemented as a composition of an expansion function E which provides an autoclave function, eight substitution boxes (S-boxes) S which *confuse* the input bits, and then a permutation P which *diffuses* the outputs from each S-box to the inputs of a number of S-boxes in the next stage. A more detailed description of these functions may be found in [NBS77], [ASA85] or [SeP89]. For the purposes of this analysis, it is convenient to regard the DES as a mixing function, as detailed in Davio et al. [DDF83], which emphasises the analysis of the functional composition $P.S.E$ (see Fig 5.1.).

In the previous chapter the following analysis of, and design rules for, permutation P were derived. This was done by analysing the $S.P.E$ functional composition, which forms one round of the S-P network in the DES:

$$R(i) = L(i-1) \oplus P(S(E(R(i-1)) \oplus K(i))). \qquad \text{(Eq 5.1)}$$

Fig 5.1 - DES as a Mixing Function

The input of permutation P may be divided into 4-bit outputs from the eight S-boxes, and the output from E is divided into 6-bit inputs to the eight S-boxes at the next stage. Instead of expressing this permutation in terms of bit positions, it may be written in terms of which S-box output is connected by $P.E$ to each input bit of the S-boxes at the next stage (see Table 5.1). Provided the S-boxes fulfil their design requirements, it may be assumed that all S-box outputs are equivalent, and that the inputs may be considered as three pairs ab, cd and ef. Note that due to the expansion function E, columns ab are identical to columns ef for the previous S-box.

Table 5.1 - P.E Permutation							
S-box	Inputs from S-boxes						Excluded S-box
	a	b	c	d	e	f	
1	7	4	**2**	5	6	**8**	3
2	6	8	**3**	7	5	**1**	4
3	5	1	**4**	6	7	**2**	8
4	7	2	**5**	8	**3**	1	6
5	3	1	2	**6**	**4**	8	7
6	4	8	**7**	1	3	**5**	2
7	3	5	4	**8**	2	**6**	1
8	2	6	3	**1**	**7**	4	5

From this the author derived the following set of empirical rules for designing permutation P:

- P1 Each of the S-box input bits ab cd ef come from the outputs of different S-boxes.
- P2 None of the input bits ab cd ef to a given S-box S(i) comes from the output of that same S-box S(i).
- P3 An output from S(i-1) goes to one of the ef input bits of S(i), and hence via E an output from S(i-2) goes to one of the ab input bits.
- P4 An output from S(i+1) goes to one of the cd input bits of S(i).
- P5 For each S-box output, two bits go to ab or ef input bits, the other two go to cd input bits as noted in [Dav82].

These rules all appear consistent with the implementation of the avalanche and completeness effects by ensuring that every output bit becomes a function of each input bit in as few rounds as possible.

Permutations satisfying these rules have been generated. A total of 178 permutations were found, from 96 possible exclusion sets (the mandatory wirings required by rules P3 and P4 were arbitrarily assigned to bits c and e/a respectively). Each of these permutations could generate a number of possible permutations by swapping the order of bits in each of the pairs ab, cd, and ef; or by rearranging the order of the four output bits from each S-box. However, the author believes that based on the tests conducted to date, these variations are not significant, and notes that they do not change the result of the dependency analysis. Davies [Dav82] and Davio et al. [DDF83] have also observed that a functionly equivalent $S.P.E$ combination can be formed by rearranging the order of the S-box output bits (by rearranging columns), and by then altering permutation P to compensate.

To provide a measure of the effectiveness of the derived permutations, Meyer's analysis [Mey78], [MeM82] of ciphertext dependence on plaintext bits was extended for the 178 permutations derived from these empirical rules. Briefly, following Meyer, this analysis may be described as follows. To provide a measure of this dependency, a $64*64$ array $G_{a,b}$ is formed. Each element $G_{a,b}(i,j)$ specifies a dependency of output bit $X(j)$ on input bit

$X(i)$, between rounds a and b. The number of marked elements in $G_{0,r}$ indicates the degree to which complete dependence was achieved by round r. Details of the derivation of this matrix, and the means by which entries are propagated, may be found in [MeM82]. The extended analysis [Bro89] for the current DES, the 178 empirically generated permutations, and the worst case P (see Table 5.2), gave results as shown in columns 2, 3, and 7 of Table 5.3.

Table 5.2 - Worst Case DES P
1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 6 6 6 6 7 7 7 7 8 8 8 8

3. Further Analysis of the Design of Permutation P

On closer examination, it was noted that these empirically derived rules for the design of permutation P, when written as in Table 5.1, actually result in a latin square. A *latin square* is a square of numbers in which each number occurs exactly once in each row and each column (see [DeK74]). Latin squares were first used as perfect randomisers in medical and agricultural experiments, and a considerable body of theory exists to show that particular latin squares are the best possible for a given scenario [Cox58, Ch 3,5], [Mon76, Ch 4]. Thus it is not unreasonable to assume that such a design would also act as an efficient randomiser of bits. Note that the permutation P used in the current DES is not a latin square, however it may be transformed into one by selectively swapping some ab/ef (these being equivalent due to E) and cd pairs to align the highlighted values in Table 5.1 into the same columns. This transformation makes no difference to the dependency result obtained. Such a result cannot be accidental, and must be related to the desired properties of permutation P, namely to provide maximal diffusion among the S-boxes.

In order to explore this further, some possible permutations which could be used in a DES type system, and which form a latin-square when written as specified above, were generated. A sample space of 657096 such permutations were generated. These were analysed for effectiveness using Meyer's ciphertext dependence on plaintext, with results as summarised in column 4 of Table 5.3.

The P-box used in the current DES has a propagation profile that falls fairly close to the median of the 178 permutations generated by the empirical rules. It thus appears to be a fairly representative example of them. In turn, these also fall within the range found for the permutations derived from latin-squares, which obviously encompass the criteria needed to design them. This appears to confirm the importance of this concept in the design of the permutation.

Table 5.3 - Dependency of Ciphertext on Plaintext Bits for various DES permutations P by Round						
Round	Current P	Empirical P	Sample Latin-Square P	Best Latin-Square P	Regular Form P	Worst P
1	6.25	6.25	6.25	6.25	6.25	6.25
2	32.06	32.03-32.10	30.49-32.20	32.02-32.23	30.47-32.23	21.09
3	73.49	73.44-73.58	70.36-73.78	73.44-73.83	70.31-73.83	43.75
4	96.90	96.87-96.95	95.34-97.05	96.88-97.07	95.31-97.07	68.75
5	100.00	100.00	100.00	100.00	100.00	89.06
6	100.00	100.00	100.00	100.00	100.00	98.44
7	100.00	100.00	100.00	100.00	100.00	100.00
8	100.00	100.00	100.00	100.00	100.00	100.00

4. Analysis of the Best Latin-Square Permutations P

From the sample space of 657096 permutations, those permutations resulting in the highest percentage values in each round of the dependency analysis were extracted. A total of 20 such permutations were found. When examined, half of the columns (namely the ab/ef columns) of these permutations were found to be identical, with values as given in Table 5.4. These columns provide inputs to two S-boxes due to expansion function E. Further, they were found to be nearly identical to the values generated by applying a *difference function* of

$$[-2 +1\ c\ \ d\ -1 +2] \qquad c, d \text{ arbitary} \tag{Eq 5.2}$$

to the input S-box number, as shown in full in Table 5.4.

Table 5.4 - Fixed Columns in Sample Permutations P
Replicated pattern in the best sample permutations 3 . . 7 8 . . 1 4 . . 5 2 . . 3 6 . . 4 7 . . 8 5 . . 6 1 . . 2
Pattern generated by a [-2 +1 c d -1 +2] difference function 2 . . 8 **3** . . 1 4 . . 2 5 . . 3 6 . . 4 7 . . 5 8 . . 6 1 . . **7**

In this context a *difference function* is defined as a function that may be used to build a regular permutation which maps each of the 6 inputs (after expansion function E has been applied) of a given S-box i from an output of an S-box. eg. Given Eq 5.2 the 6 inputs of S-box i would be mapped from an output of S-boxes $i-2$, $i+1$, $i+c$, $i+d$, $i-1$, and $i+2$ respectively (with arithmetic cyclic over the range $[1..8]$).

From this, it is suggested that permutations with a fixed ab/ef column structure whose values are those generated by difference function (Eq 5.2) may perform well. Permutations of this form were generated, a total of 264 being found. The results of the dependency analysis on these permutations is given in column 5 of Table 5.3. Those providing the best results in this analysis were extracted, 100 such permutations being found. These were found to be grouped into pairs, with only columns c and d interchanged. These pairs in turn were grouped by *exclusion set*. This is defined as the set of S-boxes from which a bit is not permuted to the current S-box, apart from the current S-box. This occurs since the 6 inputs are each permuted from a different S-box, which is never the current one. Hence one other S-box must be missed. The only difference between these pairs of permutations is the swapping of some of the *cd* bits, a transformation which makes no difference to the dependency results. The author does not know whether this swapping has any other significance, but the tests done to date suggest not. A total of 18 exclusion sets were found among the best permutations, and a sample from each is given in Table 5.5.

The author noted that among these permutations are two which may be generated with difference functions

$$[-2 +1 +4 -3 -1 +2], \qquad \text{and} \tag{Eq 5.3}$$

$$[-2 +1 +3 +4 -1 +2] \tag{Eq 5.4}$$

being the first and last entries in Table 5.5 respectively.

Table 5.5 - Best Latin-Square Permutations P

Sample Permutation	Exclusion Set	Number
2 5 6 8 3 6 7 1 4 7 8 2 5 8 1 3 6 1 2 4 7 2 3 5 8 3 4 6 1 4 5 7	4 5 6 7 8 1 2 3	2
2 5 6 8 3 7 5 1 4 6 8 2 5 1 7 3 6 8 2 4 7 2 1 5 8 3 4 6 1 4 3 7	4 6 7 8 1 3 2 5	4
2 5 6 8 3 7 5 1 4 6 7 2 5 1 8 3 6 8 2 4 7 2 1 5 8 3 4 6 1 4 3 7	4 6 8 7 1 3 2 5	8
2 5 6 8 3 7 5 1 4 6 7 2 5 1 8 3 6 8 1 4 7 2 3 5 8 4 2 6 1 3 4 7	4 6 8 7 2 1 3 5	8
2 5 6 8 3 6 5 1 4 7 8 2 5 1 7 3 6 8 2 4 7 3 1 5 8 2 4 6 1 4 3 7	4 7 6 8 1 2 3 5	4
2 5 6 8 3 6 5 1 4 7 8 2 5 1 7 3 6 8 1 4 7 2 3 5 8 4 2 6 1 3 4 7	4 7 6 8 2 1 3 5	8
2 4 6 8 3 7 5 1 4 6 8 2 5 8 7 3 6 1 2 4 7 2 1 5 8 3 4 6 1 5 3 7	5 6 7 1 8 3 2 4	4
2 4 6 8 3 7 5 1 4 6 8 2 5 1 7 3 6 8 1 4 7 2 3 5 8 3 2 6 1 5 4 7	5 6 7 8 2 1 4 3	4
2 4 6 8 3 7 5 1 4 6 7 2 5 1 8 3 6 8 1 4 7 3 2 5 8 2 4 6 1 5 3 7	5 6 8 7 2 1 3 4	4
2 4 6 8 3 6 5 1 4 7 8 2 5 8 7 3 6 1 2 4 7 3 1 5 8 2 4 6 1 5 3 7	5 7 6 1 8 2 3 4	4
2 4 6 8 3 6 5 1 4 7 8 2 5 8 7 3 6 1 2 4 7 3 1 5 8 2 3 6 1 5 4 7	5 7 6 1 8 2 4 3	8
2 4 6 8 3 6 5 1 4 7 8 2 5 1 7 3 6 8 1 4 7 2 3 5 8 3 2 6 1 5 4 7	5 7 6 8 2 1 4 3	8
2 4 5 8 3 7 6 1 4 6 8 2 5 8 7 3 6 1 2 4 7 3 1 5 8 2 3 6 1 5 4 7	6 5 7 1 8 2 4 3	8
2 4 5 8 3 7 6 1 4 6 8 2 5 8 7 3 6 1 2 4 7 2 1 5 8 3 4 6 1 5 3 7	6 5 7 1 8 3 2 4	8
2 4 5 8 3 7 6 1 4 6 8 2 5 1 7 3 6 8 2 4 7 3 1 5 8 2 3 6 1 5 4 7	6 5 7 8 1 2 4 3	4
2 4 5 8 3 7 6 1 4 6 7 2 5 1 8 3 6 8 2 4 7 3 1 5 8 2 4 6 1 5 3 7	6 5 8 7 1 2 3 4	4
2 4 5 8 3 7 6 1 4 6 7 2 5 1 8 3 6 8 2 4 7 2 1 5 8 3 4 6 1 5 3 7	6 5 8 7 1 3 2 4	8
2 4 5 8 3 5 6 1 4 6 7 2 5 7 8 3 6 8 1 4 7 1 2 5 8 2 3 6 1 3 4 7	6 7 8 1 2 3 4 5	2

5. A Regular Form for Permutation P

As further confirmation of this result, all permutations P which may be generated by a regular difference function on the target S-box were constructed, 120 being found. There cipher-plaintext bit dependency propagation is given in Table 5.3. The 8 best of these are indeed the regular permutations formed using the difference functions (Eq 5.3) and (Eq 5.4), and their equivalents formed by swapping *ab/ef* or *cd* columns. They are listed in Table 5.6. These permutations would thus seem to be excellent candidates for any reimplementation of the DES using 64-bit blocks. However, some problems are noted when the dependency results are examined further, as indicated below.

Table 5.6 - Best Regular Form Permutations P

Permutation	Difference Fn
7 4 5 3 8 5 6 4 1 6 7 5 2 7 8 6 3 8 1 7 4 1 2 8 5 2 3 1 6 3 4 2	-2 +3 +4 +2
7 5 4 3 8 6 5 4 1 7 6 5 2 8 7 6 3 1 8 7 4 2 1 8 5 3 2 1 6 4 3 2	-2 +4 +3 +2
7 5 6 3 8 6 7 4 1 7 8 5 2 8 1 6 3 1 2 7 4 2 3 8 5 3 4 1 6 4 5 2	-2 +4 -3 +2
7 6 5 3 8 7 6 4 1 8 7 5 2 1 8 6 3 2 1 7 4 3 2 8 5 4 3 1 6 5 4 2	-2 -3 +4 +2
2 4 5 8 3 5 6 1 4 6 7 2 5 7 8 3 6 8 1 4 7 1 2 5 8 2 3 6 1 3 4 7	+1 +3 +4 -1
2 5 4 8 3 6 5 1 4 7 6 2 5 8 7 3 6 1 8 4 7 2 1 5 8 3 2 6 1 4 3 7	+1 +4 +3 -1
2 5 6 8 3 6 7 1 4 7 8 2 5 8 1 3 6 1 2 4 7 2 3 5 8 3 4 6 1 4 5 7	+1 +4 -3 -1
2 6 5 8 3 7 6 1 4 8 7 2 5 1 8 3 6 2 1 4 7 3 2 5 8 4 3 6 1 5 4 7	+1 -3 +4 -1

6. A Further Criterion for Best Permutations

In the analysis so far, the measure of effectiveness being used is the rate of growth of dependence of output bits on input bits, using Meyer's analysis. However this dependence may be further characterised, by a dependence through message inputs (bits *bcde*) only, autoclave inputs (bits *af*) only, or through both. If, as a measure of effectiveness, the rate of growth of dependence on input bits through both autoclave and message bits is examined, it may be shown that a strictly regular permutation is not necessarily the most effective (see Table 5.7). Rather, the best latin-square permutations, that is with only two of four columns fixed, performs better by this criterion. Also, the original DES performs better still, but at a price of lower growth in overall dependence. There thus, seems to be some tradeoff between regularity, and best performance.

In the extended scheme being developed, the choice of some of the best latin square permutations, would thus seem to be indicated. In subsequent work on permutation PC2 and the key schedule, a subset of these, which also perform best using a measure of dependence of output bits on key bits have been found. These are listed in Table 5.8, and were used to produce the results in Table 5.7.

Table 5.7 - Dependency of Ciphertext bits on Plaintext bits via Message, Autoclave and Both Inputs

Round	Current P				Best Latin Square P				Best Regular P			
	Msg	Auto	Both	Total	Msg	Auto	Both	Total	Msg	Auto	Both	Total
1	192	64	0	6.25	192	64	0	6.25	192	64	0	6.25
2	641	499	173	32.06	656	512	152	32.23	616	512	192	32.23
3	834	806	1370	73.49	864	864	1296	73.83	784	960	1280	73.83
4	353	371	3245	96.90	368	448	3160	97.07	328	640	3008	97.07
5	0	0	4096	100.00	0	32	4064	100.00	0	128	3968	100.00
6	0	0	4096	100.00	0	0	4096	100.00	0	0	4096	100.00

Table 5.8 - Best Latin Square Permutations P by both Ciphertext-Plaintext and Ciphertext-Key Dependence
2 5 6 8 3 6 5 1 4 7 8 2 5 1 7 3 6 8 1 4 7 2 3 5 8 4 2 6 1 3 4 7
2 5 6 8 3 6 5 1 4 7 8 2 5 1 7 3 6 8 1 4 7 3 2 5 8 2 4 6 1 4 3 7
2 5 6 8 3 6 5 1 4 8 7 2 5 7 1 3 6 1 8 4 7 2 3 5 8 4 2 6 1 3 4 7
2 5 6 8 3 6 5 1 4 8 7 2 5 7 1 3 6 1 8 4 7 3 2 5 8 2 4 6 1 4 3 7
2 6 5 8 3 5 6 1 4 7 8 2 5 1 7 3 6 8 1 4 7 2 3 5 8 4 2 6 1 3 4 7
2 6 5 8 3 5 6 1 4 7 8 2 5 1 7 3 6 8 1 4 7 3 2 5 8 2 4 6 1 4 3 7
2 6 5 8 3 5 6 1 4 8 7 2 5 7 1 3 6 1 8 4 7 2 3 5 8 4 2 6 1 3 4 7
2 6 5 8 3 5 6 1 4 8 7 2 5 7 1 3 6 1 8 4 7 3 2 5 8 2 4 6 1 4 3 7

7. Design Rules for Permutation P

In the light of these experiments, the current set of rules for designing permutation P in a DES like scheme are:

- P1 Each of the input bits $a\ b\ c\ d\ e\ f$ for S-box S(i) must come from the outputs of different S-boxes in the previous round.
- P2 None of the input bits $a\ b\ c\ d\ e\ f$ for S-box S(i) may come from the output of that same S-box S(i) in the previous round.
- P3 For each input bit a, b, c, d, e, or f over all of the S-boxes, the S-boxes permuted to that bit must differ (ie each column forms a permutation of integers 1 to 8).
- P4 An output from S(i-2) goes to one of the ab input bits of S(i), and from S(i+1) to the other. Hence via E an output from S(i-1) goes to one of the ef input bits, and from S(i+2) to other.
- P5 Outputs from two of S(i-3), S(i+3), and S(i+4) go to cd input bits of S(i).
- P6 For each S-box output, two bits go to ab or ef input bits, the other two go to cd input bits as noted in [Dav82].

Rules P1, P2 and P3 constrain the permutation to be a latin-rectangle, whilst the remaining rules further constrain the permutation to select outputs which maximise the growth of bit dependence. Having used these rules to construct a permutation, the dependency analysis may then be used to verify its effectiveness.

8. Regular Permutations P in Feistel Ciphers

Permutations with strictly regular form were generated for a Feistel cipher using 128-bit blocks. The 4 best of these were produced using a difference function

$$[-2\ {+1}\ {-7}\ {+7}\ {-1}\ {+2}] \qquad \text{(Eq 5.5)}$$

and its equivalents by column swapping, as shown in Table 5.9. The cipher-plaintext dependency results are given in Table 5.10 (along with those for a worst case, and sample permutations given in the previous chapter. These best permutations alone of all those generated, resulted in complete dependency of ciphertext bits on all plaintext bits in 5 rounds, the same as for the current size scheme. Work is continuing to derive the best permutations for use in the extended scheme when dependence on both message and autoclave inputs is considered.

Table 5.9 - Best Regular Form Extended Permutations P
2 10 8 16 3 11 9 1 4 12 10 2 5 13 11 3 6 14 12 4 7 15 13 5 8 16 14 6 9 1 15 7 10 2 16 8 11 3 1 9 12 4 2 10 13 5 3 11 14 6 4 12 15 7 5 13 16 8 6 14 1 9 7 15
2 8 10 16 3 9 11 1 4 10 12 2 5 11 13 3 6 12 14 4 7 13 15 5 8 14 16 6 9 15 1 7 10 16 2 8 11 1 3 9 12 2 4 10 13 3 5 11 14 4 6 12 15 5 7 13 16 6 8 14 1 7 9 15
15 10 8 3 16 11 9 4 1 12 10 5 2 13 11 6 3 14 12 7 4 15 13 8 5 16 14 9 6 1 15 10 7 2 16 11 8 3 1 12 9 4 2 13 10 5 3 14 11 6 4 15 12 7 5 16 13 8 6 1 14 9 7 2
15 8 10 3 16 9 11 4 1 10 12 5 2 11 13 6 3 12 14 7 4 13 15 8 5 14 16 9 6 15 1 10 7 16 2 11 8 1 3 12 9 2 4 13 10 3 5 14 11 4 6 15 12 5 7 16 13 6 8 1 14 7 9 2

Table 5.10 - Dependency of Ciphertext bits on Plaintext bits in Extended (128-bit) DES

Round	Best Regular Form P	Sample XDES P	Worst XDES P
1	3.13	3.125	3.125
2	17.58	17.68	10.55
3	52.34	50.98	21.88
4	87.50	84.47	34.38
5	100.00	98.44	46.88
6	100.00	100.00	59.38
7	100.00	100.00	71.88
8	100.00	100.00	84.38
9	100.00	100.00	94.53
10	100.00	100.00	99.21
11	100.00	100.00	100.00

9. Conclusion

This chapter reviews some possible design criteria for the permutation P in new Feistel ciphers. These permutations provide the diffusion component in a substitution-permutation network. Some empirical rules which seem to account for the derivation of the permutation used in the DES are first presented. Then it is noted that these permutations may be regarded as latin-squares which link the outputs of S-boxes to their inputs at the next stage. When a sample of these latin-square permutations were generated, they were found to span those generated using the empirical rules. The best of these permutations suggested that permutations having a fixed column structure as generated by difference function (Eq 5.2) would perform well. Permutations with this structure were generated, and the best of these analysed. They were found to include the permutations generated using difference functions (Eq 5.3) and (Eq 5.4), which were strictly regular. Whilst these performed well in growth of overall bit dependence, when dependence on both message and autoclave bits were examined, these permutations were found to be slightly less than optimal. From these a set of design rules have been developed for permutation P. An extension of these results to a Feistel ciphers was then analysed, and 4 permutations formed by difference function (Eq 5.5) were found to result in complete ciphertext dependency on plaintext bits after 5 rounds, a result that is the same as with the current size scheme.

CHAPTER 6

Key Scheduling in Feistel Ciphers

1. Introduction

This chapter will concentrate on the design of the key schedule, which involves a key rotation component, and the permutation $PC2$. Together these provide for a diffusion of dependency of ciphertext bits in key bits. As a measure of effectiveness, the Meyer and Matyas analysis of output bit dependence on key bits will be used [MeM82]. Starting with some empirical rules for the key schedule discussed earlier, a discussion of some alternatives to the current schedule will be presented, followed by the results obtained from testing a number of alternate schedules. A presentation of the implications from these in the design of any Feistel Ciphers will conclude the chapter.

