IMPLEMENTATION ANALYSIS OF BLOCK CIPHER COMPONENTS AND STRUCTURES

LU XIAO

Your file Votre référence
ISBN: 0-612-99046-X
Our file Notre référence
ISBN: 0-612-99046-X

IMPLEMENTATION ANALYSIS OF BLOCK CIPHER COMPONENTS AND STRUCTURES

by

© Lu Xiao

A thesis submitted to the
School of Graduate Studies
in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy

Faculty of Engineering and Applied Science

Memorial University of Newfoundland
December 2003

St. John's Newfoundland

Abstract

This thesis analyzes the implementation and performance characterization of symmetric key block ciphers. In particular, we study block ciphers which consist of Substitution-boxes (S-boxes) and Maximum Distance Separable (MDS) mappings. New mechanisms are proposed to evaluate the performance of block ciphers in terms of complexity and security for both hardware and software implementations. Configured with parameterized components, many cipher cases are derived from two cipher structures, a nested Substitution-Permutation Network (SPN) and a class of Feistel networks. In our study of each case, the hardware complexity and speed are evaluated by considering a gate network consisting of one- or two-input logic gates, which is suitable for an Application-Specific Integrated Circuit (ASIC) realization. The software complexity (in terms of both speed and memory requirements) is evaluated through table lookup implementations, which is a classical approach used for fast software implementations. The results of the complexity evaluation are verified with implementations using 0.18 μm and 0.35 μm CMOS technologies for hardware and C/C++ compilers for software. Cipher security, in the form of resistance to differential and linear attacks, is used to normalize the performance in the analysis. Because the discussed structures are similar to many existing ciphers such as the Advanced Encryption Standard (AES) and Camellia, this mechanism enables us

to study the efficiency of existing and new ciphers through a wide comparison of security, performance, and implementation methods.

In addition to differential and linear cryptanalysis, we also examine integral, eXtended Sparse Linearization (XSL), and power attacks that may be applied to block ciphers. The XSL attack is discussed with respect to its effectiveness on the various studied cipher structures. Finally, a simple power analysis attack is implemented on Camellia's key schedule in the circumstance where the processor leaks Hamming weight information and the influence of the attack on the design of key schedules is explored.

Acknowledgments

First of all, I am deeply indebted to my supervisor, Dr. Howard M. Heys, for his guidance, encouragement, and patience in every part of this work. It is my fortune to be his student and as a supervisor he has done everything he could to benefit my work.

I am very grateful to Dr. Ramachandran Venkatesan and Dr. Theodore S. Norvell, for being my supervisor committee members, giving time and support all along, and teaching me courses. I would also like to thank Dr. Paul Gillard and Dr. John Robinson for teaching me courses.

I am grateful for this study opportunity provided by the Faculty of Engineering and Applied Science. Particularly, thanks to Dr. M.R. Haddara, Dr. Ray Gosine, and Moya Crocker in the Associate Dean Office who have helped me a lot during my graduate studies. The financial support granted by the School of Graduate Studies is highly appreciated.

I am also grateful to my wonderful colleagues and friends in the Computer Engineering Research Laboratories for their help and the good times, particularly Reza Shahidi who arranges the lab activities so pleasantly. In addition, I would like to thank Nolan White in the Department of Computer Science for his help during the utilization of VLSI CAD tools.

An important thank you goes to my dear wife, Kai Zhang, for her sincere love and trust in this adventure.

Finally, I would like to thank my parents and sister in China, whose support and encouragement throughout my studies have always been of great help.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the rapid development of computer and communication networks, the exchange of information is becoming more and more important for newly emerging applications such as electronic commerce and online database inquiry. For these applications, privacy of customer information must be protected. Since most public networks are open to malicious attackers, network security arises as a promising and significant research subject, especially as the Internet and wireless communication become a considerable part of human life.

Encryption technology is the core part of network security. The security and efficiency of encryption algorithms influence the performance of protected data services directly. Various space-consuming applications (e.g., teleconferencing and video on demand) are being implemented in broadband data networks, occupying much more bandwidth than traditional tasks. Currently, significant effort is being devoted to the throughput increase of network equipment such as routers and switches. As a vital part of secure communications, the encryption technology must now meet higher speed requirements.

Since the Data Encryption Standard (DES) [1] was proposed, block ciphers have

been playing a very important role in data encryption services because of their advantages:

- Fast speeds in both software and hardware
- Short lengths of cipher keys
- Well studied cipher components and structures

When a block cipher is used for secure communication, the key used for the cipher is shared secretly by the sender and the receiver. The key is often exchanged using public key cryptography as suggested in IEEE Standard 1363 [2].

1.1 Motivation

In 1977, the U.S. National Bureau of Standards published DES as a recommended algorithm for symmetric key block encryption. Until recently, DES had dominated in many security services. However, the current hardware technology and distributed computing make brute-force exhaustive key search attacks faster and cheaper. As a result, the security of DES is increasingly inadequate. For example, given a pair of plaintext and ciphertext, the key used for DES encryption could be found by exhaustive search using dedicated cracking hardware within 56 hours in 1998 [3] or distributed computation through the Internet within 22 hours in 1999 [4]. With a 56-bit key, DES is not secure enough due to its small number of possible keys (i.e., 2^{56}).

The Advanced Encryption Standard (AES) has been developed by the U.S. National Institute of Standards and Technology (NIST), as a symmetric key block cipher solution to efficiently provide enough security through the use of a larger key.

In January 1997, NIST called for algorithms as possible candidates as symmetric key block ciphers. The candidates needed to support at least a block size of 128 bits and key sizes of 128, 192, and 256 bits. Fifteen candidates were publicly tested and evaluated. Rijndael [5] was finally selected as the AES in October 2000.

As a 3-year project, the New European Schemes for Signature, Integrity, and Encryption (NESSIE) initiative also launched open calls for algorithms in the field of symmetric key block ciphers in 2000. Like the AES project, NESSIE's main objective is to offer cryptographic primitives with a higher security and efficiency level than the existing primitives. The cipher Camellia [6] was included together with AES into the NESSIE portfolio of 128-bit block ciphers in February 2003 [7].

Both the AES and NESSIE projects generated a great amount of activity in the study of symmetric key block ciphers including algorithm designs, software implementations, hardware implementations, and security evaluations.

Despite the need for efficient cipher designs, there has been no effort to develop a general model for simultaneous evaluation of hardware performance and security. It is not unusual that before a block cipher design is finalized, most analysis work is focused on security and software speed. Hardware suitability is often overlooked at this phase since it takes much time and energy to investigate the implementation of a cipher in an Application-Specific Integrated Circuit (ASIC). Typically, only when a block cipher is well established do researchers undertake hardware analysis. For example, as one candidate for the Advanced Encryption Standard (AES), the cipher RC6 [8] had attracted much academic interest before its hardware performance was well studied and recognized to be poorer than other algorithms. As a result, many block ciphers are inherently software-oriented and their hardware implementations

are neither fast nor compact[1].

One of the main objectives of this work is to investigate block ciphers suitable for hardware implementation and seek methods to implement efficient and secure block ciphers in hardware. We begin with the study of the design, implementation, and hardware complexity of basic cipher components. Then, a mechanism is presented to analyze different configurations of block cipher structures. The analysis integrates the hardware complexity, efficiency, and security evaluation into several performance measures. This mechanism will be utilized to discover the best secure cryptographic configurations that are hardware-oriented. Our hardware implementation will be concentrated on Application Specific Integrated Circuit (ASIC) design in order to facilitate superior performance over other targeted hardware environments such as Field-Programmable Gate Arrays (FPGAs)[2].

Following the hardware analysis, similar performance characterization can also be applied to software implementations. As a fast development technique across platforms, the table lookup approach has been selected to implement block ciphers, for example in [5, 9, 10]. Thus, a software performance metric is defined to integrate the security provided by cipher structures and the efficiency evaluated from corresponding table lookup implementations. The performance measured by this metric helps us to study the software efficiency and security of cipher structures on the same basis.

In addition to performance characterization, this thesis considers other facets of

[1]DES is a notable exception and many operations used by DES are more oriented to hardware, e.g., bit permutations and small S-boxes.

[2]With increasing gate densities and speeds, FPGAs are also used in many cryptographic applications. Due to a large variety of architectures, however, it is difficult to perform a meaningful hardware characterization for general FPGA implementations.

block cipher study which may influence the security of cipher structures or implementation. Specifically, we examine the two recently introduced cryptanalysis techniques of eXtended Sparse Linearization (XSL) and simple power attacks.

We believe that this work enhances the association between engineering and cryptography, and makes a significant contribution to the implementation and performance analysis of potential cryptographic structures.

1.2 Thesis Outline

Chapter 2 provides the cryptography background which is relevant to this thesis. A short but self-contained introduction is given on Shannon's product cipher, cipher components, structures, and examples. Differential and linear cryptanalysis are developed as the most fundamental tools for security consideration. Another attack, integral cryptanalysis, is illustrated using AES as the targeted cipher. The cipher implementation techniques are also briefly introduced.

Chapter 3 discusses the design and hardware implementation analysis of basic cipher components including MDS mappings and invertible S-boxes. The proposed S-box model and MDS implementation methods are used for two efficient AES hardware designs.

Chapter 4 proposes many SPN and Feistel cipher cases with different configurations of parameterized S-boxes and MDS mappings. Several metrics are defined to integrate hardware complexity and security evaluations. The hardware performance characterization is then undertaken on the basis of these metrics.

Chapter 5 compares the software performance of cipher cases proposed in Chapter 4. The table lookup implementation is used for performance evaluation, while

other implementation methods are briefly discussed and compared.

Chapter 6 investigates the effectiveness of an XSL attack [11], which was proposed in 2002 but has not yet been proved to be practical. In this chapter, a straightforward method is presented to evaluate the susceptibility of a block cipher to this potential attack.

Chapter 7 investigates the simple power analysis applied to a block cipher key schedule. Specifically, we apply the attack to the key schedule of Camellia. It is shown that such an attack works well even with measurement errors when the processor running Camellia leaks the Hamming weight of intermediate data. A general susceptibility evaluation and possible countermeasures are also suggested.

Chapter 8 draws conclusions and identifies future research directions.

Chapter 2

Background of Cryptography

This chapter introduces the basic design concepts, typical examples, and security of block ciphers, which are relevant to the contents later in this thesis. To consider the security of a block cipher, three fundamental attacks on block ciphers are illustrated briefly. More details can be obtained from appropriate literature, such as [9, 12, 13, 14] which contain a complete exposé of cryptography.

2.1 Encryption and Cryptosystems

Encryption is the mapping from the original message, called the *plaintext*, to a random looking message, called the *ciphertext*. During the mapping, a specific data set, called the *key*, determines the relation between them. The key should be randomly-selected information that is hard to deduce. As the inverse of encryption, *decryption* restores the plaintext from the ciphertext with knowledge of the key. The keys used for encryption and decryption may be different. Without the knowledge of the decryption key, decryption should be infeasible.

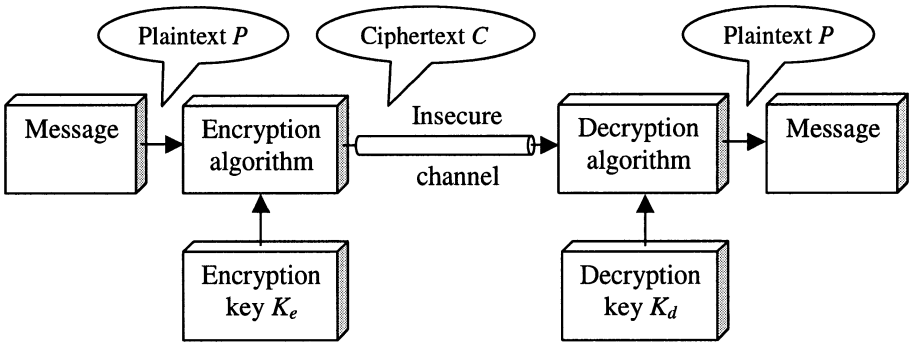


Figure 2.1: General Model of Cryptosystem

In a cryptosystem, encrypted information is transferred through an insecure channel as shown in Figure 2.1. In symmetric key ciphers, such as DES [1] , the encryption key K_e and the decryption key K_d are the same and should be kept secret; while in asymmetric key ciphers, such as RSA [15], one of the two keys is made publicly available and the other key cannot be derived from this public key through feasible computation.

Both symmetric and asymmetric key cryptography are widely used in data networks. Since one's public key can be used by others for encryption and authentication, asymmetric key cryptography (thus also called public key cryptography) is more suitable for network security applications such as key distribution and digital signatures. Although the cipher key needs to be securely distributed, symmetric key cryptography has two significant advantages:

- *High Encryption Speed*: The speed of a symmetric key cipher is much faster than an asymmetric key cipher. According to RSA Laboratories [16], DES is generally at least 100 times as fast in software and between 1,000 and 10,000 times as fast in hardware as RSA, depending on the implementation.

- *Short Key Length*: A symmetric key cipher can obtain enough security with a much shorter key. Based on current technology, a key of 128 bits is secure enough for recently proposed symmetric key ciphers. However, RSA requires a key size of at least 1,024 bits for current applications, which is not desirable for some bandwidth or memory restricted environments such as a smart card system.

The low performance of asymmetric key cryptography is due to the large arithmetic operations involved. For example, the RSA and Diffie-Hellman [17] algorithms calculate exponentials modulo a large prime and elliptic curve cryptography [18] multiplies two variables in large finite fields. These operations are inefficient in both software and hardware compared with the small components used in symmetric key ciphers. As a result, symmetric key cryptography is widely used in security applications which require high throughputs and/or small memory. Its most obvious application is confidentiality, which has a message encrypted so that the message can only be known by the sender and receiver.

2.2 Block Ciphers

According to the size of plaintext and ciphertext units, a symmetric key encryption algorithm can also be classified as a *block cipher* or a *stream cipher*. A block cipher encrypts the plaintext of a fixed bit length to the corresponding ciphertext of the same length. A block cipher with an n-bit block length is also called an n-bit block cipher. The cipher key is another block of bits with its own length. For security considerations, recently proposed block ciphers typically have a block size of 128 bits and a key size of 128 bits or more.

A stream cipher encrypts the plaintext in small units, usually bit by bit. The plaintext is combined with a bit sequence, called the *keystream*, to generate the ciphertext typically by bit-wise eXclusive-OR (XOR) operations. Stream ciphers can be designed to be very fast. Alternatively, a block cipher can be used as a stream cipher by selecting feedback modes (e.g., the Cipher Feedback and Output Feedback modes [14]), where the block cipher works as a keystream generator.

2.2.1 Product Ciphers

As introduced by C.E. Shannon in [19], the security of a block cipher can be generated by combining individual cipher steps appropriately into their "product". A product cipher usually iterates similar cipher operations for a certain number of *rounds*. The cipher key is expanded to a number of *subkeys* by a *key schedule* and the subkeys are mixed with data blocks in different rounds typically using bit-wise XOR operations.

In a product cipher, *diffusion* and *confusion* are two fundamental methods to frustrate statistical and mathematical attacks. The method of diffusion involves the dissipation of the redundancy that may be exploited by attackers into statistics across the entire block so that it is difficult for a meaningful recognition of patterns. The method of confusion is to complicate the mathematical relation between the ciphertext, plaintext, and key information so that the key is hard to derive even if plenty of ciphertexts are analyzed. The *Substitution-Permutation Network (SPN)*[1] and the *Feistel network* are two typical architectures used to achieve this [12]. Each cipher architecture is a well organized configuration of cipher *components*, which are simple cipher operations.

[1]For historical reasons, these ciphers are referred to as Substitution-Permutation Networks although the permutation layer is now typically replaced by an invertible linear transformation layer to improve resistance to differential and linear cryptanalysis [20].

2.2.2 Cipher Components

An $m \times n$ cipher component performs a simple mapping from an m-bit input to an n-bit output. A component is either linear or nonlinear. The most important nonlinear component is the *Substitution-box (S-box).* A typical linear component is a *linear transformation* involving the XOR of a subset of input bits to produce output bits.

Linear Transformations

A linear transformation enhances diffusion of a cipher. In a linear transformation from m-bit input $X = (x_{m-1}, \cdots, x_0)$ to n-bit output $Y = (y_{n-1}, \cdots, y_0)$, each output bit y_i can be expressed as an affine function of the input:

$$y_i = a_{i,m-1}x_{m-1} \oplus \cdots \oplus a_{i,0}x_0 \oplus b_i \tag{2.1}$$

where "$\oplus$" denotes an XOR operation and $a_{i,m-1}, \cdots, a_{i,0}, b_i$ are binary constants. As a result, the linear transformation can be expressed as

$$\mathcal{Y} = \mathcal{A}\mathcal{X} \oplus \mathcal{B} \tag{2.2}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are m-tuple and n-tuple representations of X and Y, respectively, $\mathcal{A}$ is an $m \times n$ binary matrix, and $\mathcal{B}$ is a binary m-tuple. Corresponding to the location of y_i in $\mathcal{Y}$, the i-th row of $\mathcal{A}$ consists of $a_{i,m-1}, \cdots, a_{i,0}$ and the i-th element of $\mathcal{B}$ is b_i.

The iterated structure of a purely linear transformation does not provide more security since the composition of multiple linear transformations is still linear. That

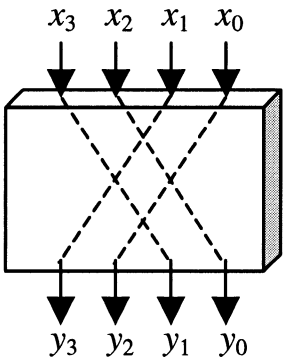


Figure 2.2: A 4×4 Bit Permutation

is,

$$\mathcal{A}_2(\mathcal{A}_1\mathcal{X} \oplus \mathcal{B}_1) \oplus \mathcal{B}_2 = \mathcal{A}_3\mathcal{X} \oplus \mathcal{B}_3$$

where $\mathcal{A}_3 = \mathcal{A}_2\mathcal{A}_1$ and $\mathcal{B}_3 = \mathcal{A}_2\mathcal{B}_1 \oplus \mathcal{B}_2$. However, the linear transformation is efficient and used to scramble the output bits of different S-boxes. By doing so, statistical relations among the plaintext, the ciphertext, and the key become complicated and difficult for attackers to analyze.

A *bit permutation* is a very simple linear transformation and used in many ciphers such as DES. For a bit permutation expressed in form of (2.2), $\mathcal{A}$ has only one nonzero element in each row and $\mathcal{B} = \mathbf{0}$. As illustrated in Figure 2.2, a bit permutation can be easily implemented by wiring input bits and output bits.

Recently, some techniques in coding theory have been absorbed into the design of linear transformations, e.g., the usage of Reed-Solomon codes in the cipher SHARK [21]. Thus, the input and output of a linear transformation are often expressed as vectors of symbols in finite fields [22]. To measure the avalanche effect of a linear transformation, the branch number of a linear transformation is defined [23]

as:

$$B = \min_{\mathcal{X} \neq 0} \{H(\mathcal{X}) + H(\mathcal{Y})\} \tag{2.3}$$

where $H(\mathcal{X})$ and $H(\mathcal{Y})$ denote the number of nonzero symbols in $\mathcal{X}$ and $\mathcal{Y}$, respectively. It is proved that a *Maximum Distance Separable (MDS)* [24] mapping has an optimal branch number B equal to $m + 1$, which is highly diffusive and effective in providing resistance to differential and linear attacks, as will be discussed in Section 2.3.

S-boxes

An S-box performs a nonlinear transformation in which the output bits cannot be expressed as affine functions. An S-box is *invertible* if a one-to-one mapping is performed. One important security measure of an S-box is nonlinearity, evaluated by the minimum Hamming distance from any linear combination of output bit functions to an affine function [20]. Figure 2.3 and Table 2.1 show an example 4×4 S-box taken from the first row of the first DES S-box. This S-box is obviously invertible and its nonlinearity can be shown to be 2. The permutation shown in Figure 2.2 cannot be used as an S-box because its nonlinearity is 0.

Table 2.1: Mapping Table of a 4×4 S-box (in hexadecimal)

$x_3x_2x_1x_0$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$y_3y_2y_1y_0$	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

Since S-boxes are the most typical components to provide confusion, many criteria and construction methods have been developed (e.g., in [20, 25, 26, 27, 28]). Different S-boxes in a cipher can have different mappings or one mapping can be used for all S-boxes in a cipher.

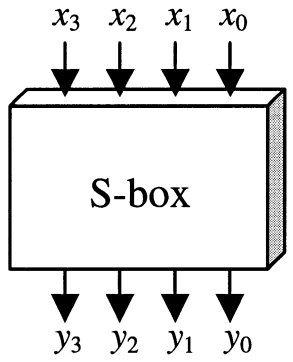


Figure 2.3: A 4×4 S-box

Other Components

Many other components are also used in block ciphers. They include addition, data dependent rotation, multiplication modulo 2^{32}, etc.. These components are not of direct relevance to structures considered in this thesis because they are not as widely studied and accepted as S-boxes and linear transformations in cipher design.

2.2.3 Cipher Structures

In this section, we consider the two best known structures for block ciphers.

Substitution Permutation Networks

During encryption using an SPN cipher, as Figure 2.4 illustrates, the input data of each round is typically mixed with subkey bits before entering the S-boxes. Each S-box performs a nonlinear mapping on small sub-blocks thus creating confusion in the data. The outputs of S-boxes are modified by a linear transformation whose purpose is to generate a diffusion of statistical effects in the data. The decryption is composed of the inverse linear transformations, the inverse S-boxes, and the key

mixtures in reverse order to encryption. To maintain similar dataflow in encryption and decryption, SPNs typically omit the linear transformation in the last round of encryption.

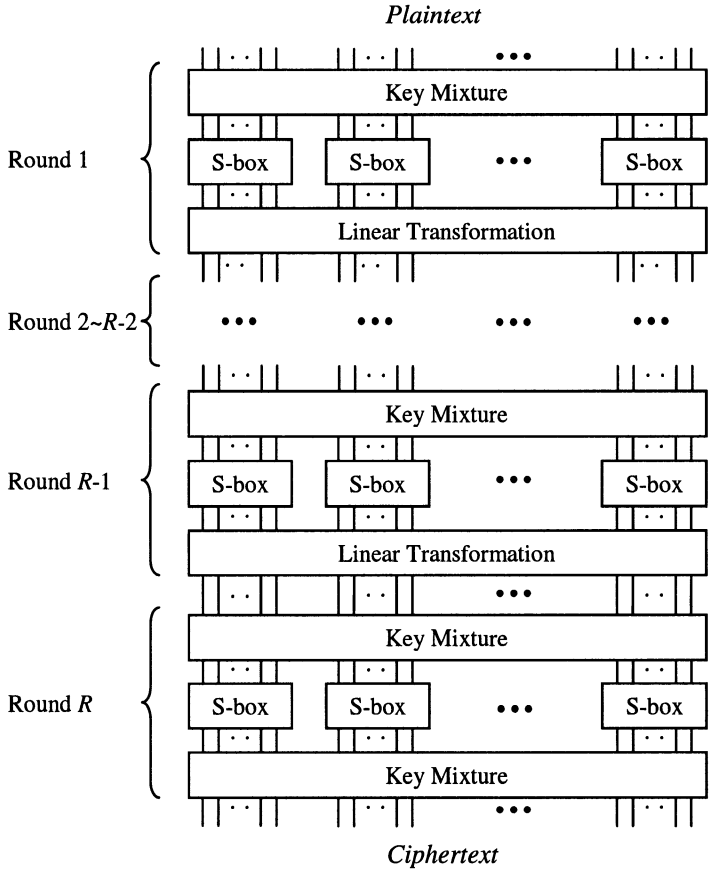


Figure 2.4: A Substitution-Permutation Network

For any input X, a function $f(X)$ is an *involution* if $f(f(X)) = X$ [29]. If the S-box layer and the linear transformation in Figure 2.3 are involutions, both the encryption and decryption operations can be performed by the same SPN except for small changes in the key schedule in the case of XOR key mixing. We refer to such a cipher as an involution SPN, of which the ciphers Anubis [30] and Khazad [31] are examples.