2. The Key Schedule in DES

The central component of the DES cryptosystem is the function g, which is a composition of expansion function E, eight substitution boxes (S-boxes) S, and a permutation P. Function g has as inputs the plaintext $[L(i-1),R(i-1)]$ from the previous round, and a selection of key bits $K(i)$ (see Fig 5.1.). This may be written as:

$$g : R(i) = L(i-1) \oplus P(S(E(R(i-1)) \oplus K(i))),\ L(i)=R(i-1) \qquad \text{(Eq 6.1)}$$

The *key schedule* in a DES algorithm is responsible for forming the sixteen 48-bit sub-keys $K(i)$ used in the rounds of the encryption procedure. This function is important since if the same key is used on successive rounds, it can weaken the resulting algorithm (see [GrT78], [MeM82], [MoS87], [MoS86], and [ASA85]). In detail, the 64-bit key is permuted by $PC1$. This permutation performs two functions: first it strips the eight parity bits out, and then distributes the remaining 56 bits over two 28-bit halves $C(0)$ and $D(0)$. The cryptographic significance of this permutation is questionable [DDF83]. Subsequently for each round, each 28-bit register is rotated left either one or two places according to the following schedule (subsequently denoted KS):

Table 6.1 - Key Schedule for DES

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Shift	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1
Total	1	2	4	6	8	10	12	14	15	17	19	21	23	25	27	28

After the shift, the resultant 28-bit vectors are permuted by $PC2$ (which in fact consists of two 28-bit permutations, each of which selects 24 bits) to form the sub-key for that round. This permutation may be written in terms of which S-box each bit is directed to, as shown in Table 6.2 (nb: an * indicates an autoclave S-box input rather than a message input; an

X specifies exclusion of that bit).

Table 6.2 - Current DES Permutation PC2
C: 1 4* 2* 3 1* 2 4 3* X 2* 1 3 4 1* 2 4* 1 X 3 4 2 X 3* 1 X 3 4 2
D: 8 6* 5 8* 6 7 X 8 5 X 7 6 5* 8 X 7* 6 8* 5 6* 7 8 6 5 7* X 5* 7

The sub-keys $K(i)$ may be written as;

$$K(i) = PC2(KS(U, i)), \quad \text{where } U = PC1(K) \qquad \text{(Eq 6.2)}$$

and $KS(U,i)$ is the key rotation schedule for input block U at round i.

3. Empirical Key Schedule Design Criteria

Previously, some empirical design rules for the key schedule were presented. The rules presented for permutation $PC2$ are (if the bits are sorted into ascending order of their input bits):

- C1 bits permuted to the same S-box input are no closer than 3 bits apart
- C2 bits permuted to an S-box input must have a span from lowest to highest input bit number of at least 22 of the 28 bits in each key half (alternatively, the average spacing must be at least 3 2/3)
- C3 bits permuted to the selector bits a, f on a given S-box must not be adjacent in the sorted list of input bits
- C4 bits not selected by $PC2$ must be at least 3 places apart

The design of the key schedule KS is obviously related to the design of $PC2$ by rules 1 and 4 given above. In chapter 4, it was noted that the key schedule KS ensures that:

- K1 each bit is used as input to each S-box
- K2 no bit is used as input to the same S-box on successive rounds
- K3 the total number of bits rotated is 56 (which implies that $K(0) = K(16)$, enabling the decryption operation to use right shifts in reverse order).

4. Ciphertext Dependence on Key Bits

This analysis is dependent on the choices of permutations P and $PC2$ as well as the KS the key schedule[1]. To quantify this dependency, a $64*56$ array F_r is formed, in which element $F_r[i,j]$ specifies a dependency of output bit $X(j)$ on key bit $U(i)$. The vector U is that formed after PC1 is applied, ie $U = PC1(K)$. The number of marked elements in G_r will be examined to provide a profile of the degree of dependence achieved by round r. Details of the derivation of this matrix, and the means by which entries are propagated, may be found in [MeM82]. This analysis technique will be used as a measure of effectiveness for possible key schedules. In particular, two criteria are used:

[1] but not on permutations IP, FP and $PC1$, which only serve to renumber the plaintext, ciphertext, and key bits respectively. The analysis done in this chapter ignores these permutations for this reason.

- rate of growth of output bit dependence on key bits by any S-box inputs
- rate of growth of output bit dependence on key bits by BOTH message and autoclave S-box inputs

5. Alternatives for the Key Schedule

The purpose of the above rules in designing a key schedule may be summarised as follows:

> to present each key bit to a message input, and to an autoclave input, of each S-box as quickly as possible.

This is achieved by a combination of the key rotation schedule KS, the key permutation $PC2$, and the function $g = S.P.E$. Trials have been performed in which each of these is varied in turn, to analyse the effect of each.

In addition to that underlying design purpose, there is a pragmatic decision on the size of the key registers. In the current scheme, the key is divided into two halves. An alternate form could be to have a single large key schedule register. We also wish to extend the size of the key, in order to ensure it is large enough to withstand any foreseeable exhaustive search style attack. One way of providing a measure of this, whilst still maintaining compatibility with existing protocols, would be to remove the notion of parity bits in the key, and use all 64-bits. This obviously implies altering $PC2$ so that all of the key bits are presented as inputs to some round of the cipher. Combining these two ideas we have the following possibilities for $PC2$:

28 -> 24 bit
56 -> 48 bit
32 -> 24 bit
64 -> 48 bit

Initially, a key schedule with the same form as the current DES was examined, in order that comparisons with the effectiveness of the current DES scheme could be made. Having obtained some guidelines from these trials, key schedules involving some of the alternatives were then tried.

6. Some Trials on New Key Schedules

In chapter 4, some empirical design criteria for permutation $PC2$ and the Key Rotation Schedule were presented. The author has subsequently used these rules to generate a set of permutations $PC2$. Since all possible 28->24 bit permutations could not be tried, permutations with the form shown in Table 6.3 were tried (that is all arrangements of the 4 excluded bits in the fixed pattern of S-boxes, subject to the rules set, were found). This form was chosen in order to distribute key bits to each of the 4 S-boxes being fed by each half of the key schedule as quickly as possible, by the simple expediant of either sending each bit to the next S-box in turn or skipping it. This table lists which S-box each of the 28 bits in the C and D key halves is permuted to, if not skipped. A total of 7315 permutations were found.

Table 6.3 - Form of generated Permutations PC2
C: 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 + X X X X
D: 5 6 7 8 5 6 7 8 5 6 7 8 5 6 7 8 5 6 7 8 5 6 7 8 + X X X X

Ciphertext-Key Dependences (CKdep) tests on these permutations produced results shown in Table 6.4, column 4, with comparisons to the current and worst case $PC2$ supplied for comparison in columns 2 and 3.

Table 6.4 - Dependency of Ciphertext bits on Key bits
Using Current DES Permutation P and Key Schedule

Round	Std $PC2$	Worst $PC2$	Generated $PC2$	Regular X 2	Regular X 1, X 3, X 4
1	5.36	5.36	5.36	5.36	5.36
2	39.17	42.19	38.50-39.06	39.06	38.62
3	82.25	81.47	80.25-82.37	82.37	81.47
4	98.44	91.29	96.65-98.66	98.66	98.21
5	100.00	96.21	99.55-100.00	100.00	100.00
6	100.00	99.55	100.00	100.00	100.00
7	100.00	100.00	100.00	100.00	100.00
8	100.00	100.00	100.00	100.00	100.00

Some of these permutations performed better than the $PC2$ used in the current DES. The best of these were selected, 15 being found. These 15 permutations were all found to have a special form, namely that the excluded bits always fell between bits permuted to S-box 1 and S-box 2 (or 5 and 6 in the D-side). These will be termed the *Regular X 2* permutations. There are exactly 15 of these, since $15 = C^4_6$ (ie the number of ways of placing the 4 excluded spots into the 6 occurences of the 1 2 3 4 regular patterns at the specified location (1 X 2 for example)). In order to investigate these permutations with a regular placing of the excluded bits, all 60 such permutations were generated. A CKdep analysis of these permutations resulted in only two results, one for permutations with the excluded bit before a bit permuted to S-box 2 (Regular X 2), and one for the others (Regular X 1, X 3, X 4). These results are also shown in Table 6.4. The reason for this variation is unknown to the author at present, as is its significance, if any, from a cryptographic point of view.

So far, we have used the first of the two criteria presented earlier, namely the growth of overall bit dependence of output bits on key bits. If we now consider the alternate measure, namely growth in dependence of output bits on key bits by both message and autoclave S-box inputs, then the results become less clear. As shown in Table 6.5, whilst growth of overall dependence is greater with the regular $PC2$'s, growth of both is worse.

Table 6.5 - Dependency of Ciphertext bits on Key bits Using Current DES Permutation P and Key Schedule by Both Message and Autoclave S-box Inputs								
Round	1	2	3	4	5	6	7	8
CKdep	Both,Either	Both,Either	Both,Either	Both,Either	Both,Either	Both,Either	Both,Either	Both,Either
PC2.std	0.0,5.36	2.01,39.17	36.50,82.25	81.03,98.44	95.87,100.0	99.33,100.0	100.0,100.0	100.0,100.0
PC2.worst	0.0,5.36	0.0,42.19	33.71,81.47	73.88,91.29	84.38,96.21	92.86,99.55	98.66,100.0	100.0,100.0
PC2 X 1	0.0,5.36	0.22,38.62	29.91,81.47	65.07,98.21	73.66,100.0	80.13,100.0	87.28,100.0	93.97,100.0
PC2 X 2	0.0,5.36	0.45,39.06	30.36,82.37	67.08,98.66	77.01,100.0	83.26,100.0	90.18,100.0	95.87,100.0
PC2 X 3	0.0,5.36	0.45,39.06	30.13,81.92	66.74,98.21	76.79,100.0	83.04,100.0	89.96,100.0	95.76,100.0
PC2 X 4	0.0,5.36	0.45,39.06	30.13,81.92	66.52,98.21	75.67,100.0	80.36,100.0	86.38,100.0	92.41,100.0

A closer look at the structure of the regular permutations shows that the autoclave input bits are clustered, due to the method used to assign them to S-box inputs. By altering the order of inputs within each S-box, a more regular arrangement of autoclave inputs was obtained. When these were tested, the growth of dependence on both was much greater, thus emphasizing the importance of this criterion on the design of $PC2$.

To obtain an indication of the relative influences of each of the components in the key schedule, a series of 40 trials were run, in which each of the following three components were varied with the specified alternatives:

P based on the results in [BrS90a], two permutations P were used:

- the current DES P and
- a strictly regular permutation generated by a difference function on the S-box number of [+1 -2 +3 +4 +2 -1]. Because of its very regular structure, the propagation of dependencies may be more easily calculated.

$PC2$ from the above work, the 4 best performing regular $PC2$ were extracted. Then these were processed to provide three levels of clustering of the autoclave inputs.

KS the key variant in the key rotation schedule appears to be the distribution of shifts of 1 verses 2 places. A set of 5 key schedules with various numbers of shifts of 1 initially were derived as shown in Table 6.6.

Table 6.6 - Trial Key Schedules																
Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
KS	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1
KS	1	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1
KS	1	1	2	2	2	2	2	2	2	2	2	2	2	2	1	1
KS	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	1
KS	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2

When this test was run, the following conclusions were made:

- the current permutation P performed slightly better (though only in minor percentage values, not in the number of rounds needed to achieve full dependence) than the strictly regular permutation. This was possibly because its less than regular structure assisted the distribution of dependencies between the autoclave and message inputs.

- the permutations $PC2$ with the best spread of autoclave inputs, performed best as expected.
- a key schedule with as many shifts of 1 initially performed best. This again would appear to be a function of the best method for spreading bits to as many S-box inputs as soon as possible.

7. Design Criteria for New Key Schedules

From the above results, the design principles for designing key schedules can now be summarised as follows:

The key schedule ensures that:

- KS1
  each key bit is used as input to each S-box in turn
- KS2
  no bit is used as autoclave inputs on successive rounds
- KS3
  no bit is excluded on successive rounds
- KS4
  the final key register value is identical to the original key register value (to enable easy reversal of the key schedule for decryption)

8. An Alternative Key Schedule Design

In the design of the DES, small key rotations were used, which required the use of permutation $PC2$ to provide a fan-out of key-bits across the S-box inputs, in order to satisfy the above principles (especially KS1). An alternative design can be envisaged in which a large key rotation interval is used, along with a null $PC2$ (ie: so called worst case or straight through $PC2$), or a local $PC2$ which only permutes bits within each block of 6 S-box inputs. Two such $PC2$ permutations tested are shown in Table 6.7. The $*$ in the table indicates a bit permuted to an autoclave input of an S-box, otherwise the bit is permuted to a message input. The autoclave inputs are assumed to be bits 1 and 6 (as in the DES) of each S-box. The 4 excluded bits are left at the end. This causes no problems, since the key rotation used of 7 places ensures that principle KS3 is satisfied.

Table 6.7 - Null and Local Permutations PC2 for Alternative Key Schedule
C: 1* 1 1 1 1 1* 2* 2 2 2 2 2* 3* 3 3 3 3 3* 4* 4 4 4 4 4* X X X X
D: 5* 5 5 5 5 5* 6* 6 6 6 6 6* 7* 7 7 7 7 7* 8* 8 8 8 8 8* X X X X
C: 1* 1 1 1 1 1* 2 2 2 2 2* 2* 2 3 3* 3* 3 3 3 4* 4 4 4 4 4* X X X X
D: 5* 5 5 5 5 5* 6 6 6 6* 6* 6 7 7* 7* 7 7 7 8* 8 8 8 8 8* X X X X

For these tests, a constant key rotation of 7 bits was used, both because it is larger than the number of inputs to an S-box, and because after sixteen rounds, the key register contents are the same as the original value (since $7*16=112=4*28=2*56$), for both split key registers or a single large key register. This key rotation schedule is shown in Table 6.8.

Table 6.8 - Alternative Constant Key Rotation Schedule

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
KS	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7

The results obtained for these $PC2$ permutations and this key schedule, using both a split key rotation register, and a single key register, are shown in Table 6.9.

Table 6.9 - Dependency of Ciphertext bits on Key bits
Using Current DES Permutation P and the Alternative Key Schedule
by Both Message and Autoclave S-box Inputs

Round	1	2	3	4	5	6	7	8
PC2	Both,Either	Both,Either	Both,Either	Both,Either	Both,Either	Both,Either	Both,Either	Both,Either
Split Key Register Used								
null	0.0,5.36	1.56,39.06	34.82,82.03	76.56,98.33	91.52,100.0	98.21,100.0	100.0,100.0	100.0,100.0
local	0.0,5.36	2.57,39.06	38.17,82.03	83.82,98.33	98.21,100.0	100.0,100.0	100.0,100.0	100.0,100.0
Single Key Register Used								
null	0.0,5.36	1.79,38.73	35.04,81.70	76.56,98.33	91.52,100.0	98.21,100.0	100.0,100.0	100.0,100.0
local	0.0,5.36	2.79,38.73	38.39,81.70	83.82,98.33	98.21,100.0	100.0,100.0	100.0,100.0	100.0,100.0

These results are very similar in performance to the key schedule used in the current DES (see Table 6.5). The null $PC2$ performs slightly worse in growth of dependencies via both autoclave and message inputs than the local $PC2$. Depending on the efficiency required, a tradeoff between best performance and ease of implementation can be made between these. There is very little difference in performance between the split and single key rotation registers, thus either could be chosen, depending on other constraints. This may be explained by realising that dependencies grow both via the inputs from the key schedule, as well as via the message paths in the cipher. The single and split key schedules are identical for the first 4 rounds, and only differ from then on. However after 4 rounds nearly all output bits already have a dependency on most input bits, and hence the differing key schedules have little effect after that point in this analysis.

9. Conclusion

The key schedule in the current DES has been analysed, and some empirical principles which could have been used in its design derived. These were used to test a number of alternative key schedules, which led to the development of a new set of generalised principles to be used in the design of a new algorithm. An alternative key schedule which either eliminates permutation $PC2$, or uses a local $PC2$, was tried and found to be as effective as that used in the current DES. These generalised rules will be used in the design of the LOKI family of ciphers, descibed in the next chapter.

CHAPTER 7

LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications

1. Introduction

This chapter provides an overview of the LOKI[1] encryption primitive which may be used to encrypt and decrypt a 64-bit block of data using a 64-bit key. This is the main achievement of this thesis. Although some of the earlier work was directed toward the design of a cipher with 128-bit data and keys, in the short term there was considerable interest in the design of ciphers which were compatible with the existing DES ciper. Thus a cipher with a full 64-bit key was the first to be designed using the principles identified. The author intends to extend this work to a cipher with 128-bit data and keys in the near future. The LOKI cipher is the first of an extensible family of encryption algorithms. These algorithms all share the same overall structure, but allow the substitution and permutation functions, central to the security of the schemes, to be tailored to permit the design of private ciphers.

The design of the LOKI primitive is based on the detailed analysis of the DES described in the previous chapters, and in [Bro89], [BrS90a], [BrS90b], and on the work of Pieprzyk on the design of good substitution functions with high non-linearity [PiF89], [Pie90], [Pie89], [PiS89]. Its overall structure has a broad resemblance to DES (see Fig 7.1), however the detailed structure has been designed to remove operations which impede analysis or hinder efficient implementation, but which do not add to the cryptographic security of the algorithm.

The LOKI primitive may be used in any mode of operation currently defined for DES, with which it is interface compatible [AAA83]. Also described are two modes of operation of the LOKI primitive which compute a 64-bit, and 128-bit, Message Authentication Code (or hash value) respectively, from an arbitrary length of message input. The modes of use are modifications of those described in [DaP89], [Win83], and [QuG90]. These modes of operation may be used to provide authentication of a communications session, or of data files held on a computer system.

The LOKI encryption primitive, and the above modes of use have been submitted to the European RIPE project [VCF90], Standards Australia, and the Defence Signals Directorate, for evaluation.

---

[1] LOKI - God of mischief and trickery in Scandinavian mythology. "He is handsome and well made, but of a very fickle mood and most evil disposition. He is of the giant race, but forced himself into the company of the gods, and seems to take pleasure in bringing them into difficulties, and in extracting them out of the danger by his cunning, wit and skill" [Bulfinch's Mythology, Avenel Books, NY 1978].

This chapter will first present the LOKI cipher as currently specified, and will then examine the detailed design choices in its definition.

1.1. Overview

LOKI is a family of ciphers designed to encrypt and decrypt blocks of data consisting of 64 bits, under control of a 64-bit key. This chapter defines a common variant of the algorithm for use when compatibility between implementations is required. The same structure, but with alternate substitution functions may be used to build private variants of this algorithm. The same key is used for both encryption and decryption, but with the schedule of addressing the key bits altered so that the decryption process is the reverse of the encryption process. A block to be encrypted is added modulo 2 to the key, is then processed in 16 rounds of a key-dependent computation, and finally is added modulo 2 to the key again. The key-dependent computation can be defined in terms of a confusion-diffusion function f, and a key schedule KS. Descriptions of the encryption operation, the decryption operation, and the definition of the function f, are provided in the following sections. The representation of the keys, key values to be avoided and guidelines for the construction of alternate private ciphers, and full results for the tests conducted to date on LOKI, are subsequently described.

Fig 7.1 - LOKI Encryption Computation

1.2. Encryption

The encryption computation is illustrated in Fig 7.1. The 64 bits of the input block to be encrypted are added modulo 2 to the key, processed in 16 rounds of a complex key-dependent computation, and finally added modulo 2 to the key again.

In detail, the 64-bit input block X is partitioned into two 32-bit blocks XL and XR. Similarly, the 64-bit key is partitioned into two 32-bit blocks KL and KR. Corresponding halves are added together modulo 2, to form the initial left and right halves for the following 16 rounds, thus:

$$L_0 = XL \oplus KL_0 \qquad KL_0 = KL \qquad \text{(Eq 7.1)}$$

$$R_0 = XR \oplus KR_0 \qquad KR_0 = KR$$

The complex key-dependent computation consists (except for a final interchange of blocks) of 16 rounds (iterations) of a set of operations. Each iteration includes the calculation of the encryption function f. This is a concatenation of a modulo 2 addition and three functions E, S, and P. Function f takes as input the 32-bit right data half R_{i-1} and the 32-bit left key half KL_i produced by the key schedule KS (denoted K_i below), and which produces a 32-bit result which is added modulo 2 to the left data half L_{i-1}. The two data halves are then interchanged (except after the last round). Each round may thus be characterised as:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, KL_i) \qquad \text{(Eq 7.2)}$$

$$f(R_{i-1}, K_i) = P(S(E(R_{i-1} \oplus K_i)))$$

The component functions E, S, and P are described later.

The key schedule KS is responsible for deriving the sub-keys K_i, and is defined as follows: the 64-bit key K is partitioned into two 32-bit halves KL and KR. In each round i, the sub-key K_i is the current left half of the key KL_{i-1}. This half is then rotated 12 bits to the left, and the key halves are interchanged. This may be defined thus:

$$K_i = KL_{i-1}$$

$$KL_i = KR_{i-1} \qquad \text{(Eq 7.3)}$$

$$KR_i = ROL(KL_{i-1}, 12)$$

Finally after the 16 rounds, the other key halves are added modulo 2 to the data halves to form two output block halves YL and YR which are then concatenated together to form the output block Y. This is defined as follows (note the swap of data and key halves to undo the final interchange in (Eq 7.2) and (Eq 7.3)):

$$YL = R_{16} \oplus KR_{16}$$

$$YR = L_{16} \oplus KL_{16} \qquad \text{(Eq 7.4)}$$

$$Y = YL \mid YR$$

1.3. Decryption

The decryption computation is identical to that used for encryption, save that the partial keys used as input to the function f in each round are calculated in reverse order, and the initial and final additions of key to data modulo 2 use the opposite halves of the key (interchange KL_0 and KR_0 in (Eq 7.1) and KL_{16} and KR_{16} in (Eq 7.3)). The calculation of the partial keys for decryption consists of first exchanging key halves, then rotating the left half 12 bits to the right, and then using the left half as the partial key. This is defined as:

$$KR_i = KL_{i-1}$$

$$KL_i = ROR\,(KR_{i-1}, 12) \qquad \text{(Eq 7.5)}$$

$$K_i = KL_i$$

1.4. Function f

The encryption function f is a concatenation of a modulo 2 addition and three functions E, S, and P, which takes as input the 32-bit right data half R_{i-1} and the 32-bit left key half KL_i, and produces a 32-bit result which is added modulo 2 to the left data half L_{i-1}. This is shown in Fig 7.2, and is defined thus:

$$f\,(R_{i-1}, K_i) = P\,(S\,(E\,(R_{i-1} \oplus K_i))) \qquad \text{(Eq 7.6)}$$

The modulo 2 addition of the key and data halves ensures that the output of f will be a complex function of both of these values.

The expansion function E takes a 32-bit input and produces a 48-bit output block, composed of four 12-bit blocks which form the inputs to four S-boxes in function f. Function E selects consecutive blocks of twelve bits as inputs to S-boxes S(4), S(3), S(2), and S(1) respectively, as follows:

$[b_3\, b_2\, \cdots\, b_0\, b_{31}\, b_{30}\, \cdots\, b_{24}]$
$[b_{27}\, b_{26}\, \cdots\, b_{16}]$
$[b_{19}\, b_{18}\, \cdots\, b_8]$
$[b_{11}\, b_{10}\, \cdots\, b_0]$

This is shown in Table 7.1 in full.

Table 7.1 - LOKI Expansion Function E

source bit for output bits 47 down to 0 respectively

3	2	1	0	31	30	29	28	27	26	25	24
27	26	25	24	23	22	21	20	19	18	17	16
19	18	17	16	15	14	13	12	11	10	9	8
11	10	9	8	7	6	5	4	3	2	1	0

Fig 7.2 - LOKI Function *f* Detail

The substitution function S provides the confusion component in the LOKI cipher. It takes a 48-bit input and produces a 32-bit output. It is composed of four S-boxes, each of which takes a 12-bit input and produces an 8-bit output, which are concatenated together to form the 32-bit output of S. The 8-bit output from S(4) becomes the most significant byte (bits [31...24]), then the outputs from S(3) (bits[23...16]), S(2) (bits[15...8]), and S(1) (bits [7...0]). In this version, the four S-boxes are identical. The form of each S-box is shown in Fig 7.3. The 12-bit input is partitioned into two segments: a 4-bit row value r formed from bits $[b_{11}\, b_{10}\, b_1\, b_0]$, and an 8-bit column value c formed from bits $[b_9\, b_8 \cdots b_3\, b_2]$. The row value r is used to select one of 16 S-functions Sfn_r, which then take as input the column value c and produce an 8-bit output value. This is defined as:

$$\mathrm{Sfn}_r = (c \oplus r)^{e_r} \bmod \mathrm{gen}_r, \qquad \text{in } GF(2^8) \qquad \text{(Eq 7.7)}$$

where gen_r is an irreducible polynomial in $GF(2^8)$, and e_r is the exponent used in forming the rth S-box. The generators and exponents to be used in the 16 S-functions Sfn_r in the standard LOKI are specified in Table 7.2.