Feistel Networks

As the other typical architecture of block ciphers, the Feistel network has been widely used and studied. In each round i of a Feistel network as shown in Figure 2.5, the right half of the round input (denoted as X_i) goes through function F parameterized by subkey K_i. Also called the *round function*, F often consists of key mixture, S-boxes, and a linear transformation. The output of F, denoted as Y_i, is XORed with the left half of the round input. The round output is the swapped result of X_i and $X_{i-1} \oplus Y_i$.

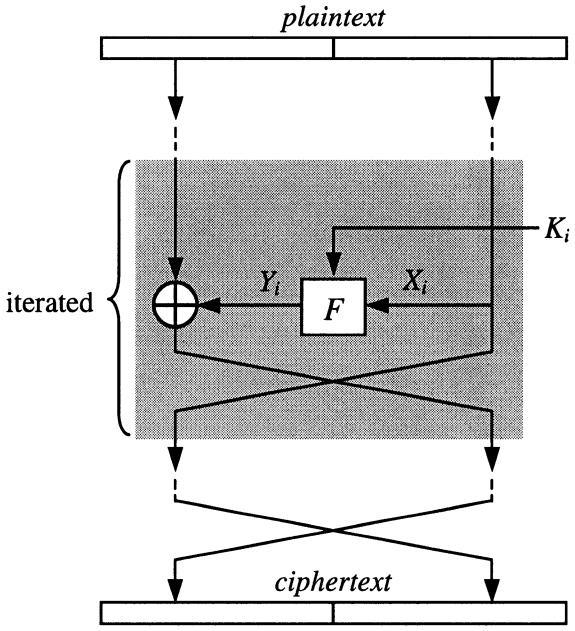


Figure 2.5: A Feistel Network

One advantage of a Feistel cipher is that, even if F is not invertible, the same cipher structure of Figure 2.5 can be used for both encryption and decryption with the appropriate modification to the key schedule.

2.2.4 Examples

Recent initiatives in cryptography have focussed on the development of new block cipher standards. As the successor of DES, the SPN cipher Rijndael [5] was selected by the U.S. National Institute of Standards and Technology as the Advanced Encryption Standard (AES) [32] in October 2000. As a Feistel network proposed in [6], Camellia was included together with AES into the NESSIE portfolio of 128-bit block ciphers in February 2003 [7]. Consequently, AES and Camellia are two important examples of ciphers that are expected to be widely used in cryptographic applications of the future.

Data Encryption Standard

Proposed in the 1970s, DES [1] is a Feistel cipher with a block size of 64 bits and a key size of 56 bits. DES conforms to the general Feistel structure illustrated in Figure 2.5 except for an initial permutation at the beginning and its inverse at the end of the cipher. After the plaintext passes through the initial permutation, the 64-bit permuted result splits to two 32-bit halves and enters a Feistel network of 16 rounds. In each round, the function F processes 32-bit X_i with subkey K_i of 48 bits. Within the function F as Figure 2.6 shows, a bit permutation expands X_i to 48 bits, which are then XORed with subkey K_i. The result after subkey mixture forms the inputs of 8 parallel 6×4 S-boxes (S_1 to S_8). The outputs of S-boxes are concatenated and pass through another bit permutation to form Y_i.

The key schedule of DES consists of two 28-bit rotating registers and two bit permutations PC1 and PC2. The key passes through PC1 to form the initial contents stored in the two registers. In each round, the two registers are rotated left independently for 1 or 2 bits. The rotated results are concatenated and pass through

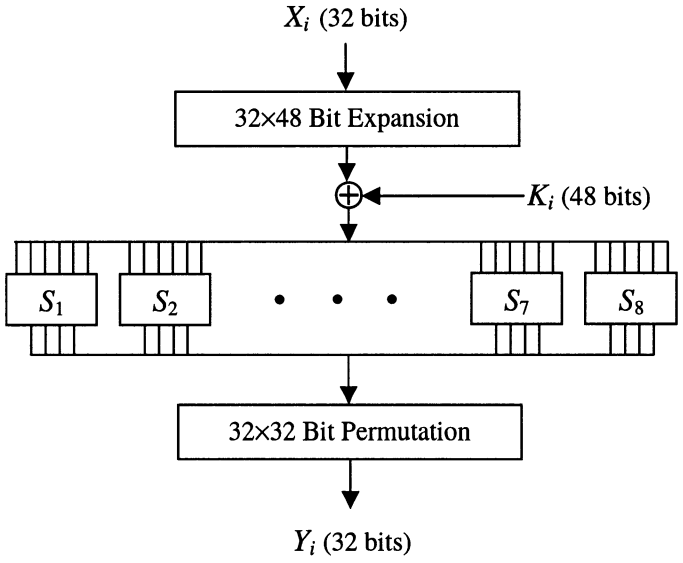


Figure 2.6: Function F of DES

PC2 to get one subkey K_i.

Advanced Encryption Standard

AES [32] is a 128-bit SPN cipher using keys with sizes of 128, 196, and 256 bits. With larger block and key sizes, AES is believed to be much more secure than DES. The number of rounds R depends on the key size, e.g., $R = 10$ when the key size is 128 bits.

At the beginning of the cipher, the 128-bit plaintext is stored in a two-dimensional array of bytes called the *State* and denoted by $\{\lambda_{i,j}\}$, $0 \leq i, j \leq 3$. There are four sequential steps in each round of the cipher. Each step takes data from the State as the input and stores the result in the State as the output. These four steps are [32]:

- *ByteSub*: This is a layer of parallel 8×8 S-boxes. Each byte enters an S-box independently. All AES S-boxes perform the same invertible mapping which consists of multiplicative inversion over GF(2^8) followed by an affine

transformation.

- *ShiftRow*: This is a byte-wise cyclic shifting in each row of the State. The shift offset of each round is fixed but different from one row to another. The updated State $\{\lambda'_{i,j}\}$ can be expressed as:

$$(\lambda'_{i,0}, \lambda'_{i,1}, \lambda'_{i,2}, \lambda'_{i,3}) = (\lambda_{i,i \bmod 4}, \lambda_{i,(i+1) \bmod 4}, \lambda_{i,(i+2) \bmod 4}, \lambda_{i,(i+3) \bmod 4}).$$

- *MixColumn*: Each column performs an MDS mapping, which can be implemented by matrix multiplication over GF(2^8):

$$\begin{pmatrix} \lambda'_{0,j} \\ \lambda'_{1,j} \\ \lambda'_{2,j} \\ \lambda'_{3,j} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} \lambda_{0,j} \\ \lambda_{1,j} \\ \lambda_{2,j} \\ \lambda_{3,j} \end{pmatrix}$$

- *AddRoundKey*: In this operation, a 128-bit subkey is mixed with the State. Each column of the State is XORed with one 4-byte word of the subkey.

It should be noted that AES still follows the general SPN structure illustrated in Figure 2.4. AES has an initial *AddRoundKey* before the first round. *ShiftRow* and *MixColumn* are linear and can be expressed together in the form of (2.2) where $\mathcal{A}$ is a 128×128 binary matrix and $\mathcal{B} = \mathbf{0}$. Such a combined linear transformation has a branch number B of 5. *MixColumn* is replaced with another *AddRoundKey* in the last round. Therefore, AES can be described in the form of Figure 2.4 by considering the round structure as *AddRoundKey*, *ByteSub*, and a linear transformation composed of *ShiftRow* and *MixColumn*.

The AES key schedule expands the cipher key into enough subkeys, which are sequentially stored in an array $W[\,]$ of 4-byte words. The first N_k words of $W[\,]$ are initialized as the cipher key, where N_k is the word length of the cipher key. Then, the next N_k words are derived from the current N_k known words using a sliding window approach. Many operations such as substitution, rotation, and constant padding are performed during the derivation.

Camellia

Camellia is a 128-bit Feistel-like cipher using keys with sizes of 128, 196, and 256 bits. A general round structure, as shown in Figure 2.5, is iterated for 18 times when 128 bit keys are used or 24 times when 192 or 256 bit keys are used.

The round function F of Camellia is illustrated in Figure 2.7. First, the 64-bit X_i is XORed with subkey K_i. The data mixed with the subkey then enters a layer of parallel 8×8 S-boxes. Camellia uses four invertible mappings for S-boxes, denoted by S_1, S_2, S_3, and S_4. A linear transformation follows the S-box layer, which is implemented by XORs as shown in the figure. Note that the branch number B of this linear transformation is 5 [6].

Two functions called FL and FL^{-1} are inserted into the Feistel network every 6 rounds. These two functions perform simple logic operations with two subkeys required and are linear when the subkeys are fixed. There are also two 128-bit key mixtures using XORs, located at the beginning and the end of the cipher, respectively. The key schedule, which will be discussed in detail in Chapter 7, expands all subkeys using a structure similar to encryption.

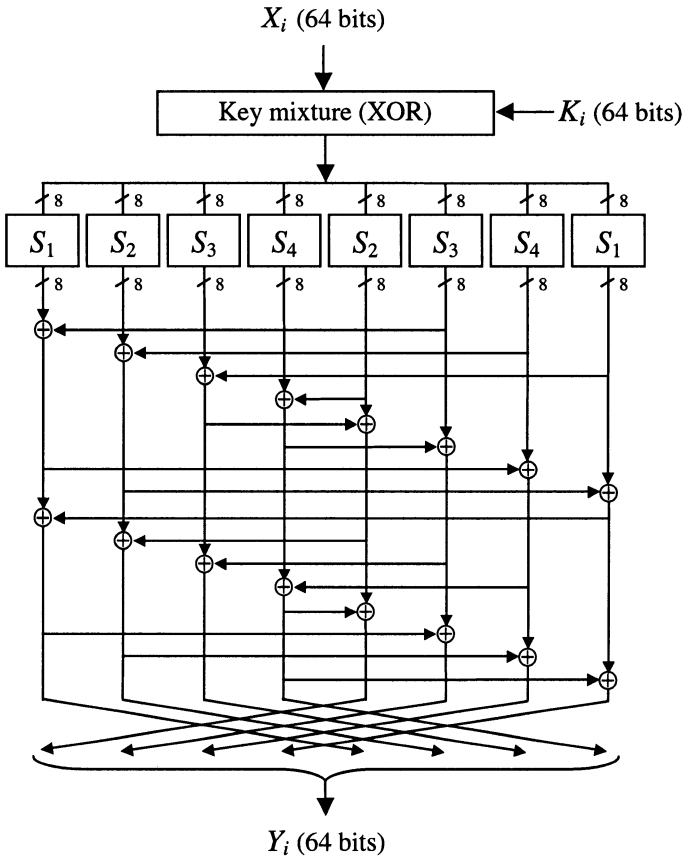


Figure 2.7: Function F of Camellia

Other Block Ciphers

In addition to DES, AES, and Camellia, many other block ciphers have been proposed with different cryptographic properties.