Table 7.2 - LOKI S-box Irreducible Polynomials[2] and Exponents		
Row	gen_r	e_r
0	375	31
1	379	31
2	391	31
3	395	31
4	397	31
5	415	31
6	419	31
7	425	31
8	433	31
9	445	31
10	451	31
11	463	31
12	471	31
13	477	31
14	487	31
15	499	31

The permutation function P provides diffusion of the outputs from the four S-boxes across the inputs of all S-boxes in the next round. It takes the 32-bit concatenated outputs from the S-boxes, and distributes them over all the inputs for the next round via a regular permutation which takes bits from the outputs of each S-box in turn, as defined in Table 7.3.

Table 7.3 - LOKI Permutation P

source bit for output bits 31 down to 0 respectively

0	8	16	24	1	9	17	25
2	10	18	26	3	11	19	27
4	12	20	28	5	13	21	29
6	14	22	30	7	15	23	31

[2] The column gen_r is a decimal representation of the binary string which defines the generator polynomial in $GF(2^8)$, the field used in the S-box function calculations.

Fig 7.3 - LOKI S-Box Detail

2. Design Philosophy

2.1. Overall LOKI Algorithm Design

The development of LOKI was motivated by the need to develop an alternative to DES for the provision of secrecy and authentication services in a range of applications, including electronic messaging, EDI, EFT, and others.

A major requirement in the short term was for LOKI to be interface compatible with DES, so that it could be used as a replacement in services based on it. The same design principles are being used in the development of a future 128-bit version of the algorithm.

LOKI is a Feistel type cipher [Fei73], a form of *Substitution-Permutation (S-P) network* originally described by Shannon [Sha49]. This family of ciphers include Lucifer [Sor84], and of course ISO DEA-1 or DES [Ins81].

As stated previously in Chapter 2, two key properties of such S-P networks are the *avalanche* property [Fei73]; and the *completeness* property [KaD79]. The *avalanche property* states that changing one input bit should result in approximately half of the output bits changing. The *completeness property* states that each output bit must be a complex function of every input bit. These properties appear to be important criteria for providing cryptographic strength in the resultant system, and the provision of them was the primary design criterion used in the design of LOKI.

More formally, these properties may be defined [WeT86] [Llo90] as follows.

A function f has a good *avalanche* effect if:

> for each bit i, $0 \le i < m$, if the 2^m plaintext vectors are divided into 2^{m-1} pairs X and X_i with each pair differing only in bit i; and if the 2^{m-1} exclusive-or sums, termed avalanche vectors
>
> $$V(X)_i = f(X) \oplus f(X_i) \qquad \text{(Eq 7.8)}$$
>
> are compared, then in each bit about half of these sums should be found to be 1.

A function f is *complete* if:

> for each bit j, $0 \le j < m$ in the ciphertext output vector, there is at least one pair of plaintext vectors X and X_i, which differ only in bit i, and for which $f(X)$ and $f(X_i)$ differ in bit j. That is, if all the avalanche vectors are summed:
>
> $$\sum_{X \in GF(2^m)} V(X)_i > 0 \text{ for all } i,\ i \le i \le m. \qquad \text{(Eq 7.9)}$$

The provision of the avalanche and completeness properties in the new algorithm was achieved by partitioning the design into two phases. First, the overall algorithm structure which incorporates the permutation P, and the key schedule, was developed to provide these properties on the assumption that the function S possessed the properties. Second, a suitable family of functions S were devised which satisfied that assumption.

2.2. Overall Design, Permutation P, E, and the Key Schedule

An overall algorithm structure incorporating the permutation P, E, and the key schedule, was developed to provide the avalanche and completeness properties on the assumption that the function S possessed those properties. The major tool used to measure this was the analysis of the growth of dependence of ciphertext bits in plaintext and key bits, a test developed by Meyer [MeM82].

This tool has been used to analyse possible design rules for these components in the DES. The results for permutation P and E are given in [Bro89], [BrS90a]. Some principles for the design of this permutation in a variant of the DES system are given in those chapters.

Expansion function E is provided to duplicate some of the input bits in each round so that they may be used to provide a self-keying (*autoclave*) property in the choice of the substitution functions within the S-boxes. Without this, the selection of a particular substitution function would depend only on the key-bits. The author and his collegues believe that this would significantly restrict the domain of the resultant function, and hinder the growth of dependencies. In the design of LOKI, an extension of the function E used in the DES was chosen.

In the design of LOKI, these principles may be generalised as follows. Principles used in the design of Permutation P:

- P1 distributes the outputs from a given S-box to the inputs of as many S-boxes as possible in the next round.
- P2 partitions the outputs from a given S-box between message (column) and autoclave (row) inputs to S-boxes in the next round.
- P3 when designed using a regular form (where each output bit from an S-box is permuted by a fixed offset), will achieve very high growth of dependencies.

Another observation from this analysis, was that the size of the S-box had a major impact on the growth of dependencies. The original choice of a 6-bit to 4-bit S-box for the DES was made because memory costs were significant. That is much less of a problem now. Thus a decision was made to use 12-bit to 8-bit S-boxes in the design of LOKI. This requires the use of 4 kbytes of storage for an S-box table. This was felt to be the largest practicable, since larger possibilities (eg three S-boxes with 16 input bits, or two with 24 input bits) would still require a prohibitively large store.

In the light of these principles, it was decided to use a regular permutation P which simply sends output bits from each S-box to the inputs of S-boxes in turn. It may be regarded as being generated by a difference function (on output bits 0 to 7 respectively) of

$$[+3\ +2\ +1\ 0\ +3\ +2\ +1\ 0] \qquad \text{(Eq 7.10)}$$

This form of permutation enables an efficient implementation in hardware and software due to the repeated wiring pattern. In conjunction with an expansion function E that selects blocks of 12-bits at a time, another efficiently implemented operation, this will provide a rapid growth in bit dependencies.

The other key component of the overall design is the key schedule. In [BrS90b], the design of key schedules in variants of the DES was examined. The principles derived in this chapter may be summarised as follows. The key schedule ensures that:

- KS1
  each key bit is used as input to each S-box in turn
- KS2
  no bit is used as autoclave inputs on successive rounds
- KS3
  no bit is excluded on successive rounds
- KS4
  the final key register value is identical to the original key register value (to enable easy reversal of the key schedule for decryption)

In the design of the DES, small key rotations were used, which required the use of permutation PC2 to provide a fan-out of key-bits across the S-box inputs. In the design of LOKI, a large key rotation is being used, which means that no PC2 is needed. A rotation value of 12 was chosen, both because it is the size of an S-box input, and because after eight rotations, the key register contents are the same as the original value (since $12*8=96=3*32$). It ensures that all 12 inputs to a given S-box are shifted to the next S-box on the next round. Further, because of the design of E, the blocks of 4 autoclave/message/autoclave input bits change roles in the next round. The key rotation is only done on the half of the key register being used as input to function f. Table 7.4 shows the partial keys used in each round (where the input key is represented as $abcdefghijklmnop$, each letter being a (4-bit) nybble. Note this bears a fair degree of similarity to that used in Lucifer, as shown in Sorkin [Sor84].

Table 7.4 - LOKI Key Schedule Overview

Input Key = abcdefgh ijklmnop

Round	Partial Keys Used			
1	hab	bcd	def	fgh
2	pij	jkl	lmn	nop
3	cde	efg	gha	abc
4	klm	mno	opi	ijk
5	fgh	hab	bcd	def
6	nop	pij	jkl	lmn
7	abc	cde	efg	gha
8	ijk	klm	mno	opi
9	def	fgh	hab	bcd
10	lmn	nop	pij	jkl
11	gha	abc	cde	efg
12	opi	ijk	klm	mno
13	bcd	def	fgh	hab
14	jkl	lmn	nop	pij
15	efg	gha	abc	cde
16	mno	opi	ijk	klm

A decision to use 16 rounds was made, both for consistency with the DES, and in the light of Shamir and Bihar [BiS90] who show that block ciphers may be cryptanalysed if they have fewer rounds (specifically less than 8 was claimed to be easy).

The final component in the overall design was the decision to use only half of the key register as input to function f on each round, by shifting the addition of key and data to before application of expansion function E. This choice was made primarily on grounds of ease of implementation. To compensate for the reduced number of key bits introduced into each round, the key K is added modulo 2 to the plaintext X before starting the 16 rounds, and to the ciphertext Y after the rounds. This greatly assists the growth of dependencies, and also hinders cryptanalysis of bit values at the output of the final stage.

The final overall form of the LOKI primitive is shown in Fig 7.1. The permutations P and E were listed in Tables 7.1 and 7.3 respectively. The dependency growth for this design is summarised in Table 7.5, with the full details being provided in Appendix 3. In brief it provides full dependence of ciphertext bits on plaintext bits in 3 rounds via either message or autoclave inputs, and via both message and autoclave inputs. This compares to 5 rounds for the current DES. It provides full dependence of ciphertext bits on key bits in 3 rounds, again via either or both message and autoclave inputs, compared to 5 and 7 rounds respectively, for the current DES. The author believes this very rapidly builds growth in dependence, and thus will satisfy the avalanche and completeness properties.

Table 7.5 - Dependency of Ciphertext bits on Plaintext and Key bits by Both Message and Autoclave S-box Inputs								
Round	1	2	3	4	5	6	7	8
Cipher	Both, Either	Both, Either	Both, Either	Both, Either	Both, Either	Both, Either	Both, Either	Both, Either
Ciphertext Dependence on Plaintext (CPdep)								
LOKI CPdep	0.00 11.72	25.00 45.31	75.00 84.38	100.0 100.0	100.0 100.0	100.0 100.0	100.0 100.0	100.0 100.0
DES CPdep	0.00 6.25	4.22 32.06	33.45 73.49	79.22 96.90	100.0 100.0	100.0 100.0	100.0 100.0	100.0 100.0
Ciphertext Dependence on Key (CKdep)								
LOKI CKdep	0.00 20.02	50.00 69.24	100.0 100.0	100.0 100.0	100.0 100.0	100.0 100.0	100.0 100.0	100.0 100.0
DES CKdep	0.00 5.36	2.01 39.17	36.50 82.25	81.03 98.44	95.87 100.0	99.33 100.0	100.0 100.0	100.0 100.0

2.3. Key Representation and Choice

LOKI keys are 64-bit blocks, numbered as specified in the Definitions. These keys may be written in hexadecimal thus: hhhhhhhhhhhhhhhh, where h is one hex (4-bit) digit. All 64 bits of the key are used in the LOKI algorithm and contribute to the confusion-diffusion process. There is no concept of parity bits in the key. Valid keys may thus cover the range 0000000000000000_{16} to $ffffffffffffffff_{16}$.

The cryptographic strength of the LOKI algorithm is greatly reduced if only a small number of internal sub-keys are generated. Thus keys which produce such sub-keys should be avoided. Such keys have been classified as weak, semi-weak, or demi-semi-weak, depending on the number of distinct sub-keys actually formed. For LOKI, keys which fall into these catagories are detailed below:

- **Weak keys** are those which result in only a single sub-key being formed on all 16 rounds. These keys thus form their own decryption key, and have the form: $hhhhhhhhhhhhhhhh_{16}$, $h\ \epsilon\ [0..f\,]$. There are 16 such keys.
- **Semi-Weak keys** are those which result in two sub-keys being formed on alternate rounds. These keys thus form mutual pairs, each being the decryption key for the other, and have the form: $hhhhhhhhiiiiiiii_{16}$, $h,i\ \epsilon\ [0..f\,]$, $h \neq i$. There are 240 such keys.
- **Demi-Semi-Weak keys** are those which result in four sub-keys being formed on successive rounds. It is not known whether these form a security risk, but it is generally accepted that they should be avoided. They have the form: $hihihihijkjkjkjk_{16}$, $h,i,j,k\ \epsilon\ [0..f\,]$, $h \neq i \neq j \neq k$. There are 65280 such keys.

In brief, the keys to be avoided may be described as all keys of the form: $hihihihijkjkjkjk_{16}$, $h,i,j,k\ \epsilon\ [0..f\,]$; that is where both the first four bytes are identical, and the second four bytes are also identical. Thus they may be very easily identified and rejected when choosing keys. There are a total of 65536 (ie 2^{16}) keys to be avoided out of a total key space of 2^{64}, a very small fraction.

2.4. LOKI S-boxes

In the design of LOKI, a suitable family of functions S possessing the avalanche and completeness properties was needed, suitable for inclusion in the overall structure detailed previously.

The key principle used in the design of the LOKI S-boxes was the concept of *non-linearity* found in [PiF89], [Pie90], [Pie89], [PiS89]. The *non-linearity* of a function f, in the set F_n of all Boolean functions of n variables, is the minimum Hamming distance between f and the set of all linear Boolean functions L_n existing in F_n. Since it is generally agreed that good substitution functions should not be linear or affine, Pieprzyk argues, following the findings of [KBS86b], that functions having high non-linearity would thus be good candidates for the substitution functions in an S-P network. Further he has shown that exponentiation in a Galois Field can be used to form Boolean functions having maximum non-linearity. The author and his colleagues thus decided to use such functions as the key element in the design of the LOKI S-boxes. Interestingly, Adams and Tavares [AdT90] have independently suggested a similar set of criteria for good S-box design.

The known design principles for the DES S-boxes may be found in [Bro89], [BMP87]. From these, and from the above discussion, the following principles were identified for use in the design of the LOKI S-boxes:

- S1 Each row function in an S-box is a permutation of the integers 0 to 255
- S2 Each row function possesses maximum non-linearity
- S3 The combination of row functions should form Boolean functions having the highest possible non-linearity.

The following principles were used as confirmation of a suitable design for the S-boxes:

- S4 Changing a given input bit to an S-box should result in a given output bit changing for about half the possible inputs with that one input bit changed (avalanche property)
- S5 Changing 1 input bit to an S-box results in changing at least 2 output bits
- S6 The S-boxes were chosen to minimise the difference between the number of 1s and 0s in any S-box output when any single input bit is held constant.
- S7 When the all inputs to the S-boxes are used in turn, the outputs when concatenated together, should form a stream possessing the properties of a good pseudorandom sequence [BeP82], [Knu81], [Gol82].

Given the decision in the overall design to use S-boxes with 12-bit inputs and 8-bit outputs, the LOKI S-box has 16 row functions of 256 columns. From principles 1,2 and 3, and the observations on exponentiation, these row functions were decided to be exponentiation in Galois field $GF(2^8)$. This field has 30 irreducible polynomials which may be used to form a multiplication group; these are listed in Table 7.8. Within this field also, there are 24 exponents (composed of 13 inverse pairs, with two pairs being self-inverse) which may be used to form Boolean functions having the maximum possible non-linearity (of 112); these are listed in Table 7.9. These values were tested by Pieprzyk[3] based on the principles he has identified. For each of the sixteen row functions (S-functions), one of the 30 irreducible polynomials, and one of the 24 exponents, may be chosen to form a function of maximum non-linearity. A number of other exponents may also be used, which form functions with slightly less than maximum non-linearity. The set of all exponents in $GF2^8$ consists of two subsets: the first contains the set $\{1,2,4,16,32,64,128\}$ of exponents which form linear permutations, and these elements must not be used to form the encryption process. The second set of exponents generates nonlinear permutations only, with nonlinearities varying from 80 to 112. Although it contradicts the design principle 2, the author has neither objections nor theoretical justification against using the full set of nonlinear exponents in private versions of the LOKI primitive. There is thus an enormous number of possible combinations which may be chosen for private versions of the LOKI primitive.

The base value for the exponentiation was formed by adding the row and column values modulo 2, in order to distribute the locations of the 0 and 1 value entries. These would fall in the same columns if just the column value was used as the base, due to the characteristic feature of exponentiation that the first two entries remain unchanged.

In the standard version of the LOKI cipher, an exponent of 31 (the smallest and hence fastest to implement) was chosen, as was a set of 16 irreducible polynomials. These were chosen from the list of 30 available based on the results of some tests of principles 4 to 7 on various sets of 16 out of the 30 polynimials. The results indicated little significant variation among them, and thus 16 were selected, somewhat arbitrarily.

The resulting S-box formed from these sixteen row functions has been tested. It was found to have very high non-linearity for each of its Boolean functions as shown in Table 7.6, out of a maximum non-linearity of 1984. This satisfies principle 3. It has also been tested to see how well it satisfies principles 4 to 7, with results shown in Table 7.7.

Table 7.6 - LOKI S-box Non-linearity for the Boolean function of all Input Bits to each Output bit

Output Bit	Non-linearity
0	1920
1	1928
2	1920
3	1896
4	1928
5	1912
6	1912
7	1936

[3] private communications

Table 7.7 - LOKI - S-box Analysis
Rule 4 - Avalanche Test 9.5% percentage sum of differences from mean
Rule 5 - Double Bit Changes (Percentage Failing test) 6.8% of inputs produce 0 or 1 bit changes
Rule 6 - Near Permutation 2.0% percentage sum of differences from mean
Rule 7 - Stream statistics for all output bits over all possible inputs frequency test - 0.000000 (chi square value) serial test - 0.000336 (chi square value) runs of block - 1.3% percentage sum of differences from mean runs of gaps - 1.3% percentage sum of differences from mean

2.5. Selection of Alternate Substitution Functions S

The substitution function S provides the confusion component in the LOKI cipher. It takes a 48-bit input and produces a 32-bit output. It is composed of four S-boxes, each of which take a 12-bit input and produce an 8-bit output, which are concatenated together to form the 32-bit output of S. The form of each S-box is shown in Fig 7.3. The 12-bit input is partitioned into two segments: a 4-bit row value r formed from bits $[b_{11}\, b_{10}\, b_1\, b_0]$, and an 8-bit column value c formed from bits $[b_9\, b_8 \cdots b_3\, b_2]$. The row value r is used to select one of 16 S-functions Sfn_r, which then take as input the column value c and produce an 8-bit output value. This is defined as:

$$\mathrm{Sfn}_r = (c \oplus r)^{e_r} \bmod \mathrm{gen}_r, \qquad \text{in } GF(2^8) \qquad \text{(Eq 7.11)}$$

where gen_r is an irreducible polynomial in $GF(2^8)$, and e_r is the exponent used in forming the rth S-box.

The public version of LOKI uses four identical boxes with irreducible polynomials and exponents defined in Table 7.2. These were chosen from the set of irreducible polynomials and exponents which form a function having maximum non-linearity in $GF(2^8)$, as detailed in [Pie90]. In this field there are a total of 30 irreducible polynomials; these are listed in Table 7.8. For each of these polynomials, any of the set of exponents listed in Table 7.9 results in a function having maximum non-linearity (though these functions may not be all independent; this is a subject for further investigation). Any sixteen of these polynomials and exponents may be chosen to form the S-functions in an S-box (and indeed four sets may be chosen to form four distinct S-boxes if desired). For the standard (public) version of LOKI, sixteen of the irreducible polynomials were selected at random, and exponent 31, being the smallest, was chosen.

For versions of LOKI implemented in software, it is also possible to replace permutation P with another if desired. Guidelines for the design of this permutation may be found in [BrS90a].

Table 7.8 - Irreducible Polynomials for LOKI S-boxes [4]	
in binary	in decimal
100011011	283
100011101	285
100101011	299
100101101	301
100111001	313
100111111	319
101001101	333
101011111	351
101100011	355
101100101	357
101101001	361
101110001	369
101110111	375
101111011	379
110000111	391
110001011	395
110001101	397
110011111	415
110100011	419
110101001	425
110110001	433
110111101	445
111000011	451
111001111	463
111010111	471
111011101	477
111100111	487
111110011	499
111110101	501
111111001	505

Table 7.9 - Exponents Inverse Pairs
Available for use in the LOKI S-boxes

31	62	91	107	109	127	173	182	191	199	223	239	254
181	218	241	143	124	253	227	248	251	214	247	”	”

[4] the polynomials are expressed with the left most bit being most significant. ie the string 100011011 represents the polynomial $x^8+x^4+x^3+x+1$.

3. Additional Modes of Use

The LOKI primitive may also be used in any mode of operation currently defined for DES, with which it is interface compatible [AAA83]. In addition, two modes of use are defined using the LOKI primitive for the purpose of providing message authentication. The Single Block Hash (SBH) mode computes a 64-bit Message Authentication Code (MAC or hash value), from an arbitrary length of message input. The Double Block Hash (DBH) mode computes a 128-bit MAC from an arbitrary length of message input.

In the following definitions, the LOKI primitive used for encryption is denoted $Y = EL_K(X)$. That is, Y is a 64-bit block formed by encrypting input X using the LOKI primitive with key K.

3.1. Single Block Hash (SBH) Mode

The SBH mode is defined as follows. Data for which a hash is to be computed is divided into 64-bit blocks, the final block being padded with nulls if required. A 64-bit key is supplied, and is used as the initial hash value IV. For each message block M_i: that block is added modulo 2 to the previous hash value to form a key. That key is used to encrypt the previous hash value. The encrypted value is added modulo 2 to the previous hash value to form the new hash value (see Fig 7.4). The SBH code is the final hash value formed. This process may be summarised as:

$$H_0 = IV$$

$$H_i = EL_{M_i \oplus H_{i-1}}(H_{i-1}) \oplus H_{i-1} \qquad \text{(Eq 7.12)}$$

$$SBH = H_n$$

Fig 7.4 - LOKI Single Block Hash (SBH)

The SBH mode is a variant of the Davies and Meyer hash function described in [DaP89], [Win83]. The major extension is the addition modulo 2 of the previous hash value to the current message block before using it as key input to the LOKI primitive. This was desired to prevent weak keys being supplied to the primitive when the message data was constant. If the Initialization Value is chosen not to be a weak key, then the chance of generating a weak key from a given message stream should be greatly reduced.

3.2. Double Block Hash (DBH) Mode

The DBH mode is shown in Fig 7.5, and is defined as follows. Data for which a hash is to be computed is divided into pairs of 64-bit blocks M_{2i+1}, M_{2i+2}, the final block being padded with nulls if required. A 128-bit key is supplied, composed of two 64-bit blocks, which are used as the initial hash values IV_{-1}, IV_0.

$$H_{-1} = IV_{-1} \qquad \text{(Eq 7.13)}$$

$$H_0 = IV_0$$

For each pair of message blocks M_{2i+1} M_{2i+2}, the following calculation is done:

$$T = EL_{M_{2i+1} \oplus H_{2i-1}}(H_{2i-1} \oplus M_{2i+2}) \oplus M_{2i+2} \oplus H_{2i} \qquad \text{(Eq 7.14)}$$

$$H_{2i+1} = EL_{M_{2i+2} \oplus H_{2i}}(T \oplus M_{2i+1}) \oplus M_{2i+1} \oplus H_{2i-1} \oplus H_{2i}$$

$$H_{2i+2} = T \oplus H_{2i-1}$$

The DBH block is formed by concatenating the final two hash values as follows:

$$DBH = H_{n-1} \mid H_n \qquad \text{(Eq 7.15)}$$

Fig 7.5 - LOKI Double Block Hash (DBH)

The DBH mode is derived from that proposed by Quisquater and Girault [QuG90]. Again it was extended by the addition modulo 2 of the previous hash value to the current message block before using it as key input to the LOKI primitive.

4. Performance and Implementations

4.1. Software Implementation

Three variations which trade size for speed have been identified for implementing the LOKI primitive.

i the smallest and slowest explicitly computes the S-function values within the LOKI S-boxes, and performs permutation P using masks and shifts. This version would be suitable for use where speed is not essential, for example within smartcards.

ii the medium size and speed version uses a 4 kbyte table to save the values of the S-functions, but still performs permutation P using masks and shifts.

iii the largest but fastest version uses a 64 kbyte table to hold all four S-box values with output bits pre-permuted by permutation P. Implementing the SP functions then consists of four bit field extracts, table lookup and inclusive ORs. This version is suitable for use on general purpose computers.

4.2. Hardware Implementation

A draft register level design has been sketched out for a possible hardware implementation. It uses a 4 kbyte ROM to store the values of the S-functions, and a set of 32-bit buses and registers to hold the intermediate values and implement the overall design. An initial timing chart has been done which indicates that about 140 register transfers or arithmetic operations need to be done to encrypt a 64-bit block using one of the standard modes.

5. Conclusion

The LOKI cryptographic primitive, and its associated modes of use for message authentication have been described. This algorithm is currently undergoing evaluation and testing by several parties.

The LOKI primitive may also be used in any mode of operation currently defined for DES, with which it is interface compatible [AAA83]. In addition, two modes of use are defined using the LOKI primitive for the purpose of providing message authentication. The Single Block Hash (SBH) mode computes a 64-bit Message Authentication Code (MAC or hash value), from an arbitrary length of message input. The Double Block Hash (DBH) mode computes a 128-bit MAC from an arbitrary length of message input.

The design of the LOKI family of encryption algorithms is the major achievement of this thesis. It builds on the earlier work identifying the need for new private-key algorithms, and the subsequent analysis of the DES leading to the development of suitable design rules. In conjunction with the work of Pieprzyk on S-box design, this lead to the development of the LOKI family of algorithms. It is hoped that this work will shed further light on the difficult problem of the design of good new encryption algorithms.

-

Chapter 7

CHAPTER 8

Conclusion

This thesis has analysed some existing encryption algorithms and used that analysis in the design of a new private key encryption algorithm, LOKI. In brief, the results of this analysis and design may be summarised as follows.

1. Results Obtained in this Thesis

Chapter 2 surveyed the field of cryptography, providing an overview on the design of modern private and public key schemes. Particular emphasis was given to the design of modern, private-key, block ciphers like the DES, and to exponentiation based public-key ciphers such as RSA. These form the basis of the subsequent work in this thesis.

Chapter 3 detailed various alternative methods for implementing the RSA scheme, this being the most widely regarded public-key cryptosystem. Methods detailed included conventional multi-precision arithmetic with modulo reduction performed either in software on conventional machines or specially designed long bit-length hardware. Alternatively specialized multiplication algorithms optimized for use on high-speed Digital Signal Processing (DSP) chips were discussed. Finally the use of carry-save multiplication techniques on specially designed chips was examined. None of these methods seem to offer much hope of implementing exponentiation, and hence the RSA scheme sufficiently fast enough to be able to encrypt all data being exchanged on modern high-speed data links. Hence there is a need for good private-key algorithms for use in a hybrid scheme which exchanges session keys and authentication information using a public key scheme such as RSA, but which then uses a private key scheme for data encryption. The remainder of the thesis was devoted to the development of a new private key algorithm, suitable for this use.

Chapter 4 examines all the known design criteria for the S-boxes, P-boxes, and key scheduling used in the Data Encryption Algorithm (DES). The DES was chosen for detailed analysis as it the only standard encryption algorithm used at present, and one which has been subjected to detailed attack over the past 12 years since its adoption. Although there are doubts about the size of the key used in the DES, in all other respects its design is believed to be sound. The design criteria all appear to be related to the avalanche and completeness effects, key concepts developed from work by Shannon [Sha49] by Feistel [Fei73] and Kam and Davida [KaD79]. These concepts are central to the subsequent work in this thesis. This chapter concluded with a discussion on how these criteria could be used to design an extended DES-like Feistel cipher, as well as aiding in identifying further design criteria.

Chapter 5 reviewed some possible design criteria for the permutation P in new Feistel ciphers. These permutations provide the diffusion component in a substitution-permutation network. Some empirical rules which seem to account for the derivation of the permutation used in the DES were first presented. Then it is noted that these permutations may be regarded as latin-squares which link the outputs of S-boxes to their inputs at the next stage. When a sample of these latin-square permutations were generated, they were found to span those generated using the empirical rules. The best of these permutations suggested that permutations having a fixed column structure would perform well. Permutations with this structure were generated, and the best of these analysed. They were found to include the permutations generated using difference functions which were strictly regular. Whilst these performed well in growth of overall bit dependence, when dependence on both message and autoclave bits were examined, these permutations were found to be slightly less than optimal. From these a set of design rules have been developed for permutation P, as follows:

- P1 Each of the input bits $a\ b\ c\ d\ e\ f$ for S-box S(i) must come from the outputs of different S-boxes in the previous round.
- P2 None of the input bits $a\ b\ c\ d\ e\ f$ for S-box S(i) may come from the output of that same S-box S(i) in the previous round.
- P3 For each input bit a, b, c, d, e, or f over all of the S-boxes, the S-boxes permuted to that bit must differ (ie each column forms a permutation of integers 1 to 8).
- P4 An output from S(i-2) goes to one of the ab input bits of S(i), and from S(i+1) to the other. Hence via E an output from S(i-1) goes to one of the ef input bits, and from S(i+2) to other.
- P5 Outputs from two of S(i-3), S(i+3), and S(i+4) go to cd input bits of S(i).
- P6 For each S-box output, two bits go to ab or ef input bits, the other two go to cd input bits as noted in [Dav82].

An extension of these results to a Feistel ciphers was then analysed, and 4 permutations were found to result in complete ciphertext dependency on plaintext bits after 5 rounds, a result that is the same as the current DES.

Chapter 6 examined the key schedule used in the current DES. Some empirical principles which could have been used in its design were then derived. These were used to test a number of alternative key schedules, which led to the development of a new set of generalized principles to be used in the design of a new algorithm. These design principles for the key schedule ensures that:

- KS1
  each key bit is used as input to each S-box in turn
- KS2
  no bit is used as autoclave inputs on successive rounds
- KS3
  no bit is excluded on successive rounds
- KS4
  the final key register value is identical to the original key register value (to enable easy reversal of the key schedule for decryption)

An alternative key schedule which either eliminates permutation $PC2$, or uses a local $PC2$, was tried and found to be as effective as that used in the current DES. These generalised rules were used in the subsequent design of the LOKI family of ciphers.

Chapter 7 describes the LOKI encryption primitive which may be used to encrypt and decrypt a 64-bit block of data using a 64-bit key. This is the main achievement of this thesis. Although some of the earlier work was directed toward the design of a cipher with 128-bit data and keys, in the short term there was considerable interest in the design of ciphers which were compatible with the existing DES ciper. Thus a cipher with a full 64-bit key was the first to be designed using the principles identified. The author intends to extend this work to a cipher with 128-bit data and keys in the near future. The LOKI cipher is the first of an extensible family of encryption algorithms. These algorithms all share the same overall structure, but allow the substitution and permutation functions, central to the security of the schemes, to be tailored to permit the design of private ciphers.

The LOKI primitive may be used in any mode of operation currently defined for DES, with which it is interface compatible. Also described in this chapter are two new modes of operation of the LOKI primitive which compute a 64-bit, and 128-bit, Message Authentication Code (or hash value) respectively, from an arbitrary length of message input. These modes of operation may be used to provide authentication of a communications session, or of data files.

The Appendices of this thesis included the detailed results from the analysis of the LOKI algorithm, the C program source for the DES dependency analysis programs and for a sample implementation of the LOKI algorithm, and finally two papers on the applications of encryption algorithms, including LOKI, in the field of secure remote logins, and the provision of Pay-TV in Australia.

2. Assumptions and Limitations of the Research

This thesis has been primarily concerned with the design of a new private-key block cipher - LOKI. The overall design of LOKI was based on design principles developed from a careful analysis of the DES and DES-like algorithms, in the light of the avalanche and completeness properties. It is a fundamental assumption in this work, that the overall structure of the DES is basically sound, and that the avalanche and completeness properties are key indicators of a good cryptographic scheme. The author believes these assumptions to be reasonable. The DES has withstood a concerted analysis by the open research community, since its adoption in 1977. The most successful attack to date, that of Biham and Shamir [BiS90] still performs worse than an exhaustive key-space search on the DES with the full 16 rounds. Further, the analyses in this thesis show that the DES performs substantially better than average in its development of the avalanche and completeness properties. The utility of these properties, in turn, may be traced back to Shannon [Sha49], who described a Substitution-Permutation network as providing confusion and diffusion of the bits. These thus form the basis on which the work in this thesis is dependent.

3. Suggestions for Further Research

In the near future, it is hoped that a 128-bit version of LOKI will be developed using the design principles identified here. There is also scope for comparing these criteria, along with the S-box design by Pieprzyk, against variations of the general avalanche and completeness properties, such as the strict avalanche property [Llo90]. The impact of Biham and Shamir's differential cryptanalysis techniques on the design principles also needs to be evaluated. In the longer term, theoretical work by Pieprzyk and others should lead to a better understanding on how to construct provably good randomiser functions. This will supplant some of the empirical design criteria developed here.

APPENDIX A

Analysis of the LOKI Primitive

1. LOKI Dependency Analysis

To provide a measure of the effectiveness of the permutation P, and the key schedule used in the LOKI cipher, Meyer's analysis [Mey78], [MeM82] of ciphertext dependence on plaintext bits was used in the design of the overall structure of LOKI. Briefly, following Meyer, this analysis may be described as follows. To provide a measure of the dependency of ciphertext bits on plaintext bits, a $64*64$ array $G_{a,b}$ is formed. Each element $G_{a,b}(i,j)$ specifies a dependency of output bit $X(j)$ on input bit $X(i)$, between rounds a and b. The number of marked elements in $G_{0,r}$ indicates the degree to which complete dependence was achieved by round r. Similarly, the dependency of ciphertext bits on key bits is measured by forming a $64*64$ array $F_{a,b}$, each element of which specifies a dependency of output bit $X(j)$ on key bit $K(i)$. Again, the number of marked elements in $F_{0,r}$ will be examined to provide a profile of the degree of dependence achieved by round r.

Details of the derivation of these matrices, and the means by which entries are propagated, may be found in [MeM82]. This derivation has obviously been adapted by the author for use with the LOKI cipher. This involves changing the dependences through the S-boxes, permutations and key schedule to that used in the LOKI scheme. The matrices found for the LOKI primitive for ciphertext bit dependence on plaintext bits (*CPdep*), and on key bits (*CKdep*), are listed in the following sections.

In these matrices, the following notation is used:

x specifies a dependence via an S-box message input bit (ie bits $b\ c\ d\ e$).

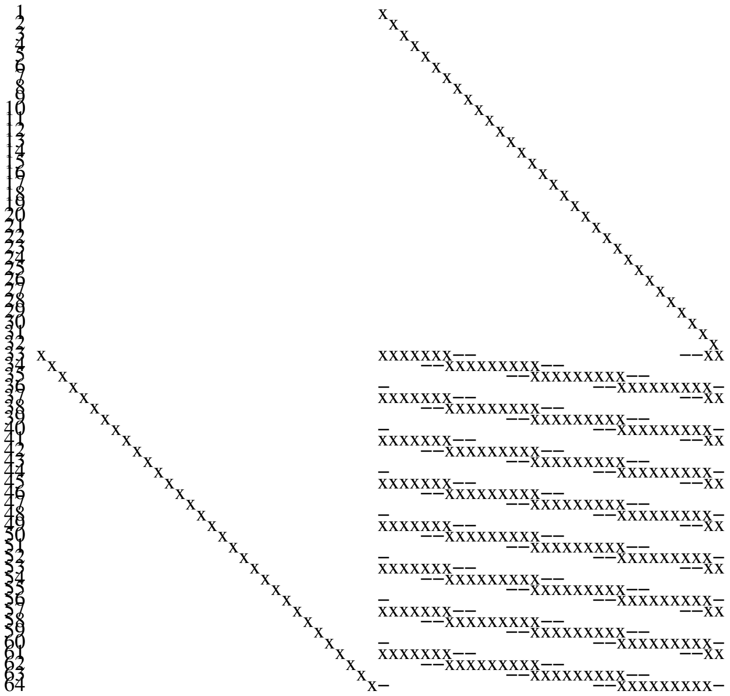
- specifies a dependence via an S-box autoclave input bit (ie bits $a\ f$).

* specifies dependencies via both message and autoclave inputs through some path of dependencies in the cipher.

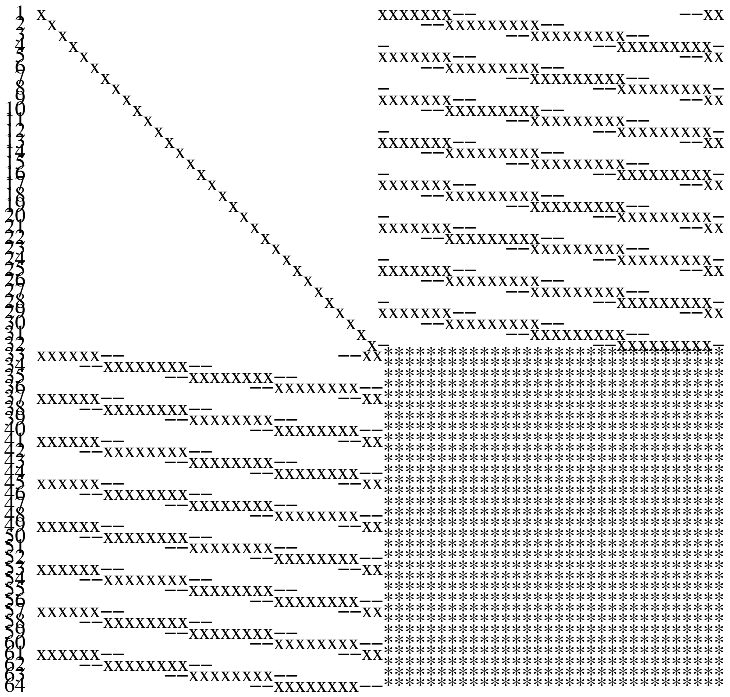
The values listed above each matrix specify the number of locations filled with each of the alternatives, and then gives percentage values for locations with any sort of dependence, and with dependence via both input types. These final two values are the ones used in the analysis earlier in this thesis.

1.1. LOKI CPdep Analysis

Round 1: None 3616, Msg 352, Autoclave 128, Both 0, Err 0 CPdep: 0.00,11.72



Round 2: None 2240, Msg 576, Autoclave 256, Both 1024, Err 0 CPdep: 25.00,45.31



Round 3: None 640, Msg 256, Autoclave 128, Both 3072, Err 0 CPdep: 75.00,84.38

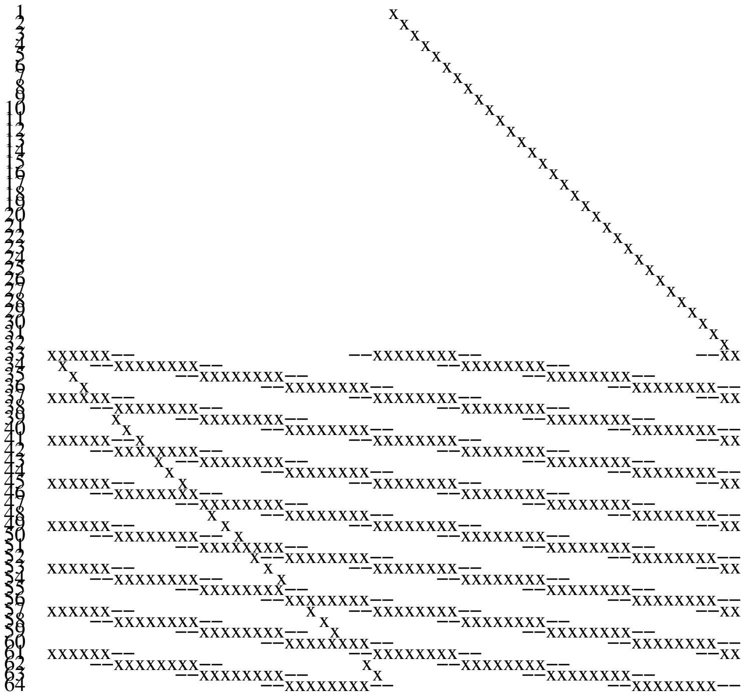
```
 1 xxxxxx--                    --xx********************************
 2     --xxxxxxxx--                ********************************
 3             --xxxxxxxx--        ********************************
 4                     --xxxxxxxx--********************************
 5 xxxxxx--                    --xx********************************
 6     --xxxxxxxx--                ********************************
 7             --xxxxxxxx--        ********************************
 8                     --xxxxxxxx--********************************
 9 xxxxxx--                    --xx********************************
10     --xxxxxxxx--                ********************************
11             --xxxxxxxx--        ********************************
12                     --xxxxxxxx--********************************
13 xxxxxx--                    --xx********************************
14     --xxxxxxxx--                ********************************
15             --xxxxxxxx--        ********************************
16                     --xxxxxxxx--********************************
17 xxxxxx--                    --xx********************************
18     --xxxxxxxx--                ********************************
19             --xxxxxxxx--        ********************************
20                     --xxxxxxxx--********************************
21 xxxxxx--                    --xx********************************
22     --xxxxxxxx--                ********************************
23             --xxxxxxxx--        ********************************
24                     --xxxxxxxx--********************************
25 xxxxxx--                    --xx********************************
26     --xxxxxxxx--                ********************************
27             --xxxxxxxx--        ********************************
28                     --xxxxxxxx--********************************
29 xxxxxx--                    --xx********************************
30     --xxxxxxxx--                ********************************
31             --xxxxxxxx--        ********************************
32                     --xxxxxxxx--********************************
33 ****************************************************************
34 ****************************************************************
35 ****************************************************************
36 ****************************************************************
37 ****************************************************************
38 ****************************************************************
39 ****************************************************************
40 ****************************************************************
41 ****************************************************************
42 ****************************************************************
43 ****************************************************************
44 ****************************************************************
45 ****************************************************************
46 ****************************************************************
47 ****************************************************************
48 ****************************************************************
49 ****************************************************************
50 ****************************************************************
51 ****************************************************************
52 ****************************************************************
53 ****************************************************************
54 ****************************************************************
55 ****************************************************************
56 ****************************************************************
57 ****************************************************************
58 ****************************************************************
59 ****************************************************************
60 ****************************************************************
61 ****************************************************************
62 ****************************************************************
63 ****************************************************************
64 ****************************************************************
```

Round 4: None 0, Msg 0, Autoclave 0, Both 4096, Err 0 CPdep: 100.00,100.00

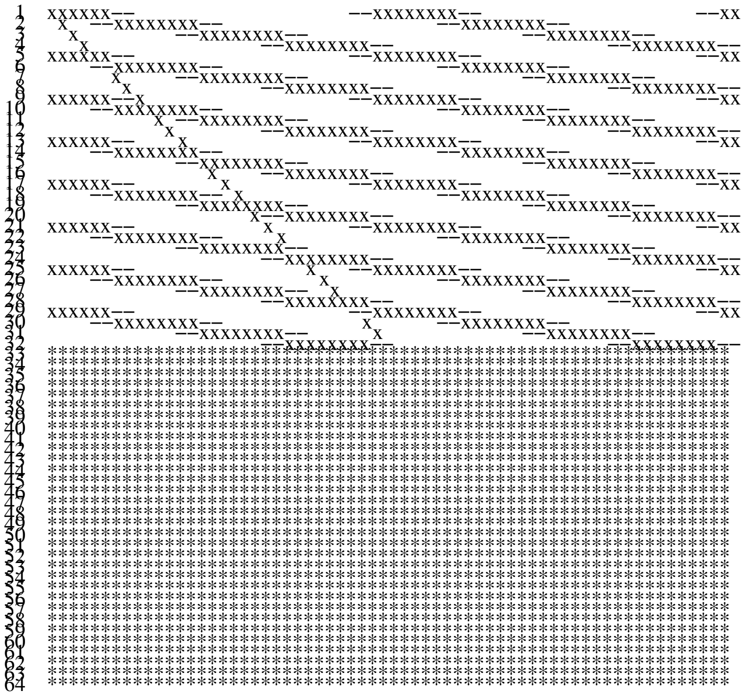
```
 1 ****************************************************************
 2 ****************************************************************
 3 ****************************************************************
 4 ****************************************************************
 5 ****************************************************************
 6 ****************************************************************
 7 ****************************************************************
 8 ****************************************************************
 9 ****************************************************************
10 ****************************************************************
11 ****************************************************************
12 ****************************************************************
13 ****************************************************************
14 ****************************************************************
15 ****************************************************************
16 ****************************************************************
17 ****************************************************************
18 ****************************************************************
19 ****************************************************************
20 ****************************************************************
21 ****************************************************************
22 ****************************************************************
23 ****************************************************************
24 ****************************************************************
25 ****************************************************************
26 ****************************************************************
27 ****************************************************************
28 ****************************************************************
29 ****************************************************************
30 ****************************************************************
31 ****************************************************************
32 ****************************************************************
33 ****************************************************************
34 ****************************************************************
35 ****************************************************************
36 ****************************************************************
37 ****************************************************************
38 ****************************************************************
39 ****************************************************************
40 ****************************************************************
41 ****************************************************************
42 ****************************************************************
43 ****************************************************************
44 ****************************************************************
45 ****************************************************************
46 ****************************************************************
47 ****************************************************************
48 ****************************************************************
49 ****************************************************************
50 ****************************************************************
51 ****************************************************************
52 ****************************************************************
53 ****************************************************************
54 ****************************************************************
55 ****************************************************************
56 ****************************************************************
57 ****************************************************************
58 ****************************************************************
59 ****************************************************************
60 ****************************************************************
61 ****************************************************************
62 ****************************************************************
63 ****************************************************************
64 ****************************************************************
```

1.2. LOKI CKdep Analysis

Round 1: None 3276, Msg 564, Autoclave 256, Both 0, Err 0 CKdep: 0.00,20.02



Round 2: None 1260, Msg 532, Autoclave 256, Both 2048, Err 0 CKdep: 50.00,69.24



Round 3: None 0, Msg 0, Autoclave 0, Both 4096, Err 0 CKdep: 100.00,100.00

```
 1  ****************************************************************
 2  ****************************************************************
 3  ****************************************************************
 4  ****************************************************************
 5  ****************************************************************
 6  ****************************************************************
 7  ****************************************************************
 8  ****************************************************************
 9  ****************************************************************
10  ****************************************************************
11  ****************************************************************
12  ****************************************************************
13  ****************************************************************
14  ****************************************************************
15  ****************************************************************
16  ****************************************************************
17  ****************************************************************
18  ****************************************************************
19  ****************************************************************
20  ****************************************************************
21  ****************************************************************
22  ****************************************************************
23  ****************************************************************
24  ****************************************************************
25  ****************************************************************
26  ****************************************************************
27  ****************************************************************
28  ****************************************************************
29  ****************************************************************
30  ****************************************************************
31  ****************************************************************
32  ****************************************************************
33  ****************************************************************
34  ****************************************************************
35  ****************************************************************
36  ****************************************************************
37  ****************************************************************
38  ****************************************************************
39  ****************************************************************
40  ****************************************************************
41  ****************************************************************
42  ****************************************************************
43  ****************************************************************
44  ****************************************************************
45  ****************************************************************
46  ****************************************************************
47  ****************************************************************
48  ****************************************************************
49  ****************************************************************
50  ****************************************************************
51  ****************************************************************
52  ****************************************************************
53  ****************************************************************
54  ****************************************************************
55  ****************************************************************
56  ****************************************************************
57  ****************************************************************
58  ****************************************************************
59  ****************************************************************
60  ****************************************************************
61  ****************************************************************
62  ****************************************************************
63  ****************************************************************
64  ****************************************************************
```

Round 4: None 0, Msg 0, Autoclave 0, Both 4096, Err 0 CKdep: 100.00,100.00

```
 1  ****************************************************************
 2  ****************************************************************
 3  ****************************************************************
 4  ****************************************************************
 5  ****************************************************************
 6  ****************************************************************
 7  ****************************************************************
 8  ****************************************************************
 9  ****************************************************************
10  ****************************************************************
11  ****************************************************************
12  ****************************************************************
13  ****************************************************************
14  ****************************************************************
15  ****************************************************************
16  ****************************************************************
17  ****************************************************************
18  ****************************************************************
19  ****************************************************************
20  ****************************************************************
21  ****************************************************************
22  ****************************************************************
23  ****************************************************************
24  ****************************************************************
25  ****************************************************************
26  ****************************************************************
27  ****************************************************************
28  ****************************************************************
29  ****************************************************************
30  ****************************************************************
31  ****************************************************************
32  ****************************************************************
33  ****************************************************************
34  ****************************************************************
35  ****************************************************************
36  ****************************************************************
37  ****************************************************************
38  ****************************************************************
39  ****************************************************************
40  ****************************************************************
41  ****************************************************************
42  ****************************************************************
43  ****************************************************************
44  ****************************************************************
45  ****************************************************************
46  ****************************************************************
47  ****************************************************************
48  ****************************************************************
49  ****************************************************************
50  ****************************************************************
51  ****************************************************************
52  ****************************************************************
53  ****************************************************************
54  ****************************************************************
55  ****************************************************************
56  ****************************************************************
57  ****************************************************************
58  ****************************************************************
59  ****************************************************************
60  ****************************************************************
61  ****************************************************************
62  ****************************************************************
63  ****************************************************************
64  ****************************************************************
```

2. Analysis of the LOKI S-Boxes

Below are the statistics generated from the analysis of the LOKI S-boxes for their compliance with the principles presented in Section II.b. The following tests were done:

Avalanche Test and One-bit Changes (Rules 4,5)

for each input bit in turn, the set of all possible pairs of vectors, differing in only that bit were generated. Then for each out put bit, the number of those pairs in which the given output bit changed was counted. The expected value was about half of the possible pairs. Also counts were made of the number of pairs for which the output vectors were identical, or differed only by one bit.