Serpent [33] and RC6 [8] were two AES candidates but not selected in the last round of competition. As an SPN cipher of 32 rounds, Serpent uses 8 different 4×4 mappings for S-boxes, while a certain mapping is used for all S-boxes in the same round. Serpent is fast in hardware and its structure is optimized for a bitslicing implementation [34] in software. RC6 is based on the cipher RC5 [35], which uses

data-dependent rotations and integer additions for encryption. RC6 also uses integer multiplications.

The block ciphers SHARK [21] and Square [36] include many features adopted by AES. SHARK is a 64-bit SPN cipher with an MDS mapping based on a Reed-Solomon code. Square is a 128-bit SPN cipher like AES. As one significant difference from AES, the linear transformation of Square does not have cyclic shifting for each row. Instead, the byte $\lambda_{i,j}$ in the State is changed to $\lambda_{j,i}$.

Hierocrypt [37], Anubis [30], Khazad [31], and MISTY [38] were all submitted to NESSIE for evaluation. Hierocrypt has a 2-level nested SPN structure which will be introduced in detail in Chapter 4. Anubis and Khazad are both involution ciphers which perform the same operations in encryption and decryption with only slight changes in the key schedule. Except for the involution feature, Anubis and Khazad are very similar to AES and SHARK, respectively. MISTY is a nested Feistel network with a block size of 64 bits.

2.3 Cryptanalysis

As the art of breaking ciphers, cryptanalysis is a valuable tool in finding the potential drawbacks in current ciphers and developing practical design principles. Differential cryptanalysis [39] and linear cryptanalysis [40] are two of the most powerful cryptanalytic techniques applied to block ciphers. They first concentrated on DES-like cryptosystems and are now used as general tools for security evaluation. Integral cryptanalysis was first applied on the cipher Square and became well known for its application to AES as described in [5]. Implementation attacks exploit the statistics existing in power, timing, and other measurable physical factors.

2.3.1 Differential Cryptanalysis

Differential cryptanalysis is a chosen-plaintext attack introduced by E. Biham and A. Shamir in CRYPTO'90 [39]. A chosen-plaintext attack needs access to the encryption machinery so that attacks can get the ciphertext corresponding to a selected plaintext.

For a system with block input and output, if two outputs Y_1 and Y_2 correspond to two inputs X_1 and X_2, respectively, then the input difference[2] is:

$$\Delta X = X_1 \oplus X_2$$

and the output difference is:

$$\Delta Y = Y_1 \oplus Y_2$$

where "$\oplus$" represents bit-wise XOR. Among all possible input pairs with a difference of ΔX, only a subset of output pairs lead to the specified difference of ΔY. A mapping from an input difference ΔX to an output difference ΔY is called a *differential*. The probability that a differential $(\Delta X, \Delta Y)$ occurs is called the *differential probability* and denoted by P_D:

$$P_D = prob\{\, Y_1 \oplus Y_2 = \Delta Y \mid X_1 \oplus X_2 = \Delta X \,\} \ .$$

Differential cryptanalysis works with the notion that the key mixture applied to the input pair (i.e., XORing key K with inputs) does not affect the differential statistics. Assuming X_1' and X_2' represent two inputs to a system with a key mixture added at

[2]Other differences are also defined, e.g., the difference calculated from modulo subtraction [41] but in the context of ciphers we shall examine, the given difference definition is the most useful.

the beginning so that

$$X'_1 = X_1 \oplus K$$
$$X'_2 = X_2 \oplus K$$

we have

$$\Delta X' = X'_1 \oplus X'_2 = (X_1 \oplus K) \oplus (X_2 \oplus K) = \Delta X \ .$$

The attack begins with seeking the highly likely differential of a system. For example, in an SPN cipher of R rounds, an attacker hopes to find a differential from some plaintext difference to some output difference of the $(R-1)$-th round, which occurs with a significant P_D. With this differential, the attacker can decrypt the corresponding ciphertexts one round with all possible subkey candidates of the last round to determine possible inputs to the last round. By checking for which subkey candidate the output difference of the differential holds for the calculated outputs of the $(R-1)$-th round most frequently, the valid subkey candidate of the last round can be distinguished.

Once the last subkey is distinguished, it is straightforward for the attacker to use the same technique to determine key bits from the $(R-1)$-th round, the $(R-2)$-th round, etc.. To thwart such an attack, cipher designers construct ciphers so that there are no large differential probabilities. To achieve this, no highly likely *differential characteristics* should exist in the cipher. A *differential characteristic* of γ rounds is a sequence denoted as $(\Delta Z_0, \Delta Z_1, \cdots, \Delta Z_i, \cdots, \Delta Z_\gamma)$, where ΔZ_0 is the input difference of the first round, ΔZ_γ is the output difference of the last round, and ΔZ_i is the output difference of the i-th round and also the input of the $(i+1)$-th round, $0 < i < \gamma$. Denote P_d as the probability that such a differential characteristic

holds. In practice, the number of chosen plaintext pairs required by differential cryptanalysis, N_D, can be approximated by

$$N_D \approx 1/P_d$$

in order to attack $\gamma + 1$ rounds of the cipher [42]. The number of plaintexts required by the attack is used to indicate the workload since both the processing time and memory requirement can be deduced from this number.

An S-box is *active* if it is involved in the differential characteristic in the attack. Considering all S-boxes, $\{S_i\}$, in a cipher, their maximum differential probability p_s is defined [39] as:

$$p_s \triangleq \max_i \max_{\Delta X \neq 0, \Delta Y} prob\{ S_i(X) \oplus S_i(X \oplus \Delta X) = \Delta Y \} .$$

If a total of n_a active S-boxes exist in the differential characteristic used for the attack, then

$$P_d \leq p_s^{n_a} .$$

A linear transformation with a large branch number can ensure a large value of n_a, thus, making the upper bound of P_d even smaller. A small P_d is desirable because of its reciprocal relation to the workload given by the number of plaintexts N_D.

Based on the basic differential attacks, many more advanced attacking techniques have been proposed and may lead to more significant results. The method to use differentials instead of characteristics for security evaluation has been presented in [41], which helps to understand the provable security of the cipher. High

order differential cryptanalysis can be applied to the block ciphers with low nonlinear degrees [43]. Truncated differential cryptanalysis [44] uses differentials with only part of the ciphertext bits involved in the output differences. Impossible differential cryptanalysis [45] examines the non-existence of differences in order to distinguish the correct key guess.

2.3.2 Linear Cryptanalysis

Linear cryptanalysis, introduced by M. Matsui in EUROCRYPT'93 [40], is mainly applicable as a known-plaintext attack, which assumes that the attacker has access to enough existent plaintext-ciphertext pairs. Linear cryptanalysis exploits the linear relationship between plaintext bits and ciphertext bits, and can be used to statistically determine subkey bits in the last round. Subsequently, the other subkeys can be determined in the same way with less workload.

The basic idea of this method is to find a linear approximation expression of the cipher algorithm. The method begins with a statistical linear path between the input and output bits of an S-box. Then, the path is spread to the entire cipher structure. By cancelling the common terms, a linear approximation expression without any intermediate bits will be obtained. For an n-bit cipher with input $X = (x_{n-1}, x_{n-2}, \cdots, x_0)$, output $Y = (y_{n-1}, y_{n-2}, \cdots, y_0)$ and an m-bit cipher key $K = (k_{m-1}, k_{m-2}, \cdots, k_0)$, the final effective linear expression [40] is of the form:

$$x_{i_1} \oplus x_{i_2} \cdots \oplus x_{i_a} \oplus y_{j_1} \oplus y_{j_2} \cdots \oplus y_{j_b} = k_{l_1} \oplus k_{l_2} \cdots \oplus k_{l_c} \tag{2.4}$$

where $0 \leq i_1, i_2, \cdots, i_a, j_1, j_2, \cdots, j_b \leq n-1$ and $0 \leq l_1, l_2, \cdots, l_c \leq m-1$.

If the bit variables in (2.4) are selected randomly, then the probability that (2.4)

holds, denoted by P_L, should be 1/2. However, since Y is obtained by encrypting X with fixed K, the values in these bit locations are not totally random. A cryptanalyst would hope that (2.4) holds with probability P_L which is not equal to 1/2. The linear probability bias ε, given by $\varepsilon = |p - 1/2|$, indicates the effectiveness of the linear approximation. The larger the bias ε is, the better linear attacking performs. Once a good linear expression is statistically found, we obtain the equivalent of one bit of information about K.

Since the right side of (2.4) is fixed to be either 0 or 1, the attack can derive more key bits by statistically testing each key candidate. To attack an SPN cipher of R rounds, for example, an attacker hopes to find a linear expression consisting of bit variables of the plaintext and output of the (R–1)-th round. Then the corresponding ciphertexts are decrypted one round with all possible candidates of the last round subkey. By checking for which subkey candidate the linear expression holds true with a probability bias ε significantly different than 1/2, the attacker can distinguish the valid subkey candidate from others. Once the last subkey is distinguished, it is straightforward to determine subkey bits of other rounds.

In order to perform an accurate statistical test, substantial plaintext-ciphertext pairs need to be processed. We take the complexity of cryptanalysis to be indicated by the data amount required by the attack. It is shown by Matsui [40] that the number of plaintext-ciphertext pairs required by linear cryptanalysis, N_L, can be approximated by

$$N_L \approx 1/(2\varepsilon)^2 \ .$$

To thwart these two attacks, cipher designers construct ciphers so that there is no large bias ε different from 1/2 for the probability that a linear expression holds.

To achieve this, no highly likely *linear characteristics* should exist in the cipher. Since any linear approximation of a data block or vector can be regarded as its inner product with a masking bit vector over GF(2), a *linear characteristic* of γ rounds is a sequence of masking values denoted as $(\Gamma Z_0, \Gamma Z_1, \cdots, \Gamma Z_i, \cdots, \Gamma Z_\gamma)$, where ΓZ_0 is the masking value for the first round input, ΓZ_γ is that for the last round output, and ΓZ_i is that for the i-th round output and also the $(i+1)$-th round input.

An S-box is *active* if it is involved in the linear characteristic in the attack. Considering all S-boxes, $\{S_i\}$, in a cipher, their maximum linear probability[3] is defined [40] as:

$$q_s \triangleq \max_i \max_{\Gamma Y \neq 0, \Gamma X} (2 \times prob\{ X \cdot \Gamma X = S_i(X) \cdot \Gamma Y \} - 1)^2$$

where "$\cdot$" denotes a bit-wise inner product and ΓX and ΓY denote masking variables.

A linear approximation is established by combining appropriate S-box linear approximation expressions into a linear characteristic with the following Piling-Up Lemma.

Theorem 2.1 (Piling-Up Lemma [40]): *Let u_i , $1 \leq i \leq n$, be independent random variables whose values are 0 with probability p_i or 1 with probability $1 - p_i$. Then the probability that $u_1 \oplus u_2 \cdots \oplus u_n = 0$ is*

$$1/2 + 2^{n-1} \prod_{i=1}^{n} (p_i - 1/2) \ .$$

[3]The terminology used here is the same as defined in [6, 38, 46] although it should be noted that it does not represent a true probability.

Hence, the bias of $u_1 \oplus u_2 \cdots \oplus u_n = 0$ *is*

$$\varepsilon = 2^{n-1} \prod_{i=1}^{n} \varepsilon_i$$

where $\varepsilon_i = |p_i - 1/2|$.

The Piling-Up Lemma is useful for approximating the overall linear probability. As a result, we can define a linear characteristic probability P_l with upper bound:

$$P_l \leq q_s^{n_a}$$

where n_a is the number of active S-boxes in the linear characteristic used for attacking. Thus, it can be shown that the workload now is also expressed as

$$N_L \approx 1/P_l \ .$$

A linear transformation with a large branch number is often used to diffuse S-box outputs and thus increase the value of n_a. As a result, the workload of the attack is enlarged.