Near Permutation (Rule 6)

for each input bit in turn, the set of all possible vectors with that bit set were generated. For each output bit, the number of those vectors for which the specified output bit was set, were counted. The expected value was about half of the possible input vectors.

Stream Statistics (Rule 7)

sets of statistics were formed from the output stream for given input combinations and output bits. The statistics found were:

Frequency Test

in which counts of single bits were done, and the Chi-square value calculated.

Serial Test

in which counts of bit pairs were done, the Chi-square value calculated, and counts of bit triples done.

Run Tests

in which counts of blocks and gaps of various lengths are done, and the sum of differences from the expected values found.

These statistics are found for the stream formed by concatenating all output bits formed from all possible input combinations (full statistics shown), from all input combinations when one bit is set; and from the stream formed from a single output bit over all possible input combinations (short form statistics shown).

LOKI - S-box Analysis

loki_s - Avalanche Test & 1-bit Changes (S-Box rule 3)
for each input bit, check for all possible inputs where it is set
that the avalache vector generated using the LOKI S-boxes changes
each output bit some times

Input Bit I:	0	1	2	3	4	5	6	7	8	9	10	11	Diff
output bit 0:	1046	1036	1084	1056	1036	1054	1088	1020	1034	1032	1036	1028	(262)
output bit 1:	984	1038	990	1048	1054	1024	1042	994	1056	1016	1042	982	(-18)
output bit 2:	1020	1042	930	1066	1026	1038	1010	1036	1014	1026	1042	1038	(0)
output bit 3:	1000	986	968	1028	996	1032	1052	978	1046	1072	1060	1004	(-66)
output bit 4:	1024	1012	982	1020	1018	1032	1034	990	1000	1034	1032	1004	(-106)
output bit 5:	1012	1054	972	1028	1026	1092	1040	990	1016	1056	1046	1060	(104)
output bit 6:	1016	1004	954	1038	1030	1050	1058	1050	1002	1044	968	1038	(-36)
output bit 7:	1028	1028	974	1030	1002	1034	1056	1042	1040	1012	1000	998	(-44)
column totals	-62	8	-338	122	-4	164	188	-92	16	100	34	-40	(96)
No avalanche	3	4	0	0	0	0	0	0	0	0	6	8	(21)
1-bit avalanche	90	60	168	46	51	47	51	76	44	47	62	63	(805)

loki_s - Test for nearness to true permutation (S-box Rule 6)
check how many input combinations result in a 1 being
generated for a given output bit in LOKI S-boxes

Input Bit I:	0	1	2	3	4	5	6	7	8	9	10	11	Diff
output bit 0:	1024	1024	1024	1000	996	1020	1044	992	1016	1016	1024	1024	(-84)
output bit 1:	1024	1024	1032	1028	1008	1000	1032	1020	1032	1032	1024	1024	(-8)
output bit 2:	1024	1024	1032	1024	1036	1028	1044	1024	1028	1056	1024	1024	(80)
output bit 3:	1024	1024	1020	1028	1024	1028	1068	1012	1032	1032	1024	1024	(52)
output bit 4:	1024	1024	1020	1060	1008	1020	1084	1012	1052	1024	1024	1024	(88)
output bit 5:	1024	1024	1008	1016	1016	1032	1036	1088	1016	1020	1024	1024	(40)
output bit 6:	1024	1024	1020	1016	992	1048	1004	1040	1084	1020	1024	1024	(32)
output bit 7:	1024	1024	988	1048	1040	1016	1012	1028	1028	1080	1024	1024	(48)
column totals	0	0	-48	28	-72	0	132	24	96	88	0	0	(248)

Streamstat analysis on S-box for all inputs & all outputs
streamstat: len 32768; ln2(len) 15.0
freq test: single 0: 16384, 1: 16384, chi^2: 0.000000
serial test: double 00: 8192, 01: 8191, 02: 8191, 03: 8193, chi^2: 0.000610
poker triples 00: 1357, 01: 1400, 02: 1325, 03: 1370,
04: 1395, 05: 1320, 06: 1381, 07: 1374
3-bit 4.524542, 4-bit 0.000000, 8-bit 0.000000

len	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	>=16	chi^2
gaps	4070	2076	1022	520	261	134	58	18	17	7	0	1	0	0	7	1	13.51
blocks	4081	2063	1042	480	268	135	57	35	15	7	4	2	1	0	0	1	4.37
mean	4095	2048	1024	511	255	127	63	32	15	7	4	2	1	0	0	0	8184.00

Streamstat analysis on S-box for input bit 0 set true
ss: 16384 0.000 0.018 3.453 0.000 0.000 6.621 4.730

Streamstat analysis on S-box for input bit 1 set true
ss: 16384 0.000 0.045 2.028 0.000 0.000 15.636 9.738

Streamstat analysis on S-box for input bit 2 set true
ss: 16384 0.562 -0.015 10.582 6.344 123.250 12.442 7.671

Streamstat analysis on S-box for input bit 3 set true
ss: 16384 0.191 0.117 12.562 8.281 129.000 10.937 2.355

Streamstat analysis on S-box for input bit 4 set true
ss: 16384 1.266 0.347 12.028 8.688 127.500 13.198 6.610

Streamstat analysis on S-box for input bit 5 set true
ss: 16384 0.000 0.114 5.632 4.844 121.000 7.747 17.444

Streamstat analysis on S-box for input bit 6 set true
ss: 16384 4.254 0.897 5.260 10.906 151.000 9.562 13.566

Streamstat analysis on S-box for input bit 7 set true
ss: 16384 0.141 0.018 5.834 9.500 130.000 16.927 10.055

Streamstat analysis on S-box for input bit 8 set true
ss: 16384 2.250 0.002 5.394 6.250 135.750 13.665 11.307

Streamstat analysis on S-box for input bit 9 set true
ss: 16384 1.891 2.511 5.512 9.812 130.000 21.280 10.513

Streamstat analysis on S-box for input bit 10 set true
ss: 16384 0.000 0.084 2.474 0.000 0.000 17.074 8.891

Streamstat analysis on S-box for input bit 11 set true
ss: 16384 0.000 0.006 9.767 0.000 0.000 12.108 5.660

Streamstat analysis on S-box for output bit 0 over all inputs
ss: 4096 0.000 1.166 3.418 14.344 294.000 5.097 8.305

Streamstat analysis on S-box for output bit 1 over all inputs
ss: 4096 0.000 5.283 13.088 15.312 271.000 5.894 12.999

Streamstat analysis on S-box for output bit 2 over all inputs
ss: 4096 0.000 0.912 5.512 5.281 251.000 13.358 12.166

Streamstat analysis on S-box for output bit 3 over all inputs
ss: 4096 0.000 0.498 1.829 10.969 266.000 14.383 4.480

Streamstat analysis on S-box for output bit 4 over all inputs
ss: 4096 0.000 0.638 2.924 14.625 240.000 3.034 3.506

Streamstat analysis on S-box for output bit 5 over all inputs
ss: 4096 0.000 0.853 4.335 9.125 264.000 7.146 16.126

Streamstat analysis on S-box for output bit 6 over all inputs
ss: 4096 0.000 0.044 5.606 12.406 208.000 6.077 18.239

Streamstat analysis on S-box for output bit 7 over all inputs
ss: 4096 0.000 0.091 7.818 6.812 260.000 6.689 7.099

Appendix A

APPENDIX B

Dependency Analysis Programs

1. cp_dep.c

```
/*
 * cp_dep.c – build dependency matrix for each ciphertext bit, based on
 *         plaintext bits, for a given P (assuming S–boxes
 *         do their thing, and that their outputs are equivalent)
 *
 * Lawrence Brown <lpb@cs.adfa.oz>
 */

#include <stdio.h>

#define HSIZE 32
#define FSIZE (2 * HSIZE)

char    *name = "cp_dep";   /* program name */
char    *depnam = "CPdep";  /* dependency flag name sent to output file */
char    *usage = "[–vV] [–dD] [–rR rounds] [–P P_values] "; /* usage string */

int P[HSIZE] = {4, 2, 5, 6,
          8, 3, 7, 5,
          1, 4, 6, 7,
          2, 5, 8, 3,
          1, 2, 6, 4,
          8, 7, 1, 3,
          5, 4, 8, 2,
          6, 3, 1, 7};
      /* Permutation matrix by which S–box input bit is from */

int G[FSIZE][FSIZE]; /* dependency matrix G[i][j] gives dep of i on j */

int sp,    /* number of bits in G with no dependency   */
   me,    /*  message only dependency             */
```

```
   ac,    /*  autoclave only dependency           */
   bt,    /*  both message & autoclave dependency */
   tot,   /* percentage of bits with a dependency     */
   err;   /* number with bad specification            */

int verbose = 0; /* verbose output flag */
int dstat= 0;   /* print detailed stats (but less than verbose) flag */
int pboth= 0;   /* print % of BOTH msg+auto & Overall dependecy      */
int maxrounds = 8;  /* number of rounds to analyse */

char    disp[4] = {´ ´, ´x´, ´−´, ´*´}; /* display char for each class */

main(argc, argv)                                                        main
int argc;
char    **argv;
{
   int i;

   init(argc, argv);

   prP();
   G_init();
   if (verbose)    printf("Round 1:");
   else        printf("%s: ", depnam);
   scanG();
   if (verbose)    prG();

   for (i=2; i<=maxrounds; i++){
     nextG();
     if (verbose) printf("Round %d:", i);
     scanG();
     if (verbose) prG();
   }
   printf("\n");
}

G_init()
{
   int i, j, pbase;

   /* clear G, nb LL is 0 */
   for (i=0; i<FSIZE; i++)
    for (j=0; j<FSIZE; j++)
      G[i][j] = 000;
```

```
    /* Set LR, RL to Identity */
    for (i=0; i<HSIZE; i++)
      G[i][i+HSIZE] = G[i+HSIZE][i] = 001;

    /* Set RR based on f */
    for (i=0; i<HSIZE; i++) {
      pbase = (P[i] − 1) * 4;     /* use P to find which S−box */
      G[i+HSIZE][((pbase + HSIZE − 1) % HSIZE) + HSIZE] = 002;
      G[i+HSIZE][((pbase     ) % HSIZE) + HSIZE] = 001;
      G[i+HSIZE][((pbase +  1) % HSIZE) + HSIZE] = 001;
      G[i+HSIZE][((pbase +  2) % HSIZE) + HSIZE] = 001;
      G[i+HSIZE][((pbase +  3) % HSIZE) + HSIZE] = 001;
      G[i+HSIZE][((pbase +  4) % HSIZE) + HSIZE] = 002; /* use E */
    }
}
```

nextG

```
nextG()
{
    int s, j, pbase;
    int m1, m2, m3, m4, m5, m6;
    int nG[FSIZE][FSIZE];       /* tmp store for new array G */

    /* Do L(i+1) = R(i) */
    for (s=0; s<HSIZE; s++)
     for (j=0; j<FSIZE; j++)
      nG[s][j] = G[s+HSIZE][j];

    /* Do R(i+1) = L(i) + f(R(i)) */
    for (s=0; s<HSIZE; s++) {
      pbase = (P[s] − 1) * 4;             /* use P to find which S−box */
      m1 = ((pbase + HSIZE − 1) % HSIZE) + HSIZE; /* find input 1  */
      m2 = ((pbase     ) % HSIZE) + HSIZE; /*   find source of input 2  */
      m3 = ((pbase +  1) % HSIZE) + HSIZE; /*   find source of input 3  */
      m4 = ((pbase +  2) % HSIZE) + HSIZE; /*   find source of input 4  */
      m5 = ((pbase +  3) % HSIZE) + HSIZE; /*   find source of input 5  */
      m6 = ((pbase +  4) % HSIZE) + HSIZE; /*   find source of input 6  */
      for (j=0; j<FSIZE; j++) {
        nG[s+HSIZE][j] = G[s][j] |
                  (G[m1][j] == 00 ? 00 : 02) |
                  G[m2][j] |
                  G[m3][j] |
                  G[m4][j] |
                  G[m5][j] |
```

```
                 (G[m6][j] == 00 ? 00 : 02) ;
        }
    }

    /* copy nG into G */
    for (s=0; s<FSIZE; s++)
      for (j=0; j<FSIZE; j++)
        G[s][j] = nG[s][j];
}
```

scanG() *scanG*

```
{
    int i, j;
    float   both, ptot;

    sp = me = ac = bt = tot = err = 0;

    for (i=0; i<FSIZE; i++) {
        for (j=0; j<FSIZE; j++) {
        switch(G[i][j]) {
          case 000: sp++;   break;
          case 001: me++;   break;
          case 002: ac++;   break;
          case 003: bt++;   break;
          default:  err++;  break;
            }
        }
    }
    tot = (me + ac + bt);
    ptot = (float)tot * 100.0 / (float)(FSIZE * FSIZE);
    both = (float)bt * 100.0 / (float)(FSIZE * FSIZE);
    if (verbose) {
        printf("  None %d, Msg %d, Autoclave %d, Both %d, Err %d",
        sp, me, ac, bt, err);
        printf("  %s: %4.2,%4.2f\n", depnam, both, ptot);
    } else if (dstat) {
        printf(" %d %d %d %4.2,%4.2f", me, ac, bt, both, ptot);
    } else if (pboth) {
        printf("%4.2f,%4.2f ", both, ptot);
    } else
        printf("%4.2f ", ptot);
}
```

prG() *prG*

```
{
   int i, j;

   for (i=0; i<FSIZE; i++) {
      printf("%2d ", i+1);
      for (j=0; j<FSIZE; j++) putchar(disp[ G[i][j] ]);
      putchar('\n');
   }
   printf("\n\n\n");
}

prP()                                                              prP
{
   int i;

   printf("P: ");
   for (i=0; i<HSIZE; i++) {
      printf("%1d ", P[i]);
   }
   printf("\n");
}

init(argc, argv)                                                   init
int argc;
char    **argv;
{
   int i;

   argv++;
   while ( --argc > 0 ) {
      if ( (*argv)[0] == '-' )
      switch ( (*argv)[1] ) {
        case 'v':
        case 'V': verbose++;  argv++; break;
        case 'b':
        case 'B': pboth++;    argv++; break;
        case 'd':
        case 'D': dstat++;    argv++; break;
        case 'r':
        case 'R': argv++;
              maxrounds = atoi(*argv++);
              argc--;
              break;
        case 'P': if (argc < HSIZE) {
```

```
            fprintf(stderr, "Too few P values!!!\n");
            exit(1);
          }
          argv++;
            for (i=1; i<=HSIZE; i++)
            P[i-1] = atoi(*argv++);
          argc -= HSIZE;
          break;
      default:  fprintf(stderr,"%s - usage: %s\n", name, usage);
          break;
       }
    else
    fprintf(stderr, "Unknown argument %s\n", *argv);
  }
}
```

2. ck_dep.c

```
/*
 * ck_dep.c – build dependency matrix for each ciphertext bit, based on
 *       keytext bits, for a given P, PC2 & key schedule
 *       (assuming S–boxes do their thing, and that their
 *       outputs are equivalent)
 *
 * Lawrence Brown <lpb@cs.adfa.oz>     5/88
 */

#include <stdio.h>

#define HSIZE 32        /* size of input data block half */
#define FSIZE (2 * HSIZE)  /* size of input data block */
#define USIZE 56        /* size of key vector used in key schedule */
#define P2SIZE 48       /* size of subkey vector input to f */
#define ROTSIZE 28      /* size of key schedule register (key half) */
#define ROUNDS  16      /* Number of DES rounds */

char    *name = "ck_dep";   /* program name */
char    *depnam = "CKdep";  /* dependency flag name sent to output file */
char    *usage = "[–vV] [–dD] [–rR rounds] [–P P_values] [–C PC2_values]
                          [–f] [–K key_sched]";  /* usage string */



int P[HSIZE]    /* Permutation matrix P of which S–box each input bit is from */
= { 4, 2, 5, 6,
    8, 3, 7, 5,
    1, 4, 6, 7,
    2, 5, 8, 3,
    1, 2, 6, 4,
    8, 7, 1, 3,
    5, 4, 8, 2,
    6, 3, 1, 7};

char PC2[P2SIZE]    /* Permutation matrix PC2 of key sched bit perm */
= { 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 };

int keyrot[ROUNDS]     /* key rotation schedule       */
= { 1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1 };

int totrot[ROUNDS+1]   /* total key rotation sched   */
= { 0,1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28 };

int rotsize = ROTSIZE;  /* number of bits rotated in key sched */

int F[FSIZE][USIZE]; /* dependency matrix F[i][j] gives dependency       */
                  /*   of output bit i on key bit j at current round */

int sp,    /* number of bits in F with no dependency */
   me,     /*  message only dependency            */
   ac,    /*  autoclave only dependency           */
   bt,    /*  both message & autoclave dependency */
   tot,   /* percentage of bits with a dependency */
   err;   /* number with bad specification        */

int verbose = 0;   /* verbose output flag */
int dstat= 0;      /* print detailed stats (but less than verbose) flag */
int pboth= 0;      /* print % of BOTH msg+auto & Overall dependecy       */
int maxrounds = 8;  /* number of rounds to analyse */

char    disp[4] = {´ ´, ´x´, ´−´, ´*´}; /* display char for each class */

main(argc, argv)                                                    main
int argc;
char    **argv;
{
   int i;

   init(argc, argv);

   F_init();
   if (verbose)    printf("Round 1:");
   else        printf("%s: ", depnam);
   scanF();
   if (verbose)    prF();

   for (i=2; i<=maxrounds; i++){
     nextF(i);
     if (verbose)     printf("Round %2d:", i);
```

```
      scanF();
      if (verbose) prF();
   }
   printf("\n");
}

F_init()
{
   int i, j, kbase;

   /* clear F, nb FL is 0 */
   for (i=0; i<FSIZE; i++)
     for (j=0; j<USIZE; j++)
       F[i][j] = 000;

   /* Set FR based on f */
   for (i=0; i<HSIZE; i++) {
       kbase = (P[i] − 1) * 6; /* use P to find which S−box */
       /* kbase is the number of the first key−bit used as input    */
       /* then use PC2 to find which key−bits are input to S[kbase] */
       F[i+HSIZE][PC2[(kbase    )] − 1] = 002;
       F[i+HSIZE][PC2[(kbase + 5)] − 1] = 002;
       F[i+HSIZE][PC2[(kbase + 1)] − 1] = 001;
       F[i+HSIZE][PC2[(kbase + 2)] − 1] = 001;
       F[i+HSIZE][PC2[(kbase + 3)] − 1] = 001;
       F[i+HSIZE][PC2[(kbase + 4)] − 1] = 001;
   }
}
```

nextF

```
nextF(round)
int round;
/*  nextF − calculate the dependencies in F(i)
 *
 *     FL(i) = FR(i−1)     since L(i) = R(i)
 *     FR(i) = row is a combination of rows
 *            s in FL(i−1)               L(i−1)
 *            m1(s) to m6(s) of FR(i−1)   R(i−1)
 *            + influence of K(i) (via key sched & PC2)
 */
{
   int s, j, kbase, pbase;
   int k1, k2, k3, k4, k5, k6;
   int m1, m2, m3, m4, m5, m6;
```

```
int nF[FSIZE][USIZE];       /* tmp store for new array F */

/* Do FL(i) = FR(i−1) */
for (s=0; s<HSIZE; s++)
  for (j=0; j<USIZE; j++)
    nF[s][j] = F[s+HSIZE][j];

/* Do FR(i+1) = L(i−1) + f(K(i),R(i−1)) */
for (s=0; s<HSIZE; s++) {
    pbase = (P[s] − 1) * 4;     /* use P to find which S−box */
    /* pbase is the number of the first msg−bit used as input    */
    m1 = ((pbase + HSIZE − 1) % HSIZE) + HSIZE; /* find src of inp 1   */
    m2 = ((pbase     ) % HSIZE) + HSIZE; /*   find source of input 2  */
    m3 = ((pbase +  1) % HSIZE) + HSIZE; /*   find source of input 3  */
    m4 = ((pbase +  2) % HSIZE) + HSIZE; /*   find source of input 4  */
    m5 = ((pbase +  3) % HSIZE) + HSIZE; /*   find source of input 5  */
    m6 = ((pbase +  4) % HSIZE) + HSIZE; /*   find source of input 6  */
    for (j=0; j<USIZE; j++) {
       nF[s+HSIZE][j] = F[s][j] |          /* L(i−1)      */
                 (F[m1][j] == 00 ? 00 : 02) | /* R(i−1)     */
                 F[m2][j] |
                 F[m3][j] |
                 F[m4][j] |
                 F[m5][j] |
                 (F[m6][j] == 00 ? 00 : 02) ;
    }
    kbase = (P[s] − 1) * 6; /* use P to find which S−box */
    /* kbase is the number of the first key−bit used as input    */
    /* then use PC2 to find which key−bits are input to S[kbase] */
    k1 = krot(round, PC2[(kbase     )]);   /* K(i) */
    k2 = krot(round, PC2[(kbase + 1)]);
    k3 = krot(round, PC2[(kbase + 2)]);
    k4 = krot(round, PC2[(kbase + 3)]);
    k5 = krot(round, PC2[(kbase + 4)]);
    k6 = krot(round, PC2[(kbase + 5)]);
    nF[s+HSIZE][k1 − 1] |= 002;
    nF[s+HSIZE][k2 − 1] |= 001;
    nF[s+HSIZE][k3 − 1] |= 001;
    nF[s+HSIZE][k4 − 1] |= 001;
    nF[s+HSIZE][k5 − 1] |= 001;
    nF[s+HSIZE][k6 − 1] |= 002;
}
```

```
  /* copy nF into F */
  for (s=0; s<FSIZE; s++)
    for (j=0; j<USIZE; j++)
      F[s][j] = nF[s][j];
}

krot(i, x)                                                    krot
int i,x;
/*
 * krot – rotate key bit x by totrot[i]
 */
{
  if (x > rotsize)
      x = (x + totrot[i–1] – 1) % rotsize + rotsize + 1;
  else
      x = (x + totrot[i–1] – 1) % rotsize + 1;
  return(x);
}


scanF()                                                       scanF
{
  int i, j;
  float   ptot, both;

  sp = me = ac = bt = tot = err = 0;

  for (i=0; i<FSIZE; i++) {
      for (j=0; j<USIZE; j++) {
      switch(F[i][j]) {
        case 000: sp++;   break;
        case 001: me++;   break;
        case 002: ac++;   break;
        case 003: bt++;   break;
        default:  err++;  break;
          }
      }
  }
  tot = (me + ac + bt);
  ptot = (float)tot * 100.0 / (float)(FSIZE * USIZE);
  both = (float)bt * 100.0 / (float)(FSIZE * USIZE);
  if (verbose) {
      printf(" None %d, Msg %d, Autoclave %d, Both %d, Err %d",
      sp, me, ac, bt, err);
```

```
      printf(" %s: %4.2f,%4.2f\n", depnam, both, ptot);
   } else if (dstat) {
      printf(" %d %d %d %4.2f,%4.2f", me, ac, bt, both, ptot);
   } else if (pboth) {
      printf("%4.2f,%4.2f ", both, ptot);
   } else
      printf("%4.2f ", ptot);
}
```

prF

```
prF()
{
   int i, j;

   for (i=0; i<FSIZE; i++) {
      printf("%2d  ", i+1);
      for (j=0; j<USIZE; j++) putchar(disp[ F[i][j] ]);
      putchar(´\n´);
   }
      printf("\n\n\n");
}
```

prP

```
prP()
{
   int i;

   printf("P: ");
   for (i=0; i<HSIZE; i++) {
      printf("%d ", P[i]);
   }
   printf("\n");
}
```

prPC2

```
prPC2()
{
   int i;

   printf("PC2: ");
   for (i=0; i<P2SIZE; i++) {
      printf("%d ", PC2[i]);
   }
   printf("\n");
}
```

prKS

```
prKS()
{
    int i;

    printf("KS: ");
    for (i=0; i<ROUNDS; i++) {
        printf("%d ", keyrot[i]);
    }
    printf("\n");
    if (verbose|dstat|pboth) {
        printf("KStot: ");
        for (i=0; i<ROUNDS+1; i++) {
            printf("%d ", totrot[i]);
        }
        printf("\n");
    }
}
```

init

```
init(argc, argv)
int argc;
char    **argv;
{
    int i;

    argv++;
    while ( −−argc > 0 ) {
        if ( (*argv)[0] == ´−´ )
        switch ( (*argv)[1] ) {
          case ´v´:
          case ´V´: verbose++;  argv++; break;
          case ´b´:
          case ´B´: pboth++;    argv++; break;
          case ´d´:
          case ´D´: dstat++;    argv++; break;
          case ´r´:
          case ´R´: argv++;
                maxrounds = atoi(*argv++);
                argc−−;
                break;
          case ´P´: if (argc < HSIZE) {
                    fprintf(stderr, "Too few P values!!!\n");
                    exit(1);
                }
```

```
            argv++;
              for (i=1; i<=HSIZE; i++)
              P[i−1] = atoi(*argv++);
            argc −= HSIZE;
            prP();
            break;
        case ´C´: if (argc < P2SIZE) {
              fprintf(stderr, "Too few PC2 values!!!\n");
              exit(1);
            }
            argv++;
              for (i=1; i<=P2SIZE; i++)
              PC2[i−1] = atoi(*argv++);
            argc −= P2SIZE;
            prPC2();
            break;
        case ´f´: /* use full size key rotation registers */
        case ´F´: rotsize = 2 * ROTSIZE;
            argv++;
            break;
        case ´K´: if (argc < ROUNDS) {
              fprintf(stderr, "%d – Too few KeySched values!!!\n", argc);
              exit(1);
            }
            argv++;
              for (i=1; i<=ROUNDS; i++) /* read key rotation sched */
              keyrot[i−1] = atoi(*argv++);
              for (i=1; i<=ROUNDS; i++) /* calc total rotation sched */
              totrot[i] = totrot[i−1] + keyrot[i−1];
            argc −= ROUNDS;
            prKS();
            break;
        default:  fprintf(stderr,"%s – usage: %s\n", name, usage);
            break;
          }
      else
      fprintf(stderr, "Unknown argument %s\n", *argv);
    }
}
```

APPENDIX C

LOKI - Sample Implementation

1. Overall Program Structure

This appendix contains a listing of a sample implementation of the LOKI cipher. These implement the first (small but slow) of the possible software implementations identified previously. They modules are:

loki.h
: function declarations and typedefs for the library routines to implement the LOKI cryptographic primitive

lokimode.h
: declarations of the standard modes of operation using the LOKI primitive

loki64.i
: specification of the public constants used in the library routines to implement the LOKI cryptographic primitive

loki64.c
: the library routines to implement the LOKI cryptographic primitive

lokimac.c
: a program using the loki64.c routines which reads a stream of data and computes the SBH message authentication code for that stream.

lokicert.c
: a program using the loki64.c routines which reads a file of (key,plaintext,ciphertext) triplets and certifies that the loki64.c library routines are functioning correctly.

gf8.c
: utility routines implementing arithmetic in $ GF(2 sup 8) $.

lokimode.c
: utility routines used in implementing the standard modes of operation using the LOKI primitive

test1, test2
: files of (key,plaintext,ciphertext) triplets used by the lokicert.c program.

cert.test1
: trace of output from "lokicert test1" when "loki64.c" has been compiled with "TRACE=2" defined.

Listings are provided for the following files:

loki.h

lokimode.h

loki64.i

loki64.c

which form the core routines of the LOKI cryptographic primitive, and define macros for the standard modes of operation. The sources for the remaining routines, and for the test data and trace output, are available if required.

2. Test Data

A single test triplet is listed below (being the contents of "test1").

```
# Single LOKI Certification triplet
# data is saved as (key, plaintext, ciphertext) triplets
#
5b5a57676a56676e 675a69675e5a6b5a 3c61fa7e2e99d048
```

A set of test triplets is listed below (being the contents of "test2").

```
#
#      LOKI Validation Data Suite Triples
#      data is saved as (key, plaintext, ciphertext) triplets
#
0000000000000000 0000000000000000 0000000000000000
0000000000000000 355550b2150e2451 8e2a251b94704c69
0000000000000000 35a7bae825c0d73b 8ca64de9c1b123a7
0000000000000000 8ca64de9c1b123a7 35a7bae825c0d73b
0000000000000000 8e2a251b94704c69 355550b2150e2451
0000000000000000 ffffffffffffffff 61f38c55061e3161
0101010101010101 0123456789abcdef f60b54c240d7ed14
0101010101010101 617b3a0ce8f07100 38ae088ae853f7fb
0101010101010101 9b38f6ce85aab9c3 617b3a0ce8f07100
0113b970fd34f2ce 059b5e0851cf143a 88c37b8c809c6dcd
0113b970fd34f2ce 7514cdb961b6760d 86a560f10ec6d85b
0113b970fd34f2ce 86a560f10ec6d85b 23e9b229fdea123f
0123456789abcdef 0000000000000000 d853533a6c1beb30
0123456789abcdef 1111111111111111 c4d29774e5d5247c
0123456789abcdef 17668dfc7292532d cc9feed345a9f7a1
0123456789abcdef 23c086665917b8e1 17668dfc7292532d
0123456789abcdef d5d44ff720683d0d 13d2967efa3aa3c2
0123456789abcdef fce30226576320bd d5d44ff720683d0d
0131d9619dc1376e 5cd54ca83def57da c251a566d109cfc2
0131d9619dc1376e 65e160aed7b773a9 7a389d10354bd271
```

0131d9619dc1376e 7a389d10354bd271 94c33356f2ee1adf
0170f175468fb5e6 0756d8e0774761d2 49a480651f03b452
0170f175468fb5e6 0cd3da020021dc09 afec86ec719000ad
0170f175468fb5e6 914c1806fccbce33 0cd3da020021dc09
018310dc409b26d6 1d9d5c5018f728c2 87bd17440173768b
018310dc409b26d6 5a0bf934fd6009f8 5f4c038ed12b2e41
018310dc409b26d6 5f4c038ed12b2e41 773740aa56ef058e
025816164629b007 480d39006ee762f2 719ebffea9a16c35
025816164629b007 a1f9915541020b56 16f949d4011073b5
025816164629b007 ec92e65da168b46f a1f9915541020b56
04689104c2fd3b2f 26955f6835af609a 8139b3c65393ea37
04689104c2fd3b2f 5265227fe08a28ec 5c513c9c4886c088
04689104c2fd3b2f 5c513c9c4886c088 bfd4c8c383533bdb
04b915ba43feb5b6 42fd443059577fa2 e7244bf3b0e5715d
04b915ba43feb5b6 a483ea7ccf2e0e5a af37fb421f8c4095
04b915ba43feb5b6 af37fb421f8c4095 762bd5f9b296f82e
07a1133e4a0b2686 0248d43806f67172 dd02c943fb4537a3
07a1133e4a0b2686 624f2e2dfa008142 868ebb51cab4599a
07a1133e4a0b2686 868ebb51cab4599a 235c5997034a503d
07a7137045da2a16 28e686668c3bd6d9 dfd64a815caf1a0f
07a7137045da2a16 3bdd119049372802 3de59f157f8b6bf9
07a7137045da2a16 dfd64a815caf1a0f 00b3f10e6f1063ed
1111111111111111 0123456789abcdef b0c9897c2765e214
1111111111111111 1111111111111111 f0a12c83c4529223
1111111111111111 24900548c21a3567 f40379ab9e0ec533
1111111111111111 8a5ae1f81ab8f2dd a273d7cb7d390531
1111111111111111 a273d7cb7d390531 8a5ae1f81ab8f2dd
1111111111111111 f40379ab9e0ec533 24900548c21a3567
1c587f1c13924fef 305532286d6f295a 8b62933a45dcf71f
1c587f1c13924fef 63fac0d034d9f793 b061dff8054771a8
1c587f1c13924fef f63de067f58c38ed 63fac0d034d9f793
1f08260d1ac2465e 2c0a241eb9f05999 ef1bf03e5dfa575a
1f08260d1ac2465e 6b056e18759f5cca c07261a9596039c3
1f08260d1ac2465e ef1bf03e5dfa575a b8705cc3a19ae567
1f1f1f1f0e0e0e0e 0123456789abcdef 16be5183d694e578
1f1f1f1f0e0e0e0e 322f206b2d39d65d db958605f8c8c606
1f1f1f1f0e0e0e0e db958605f8c8c606 1c37794afacf9e4e
3000000000000000 1000000000000001 85884d224ec9e63d
3000000000000000 958e6e627a05557b 389b4e8dc846c4aa
3000000000000000 d3c4539579b96231 958e6e627a05557b
37d06bb516cb7546 0a2aeeae3ff4ab77 45127fb1f43c5304
37d06bb516cb7546 164d5e404f275232 70f6cf2136677443
37d06bb516cb7546 19acae3136c0bc7c 0a2aeeae3ff4ab77
3849674c2602319e 126898d55e911500 7178876e01f19b2a

3849674c2602319e 51454b582ddf440a 57df30a9420a50ee
3849674c2602319e 7178876e01f19b2a 9ca72dbcd502b0ba
43297fad38e373fe 4c974f1caa59f5d4 ea676b2cb7db2b7a
43297fad38e373fe 762514b829bf486a 1d3eba477e45f553
43297fad38e373fe ea676b2cb7db2b7a 547018835947a9b3
49793ebc79b3258f 437540c8698f3cfa 7ce143bcd1108178
49793ebc79b3258f 6fbf1cafcffd0556 315e66ab98b26ded
49793ebc79b3258f a0cb2871752053f0 6fbf1cafcffd0556
49e95d6d4ca229bf 00b0024eaac70ae3 5a6b612cc26cce4a
49e95d6d4ca229bf 02fe55778117f12a 85028e1e6b858ec4
49e95d6d4ca229bf 5a6b612cc26cce4a 1b1789391a840fea
4fb05e1515ab73a7 072d43a077075292 cfeeb0bdeb67fef4
4fb05e1515ab73a7 2f22e49bab7ca1ac 237c628bb6892693
4fb05e1515ab73a7 ad87789bc00718c2 2f22e49bab7ca1ac
584023641aba6176 004bd6ef09176062 fcdebb14f3b2babb
584023641aba6176 88bf0db6d70dee56 d09730ee2f16f0d1
584023641aba6176 c7d2845ea6d01c70 88bf0db6d70dee56
5b5a57676a56676e 974affbf86022d1f 429941c32d5126e8
7ca110454a1a6e57 01a1d6d039776742 098c16c1e1f4a681
7ca110454a1a6e57 690f5b0d9a26939b c10935a58a249fbb
7ca110454a1a6e57 ecd1c2f929f33ced 690f5b0d9a26939b
e0fee0fef1fef1fe 0123456789abcdef e29d11faf2d133ce
e0fee0fef1fef1fe b7687facf9b1a656 edbfd1c66c29ccc7
e0fee0fef1fef1fe edbfd1c66c29ccc7 c0f43c83cce0e0df
fedcba9876543210 0123456789abcdef cfee5c8eb79153d9
fedcba9876543210 2a2bb008df97c2f2 63f983b15c642726
fedcba9876543210 7f46aa73f7fcde02 2a2bb008df97c2f2
fedcba9876543210 c44c1b3668a0f2cf ed39d950fa74bcc4
fedcba9876543210 ed39d950fa74bcc4 7420eb3d2a7aa178
fedcba9876543210 ffffffffffffffff fce524b60ec5b3fc
ffffffffffffffff 0000000000000000 0000000000000000
ffffffffffffffff 16b15028f06a5ab8 caaaaf4deaf1dbae
ffffffffffffffff 3c7188775253884d 7359b2163e4edc58
ffffffffffffffff 7359b2163e4edc58 3c7188775253884d
ffffffffffffffff caaaaf4deaf1dbae 16b15028f06a5ab8
ffffffffffffffff ffffffffffffffff 61f38c55061e3161