The maximum probability of a linear characteristic is the primitive way to evaluate the resistance of a block cipher to linear cryptanalysis. Sometimes, a linear attack can be improved because different linear characteristics with the same linear approximation as shown in (2.4) can be combined to form a *linear hull* [47] with a higher probability. To understand provable security against a linear attack, it is desirable to estimate the expected probability of a linear hull. A method to seek the upper bound of such a probability has been suggested for SPNs in [48]. The basic linear attack can also be modified to utilize multiple linear approximations [49] or

nonlinear approximations [50].

Similar to each other as examined in [51, 52], both differential cryptanalysis and linear cryptanalysis begin with statistical recognition of specific cipher structures. For each attack, the maximum probability of any characteristics is calculated straightforwardly and used widely for security evaluation of proposed ciphers. The best linear approximation or differential characteristic can be searched for using the method presented in [53]. Provable security, in terms of the maximum probability of differentials or linear hulls, is a more accurate measure, but is generally difficult to compute or intractable to determine the required probabilities. As a result, considering the provable security of block ciphers in the context of differential and linear cryptanalysis appears to be generally impractical.

In this thesis, all 4×4 S-boxes are assumed to satisfy $p_s, q_s \leq 2^{-2}$ and all 8×8 S-boxes are assumed to satisfy $p_s, q_s \leq 2^{-6}$. Many proposed ciphers such as Serpent [33], AES, Hierocrypt [37, 54], and Camellia have S-boxes satisfying these requirements; others such as Anubis [30] and Khazad [31] have slightly higher p_s and q_s.

2.3.3 Integral Cryptanalysis

Integral cryptanalysis [55] is a chosen-plaintext attack first proposed by the authors of the cipher Square [36]. This attack exploits the effect of *balancing* caused by invertible components used in the cipher.

To attack AES using integral cryptanalysis as presented in [5], a special set of 256 plaintexts can be selected such that each plaintext has one distinct value at one common byte location and has the same values at the other 15 byte locations. The byte location with all different values is called *active*. Figure 2.8 shows a set of

plaintexts with an active byte located at the right bottom of the State (denoted by "**A**" in the figure). It should be noted that AES substitution and key mixture does not change the status of the active byte in the State. *ShiftRow* shifts the active byte to the left bottom and *MixColumn* propagates the active status to all the 4 bytes in current column.

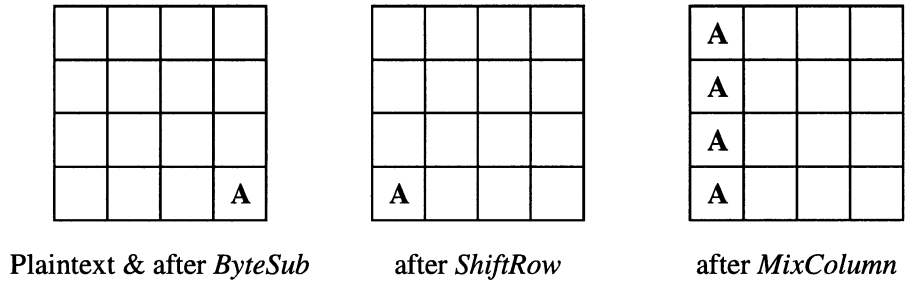


Figure 2.8: Active Status of the State in the AES First Round

Figure 2.9 illustrates the status change occurring in the second round. *ShiftRow* shifts the bytes in the active column to each column of the State because of a different byte offset per row. After *MixColumn*, all the bytes are active. The State keeps the same active status until *MixColumn* of the third round. However, each byte after *MixColumn* in the third round is the XOR sum of the products of active bytes and constants. Such an XOR sum is not active but has a property called *balancing*. That is, the 256 values at this location corresponding to the 256 plaintexts can be XORed to get 0 as the result. Such a balancing property is compromised by *ByteSub* of the fourth round.

For AES of 4 rounds, an attacker collects the ciphertexts associated with the above 256 plaintexts of the same set. Assuming a 32-bit partial key associated with one column of the State is known for the last subkey, 4 bytes at the beginning of the fourth round can be restored. By checking whether the balancing property exists

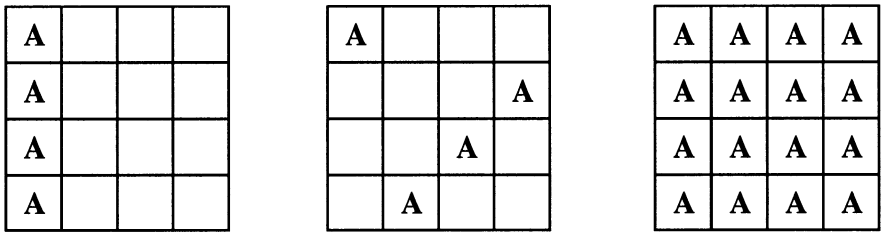


Figure 2.9: Active Status of the State in the AES Second Round

for all 256 restored values at the same byte locations, the valid 32-bit partial key candidate of the last round can be distinguished. This procedure can be repeated for other 32-bit partial subkeys associated with other columns of the State until all 128 bits of the last subkey are deduced.

The above attack against 4 rounds of AES can be extended up to 7 rounds by processing more sets of plaintexts and guessing more partial key candidates in different rounds [5]. Further, high order integral cryptanalysis and its application to block ciphers have been discussed in [55].

2.3.4 Implementation Attacks

During a cipher implementation execution, some physical information may be leaked to the external environment. If the leakage information can be measured by the attacker, an implementation attack (also called a side-channel attack) may be launched. These types of attacks are usually targeted to smart-card solutions because it is relatively easy to model their relation between internal circuits and physical leakage.

The attack based on power analysis was first introduced by P. Kocher et al. to deduce the key of DES from tamper-resistant devices [56]. To measure the power

reconfigured once produced. With a relatively longer development time, ASICs contain far more gates and run much faster than FPGAs. When manufactured in a large volume, an ASIC has a very low product cost. Therefore, this type of circuit is desirable for widely-used applications with high performance requirement.

Cryptographic hardware can be optimized for area requirement or throughput by choosing different design methods:

- *Round iterated design*: A round structure of the cipher is implemented and used in an iterative fashion to produce the encrypted output. The additional logic is needed to switch the data for input, iteration, and output. Such a design has the smallest area requirement but produces the lowest throughput.

- *Pipelined design*: A task is partitioned into several sequential stages, which have roughly equal delays. Registers are used to separate the adjacent stages and temporarily store the intermediate data and control signals, which enable each stage to work independently. A stage may contain one or several rounds of the cipher. Pipelining within one round is sometimes possible but limited due to large delay discrepancies of cipher components.

- *Loop unrolled design*: Several sequential rounds are implemented as a single combinational logic. Since the redundancy between rounds can be further reduced by CAD tools, such a design is faster than an iterated design but requires more area.

- *Block parallel design*: The design contains several independent encryption blocks. An I/O port controller assigns the input data to each block and assembles the encrypted data for output.

It should be noted that, although producing high throughputs, the pipelined design and parallel design can only be used when non-feedback block cipher modes are used. They cannot be used for example to implement the Cipher Block Chaining or Cipher Feedback modes [14]. During logic synthesis, an optimization strategy can be specified to give area or delay a higher priority.

Several implementation cases are listed, in which different implementation methods and technologies were used.

DES

DES has been implemented commercially by many hardware developers. As a recent case, Helion Technology claimed a hybrid DES design, which was an iterated structure of 2-round loop unrolling core [63]. The ASIC design in 0.18 μm CMOS technology has a throughput of 1.25 Gbits/s with less than 6,000 gates, while the FPGA design using a VirtexE-8 chip has a throughput of 526 Mbits/s with 855 LUTs (i.e., Lookup Tables).

AES

Many hardware implementations had been performed since the AES project was announced. Table 2.2 lists some typical published results. A round iterated design in an ASIC is significantly faster than that in an FPGA. However, when a chip with large area capacity is used, a fully pipelined AES implementation in FPGA can also run very fast.

Table 2.2: Several Published AES Hardware Implementations

Implementors	Throughput (Gbits/s)	Area	Design Method	Technology
B. Weeks et al. [64]	0.443	46 mm^2	iterated	ASIC, 0.5 μm
	5.163	471 mm^2	pipelined	
T. Ichikawa et al. [65]	1.95	612K gates	iterated	ASIC, 0.35 μm
H. Kuo et al. [66]	1.82	3.96 mm^2	iterated	ASIC, 0.18 μm
Helion Tech. [63]	2	27K gates	iterated	
	25	n.a.	pipelined	
A. Elbirt et al. [67]	0.300	5.3K CLBs	loop unrolling (2-round)	FPGA, Vertex XCV1000-BG560-4
	1.938	11K CLBs	pipelined (5-round)	
K. Gaj et al. [68]	0.332	2.9K CLBs	iterated	

CLBs: Configurable Logic Blocks

Camellia

Three types of implementations were presented by Camellia's authors in [10]. Type 1 used a fully loop unrolling design optimized for throughput and achieved a throughput of 1.17 Gbits/s in 0.35 μm CMOS technology, about 40% slower than AES implemented using the same methodology. The area requirement of the Camellia implementation of this type is 272,819 gates, which is about 55% less than the AES counterpart. Optimized for logic area, the ASIC implementation of Type 2 used a round iterated structure and achieved a throughput of 220 Mbits/s with a gate count of 11,350. Targeted on FPGA XC4000XL series, Type 3 was also a round iterated design and achieved a throughput of 122 Mbits/s with 874 CLBs.

2.4.2 Software Implementations

During software development, many factors need to be considered, such as the processor word size, the operating system, the software language, and the compiler. Many cryptographic operations include bit permutations and finite field mathematics, which are hard to be coded directly. In this case, a set of tables are usually generated in the memory to store the computation results based on different inputs as indices. As a result, these time-consuming operations can be realized as fast as memory access. Especially when the machine has a large word size (e.g., 32 or 64 bits), several operations caused by the change at the same small sub-block can be combined into one table lookup. This table lookup method is used in the fast implementations of DES [9], AES [5], and Camellia [10].

Table 2.3: Software Implementations on Different Platforms
(Selected from Tables 30 and 31 in [69])

Ciphers	PIII, MS	PIV, Linux	Alpha, OSF1	Mac
DES	62/ 62	61/ 61	37/ 37	60/ 59
AES	23/ 23	24/ 25	17/ 17	29/ 28
Camellia	37/ 37	64/ 63	36/ 35	31/ 31

Each entry: # cycles per byte for encryption/decryption

Table 2.3 lists the implementation results of the three ciphers of interest on different platforms measured by NESSIE [69]. The encryption and decryption times are indicated as the numbers of clock cycles per byte of output produced.

2.5 Summary

This chapter reviewed the basic design concepts of block ciphers as well as their security and implementations. The security of a block cipher is usually evaluated with respect to cryptanalysis. Several types of cryptanalysis have been introduced with emphasis on differential and linear attacks, which are the most well-known and fundamental attacks applied to block ciphers. The technology used for cipher implementations was briefly described with results of typical published cases presented.

Chapter 3

Hardware Design and Analysis of Block Cipher Components

Both S-boxes and MDS mappings are widely used components in current block cipher design. An MDS mapping can be performed through multiplications and additions over a finite field. In finite field arithmetic [22] with base 2, additions are bit-wise XORs, and multiplications can be calculated as polynomial multiplications modulo an irreducible polynomial. The MDS mapping used in AES encryption is implemented efficiently by several applications of "*xtime*" [5] (i.e., one-bit left shifting followed by addition with the irreducible polynomial). However, this method only suits the case that all entries in the generation matrix have both low Hamming weights and small magnitudes.