3. Program Listings

```
/*
 * loki.h - specifies the interface to the LOKI encryption library.
 *     This library provides routines to expand a key, encrypt and
 *     decrypt 64-bit data blocks.  The LOKI Data Encryption Algorithm
 *     is a block cipher which ensures that its output is a complex
 *     function of its input and the key.
 *
 *
 * Author: Lawrence Brown <lpb@cs.adfa.oz.au>      Aug 1989
 *     Computer Science, UC UNSW, Australian Defence Force Academy,
 *         Canberra, ACT 2600, Australia.
 *
 * Version:
 *     v1.0 - of loki64.o is current 7/89
 *
 * Copyright 1989 by Lawrence Brown and CCCSR Uni NSW. All rights reserved.
 *     This program may not be sold or used as inducement to buy a
 *     product without the written permission of the author.
 *
 * Description:
 * The routines provided by the library are:
 *
 * lokikey(key)        - expands a key into subkey, for use in
 *   char  key[8];           encryption and decryption operations.
 *
 * enloki(b)           - main LOKI encryption routine, this routine
 *   char  b[8];             encrypts one 64-bit block b with subkey
 *
 * deloki(b)           - main LOKI decryption routine, this routine
 *   char  b[8];             decrypts one 64-bit block b with subkey
 *
 * The 64-bit data & key blocks used in the algorithm are specified as eight
 *     unsigned chars. For the purposes of implementing the LOKI algorithm,
 *     the bits are numbered as follows:
 *         [63..56] [55..48]       ...       [7..0]
 *     in  b[0]     b[1]  b[2]  b[3]  b[4]  b[5]  b[6]  b[7]
 *
 */

#define LOKIBLK 8       /* No of bytes in a LOKI data-block */
```

```
#define ROUNDS  16      /* No of LOKI rounds */

typedef unsigned long   Long;   /* type specification for aligned LOKI blocks */

extern Long lokikey[2];     /* 64−bit key used by LOKI routines */
extern char *loki_lib_ver;  /* String with version no. & copyright */

#ifdef ansi         /* declare prototypes for library functions  */
extern void enloki(char b[LOKIBLK]);
extern void deloki(char b[LOKIBLK]);
extern void setlokikey(char key[LOKIBLK]);
#else              /* else just declare library functions extern */
extern void enloki(), deloki(), setlokikey();
#endif ansi
```

```
/*
 *
 * lokimode.h - define macros for each of the common modes of operation
 * of the LOKI encryption primitive.
 *     Details of the modes may be found in:
 *         ANSI Standard X3.106-1983  DEA - Modes of Operation
 *
 * Author: Lawrence Brown <lpb@cs.adfa.oz.au>      Aug 1989
 *     Computer Science, UC UNSW, Australian Defence Force Academy,
 *         Canberra, ACT 2600, Australia.
 *
 * Copyright 1989 by Lawrence Brown and CCCSR Uni NSW. All rights reserved.
 *     This program may not be sold or used as inducement to buy a
 *     product without the written permission of the author.
 *
 */

/*
 * define valid flag values for the required mode and operation
 */
#define OP_EN   0x01   /* op is encrypting */
#define OP_DE   0x02   /* op is decrypting */

#define M_ECB   0x10   /* mode is ECB */
#define M_CBC   0x20   /* mode is CBC */
#define M_CFB1  0x30   /* mode is CFB1 with  8-bit feedback */
#define M_CFB8  0x40   /* mode is CFB8 with 64-bit feedback */
#define M_OFB8  0x50   /* mode is OFB8 with 64-bit feedback */
#define M_SBH   0x60   /* mode is SBH single block (64-bit) hash */
#define M_DBH   0x70   /* mode is DBH double block (128-bit) hash */

extern Long dea_IV[2];     /* current 64-bit IV vector */
extern void setIV();       /* routine to set IV vector */

/*
 * define macros for each of the common modes
 *     nb: input block must be type "Long b[2]"
 */

#define en_ecb(b) enloki((char *)b) /* Encrypt in ECB mode */
```

```
#define de_ecb(b) deloki((char *)b) /* Decrypt in ECB mode */

#define en_cbc(b) {        /* Encrypt in CBC mode */\
   ((Long *)b)[0] ^= dea_IV[0];    /* b = b XOR IV */\
   ((Long *)b)[1] ^= dea_IV[1]; \
   enloki((char *)b);  /* encrypt data block */\
   dea_IV[0] = ((Long *)b)[0]; /* save b as new IV value */\
   dea_IV[1] = ((Long *)b)[1]; \
   }

#define de_cbc(b) {        /* Decrypt in CBC mode */\
   register Long   nextiv0, nextiv1; /* to save each block as next iv */\
   nextiv0 = ((Long *)b)[0];    /* save current ciphertext for next IV */\
   nextiv1 = ((Long *)b)[1]; \
   deloki((char *)b);  /* decrypt data block */\
   ((Long *)b)[0] ^= dea_IV[0];    /* b = b XOR IV */\
   ((Long *)b)[1] ^= dea_IV[1]; \
   dea_IV[0] = nextiv0;    /* & move next IV to current */\
   dea_IV[1] = nextiv1; \
   }

#define en_cfb1(c) {          /* Encrypt char in 8-bit CFB mode */\
   Long    work[2];    /* work store for block to encrypt */\
   work[0] = dea_IV[0]; work[1] = dea_IV[1]; \
   enloki((char *)work);   /* encrypt data block */\
   c ^= (work[0] >> 24);   /* c = c XOR MSB(LOKI(IV)) */\
   dea_IV[0] = (dea_IV[0] << 8) | (dea_IV[1] >> 24); /* IV = (IV<<8)|c */\
   dea_IV[1] = (dea_IV[1] << 8) | c; \
   }

#define en_cfb8(b) {          /* Encrypt block in 64-bit CFB mode */\
   enloki((char *)dea_IV); /* encrypt data block */\
   ((Long *)b)[0] ^= dea_IV[0];    /* b = b XOR IV */\
   ((Long *)b)[1] ^= dea_IV[1]; \
   dea_IV[0] = ((Long *)b)[0];  \
   dea_IV[1] = ((Long *)b)[0];  \
   }

#define en_ofb8(b) {          /* Encrypt block in 64-bit OFB mode */\
   enloki((char *)dea_IV); /* encrypt data block */\
   ((Long *)b)[0] ^= dea_IV[0];    /* b = b XOR IV */\
   ((Long *)b)[1] ^= dea_IV[1]; \
   }
```

```
#define en_sbh(M) {     /* Compute a 64–bit MAC using SBH mode */ \
   Long    work[2];    /* work store for block to encrypt */ \
   work[0] = ((Long *)M)[0] ^ dea_IV[0];   /* work is M XOR H */ \
   work[1] = ((Long *)M)[1] ^ dea_IV[1]; \
   setlokikey((char *)work);       /* set key to be M XOR H */ \
   work[0] = dea_IV[0];    /* work is H */ \
   work[1] = dea_IV[1]; \
   enloki((char *)dea_IV); /* encrypt previous hash value */ \
   dea_IV[0] ^= work[0];   /* new H is dea_IV XOR H */ \
   dea_IV[1] ^= work[1]; \
   }
```

```
/*
 *    loki64.i – contains the fixed permutation and substitution tables
 *              for a 64 bit LOKI implementation
 *
 * Author:    Lawrence Brown <lpb@csadfa.oz>        Aug 1989
 *           Computer Science, UC UNSW, ADFA, Canberra, ACT 2600, Australia.
 *
 * Copyright 1989 by Lawrence Brown and CCCSR Uni NSW. All rights reserved.
 *    This program may not be sold or used as inducement to buy a
 *    product without the written permission of the author.
 */

/* 32–bit permutation function P */
char P[32] = {
      31, 23, 15, 7, 30, 22, 14, 6,
      29, 21, 13, 5, 28, 20, 12, 4,
      27, 19, 11, 3, 26, 18, 10, 2,
      25, 17, 9, 1, 24, 16, 8, 0
      };

/*
 *    sfn_desc – a desriptor table specifying which irreducible polynomial
 *           and exponent is to be used for each of the 16 S functions in
 *           the Loki S–box
 */
typedef struct {
      short  gen;          /* irreducible polynomial used in this field */
      short  exp;          /* exponent used to generate this s function */
      } sfn_desc;

/*
 *    sfn – the table specifying the irreducible polys & exponents used
 *           in the Loki S–boxes for the Loki algorithm
 *           (nb: the first 16 are used unless otherwise juggled)
 */
sfn_desc sfn[] = {
      { /* 101110111 */ 375, 31},
      { /* 101111011 */ 379, 31},
      { /* 110000111 */ 391, 31},
      { /* 110001011 */ 395, 31},
      { /* 110001101 */ 397, 31},
```

```
    { /* 110011111 */ 415, 31},
    { /* 110100011 */ 419, 31},
    { /* 110101001 */ 425, 31},
    { /* 110110001 */ 433, 31},
    { /* 110111101 */ 445, 31},
    { /* 111000011 */ 451, 31},
    { /* 111001111 */ 463, 31},
    { /* 111010111 */ 471, 31},
    { /* 111011101 */ 477, 31},
    { /* 111100111 */ 487, 31},
    { /* 111110011 */ 499, 31},
#ifdef ALLPOLYS      /* all other possible irreducible polys in GF(2^8) */
    { /* 111110101 */ 501, 31},
    { /* 111111001 */ 505, 31},
    { /* 100011011 */ 283, 31},
    { /* 100011101 */ 285, 31},
    { /* 100101011 */ 299, 31},
    { /* 100101101 */ 301, 31},
    { /* 100111001 */ 313, 31},
    { /* 100111111 */ 319, 31},
    { /* 101001101 */ 333, 31},
    { /* 101011111 */ 351, 31},
    { /* 101100011 */ 355, 31},
    { /* 101100101 */ 357, 31},
    { /* 101101001 */ 361, 31},
    { /* 101110001 */ 369, 31},
#endif
    { 00, 00}
    };
```

```
/*
 * loki64.c - library routines for a 64 bit LOKI implementation
 *
 * Author: Lawrence Brown <lpb@csadfa.oz>      Aug 1989
 *     Computer Science, UC UNSW, ADFA, Canberra, ACT 2600, Australia.
 *
 * Modifications:
 *     v1.0 - original set of code (based on loki64.c v4.0) 9/89 lpb
 *
 *
 * Copyright 1989 by Lawrence Brown and CCCSR Uni NSW. All rights reserved.
 *     This program may not be sold or used as inducement to buy a
 *     product without the written permission of the author.
 *
 * nb: if this program is compiled on a little-endian machine (eg Vax)
 *     #define LITTLE_ENDIAN
 *     in order to enable the byte swapping  routines
 *
 *     if a detailed trace of LOKI function f is required for debugging
 *     #define TRACE
 *
 *     these routines assume that the 8-byte char arrays used to pass
 *     the 64-bit blocks fall on a word boundary, so that the blocks
 *     may be read as longwords for efficiency reasons. If this is
 *     not true, the load & save of the parameters will need changing.
 */
#include <stdio.h>
#include "loki.h"   /* include Interface Specification header file       */
#include "loki64.i" /* include LOKI Permutation, Substitution & Key tables*/
```

```
/*
 *  lokikey – the 64–bit key for the LOKI library routines
 */
Long    lokikey[2] = {0, 0};

/*
 *  string specifying version and copyright message
 */
char    *loki_lib_ver = "64–bit LOKI library v1.0, Copyright (C) 1989
                                        Lawrence Brown & CCCSR Uni NSW";

Long    f();        /* declare LOKI function f          */
short   s();        /* declare LOKI s–box routine (for small ver) */
extern short    exp8();     /* exponentiation routine for GF(2^8) */


/*
 *  ROL12(b) – macro to rotate 32–bit block b left  by 12 bits
 *  ROR12(b) – macro to rotate 32–bit block b right by 12 bits
 */
#define ROL12(b) b = ((b << 12) | (b >> 20));
#define ROR12(b) b = ((b >> 12) | (b << 20));

/*
 *      bswap(b) – exchanged bytes in each longword of a 64–bit data block
 *                 on little–endian machines where byte order is reversed
 */
#ifdef  LITTLE_ENDIAN
#define bswap(cb) {                     \
        register char   c;              \
        c = cb[0]; cb[0] = cb[3]; cb[3] = c;    \
        c = cb[1]; cb[1] = cb[2]; cb[2] = c;    \
        c = cb[4]; cb[4] = cb[7]; cb[7] = c;    \
        c = cb[5]; cb[5] = cb[6]; cb[6] = c;    \
}
#endif
```

```
/*
 *  setlokikey(key) – save 64–bit key for use in encryptions & decryptions
 */

void
setlokikey(key)                                                        setlokikey
char    key[LOKIBLK];       /* Key to use, stored as an array of Longs    */
{
    lokikey[0] = ((Long *)key)[0];
    lokikey[1] = ((Long *)key)[1];

#ifdef LITTLE_ENDIAN
    bswap(lokikey);         /* swap bytes round if little–endian */
#endif

#if TRACE >= 1
    fprintf(stderr,"  keyinit(%08lx, %08lx)\n", lokikey[0], lokikey[1]);
#endif
}
```

```
/*
 *  enloki(b) – main LOKI encryption routine, this routine encrypts one
 *      64–bit block b using the LOKI algorithm with lokikey
 *
 *      The encryption operation involves XOR´ing the input block with
 *      the key, applying a LOKI round sixteen times
 *      (which ensures the output is a complex function of the input,
 *      and the key), and finally XOR´ing the input block with the key
 *
 *      Each round performs the following calculation using a 48–bit
 *      subkeys:
 *          L(i) = R(i–1)
 *          R(i) = L(i–1) XOR f(R(i–1), K(i))
 *
 *      To save swapping, alternate calls to f use L & R respectively
 *
 *      nb: The 64–bit block is passed as two longwords. For the
 *          purposes of the LOKI algorithm, the bits are numbered:
 *              [63 62 .. 33 32] [31 30 ... 1 0]
 *          The L (left) half is b[0], the R (right) half is b[1]
 *
 */
void
enloki(b)                                                              enloki
char    b[LOKIBLK];
{
    register Long   L, R;           /* left & right data halves  */
    register Long   KL, KR;         /* left & right key  halves  */

#ifdef LITTLE_ENDIAN
    bswap(b);           /* swap bytes round if little–endian */
#endif

#if TRACE >= 1
    fprintf(stderr,"  enloki(%08lx, %08lx)\n", ((Long *)b)[0], ((Long *)b)[1]);
#endif

    KL = lokikey[0];            /* load keys in register vars */
    KR = lokikey[1];

    L = ((Long *)b)[0] ^ KL;        /* LR = X XOR K */
    R = ((Long *)b)[1] ^ KR;
```

```
   /* Perform the 16 rounds using sub−keys in usual order              */
   L ^= f(R, KL);  ROL12(KL);
   R ^= f(L, KR);  ROL12(KR);
   L ^= f(R, KL);  ROL12(KL);
   R ^= f(L, KR);  ROL12(KR);
   L ^= f(R, KL);  ROL12(KL);
   R ^= f(L, KR);  ROL12(KR);
   L ^= f(R, KL);  ROL12(KL);
   R ^= f(L, KR);  ROL12(KR);
   L ^= f(R, KL);  ROL12(KL);
   R ^= f(L, KR);  ROL12(KR);
   L ^= f(R, KL);  ROL12(KL);
   R ^= f(L, KR);  ROL12(KR);
   L ^= f(R, KL);  ROL12(KL);
   R ^= f(L, KR);  ROL12(KR);
   L ^= f(R, KL);  ROL12(KL);
   R ^= f(L, KR);  ROL12(KR);

   ((Long *)b)[0] = R ^ KR;        /* Y = swap(LR) XOR K */
   ((Long *)b)[1] = L ^ KL;

#if TRACE >= 1
   fprintf(stderr,"  enloki returns %08lx, %08lx\n", ((Long *)b)[0], ((Long *)b)[1]);
#endif

#ifdef LITTLE_ENDIAN
   bswap(b);          /* swap bytes round if little−endian */
#endif
}
```

```
/*
 * deloki(b) – main LOKI decryption routine, this routine decrypts one
 *     64–bit block b using the LOKI algorithm with lokikey
 *
 *     Decryption uses the same algorithm as encryption, except that
 *     the subkeys are used in reverse order.
 */
void
deloki(b)                                                          deloki
char    b[LOKIBLK];
{
   register Long   L, R;          /* left & right data halves  */
   register Long   KL, KR;        /* left & right key  halves  */

#ifdef LITTLE_ENDIAN
   bswap(b);          /* swap bytes round if little–endian */
#endif

#if TRACE >= 1
   fprintf(stderr,"  deloki(%08lx, %08lx)\n", ((Long *)b)[0], ((Long *)b)[1]);
#endif

   KL = lokikey[0];           /* load keys in register vars */
   KR = lokikey[1];

   L = ((Long *)b)[0] ^ KR;       /* LR = X XOR K */
   R = ((Long *)b)[1] ^ KL;

   /* Perform the 16 rounds using sub–keys in usual order              */
   ROR12(KR);  L ^= f(R, KR);
   ROR12(KL);  R ^= f(L, KL);
   ROR12(KR);  L ^= f(R, KR);
   ROR12(KL);  R ^= f(L, KL);
   ROR12(KR);  L ^= f(R, KR);
   ROR12(KL);  R ^= f(L, KL);
   ROR12(KR);  L ^= f(R, KR);
   ROR12(KL);  R ^= f(L, KL);
   ROR12(KR);  L ^= f(R, KR);
   ROR12(KL);  R ^= f(L, KL);
   ROR12(KR);  L ^= f(R, KR);
   ROR12(KL);  R ^= f(L, KL);
   ROR12(KR);  L ^= f(R, KR);
   ROR12(KL);  R ^= f(L, KL);
   ROR12(KR);  L ^= f(R, KR);
```

```
    ROR12(KL);  R ^= f(L, KL);

    ((Long *)b)[0] = R ^ KL;        /* Y = LR XOR K */
    ((Long *)b)[1] = L ^ KR;

#if TRACE >= 1
    fprintf(stderr,"  deloki returns %08lx, %08lx\n", ((Long *)b)[0], ((Long *)b)[1]);
#endif

#ifdef LITTLE_ENDIAN
    bswap(b);           /* swap bytes round if little-endian */
#endif
}
```

```
/*
 * f(r, k) – is the complex non–linear LOKI function, whose output
 *     is a complex function of both input data and sub–key.
 *     The input data R(i–1) is:
 *         added modulo 2 to the key K(i)
 *         expanded to 48–bits via expansion fn E
 *         substituted into the S–boxes
 *         permuted by P.
 *     ie the calculation is:
 *         A = E(R(i–1) XOR K(i))      (a 48–bit value)
 *         B = S(A)            (a 32–bit value)
 *         f = P(B)            (a 32–bit value)
 *
 * nb: the 12–bit S–box input value [x x x x 11 10 9 ... 3 2 1 0]
 *     is interpreted as:
 *     bits [11 10 1 0] select 1 of 16 rows within each box,
 *     bits [9 8 ... 3 2] then select a column within that row
 */

#define MASK12  0x0fff          /* 12 bit mask for expansion E */

static Long
f(r, k)                                                              f
register Long   r;  /* Data value R(i–1) */
Long        k;  /* Key     K(i)   */
{
   Long    a, b, c;        /* 32 bit S–box output, & P output */

   a = r ^ k;              /* A = R(i–1) XOR K(i) */
   b = ((Long)s((a        & MASK12))      ) | /* B = S(E(R(i–1)) XOR K(i)) */
      ((Long)s(((a >>  8) & MASK12)) <<  8) |
      ((Long)s(((a >> 16) & MASK12)) << 16) |
      ((Long)s((((a >> 24) | (a << 8)) & MASK12)) << 24);
   perm32(&c, &b, P);      /* C = P(S( E(R(i–1)) XOR K(i))) */

#if TRACE >= 2  /* If Tracing, dump A, K(i), and f(R(i–1),K(i)) */
   fprintf(stderr,"  f(%08lx, %08lx) = P.S(%08lx) = P(%08lx) = %08lx\n",
      r, k, a, b, c);
#endif
   return(c);          /* f returns the result C */
}
```

```
/*
 *  s(i) – return S–box value for input i
 */
static short s(i)
register Long i;    /* return S–box value for input i */
{
    register short  r, c, v, t;

    r = ((i>>8) & 0xc) | (i & 0x3);     /* row value */
    c = (i>>2) & 0xff;                  /* column value */
    t = c ^ r;                          /* base value for Sfn */
    v = exp8(t, sfn[r].exp, sfn[r].gen); /* Sfn[r] = t ^ exp mod gen */
#if TRACE >= 2
    fprintf(stderr, "   s(%lx=[%d,%d]) = %x sup %d mod %d = %x\n",
            i, r, c, t, sfn[r].exp, sfn[r].gen, v);
#endif
    return(v);
}

/*
 *  perm32(out, in, perm) is the general permutation of a 32–bit input
 *      block to a 32–bit output block, under the control of a
 *      permutation array perm. Each element of perm specifies which
 *      input bit is to be permuted to the output bit with the same
 *      index as the array element.
 *
 *  nb: to set bits in the output word, as mask with a single 1 in it is
 *      used. On each step, the 1 is shifted into the next location
 */

#define MSB 0x80000000L     /* MSB of 32–bit word */

perm32(out, in , perm)                                              perm32
Long    *out;       /* Output 32–bit block to be permuted               */
Long    *in;        /* Input  32–bit block after permutation            */
char    perm[32];   /* Permutation array                                */
{
    Long    mask = MSB;         /* mask used to set bit in output    */
    register int    i, o, b;    /* input bit no, output bit no, value */
    register char   *p = perm;  /* ptr to permutation array   */
```

```
  *out = 0;              /* clear output block */
  for (o=0; o<32; o++) {  /* For each output bit position o */
    i =(int)*p++;        /* get input bit permuted to output o */
    b = (*in >> i) & 01; /* value of input bit i */
    if (b)               /* If the input bit i is set */
      *out |= mask;      /*  OR in mask to output i */
    mask >>= 1;          /* Shift mask to next bit     */
  }
}
```

Appendix C

Bibliography

[ASA85] ASA, "*Electronics Funds Transfer - Requirements for Interfaces, Part 5, Data Encryption Algorithm*," AS2805.5-1985, Standards Association of Australia, Sydney, Australia, 1985.

[AdS82] W. Adams and D. Shanks, "Strong Primality Tests That Are Not Sufficient," *Mathematics of Computation*, vol. 39, no. 159, pp. 255-300, |July| 1982.

[AdT90] C. Adams and S. Tavares, "Good S-boxes are easy to find," in *Advances in Cryptology - CRYPTO'89*, Lecture Notes in Computer Science, no. 435, G. Brassard (editor), pp. 612-615, |Springer Verlag|, Berlin, 1990.

[APR83] L. M. Adleman, C. Pomerance and R. S. Rumlely, "On Distinguishing Prime Numbers from Composite Numbers," *Annals of Mathematics*, vol. 117, pp. 173-206, 1983.

[AHU74] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, |Addison Wesley, Reading, Mass., 1974.

[AAA83] "*Information Interchange - Data Encryption Algorithm - Modes of Operation*," American National Standards Institute X3.106-1983, American National Standards Institute, New York, 1983.

[AnR82] D. Andelman and J. Reeds, "On the Cryptanalysis of Rotor Machines and Substitution-Permutation Networks," |*IEEE Trans. Inf. Theory*|, vol. IT-28, no. 4, pp. 578-584, |July| 1982.

[Aru85] A. A. Aruliah, "*Pascal Implementation of the RSA Algorithm*," NPL-DITC-66/85, National Physical Lab., Div. of Information Technology and Computing, Teddington (England), |September.| 1985.

[BeP82] H. Beker and F. Piper, *Cipher Systems: The Protection of Communications*, Northwood Books, London, 1982.

[Bek84] H. J. Beker, "A Survey of Encryption Algorithms," in |*Proc.*| *1984* |*Int.*| |*Conf.*| *on Secure Communications Systems*, pp. 14-19, IEE, London, 22-23 |February.| 1984.

[BKS79] S. Berkovits, J. Kowalchuk and B. Schanning, "Implementing Public Key Scheme," *IEEE Communications Magazine*, vol. 17, pp. 2-3, |May| 1979.

[Ber82] T. A. Berson, "Long Key Variants of DES," in *Advances in Cryptology* - |*Proc.*| *of Crypto 82*, D. Chaum, R. L. Rivest and A. T. Sherman (editors), pp. 311-313, Plenum Press, New York, |August.|

23-25, 1982.

[BiS90] E. Biham and A. Shamir, "*Differential Cryptanalysis of DES-like Cryptosystems*," Technical Report, Weizmann Institute of Science, Rehovot, Israel, 19 July 1990.

[Bla83] G. R. Blakley, "A Computer Algorithm for Calculating the Product AB Modulo M," *IEEE |Trans.| on Computers*, vol. C-32, no. 5, pp. 497-500, |May| 1983.

[Boe89] B. D. Boer, "Cryptanalysis of F.E.A.L," in *Advances in Cryptology - Eurocrypt'88*, Lecture Notes in Computer Science, no. 330, C. G. Gunther (editor), pp. 293-300, |Springer Verlag|, Berlin, 1989.

[BoB86] A. W. Bojanczyk and R. P. Brent, "A Systolic Algorithm for Extended GCD Computation," in *|Proc.| 9th Australian Computer Science |Conf.|*, pp. 129-137, ANU, Canberra, Australia, |January.| 29-31, 1986.

[Bon84] D. J. Bond, "Practical Primality Testing," in *|Proc.| of |Int.| |Conf.| on Secure Communication Systems*, pp. 50-53, IEE, 22-23 |February.| 1984.

[BGK77] D. K. Branstead, J. Gait and S. Katzke, "*Report of the Workshop on Cryptography in Support of Computer Security*," NBSIR 77-1291, National Bureau of Standards, |September.| 1977.

[Bra79] G. Brassard, "A Note on the Complexity of Cryptography," *|IEEE Trans. Inf. Theory|*, vol. IT-25, no. 2, pp. 232-233, |March.| 1979.

[Bre86] R. P. Brent, "Some Integer Factorization Algorithms using Elliptic Curves," in *|Proc.| 9th Australian Computer Science |Conf.|*, pp. 149-163, ANU, Canberra, Australia, |January.| 29-31, 1986.

[Bri82] E. F. Brickell, "A Fast Modular Multiplication Algorithm with Application to Two Key Cryptography," in *Advances in Cryptology - |Proc.| of Crypto 82*, D. Chaum, R. L. Rivest and A. T. Sherman (editors), pp. 51-60, Plenum Press, New York, |August.| 23-25, 1982.

[BMP87] E. F. Brickell, J. H. Moore and M. R. Purtill, *Advances in Cryptology - |Proc.| of CRYPTO'86*, Lecture Notes in Computer Science, no. 263, pp. 3-8, |Springer Verlag|, Berlin, 1987.

[Bri90] E. F. Brickell, "A survey of hardware implementations of RSA," in *Advances in Cryptology - CRYPTO'89*, Lecture Notes in Computer Science, no. 435, G. Brassard (editor), pp. 368-370, |Springer Verlag|, Berlin, 1990.

[BBB86] "*Advance Information on the Meteor VLSI Public Key Device*," TA10.1.2, British Telecom, Ipswich UK, |April.| 1986.

[Bro87] L. P. Brown, "Implementation of RSA ," in *Secure Data Communications Workshop - Digest of Papers*, J. Snare (editor), IEEE, Melbourne, Australia, 31 |July| 1987.

[BPS89] L. Brown, J. Pieprzyk and J. Seberry, "*LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications*," RIPE Submission Annex, CCSR, Canberra, Australia, |September.| 1989.

[Bro89] L. Brown, "A Proposed Design for an Extended DES," in *Computer Security in the Age of Information*, W. J. Caelli (editor), North-Holland, Amsterdam, 1989.

[Bro90a] L. Brown, "Technical Options for Implementing Pay-TV in an Australian Context," *Australian Telecommunications Review* , |November.| 1990.

[Bro90b] L. Brown, "SECLOG - A Secure Remote Login Frontend and Shell," *Australian Unix Users Group Newsletter*, vol. 11, no. 2, pp. 18-26, |April.| 1990.

[BrS90a] L. Brown and J. Seberry, "On the Design of Permutation P in DES Type Cryptosystems," in *Advances in Cryptology - Eurocrypt'89*, Lecture Notes in Computer Science, no. 434, J. J. Quisquater and J. Vanderwalle (editors), pp. 696-705, |Springer Verlag|, Berlin, 1990.

[BrS90b] L. Brown and J. Seberry, "Key Scheduling in DES Type Cryptosystems," in *Advances in Cryptology: Auscrypt '90*, Lecture Notes in Computer Science, no. 453, pp. 221-228, |Springer Verlag|, Berlin, 1990.

[Bro90] L. Brown, "Secure Remote Login - the SECLOG option," in *AUUG 90 Conference Proceedings*, pp. 309-320, Australian UNIX Systems Users Group, Sydney, NSW, Australia, |September.| 1990.

[Bur84] C. E. Burton, "RSA: A Public Key Cryptosystem, Parts 1 and 2," *Doctor Dobbs Journal*, vol. 9, no. 3,4, pp. 16-43, 32-59, March - April 1984.

[Chi84] H. R. Chivers, "A Practical Fast Exponentiation Algorithm for Public-Key," in *|Proc.| of |Int.| |Conf.| on Secure Communication Systems*, pp. 54-58, IEE, 22-23 |February.| 1984.

[CoL84] H. Cohen and H. W. Lenstra, "Primality Testing and Jacobi Sums," *Mathematics of Computation*, vol. 42, no. 165, pp. 297-330, |January.| 1984.

[Coo83] S. A. Cook, "An Overview of Computational Complexity," *|Comm. of the ACM|*, vol. 26, no. 6, pp. 401-408, |June| 1983.

[CoQ82] C. Couvreur and J. J. Quisquater, "An Introduction to Fast Generation of Large Prime Numbers," *Philips J. Research*, vol. 37, no. 5-6, pp. 231-264, Philips Research Labs, Brussels, Belgium, 1982.

[Cox58] D. R. Cox, *Planning of Experiments*, Wiley Series in Probability and Mathematical Statistics, |John Wiley & Sons, New York, 1958.

[Coy85] M. Coyne, "*An Implementation of a Public Key Cryptosystem*," Honours Thesis, Basser Department of Computer Science, University of Sydney, Australia, |November.| 13, 1985.

[Dav82] D. W. Davies, "Some Regular Properties of the Data Encryption Standard," in *Advances in Cryptology - |Proc.| of Crypto 82*, D. Chaum, R. L. Rivest and A. T. Sherman (editors), pp. 89-96, Plenum Press, New York, |August.| 23-25, 1982.

[Dav83] D. W. Davies, "Applying the RSA Digital Signature to Electronic Mail," |*IEEE Computers*|, vol. C-16, no. 2, pp. 55-62, |February.| 1983.

[DHS84] J. A. Davies, D. B. Holdridge and G. J. Simmons, "Status Report on Factoring (At the Sandia National Laboratories)," in *Advances In Cryptology: |Proc.| of Eurocrypt 84*, Lecture Notes in Computer Science, no. 209, T. Beth, N. Cot and I. Ingemarsson (editors), pp. 183-215, |Springer Verlag|, Berlin, |April.| 1984.

[DaP89] D. W. Davies and W. L. Price, *Security for Computer Networks*, John Wiley and Sons, New York, 1989. (2nd edn).

[DDF83] M. Davio, Y. Desmedt, M. Fosseprez, R. Govaerts, J. Hulsbosch, P. Neutjens, P. Piret, J. Quisquater, J. Vanderwalle and P. Wouters, "Analytical Characteristics of the DES," in *Advances in Cryptology - |Proc.| of Crypto 83*, D. Chaum, R. L. Rivest and A. T. Sherman (editors), pp. 171-202, Plenum Press, New York, |August.| 22-24, 1983.

[Dem89] N. Demytko, "Generating Multiprecision Integers with Guaranteed Primality," in *Computer Security in the Age of Information*, W. J. Caelli (editor), North-Holland, Amsterdam, 1989.

[DeK74] J. Denes and A. D. Keedwell, *Latin Squares and their Applications*, English Universities Press Limited, London UK, 1974.

[Den82] D. E. Denning, *Cryptography and Data Security*, Addison-Wesley, 1982.

[Den83] D. E. Denning, "Protecting Public Keys and Signature Keys," |*IEEE Computers*|, vol. C-16, no. 2, pp. 27-35, |February.| 1983.

[Den84] D. E. Denning, "Digital Signatures with RSA and Other Public-Key Cryptosystems," |*Comm. of the ACM*|, vol. 27, no. 4, pp. 388-392, |April.| 1984.

[Des84] Y. Desmedt, "*Analysis of the Security and New Algorithms for Modern Industrial Cryptography*," PhD Disertation, Department Elektrorechniek, Faculteit der Toegepaste Wetenschappen, Katholieke Universiteit, Leuven, |October.| 1984.

[DiH76] W. Diffie and M. E. Hellman, "New Directions In Cryptography," *IEEE |Trans.| on Information Theory*, vol. IT-22, no. 6, pp. 644-654, |November.| 1976.

[DiH77] W. Diffie and M. E. Hellman, "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," *Computer*, vol. 10, no. 6, pp. 74-84, |June| 1977.

[DiH79] W. Diffie and M. E. Hellman, "Privacy and Authentication: An Introduction to Cryptography," *Proc IEEE*, vol. 67, no. 3, pp. 397-427, |March.| 1979.

[Dus91] S. Dusse, "A Cryptographic Library for the Motorola DSP 56000," in *Advances in Cryptology - Eurocrypt'90*, Lecture Notes in Computer Science, no. 473, I. B. Damgard (editor), pp. 230-244, |Springer Verlag|, Berlin, 1991.

[ElG85] T. ElGamal, "A Public Key Cryptosystem and a Signature System Based in Discrete Logarithms," *IEEE |Trans.| on Information Theory*, vol. IT-31, no. 4, pp. 469-472, |July| 1985.

[Fam82] B. W. Fam, "A Parallel/Pipeline Processor For Fast Exponentiation," *|Proc.| 1982 |Int.| |Conf.| on Parallel Processing* , pp. 316-318, 1982.

[Fei73] H. Feistel, "Cryptography and Computer Privacy," *Scientific American*, vol. 228, no. 5, pp. 15-23, |May| 1973.

[FNS75] H. Feistel, W. A. Notz and J. L. Smith, "Some Cryptographic Techniques for Machine-to-Machine Data Communications," *|Proc.| IEEE*, vol. 63, no. 11, pp. 1545-1554, |November.| 1975.

[Gai79] J. Gait, "Can the Mitre Public Key System Be Short-Cycled?," *IEEE Communications Magazine*, vol. 17, pp. 2-3, |November.| 1979.

[Gir71] M. B. Girsdansky, "Data Privacy - Cryptology and the Computer at IBM Research," *IBM Research Reports*, vol. 7, no. 4, pp. 1-12, IBM Research, Yorktown Heights, New York, 1971.

[Gol82] S. Golomb, *Shift Register Sequences*, Aegean Park Press, 1982.

[Gor84] J. Gordon, "Strong RSA keys," *Electronics Letters*, vol. 20, no. 12, pp. 514-516, |June| 1984.

[GrT78] E. K. Grossman and B. Tuckerman, "Analysis of a Weakened Feistel-Like Cipher," in *|Proc.| 1978 IEEE |Conf.| On Communications*, pp. 46.3.1-5, IEEE, 1978.

[Hel79a] M. E. Hellman, "DES will be totally insecure within ten years," *SPECTRUM*, vol. 16, no. 7, pp. 31-41, |July| 1979. With rebuttals from George I. Davida, Walter Tuchman, and Dennis Branstad.

[Hel79b] M. E. Hellman, "The Mathematics of Public-Key Cryptography," *Scientific American*, vol. 241, pp. 146-157, 1979.

[Hen81] P. S. Henry, "Fast Decryption Algorithm for the Knapsack Cryptographic System," *Bell System Technical Journal*, vol. 60, no. 5, pp. 767-773, Bell Labs, May-June 1981.

[Her78] T. E. Herlestam, "Critical Remarks on Some Public Key Cryptosystems," *Bit*, vol. 18, no. 4, pp. 493-496, 1978.

[HeJ81] T. Herlestam and R. Johannesson, "On Computing Logarithms Over GF(2^p), or an Attempt to Swindle Mitre Corporation," in *1981 IEEE |Int.| Symposium on Information Theory - Abstracts of Papers* , p. 47, 1981.

[Her80] J. E. Hershey, "Implementation of Mitre Public Key Cryptographic System," *Electronics Letters*, vol. 16, no. 24, pp. 930-931, |November.| 1980.

[HGD84] F. Hoornaert, J. Goubert and Y. Desmedt, "Efficient Hardware Implementation of the DES," in *Advances in Cryptology: |Proc.| of Crypto 84*, Lecture Notes in Computer Science, no. 196, G. R. Blakley and D. Chaum (editors), pp. 147-173, |Springer Verlag|, Berlin, |August.| 1984.

[HDV89] F. Hoornaert, M. Decroos, J. Vanderwalle and R. Govaerts, "Fast RSA Hardware: Dream or Reality?," in *Advances in Cryptology - Eurocrypt'88*, Lecture Notes in Computer Science, no. 330, C. G. Gunther (editor), pp. 257-266, |Springer Verlag|, Berlin, 1989.

[Ins81] "*Data Encryption Algorithm*," American National Standards Institute X3.92-1981, American National Standards Institute, New York, 1981.

[JMN84] A. M. Jackson, N. A. McEvoy and B. B. Newman, *The Project Universe Experiment*, pp. 14-19, IEE, London, 22-23 |February.| 1984.

[Jan87] A. Janiak, "*EDSP-247 Encryption Chipset*," Product Info, Kencomp - A Division of Janiak Holdings Pty. Ltd., 2 Regina Street, Kilsyth, Vic 3137, Australia, 1987.

[KaD79] J. B. Kam and G. I. Davida, "Structured Design of Substitution-Permutation Encryption Networks," *|IEEE Trans. on Computers|*, vol. C-28, no. 10, pp. 747-753, |October.| 1979.

[KBS86a] D. Khoo, D. J. Bird and J. Seberry, "*Encryption exponents 3 and the security of RSA*," Tech. Rep. 275, Basser Dept. of Computer Science, Uni. of Sydney, 1986.

[KBS86b] D. Khoo, D. J. Bird and J. Seberry, "Encryption Exponent 3 and the Security of RSA," *Eurocrypt 86 - Abstracts of Papers* , Linkoping, Sweden, 20-22 May 1986. also Journal Cryptology (to appear).

[Knu81] D. E. Knuth, *Seminumerical Algorithms*, The Art of Computer Programming, no. 2, |Addison Wesley, London, 1981.

[Koh78] L. M. Kohnfelder, "On the Signature Reblocking Problem in Public-Key Cryptosystems," *|Comm. of the ACM|*, vol. 21, no. 2, p. 179, |February.| 1978.

[Kon86] A. G. Konheim, *Cryptography*, Seminars of Excellence, ORSYS Institute, Amsterdam, |June| 9-11, 1986. notes from a seminar series.

[KrR82a] D. W. Kravitz and I. S. Reed, "Extension of RSA Crypto-Structure: A Galois Approach," *Electronics Letters*, vol. 18, no. 6, pp. 255-256, |March.| 1982.

[KrR82b] D. W. Kravitz and I. S. Reed, "Comment: on Extension of RSA Cryptostructure: A Galois Approach," *Electronics Letters*, vol. 18, no. 13, pp. 582-583, |June| 1982.

[Lem79] A. Lempel, "Cryptology In Transition," |*Computing Surveys*|, vol. 11, no. 4, pp. 285-303, |December.| 1979.

[Llo90] S. Lloyd, "Counting Functions Satisfying a Higher Order Strict Avalanche Criterion," in *Advances in Cryptology - Eurocrypt'89*, Lecture Notes in Computer Science, no. 434, J. J. Quisquater and J. Vanderwalle (editors), pp. 63-74, |Springer Verlag|, Berlin, 1990.

[MTI86] T. Matsumoto, Y. Takashima and H. Imai, "On Seeking Smart Public-Key-Distribution Systems," |*Trans.*| *IECE of Japan*, vol. E69, no. 2, pp. 99-106, IECE, |February.| 1986.

[Mau90] U. M. Maurer, "Fast generation of secure RSA-products with almost maximal diversity," in *Advances in Cryptology - Eurocrypt'89*, Lecture Notes in Computer Science, no. 434, J. J. Quisquater and J. Vanderwalle (editors), pp. 636-647, |Springer Verlag|, Berlin, 1990.

[Mer78] R. C. Merkle, "Secure Communications Over Insecure Channels," |*Comm. of the ACM*|, vol. 21, no. 4, pp. 294-299, |April.| 1978.

[Mer89] R. C. Merkle, "*A Software Encryption Function*," Technical Report, Xerox PARC, Pala Alto, CA, 1989. also released as USEnet news article in "sci.crypt" on 13 July 1989.

[Mey78] C. H. Meyer, "Ciphertext/plaintext and ciphertext/key dependence vs number of rounds for the data encryption standard," in *AFIPS* |*Conf.*| |*Proc.*| *47*, pp. 1119-1126, AFIPS Press, Montvale NJ, USA, |June| 1978.

[MeM82] C. H. Meyer and S. M. Matyas, *Cryptography: A New Dimension in Data Security*, |John Wiley & Sons, New York, 1982.

[Miy88] S. Miyaguchi, "FEAL-8 LSI chip with the speed of 12Mbyte/s," *Nikkei Electronics* , 8 |February.| 1988.

[Miy90] S. Miyaguchi, "The FEAL-8 cryptosystem and a call for attack," in *Advances in Cryptology - CRYPTO'89*, Lecture Notes in Computer Science, no. 435, G. Brassard (editor), pp. 624-627, |Springer Verlag|, Berlin, 1990.

[MoA85] S. B. Mohan and B. S. Adiga, "Fast Algorithms for Implementing RSA Public Key Cryptosystem," *Electronics Letters*, vol. 21, no. 17,

p. 761, |August.| 1985.

[Mon76] D. C. Montgomery, *Design and Analysis of Experiments*, |John Wiley & Sons, New York, 1976.

[MoT86] T. E. Moore and S. E. Tavares, "A Layered Approach to the Design of Private Key Crypto systems," in *Proceedings of Crypto 85*, Lecture Notes in Computer Science, no. 218, pp. 227-245, Springer-Verlag, 1986.

[MoS86] J. H. Moore and G. J. Simmons, "Cycle Structure of the Weak and Semi-Weak DES Keys," *Eurocrypt 86 - Abstracts of Papers* , p. 2.1, Linkoping, Sweden, 20-22 |May| 1986.

[MoS87] J. H. Moore and G. J. Simmons, *Advances in Cryptology - |Proc.| of CRYPTO'86*, Lecture Notes in Computer Science, no. 263, pp. 9-32, |Springer Verlag|, Berlin, 1987.

[Mor86] T. M. Morris-Yates, "*An nMOS VLSI Implementation of a Data Encryption Algorithm*," BEng Honours Thesis, School of Electrical Engineering, Uni. Sydney, |November.| 1986.

[Mor83] D. R. Morrison, "Subtractive Encryptors-Alternatives to the DES," |*SIGACT News*|, vol. 15, no. 1, pp. 67-77, SPRING 1983.

[Mul83] C. Muller-Schloer, "A Microprocessor-based Cryptoprocessor," *IEEE Micro* , pp. 5-15, |October.| 1983.

[MNW84] R. Mullin, E. Nemeth and N. Weidenhofer, "Will Public Key Cryptosystems Live up to Their Expectations?," in *|Proc.| 1984 |Int.| Joint |Conf.| on Parallel Processing*, pp. 193-196, 1984.

[NBS77] NBS, "*Data Encryption Standard (DES)*," FIPS PUB 46, US National Bureau of Standards, Washington, DC, |January.| 1977.

[Odl84] A. M. Odlyzko, "Discrete Logarithms in Finite Fields and their Cryptographic Significance," in *Advances In Cryptology: |Proc.| of Eurocrypt 84*, Lecture Notes in Computer Science, no. 209, T. Beth, N. Cot and I. Ingemarsson (editors), pp. 224-316, |Springer Verlag|, Berlin, |April.| 1984.

[Oor91] P. C. Oorschot, "A Comparison of Practical Public-Key Cryptosystems based on Integer Factorization and Discrete Logarithms," *Advances in Cryptology - Crypto'90*, vol. 537, pp. 576-581, |Springer Verlag|, Berlin, 1991.

[ORS87] G. A. Orton, M. P. Roy, P. A. Scott, L. E. Peppard and S. E. Tavares, *Advances in Cryptology - CRYPTO'86*, Lecture Notes in Computer Science, no. 263, pp. 277-301, |Springer Verlag|, Berlin, 1987.

[Out86] R. Outerbridge, "Some Design Criteria for Feistel-Cipher Key Schedules," *Cryptologia*, vol. 10, no. 3, pp. 142-156, |July| 1986.

[Out89] R. Outerbridge, "*A 128-bit Key Schedule for the "Data Encryption Standard" Algorithm*," (private communication), Toronto, Canada, |January.| 15, 1989.

[PiS89] J. Pieprzyk and J. Seberry, "*Remarks on Extension of DES - Which Way to Go?*," Tech. Rep. CS89/4, |Dept. of Computer Science, UC UNSW, Australian Defence Force Academy|, Canberra, Australia, |February.| 1989.

[PiF89] J. Pieprzyk and G. Finkelstein, "Permutations that Maximize Non-Linearity and Their Cryptographic Significance," in *Computer Security in the Age of Information*, W. J. Caelli (editor), North-Holland, Amsterdam, 1989.

[Pie89] J. Pieprzyk, "Error Propagation Property and Application in Cryptography," *IEE Proceedings-E, Computers and Digital Techniques*, vol. 136, no. 4, pp. 262-270, |July| 1989.

[Pie90] J. Pieprzyk, "Non-Linearity of Exponent Permutations," in *Advances in Cryptology - Eurocrypt'89*, Lecture Notes in Computer Science, no. 434, J. J. Quisquater and J. Vanderwalle (editors), pp. 80-92, |Springer Verlag|, Berlin, 1990.

[Pom81] C. Pomerance, "Recent Developments in Primality Testing," *The Mathematical Intelligencer*, vol. 3, no. 3, pp. 97-105, 1981.

[PoK79] G. J. Popek and C. S. Kline, "Encryption and Secure Computer Networks," |*Computing Surveys*|, vol. 11, no. 4, pp. 331-356, |December.| 1979.

[Pur83] G. B. Purdy, "A Carry-Free Algorithm for Finding the Greatest Common Divisor of Two Integers," *Computers and Mathematics with Applications*, vol. 9, no. 2, pp. 311-316, 1983.

[QuC82] J. J. Quisquater and C. Couvreur, "Fast Decipherment Algorithm for RSA Public-Key Cryptosystem," *Electronics Letters*, vol. 18, no. 21, pp. 905-907, |October.| 1982.

[QDD86] J. Quisquater, Y. Desmedt and M. Davio, "The Importance of Good Key Scheduling Schemes (How to make a Secure DES scheme with <= 48 bit keys)," in *Advances in Cryptology - Crypto 85*, Lecture Notes in Computer Science, no. 218, H. C. Williams (editor), pp. 537-542, |Springer Verlag|, Berlin, 1986.

[QuG90] J. Quisquater and M. Girault, "2n-Bit Hash Functions Using n-Bit Symmetric Block Cipher Algorithms," in *Advances in Cryptology - Eurocrypt'89*, Lecture Notes in Computer Science, no. 434, J. J. Quisquater and J. Vanderwalle (editors), pp. 102-109, |Springer Verlag|, Berlin, 1990.

[Rab79] M. O. Rabin, "*Digitalized Signatures and Public-Key Functions as Intractable as Factorization*," MIT/LCS/Tech. Rep.-212, MIT Lab.

Computer Science,, Cambridge, Mass., |January.| 1979.

[RSW82] R. F. Rieden, J. B. Snyder, R. J. Widman and W. J. Barnard, "*A Two-Chip Implementation of the RSA Public-Key Encryption Algorithm*," SAND82-0994, Sandia National Labs, Albuquerque, NM 87185, 1982.

[RSA78] R. L. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," |*Comm. of the ACM*|, vol. 21, no. 2, pp. 120-126, |February.| 1978.

[Riv78] R. L. Rivest, "Remarks on a Proposed Cryptanalytic Attack on the MIT Cryptosystem," *Cryptologia*, vol. 2, no. 1, pp. 62-65, |January.| 1978.

[Riv79] R. L. Rivest, "Critical Remarks on Critical Remarks on Some Public-Key Cryptosystems," *Bit*, vol. 19, no. 2, pp. 274-275, 1979.

[Riv80] R. L. Rivest, "A Description of a Single-Chip Implementation of the RSA cipher," *Lambda*, vol. 1, no. 3, pp. 14-18, 1980.

[Riv84] R. L. Rivest, "RSA Chips (Past/Present/Future)," in *Advances in Cryptology: |Proc.| of Eurocrypt 84*, Lecture Notes In Computer Science, no. 209, T. Beth (editor), pp. 159-165, |Springer Verlag|, Berlin, |April.| 1984.

[ScA83] G. C. Schaller and J. F. Arnold, "Microprocessor Based Implementation of the Public Key Encryption System," in *19th |Int.| Electronics Convention and Exhibition. Digest of Papers.*, pp. 300-302, IREE, 1983.

[Sch82] I. Schaumuller-Bichl, "Cryptanalysis of the Data Encryption Standard by the Method of Formal Coding," in *Cryptography: |Proc.| of the workshop on Cryptography (Eurocrypt 82)*, Lecture Notes in Computer Science, no. 149, T. Beth (editor), pp. 235-255, |Springer Verlag|, Berlin, |March.| 29 - |April.| 2, 1982.

[Sch84] M. R. Schroeder, *Number Theory in Science and Communication*, Springer-Verlag, 1984.

[Sch85] M. R. Schroeder, "Number theory and the Real World," *The Mathematical Intelligencer*, vol. 7, no. 4, pp. 18-26, 1985.

[SeP89] J. Seberry and J. Pieprzyk, *Cryptography: An Introduction to Computer Security*, |Prentice Hall, Englewood Cliffs, NJ|, 1989.

[SBC84] S. C. Serpell, C. B. Brookson and B. L. Clark, "A Prototype Encryption System Using Public Key," in *Advances in Cryptology: |Proc.| of Crypto 84*, Lecture Notes in Computer Science, no. 196, G. R. Blakley and D. Chaum (editors), pp. 3-9, |Springer Verlag|, Berlin, |August.| 1984.

[Sha86] A. Shamir, "On the Security of DES," in *Advances in Cryptology - Crypto 85*, Lecture Notes in Computer Science, no. 218, H. C. Williams (editor), pp. 280-281, |Springer Verlag|, Berlin, 1986.

[Sha49] C. E. Shannon, "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, vol. 28, no. 10, pp. 656-715, |October.| 1949.

[ShM88] A. Shimizu and S. Miyaguchi, "Fast Data Encipherment Algorithm FEAL," *Eurocrypt 87 Abstracts* , pp. 11-14, 1988.

[SiN77] G. J. Simmons and J. J. Norris, "Preliminary Remarks on the MIT Cryptosystem," *Cryptologia*, vol. 1, no. 4, pp. 406-414, 1977.

[Sim79a] G. J. Simmons, "Symmetric and Asymmetric Encryption," |*Computing Surveys*|, vol. 11, no. 4, pp. 305-330, |December.| 1979.

[Sim79b] G. J. Simmons, "Cryptology: The Mathematics of Secure Communication," *The Mathematical Intelligencer*, vol. 1, no. 4, pp. 233-246, 1979.

[Sim82] G. J. Simmons, ed., *Secure Communications and Asymmetric Cryptosystems*, AAAS Selected Symposium, no. 69, Westview Press, 1982.

[SNO72] J. L. Smith, W. A. Notz and P. R. Osseck, "An Experimental Application of Cryptography to a Remotely Accessed Data System," in |*Proc.*| *ACM* |*Conf.*|, p. 282 297, 1972.

[SoS77] R. Solovay and V. Strassen, "A Fast Monte Carlo Test for Primality," *Siam Journal on Computing*, vol. 6, no. 1, pp. 84-85, |March.| 1977.

[Sor84] A. Sorkin, "Lucifer, a Cryptographic Algorithm," *Cryptologia*, vol. 8, no. 1, pp. 22-41, |January.| 1984.

[VCF90] J. Vandewalle, D. Chaum, W. Fumy, C. Janssen, P. Landrock and G. Roelofsen, "A European Call for Cryptographic Algorithms: RIPE RACE Integrity Primitives Evaluation," in *Advances in Cryptology - Eurocrypt'89*, Lecture Notes in Computer Science, no. 434, J. J. Quisquater and J. Vanderwalle (editors), pp. 267-271, |Springer Verlag|, Berlin, 1990.

[WeT86] A. F. Webster and S. E. Tavares, "On the design of S-Boxes," in *Advances in Cryptology - Crypto 85*, Lecture Notes in Computer Science, no. 218, H. C. Williams (editor), pp. 523-534, |Springer Verlag|, Berlin, 1986.

[Wei90] M. J. Weiner, "Cryptanalysis of short RSA secret exponents," in *Advances in Cryptology - Eurocrypt'89*, Lecture Notes in Computer Science, no. 434, J. J. Quisquater and J. Vanderwalle (editors), p. 372, |Springer Verlag|, Berlin, 1990. full paper in IEEE-IT, IT-36(3), pp 553-558.

[WiS79] H. C. Williams and B. Schmid, "Some Remarks Concerning the MIT Public Key Cryptosystem," *Bit*, vol. 19, no. 4, pp. 525-538, 1979.

[Wil80] H. C. Williams, "A Modification of the RSA Public-Key Encryption Procedure," *IEEE* |*Trans.*| *on Information Theory*, vol. IT-26, no. 6, pp. 726-729, |November.| 1980.

[Wil82] H. C. Williams, "Computationally Hard Problems as a Source for Cryptosystems," in *Secure Communications and Asymmetric Cryptosystems*, AAAS Selected Symposia Series, no. 69, G. J. Simmons (editor), pp. 11-39, Westview Press, 1982.

[Wil84] H. C. Williams, "Some Public-Key Crypto-Functions as Intractable as Factorization," in *Advances in Cryptology: |Proc.| of Crypto 84*, Lecture Notes in Computer Science, no. 196, G. R. Blakley and D. Chaum (editors), pp. 66-70, |Springer Verlag|, Berlin, |August.| 1984.

[Wil86] H. C. Williams, "An M^3 Public-Key Encryption Scheme," in *Advances in Cryptology - Crypto 85*, Lecture Notes in Computer Science, no. 218, H. C. Williams (editor), pp. 358-368, |Springer Verlag|, Berlin, 1986.

[WiC81] R. Willoner and I. Chen, "An Algorithm for Modular Exponentiation," in *|Proc.| 5th Symposium on Computer Arithmetic, 1981.*, pp. 135-138, 1981.

[Win83] R. S. Winternitz, "Producing a One-Way Hash Function from DES," in *Advances in Cryptology - |Proc.| of Crypto 83*, D. Chaum, R. L. Rivest and A. T. Sherman (editors), pp. 203-207, Plenum Press, New York, |August.| 22-24, 1983.

[Yac91] Y. Yacobi, "Exponentiating faster with addition chains," in *Advances in Cryptology - Eurocrypt'90*, Lecture Notes in Computer Science, no. 473, I. B. Damgard (editor), pp. 222-229, |Springer Verlag|, Berlin, 1991.

[Yao82] A. C. Yao, "Theory and Applications of Trapdoor Functions," in *|Proc.| Annual Symposium On Foundations of Computer Science*, pp. 80-91, IEEE, 1982.

[YiP82] K. Yiu and K. Peterson, "A Single-Chip VLSI Implementation of the Discrete Exponential Public-Key Distribution System," in *Globecom '82. IEEE Global Telecommunications |Conf.|, Vol. 1*, pp. 173-179, 1982.

[Zim86] P. Zimmermann, "A Proposed Standard Format for RSA Cryptosystems," *Computer*, vol. 19, no. 9, pp. 21-34, |September.| 1986.

Analysis of the DES
and the Design of the LOKI Encryption Scheme

unicrest here

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY COLLEGE, UNIVERSITY OF NEW SOUTH WALES
AUSTRALIAN DEFENCE FORCE ACADEMY
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Lawrence Peter Brown
1991

Certificate of Originality

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of a university or other institute of higher learning, except where due acknowledgement is made in the text.

I also hereby declare that this thesis is written in accordance with the University's Policy with respect to the use of Project Reports and Higher Degree Theses.

Lawrence Peter Brown

Abstract

This thesis will analyse some existing encryption algorithms used to provide secrecy in communications and data storage, and will detail the development of a new private key algorithm. It starts with a brief survey of modern encyption algorithms and their uses. It then takes a closer look at the RSA public key algorithm, with particular reference to its practical implementation. It shows that whilst RSA has some very desirable properties for use in public networks in that keys need not be exchanged previously, limitations in implementation speeds mean that RSA alone is not sufficient. Consequently a hybrid scheme is required which combines a public key scheme for authentication and key exchange, with a private key scheme for privacy. At present the DES algorithm is the sole certified private key scheme. It has been extensively analysed for possible weaknesses since its adoption in 1977. It is generally agreed that whilst the structure is sound, its key size of 56-bits has become too small with the improvements in computational speed on modern computers. Whilst the DES algorithm is public, its design criteria are classified. This thesis examines the design of the DES data encryption algorithm and related schemes, particularly with reference to the fundamental avalanche and completeness properties. From this is developed some possible design criteria for such schemes. Using these criteria, along with insights based on generic substitution-permutation cipher construction, suggestions and support from the authors colleagues, and with some results on substitution box design by Pieprzyk and Seberry, a new group of private key schemes, the LOKI family, is designed and presented as the major achievement of this thesis.

Foreword

To my supervisors Jennifer Seberry, Ah Chung Tsoi, and David Anderson, for their help, advice, support and encouragement both directly on the thesis material, and indirectly though their help in supporting this research environment.

To the current and former members of the Department of Computer Science, and particularly to Geoff Collins, George Gerrity, Andrzej Goscinski, Mike and Cathy Newberry, Josef Pieprzyk, Rei Safavi-Naini, and Chris Vance for their support, help and suggestions. Thankyou.

This work could not have been completed without the support of ARC grant A48830241, ATERB, and Telecom Australia contract 7027. To all of those organisations, thankyou for your encouragement of this research.

Contents

Index of Figures

Index of Tables