As typically the only nonlinear components in a block cipher, S-boxes must be designed to promote high security. As a result, each bit of an S-box output is a complicated Boolean function of input bits with a high algebraic order, which makes it difficult to optimize or evaluate the complexity of S-boxes generally in

hardware[1]. We propose an efficient hardware model of invertible S-boxes through the logic minimization of a decoder-switch-encoder circuit. By use of this model, a good upper bound of the minimum hardware complexity can be deduced for the S-boxes used in SPNs and some Feistel networks (e.g., Camellia [6]). The model can be used as a technique for the construction of S-boxes in hardware so that the space and time complexities are low.

In our work, we take the conventional approach that the space complexity of a hardware implementation is evaluated by the number of 2-input gates and bit-wise inverters; the time complexity is evaluated by the gate delay as measured by the number of traversed layers in the gate network. As a general complexity evaluation, these measures are not exactly proportional to the real area and delay in a synthesized VLSI design because logic synthesis involves technology-dependent optimization and maps a general design to different sets of cells based on targeted technologies. For example, a 2-input XOR gate is typically larger in area and delay than a 2-input AND gate in most technologies. As well, it is assumed that the overhead caused by routing after logic minimization can be ignored. Although routing affects the performance in a place-and-routed implementation, it is difficult to estimate its complexity accurately before synthesis into the targeted technology.

From previous FPGA and ASIC implementations of block ciphers, such as those listed in [70], it is well established that S-boxes normally contribute to most of a cipher's area requirement and delay. Although linear components such as MDS mappings are known to be much more efficient than S-boxes, it is important for cipher designers to characterize hardware properties of both S-boxes and MDS mappings on the same basis as is done through the analysis in this chapter.

[1]Some special cases with algebraic structure as in the AES S-box can be efficiently optimized.

The content of this chapter is also presented in [71].

3.1 Optimized MDS Mappings for Hardware

3.1.1 MDS Mappings

A linear code over Galois field GF(2^n) is denoted as an (l, k, d)-code, where l is the symbol length of the encoded message, k is the symbol length of the original message, and d is the minimal symbol distance between any two encoded messages. An (l, k, d)-code is MDS if $d = l-k+1$. A $(2k, k, k+1)$-code with generation matrix $\mathcal{G} = [\mathcal{I}|\mathcal{C}]$, where $\mathcal{C}$ is a $k \times k$ matrix and $\mathcal{I}$ is an identity matrix, determines an MDS mapping from the input $\mathcal{X}$ to the output $\mathcal{Y}$ through matrix multiplication over a finite field as follows:

$$f_M : \mathcal{X} \mapsto \mathcal{Y} = \mathcal{C} \cdot \mathcal{X} \tag{3.1}$$

where

$$\mathcal{X} = \begin{pmatrix} X_{k-1} \\ \vdots \\ X_0 \end{pmatrix}, \quad \mathcal{Y} = \begin{pmatrix} Y_{k-1} \\ \vdots \\ Y_0 \end{pmatrix}, \quad \mathcal{C} = \begin{pmatrix} C_{k-1,k-1} & \cdots & C_{k-1,0} \\ \vdots & \ddots & \vdots \\ C_{0,k-1} & \cdots & C_{0,0} \end{pmatrix}.$$

Each entry in $\mathcal{X}, \mathcal{Y}$, and $\mathcal{C}$ is an element in GF(2^n).

For a linear transformation, the branch number was defined in (2.3) as the minimum number of nonzero elements in the input and output when the input elements are not all zero. It is desirable that a linear transformation has a high branch number when it is used after a layer of S-boxes in a block cipher, in order for there to be low probabilities for differential and linear characteristics [39, 40]. A mapping based on

a $(2k, k, k+1)$-code has an optimal branch number of $k+1$.

3.1.2 Bit-Parallel Multipliers

An MDS mapping can be regarded as matrix multiplication in a finite field. Since the generation matrix is constant, each element in the encoded message is the XOR of several outputs of constant multipliers. As basic operators, bit-parallel multipliers given in a standard base [72, 73] are selected in this research. A constant multiplier can be written as a function from an element A to an element B over GF(2^n) as follows:

$$f_C : A \mapsto B = C \cdot A \tag{3.2}$$

where C is the constant element in GF(2^n). The expression in binary polynomial form is given as

$$b_{n-1}x^{n-1} + \cdots + b_0 = (c_{n-1}x^{n-1} + \cdots + c_0)(a_{n-1}x^{n-1} + \cdots + a_0) \bmod P(x) \tag{3.3}$$

where $P(x)$ denotes the irreducible polynomial of degree n for the field. An $n \times n$ binary matrix $\mathcal{F}_C$ is associated with this constant multiplier such that:

$$\begin{pmatrix} b_{n-1} \\ b_{n-2} \\ \vdots \\ b_0 \end{pmatrix} = \mathcal{F}_C \times \begin{pmatrix} a_{n-1} \\ a_{n-2} \\ \vdots \\ a_0 \end{pmatrix} \tag{3.4}$$

where

$$\mathcal{F}_C = \begin{pmatrix} f_{n-1,n-1} & \cdots & f_{n-1,0} \\ \vdots & \ddots & \vdots \\ f_{0,n-1} & \cdots & f_{0,0} \end{pmatrix}$$

and $f_{i,j} \in \{0,1\}, 0 \leq i,j \leq n-1$. The entries in each column of $\mathcal{F}_C$ are determined by

$$f_{n-1,j}x^{n-1} + \cdots + f_{0,j} = x^j(c_{n-1}x^{n-1} + \cdots + c_0) \bmod P(x). \tag{3.5}$$

Since $\mathcal{F}_C$ is constant, it is trivial to implement a constant bit-parallel multiplier by bit-wise XOR operations. For example, considering a constant multiplier to perform $B = 19\text{H} \times A$ over GF(2^8) where "H" indicates hexadecimal format and $P(x) = x^8 + x^4 + x^3 + x + 1$, we get the binary product matrix $\mathcal{F}_{19\text{H}}$ and the corresponding Boolean expressions for all bit outputs as the following:

$$\mathcal{F}_{19\text{H}} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{cases} b_7 = a_4 \bigoplus a_3 \\ b_6 = a_3 \bigoplus a_2 \\ b_5 = a_7 \bigoplus a_2 \bigoplus a_1 \\ b_4 = a_7 \bigoplus a_6 \bigoplus a_1 \bigoplus a_0 \\ b_3 = a_6 \bigoplus a_5 \bigoplus a_4 \bigoplus a_3 \bigoplus a_0 \\ b_2 = a_7 \bigoplus a_5 \bigoplus a_2 \\ b_1 = a_6 \bigoplus a_4 \bigoplus a_1 \\ b_0 = a_5 \bigoplus a_4 \bigoplus a_0 \end{cases}.$$

If we define $w(\mathcal{F}_C)$ as the count of nonzero entries in $\mathcal{F}_C$ and $w_i(\mathcal{F}_C)$ as the count of nonzero entries in row i of $\mathcal{F}_C$, the number of 2-input XOR gates used for the multiplier is upper bounded by $w(\mathcal{F}_C) - n$ and the delay of gate levels is

$\max_i\{\lceil \log_2 w_i(\mathcal{F}_C)\rceil\}$.

3.1.3 Complexity of MDS Mappings

An MDS mapping has been defined in (3.1) where each entry $C_{i,j}$ of matrix $\mathcal{C}$ is associated with a product matrix $\mathcal{F}_{C_{i,j}}$. Replacing each $C_{i,j}$ in matrix $\mathcal{C}$ with $\mathcal{F}_{C_{i,j}}$ as a submatrix, we get an $nk \times nk$ binary matrix $\mathcal{F}_\mathcal{C}$ as the following:

$$\mathcal{F}_\mathcal{C} = \begin{pmatrix} \mathcal{F}_{C_{k-1,k-1}} & \cdots & \mathcal{F}_{C_{k-1,0}} \\ \vdots & \ddots & \vdots \\ \mathcal{F}_{C_{0,k-1}} & \cdots & \mathcal{F}_{C_{0,0}} \end{pmatrix}.$$

Because $\mathcal{Y}$ is the matrix product of $\mathcal{F}_\mathcal{C}$ and $\mathcal{X}$, the MDS mapping can be straightforwardly implemented by a number of XOR gates. The gate count of 2-input XORs is upper bounded by

$$G_{MDS} = w(\mathcal{F}_\mathcal{C}) - nk \tag{3.6}$$

and the delay is upper bounded by

$$D_{MDS} = \max_i\{\lceil \log_2 w_i(\mathcal{F}_\mathcal{C})\rceil\} \tag{3.7}$$

where $0 \leq i \leq n-1$.

3.1.4 Three Types of Matrices

In the search of optimized MDS mappings in the next section, we will use three types of matrices which suit different applications. When an exhaustive matrix search is impractical, we will limit the search scope to one of the following three matrix types.

- *Circulant matrices*: Given k elements $\alpha_0, \ldots, \alpha_{k-1}$, a circulant matrix $\mathcal{A}$ is constructed with each entry $A_{i,j} = \alpha_{(i+j) \bmod k}$. The probability that a circulant matrix is suitable for an MDS mapping $\mathcal{C}$ is much higher than that of a normal square matrix [36].

- *Hadamard matrices*: Given k elements $\alpha_0, \ldots, \alpha_{k-1}$, a Hadamard matrix $\mathcal{A}$ is constructed with each entry $A_{i,j} = \alpha_{i \oplus j}$. Each Hadamard matrix $\mathcal{A}$ over a finite field has the following properties: $\mathcal{A}^2 = \gamma \cdot \mathcal{I}$ where γ is a constant. When $\gamma = 1$, $\mathcal{A}$ is an involution matrix. An involution MDS mapping is required by an involution SPN.

- *Cauchy matrices*: Given $2k$ elements $\alpha_0, \ldots, \alpha_{k-1}$, $\beta_0, \ldots, \beta_{k-1}$, a Cauchy matrix $\mathcal{A}$ is constructed with each entry $A_{i,j} = 1/(\alpha_i \oplus \beta_j)$. Any Cauchy matrix is MDS when $\alpha_0, \ldots, \alpha_{k-1}$ are distinct, $\beta_0, \ldots, \beta_{k-1}$ are distinct, and $\alpha_i \neq \beta_j$ for all i, j [24]. Although a Cauchy matrix can be conveniently used as matrix $\mathcal{C}$ for an MDS mapping, the relation between selected coefficients (i.e., $\alpha_0, \ldots, \alpha_{k-1}, \beta_0, \ldots, \beta_{k-1}$) and corresponding MDS complexity is not as straightforward as in the former two matrix types. Therefore, it is difficult to select coefficients to construct a Cauchy matrix that can be efficiently implemented in hardware.

3.1.5 The Optimization Method

The hardware complexity of an MDS mapping is determined directly by matrix $\mathcal{C}$. In order to improve hardware performance, matrix $\mathcal{C}$ should be designed to produce low hardware complexity. However, not every matrix with low complexity is suitable as an MDS mapping. The mapping associated with matrix $\mathcal{C}$ can be tested using the

following theorem:

Theorem 3.1 [24]: *An (l, k, d)-code with generation matrix $\mathcal{G} = [\mathcal{I}|\mathcal{C}]$ is MDS if, and only if, every square submatrix of $\mathcal{C}$ is nonsingular.*

To minimize gate count and delay in hardware, we want to find an MDS mapping based on a $(2k, k, k+1)$-code over GF(2^n) with low Hamming weights of $w(\mathcal{F}_\mathcal{C})$ and $w_i(\mathcal{F}_\mathcal{C})$. Theorem 3.1 provides us a way to determine whether a matrix candidate is MDS. Theoretically, the optimal MDS mapping can always be determined through an exhaustive search of all matrix candidates of $\mathcal{C}$. However, such a search is computationally impractical when k and n get large. In this case, it is reasonable to focus the search on some subsets of candidates which are likely to yield MDS mappings. The search scope can thus be limited to circulant, Hadamard, and Cauchy matrices.

Table 3.1: Four Choices for MDS Search

Search Options	# of Candidates	Applicable Cases
Exhaustive	$2^{k^2 n}$	small k, n
Circulant Matrices	2^{kn}	large k, n
Hadamard Matrices	2^{kn}	large k, n as well as involution
Cauchy Matrices	2^{2kn}	hard to find MDS mappings in other matrix categories

Table 3.1 describes four choices for the MDS search. We adopt an appropriate searching method based on the number of candidates to be tested and the required MDS features (involution or not). If computation permits, exhaustive search is preferred. When an exhaustive search is impractical, a search in circulant matrices may be performed for non-involution MDS mappings or a search in Hadamard matrices may be performed for MDS mappings which are involutions. Since only a subset of MDS mappings can be derived from circulant, Hadamard, or Cauchy matrices,

only exhaustive search over all possible matrices (and therefore all MDS mappings) is guaranteed to find a truly optimized MDS mapping. However for large k and n, searching over a subset of MDS mappings is the best that can be achieved. The objective is to find the candidate with the MDS property and a low hardware cost. The hardware "cost" could be gate count, delay, or both. Sometimes, no candidates in the sets of circulant and Hadamard matrices pass the MDS test. In this case, the optimal mapping will be determined through a search of Cauchy matrices, where each candidate is deterministically MDS.

Once a candidate is proved to be MDS (or involution MDS), those remaining candidates with higher hardware cost can be ignored narrowing the search space. The results generated in this searching method can be used for the hardware characterization of ciphers with MDS mappings of a specified size.

It is noted that $w(\mathcal{F}_\mathcal{C}) - nk$ just indicates the upper bound of XORs in the circuit. Two greedy methods introduced in [73] can be applied to the MDS matrix multiplication in order to further reduce redundancy in the circuit. However, the improvement of using greedy methods is not significant when $w(\mathcal{F}_\mathcal{C})$ is already low.

3.1.6 MDS Search Results

We have implemented a search for the best MDS mappings of various sizes. During the search, gate reduction is given higher priority than delay reduction because the delay difference among mappings is generally not evident. The optimal[2] non-involution MDS mappings for bit-parallel implementations for various sizes of MDS

[2]Here "optimal" means "locally optimal" when the MDS mapping is constrained to a particular matrix category.

mappings are given in Table 3.2. As in AES, SPNs using these optimal MDS mappings are more efficient in encryption than decryption. In Table 3.2, the average weight is determined by computing the number of matrix entries and dividing by two. That is, it represents the average number of ones in a matrix across all $nk \times nk$ matrices. These average weight values are included to show how effective the optimization work is for each MDS category.

Table 3.2: MDS Search Results

MDS	Galois Field	$P(x)$	Average Weight	Involution	$w(\mathcal{F}_C)$	Delay (# layers)	Matrix Type
$(4,2,3)$	GF(2^2)	7 H	8	No	9	2	exhaustive
$(4,2,3)$	GF(2^4)	13 H	32	No	17	2	exhaustive
$(4,2,3)$	GF(2^8)	11D H	128	No	35	3	exhaustive
$(8,4,5)$	GF(2^4)	13 H	128	No	76	3	circulant
$(8,4,5)$	GF(2^8)	11D H	512	No	164	3	circulant
$(16,8,9)$	GF(2^4)	13 H	512†	No	464	4	Cauchy
$(16,8,9)$	GF(2^8)	11D H	2048	No	784	4	circulant
$(4,2,3)$	GF(2^2)	7 H	8	Yes	11	2	exhaustive
$(4,2,3)$	GF(2^4)	13 H	32	Yes	21	2	exhaustive
$(4,2,3)$	GF(2^8)	11D H	128	Yes	48	3	exhaustive
$(8,4,5)$	GF(2^4)	13 H	128	Yes	88	3	Hadamard
$(8,4,5)$	GF(2^8)	11D H	512	Yes	200	4	Hadamard
$(16,8,9)$	GF(2^4)	13 H	512†	Yes	544	5	Cauchy
$(16,8,9)$	GF(2^8)	11D H	2048	Yes	928	5	Hadamard

†: Most randomly generated matrices are not MDS due to a small field and the requirement of a large branch number.

The optimal involution MDS mappings in terms of our complexity analysis are also given in Table 3.2. Since the MDS test of Theorem 3.1 is computationally intensive, an involution test will be performed first to eliminate wrong candidates. In [29], an algebraic construction of an involution MDS mapping based on Cauchy matrices is described. This known MDS mapping is used to eliminate remaining

candidates that produce higher complexity and therefore reduce search space before a better mapping is found.

The categories in Table 3.2 correspond to many MDS mappings in real ciphers (although there are minor differences in finite field selection). For example, Square, AES, and Hierocrypt at the lower level have non-involution MDS mappings based on $(8, 4, 5)$-codes over GF(2^8) [32, 36, 37]. SHARK has an non-involution MDS mapping based on $(16, 8, 9)$-codes over GF(2^8) [21]. Hierocrypt at the higher level has two choices of non-involution MDS mappings, based on $(8, 4, 5)$-codes over GF(2^4) and GF(2^{32}), respectively [37]. Anubis has an involution MDS mapping based on an $(8, 4, 5)$-code over GF(2^8) [30]. Khazad has an involution MDS mapping based on a $(16, 8, 9)$-code over GF(2^8) [31]. None these ciphers have MDS mappings with complexity as low as their corresponding cases listed in the tables. The mappings of AES, Anubis, and Khazad have MDS mappings that are close to the optimal cases in terms of gate counts (i.e., $w(\mathcal{F}_C) = 184, 216,$ and 1296, respectively), while Hierocrypt's MDS mappings have high complexity, similar to the average gate counts.

As Table 3.2 indicates, the involution MDS mappings are not as efficient as non-involution MDS mappings after optimization. However, the performance difference between them is quite small. When used in an SPN, the involution MDS mapping produces equally optimized performance for both encryption and decryption. When an SPN uses a non-involution MDS mapping optimized only for encryption, the inverse MDS mapping used in decryption has a higher complexity. For example, the MDS mapping used in AES decryption has $w(\mathcal{F}_C) = 472$ and, hence, needs more gates in hardware than the MDS mapping used for encryption which has $w(\mathcal{F}_C) = 184$. When a non-involution MDS mapping is optimized for both encryption and decryption, the overall hardware cost is similar to an optimized involution MDS

mapping.

3.1.7 Synthesis Results

We implemented the optimized MDS mappings in hardware using both 0.18 μm and 0.35 μm CMOS technologies. Synopsys Design Compiler was used for synthesis and the default optimization strategy gave the space concern a higher priority [74]. TSMC's cell library was targeted to 0.18 μm technology, where a specific area size was reported after synthesis. The cell library lsi_10k.db was targeted to 0.35 μm technology, where the area was reported in the number of equivalent NAND gates.

Tables 3.3 and 3.4 show the synthesis results of optimized MDS mappings listed in Table 3.2. The synthesis circuits of these MDS mappings produce space complexities with roughly the same trends as shown in Table 3.2. Because some cells in the target libraries have more than 2 inputs, the ratio between experimental values and the corresponding estimates vary slightly when the fields and minimum distances of MDS mappings are both small. This variance becomes insignificant as the complexity of an MDS mapping increases. The delay time of an MDS mapping may be larger than its estimate when the circuit becomes larger (e.g., mappings based on (16, 8, 9)-codes when using 0.35 μm CMOS), which is due to technology related wiring overhead and optimization strategy.

Table 3.3: Synthesis Results of Non-Involution MDS Mappings

MDS	Hamming Weight	Gate Delay	.18 μm CMOS		.35 μm CMOS	
			Area (μm^2)	Delay (ns)	Area†	Delay (ns)
(4, 2, 3), GF(2^2)	9	2	105.7	0.80	12	2.25
(4, 2, 3), GF(2^4)	17	2	260.2	0.42	28	2.25
(4, 2, 3), GF(2^8)	35	3	544.8	0.42	57	2.25
(8, 4, 5), GF(2^4)	76	3	1549.0	1.30	153	3.62
(8, 4, 5), GF(2^8)	164	3	3659.0	1.33	375	3.51
(16, 8, 9), GF(2^4)	464	4	8863.0	2.01	844	8.59
(16, 8, 9), GF(2^8)	784	4	17376.4	2.01	1636	9.49

† : # equivalent NANDs

Table 3.4: Synthesis Results of Involution MDS Mappings

MDS	Hamming Weight	Gate Delay	.18 μm CMOS		.35 μm CMOS	
			Area (μm^2)	Delay (ns)	Area†	Delay (ns)
(4, 2, 3), GF(2^2)	11	2	113.8	1.59	12	3.44
(4, 2, 3), GF(2^4)	21	2	280.5	0.95	27	2.43
(4, 2, 3), GF(2^8)	48	3	703.3	1.10	63	3.85
(8, 4, 5), GF(2^4)	88	3	1687.2	1.70	174	3.77
(8, 4, 5), GF(2^8)	200	4	4260.8	1.33	398	6.34
(16, 8, 9), GF(2^4)	544	5	9371.2	2.64	891	11.45
(16, 8, 9), GF(2^8)	928	5	19559.6	2.36	1850	10.96

† : # equivalent NANDs

3.2 General Hardware Model of Invertible S-boxes

3.2.1 Biham's Method to Simplify S-box Circuits

In [34], a method of generating a Boolean function through nested multiplexing is introduced to optimize gate circuits for the 6×4 S-boxes in DES implementations. Consider that a Boolean function $f(a,b,c)$ with three input bits a, b, and c can be written as

$$f(a,b,c) = f_1(a,b)\cdot c + f_2(a,b)\cdot \overline{c}$$

where $f_1(a,b)$ and $f_2(a,b)$ are two Boolean functions and "+" denotes OR. If $f_3(a,b) = f_1(a,b) \oplus f_2(a,b)$, then

$$f(a,b,c) = f_2(a,b) \oplus (f_3(a,b)\cdot c)\ .$$

Similarly, a Boolean function with an input of 4 bits can be regarded as a multiplexor using one input bit to select two boolean functions determined by the other three input bits. This procedure is repeated until a Boolean function has 6 input bits. A 6×4 DES S-box contains four of these 6-bit Boolean functions. This general approach can be taken for any size S-box and works well for optimization of small S-boxes such as the 4×4 S-boxes in Serpent [33]. However, in the case of general invertible 8×8 S-boxes used by many ciphers, this method can be improved upon, as we shall see.

3.2.2 Decoder-Switch-Encoder Model

In this section, we derive a general hardware model of $n\times n$ invertible S-boxes by simplification of a decoder-switch-encoder structure. Using this model, the upper

bounds of optimized gate counts and delays for S-boxes can be deduced.

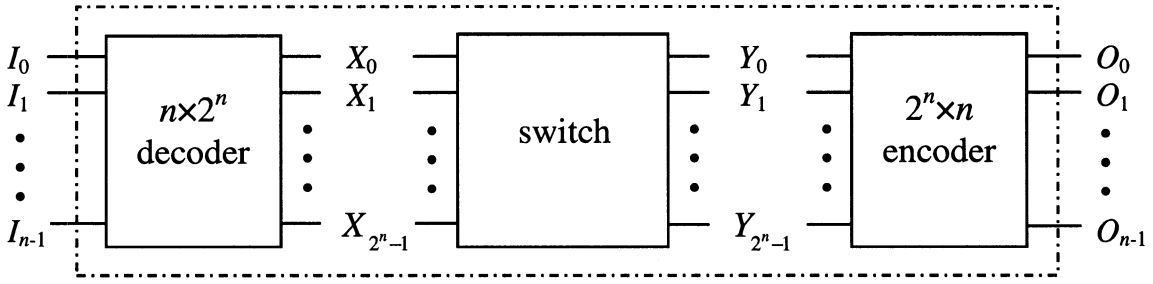


Figure 3.1: A General Hardware Structure of Invertible S-boxes

As shown in Figure 3.1, the $n \times 2^n$ decoder outputs 2^n distinct minterms from the n-bit S-box input. The switch is a wiring area composed of 2^n wires. Each wire connects an input port X_i to an output port Y_j, $0 \leq i, j \leq 2^n - 1$. Since the S-box is invertible, only one input port is connected to an output port. Although the wiring scheme embodies the S-box mapping, the switch does not cost any gates. The output of the switch is encoded through a $2^n \times n$ encoder, which produces the n-bit output of the S-box. A detailed example is presented in Figure 3.2, which is chosen for DES (the first row of the first S-box with the mapping shown in Table 2.1).

Decoder

The $n\times 2^n$ decoder is implemented by n NOT gates and a number of AND gates. The NOT gates generate complementary variables of n inputs. The AND gates produce all 2^n minterms from n binary inputs and their complements.

The most straightforward approach is to generate every minterm separately, which costs $2^n \cdot (n-1)$ 2-input AND gates plus n bit-wise NOT gates, and a delay of $\lceil \log_2 n \rceil + 1$ gate levels. This approach can be improved by eliminating redundant

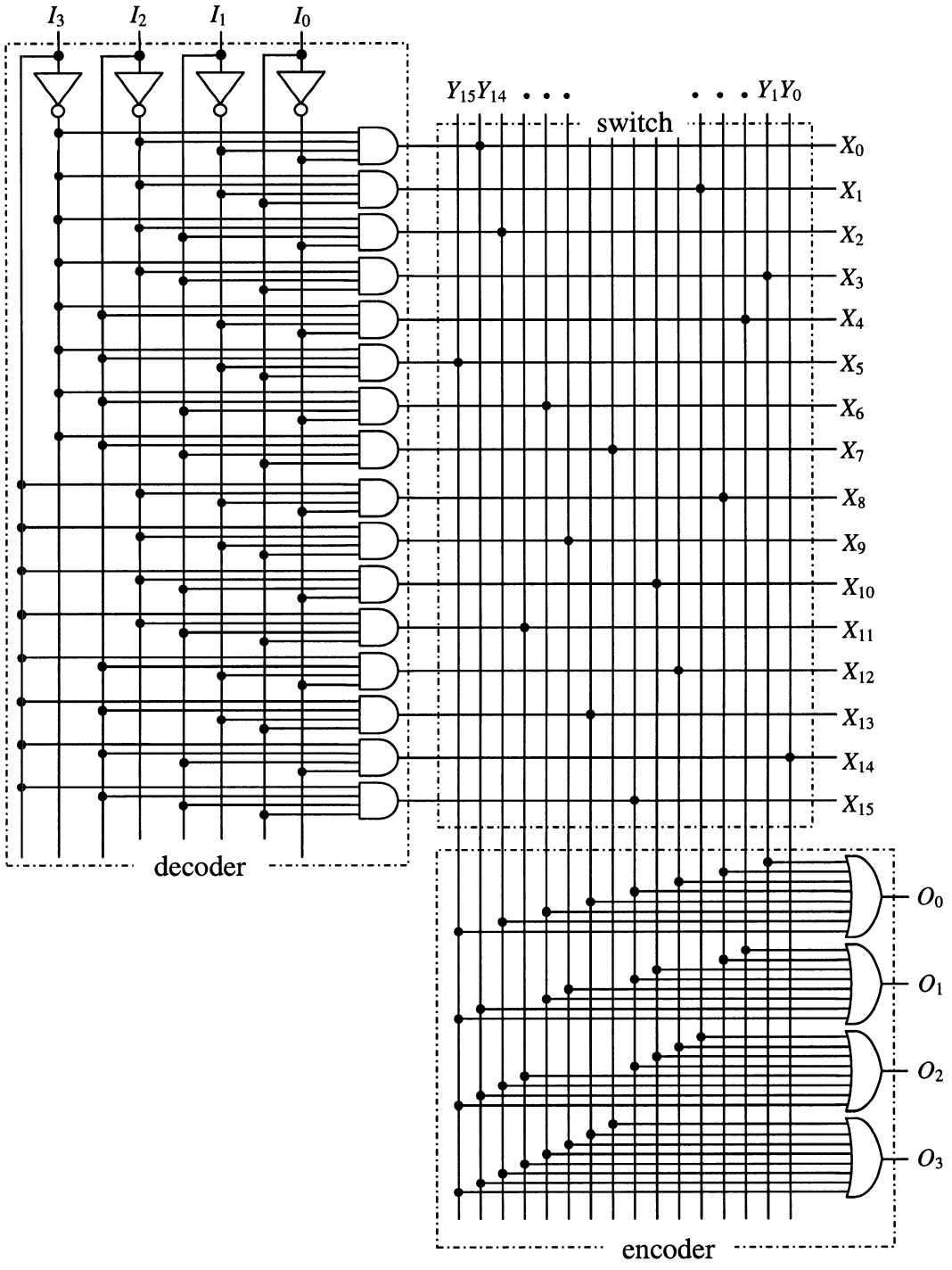


Figure 3.2: The Circuit of a 4×4 Invertible S-box

AND gates in the circuit. The gate count of the optimized circuit can be generated using a dynamic programming method.

for $i \leftarrow 0$ **to** $n-1$ **do**
  $D(i,i) \leftarrow 0$
for $step \leftarrow 1$ **to** $n-1$ **do**
  for $i \leftarrow 0$ **to** $n-1-step$ **do**
    $j = i + step$
    $D(i,j) \leftarrow \infty$
    for $k \leftarrow i$ **to** $j-1$ **do**
      $temp = D(i,k) + D(k+1,j) + 2^{j-i+1}$
    if $temp < D(i,j)$ **then** $D(i,j) \leftarrow temp$
return $D(0,n-1)$

Figure 3.3: Algorithm to Determine Decoder AND-Gate Count

Consider the dynamic programming algorithm in Figure 3.3, used to compute the minimum number of AND gates in the decoder. Let $D(i,j)$ be the minimal number of 2-input AND gates used for generating all possible minterms composed of literals $I_i, \cdots, I_j$ and their complements. Thus, $D(i,j) = 0$ when $i = j$. If we know two optimal results of subproblems, say $D(i,k)$ and $D(k+1,j)$ where $i \leq k < j$, all minterms for $I_i, \cdots, I_j$ can be obtained by using AND gates to connect two different minterms in the subproblems, respectively. Since the number of these pairs is 2^{j-i+1}, this solution needs $D(i,k) + D(k+1,j) + 2^{j-i+1}$ AND gates in total. The algorithm of Figure 3.3 can be easily modified to determine the actual gate network used for the decoder. When $n = 2^k$, it can be shown that the number of 2-input AND gates and bit-wise NOT gates in the decoder is given by

$$G_{Dec}(n) = n \sum_{i=1}^{k} 2^{2^i - i} + n \ . \tag{3.8}$$

The delay, in terms of the number of gate levels, of the decoder is

$$D_{Dec}(n) = \lceil \log_2 n \rceil + 1 \ .$$

Encoder

The $2^n \times n$ binary encoder can be implemented using a number of 2-input OR gates. Table 3.5 gives the truth table of a 16×4 binary encoder. Each output signal O_i is the OR of the 2^{n-1} input signals that produce "1" in column O_i in the truth table; this is denoted as $O_i = \sum Y_k$. If we separately construct circuits for these output signals, it would cost $n \cdot (2^{n-1} - 1)$ 2-input OR gates and a delay of $n-1$ gate levels. Fortunately, most OR gates can be saved if the same intermediate ORed signals are reused.

Considering that the OR is done in a dynamic programming method, some subproblems used in calculating O_i are also used in calculating O_j if $i > j > 0$. For example, as shown in Table 3.5, the task of calculating O_{n-1} includes the subproblems of calculating the OR from $Y_{5 \cdot 2^{n-3}}$ to $Y_{6 \cdot 2^{n-3}-1}$ and calculating the OR from $Y_{6 \cdot 2^{n-3}}$ to Y_{2^n-1}. These two subproblems are also included in the calculation of O_{n-3} and O_{n-2}, respectively. As a result, the OR gates needed to solve the recurrent subproblems can be saved. Actually, in the procedure of calculating O_i, only the subproblem of calculating the OR from Y_{2^i} to $Y_{2^{i+1}-1}$ has to be solved because all other $2^{n-i-1}-1$ subproblems have been solved in the procedures of calculating $O_{n-1}, \cdots, O_{i+1}$. In this sense, we need $2^i - 1$ OR gates for the subproblem that has not been solved and $2^{n-i-1}-1$ OR gates to OR the results of all 2^{n-i-1} subproblems. In total, the count

Table 3.5: Truth Table of a $2^n \times n$ Encoder

Input	Output			
Y_k	O_3	O_2	O_1	O_0
Y_0	0	0	0	0
Y_1	0	0	0	1
Y_2	0	0	1	0
Y_3	0	0	1	1
Y_4	0	1	0	0
Y_5	0	1	0	1
Y_6	0	1	1	0
Y_7	0	1	1	1
Y_8	1	0	0	0
Y_9	1	0	0	1
Y_{10}	1	0	1	0
Y_{11}	1	0	1	1
Y_{12}	1	1	0	0
Y_{13}	1	1	0	1
Y_{14}	1	1	1	0
Y_{15}	1	1	1	1

(a) $n = 4$

Input	Output			
Y_k	O_{n-1}	O_{n-2}	O_{n-3}	$\cdots$
$Y_0, \cdots, Y_{2^{n-3}-1}$	0	0	0	$\cdots$
$Y_{2^{n-3}}, \cdots, Y_{2^{n-2}-1}$	0	0	1	$\cdots$
$Y_{2^{n-2}}, \cdots, Y_{3 \cdot 2^{n-3}-1}$	0	1	0	$\cdots$
$Y_{3 \cdot 2^{n-3}}, \cdots, Y_{2^{n-1}-1}$	0	1	1	$\cdots$
$Y_{2^{n-1}}, \cdots, Y_{5 \cdot 2^{n-3}-1}$	1	0	0	$\cdots$
$Y_{5 \cdot 2^{n-3}}, \cdots, Y_{6 \cdot 2^{n-3}-1}$	1	0	1	$\cdots$
$Y_{6 \cdot 2^{n-3}}, \cdots, Y_{7 \cdot 2^{n-3}-1}$	1	1	0	$\cdots$
$Y_{7 \cdot 2^{n-3}}, \cdots, Y_{2^n-1}$	1	1	1	$\cdots$

(b) $n \geq 4$

of OR gates for the encoder is

$$G_{Enc}(n) = \sum_{i=0}^{n-1}[(2^i - 1) + (2^{n-i-1} - 1)] = 2^{n+1} - 2n - 2 \tag{3.9}$$

which is less than $n(2^{n-1} - 1)$ for $n > 2$ and the gate delay is

$$D_{Enc}(n) = n - 1$$

which is the same as the delay before simplification.

3.2.3 S-box Complexity

Based on the analysis of the decoder-switch-encoder structure, the hardware complexity of invertible S-boxes is estimated. Since 8×8 S-boxes are very popular in current block ciphers (e.g., AES [32], Hierocrypt [37], and Camellia [6]), let us examine the usability of this model in this case. According to (3.8) and (3.9), the upper bound of the optimal gate count for an 8×8 invertible S-box is 806, while the gate count before logic minimization is 2816.

Through experimental simplifications using the Synopsys logic synthesis tool [74], we realized 8×8 invertible S-boxes with a count of equivalent gates close to 800 when the target library was lsi_10k.db, as shown in Table 3.6. In addition to the S-boxes of AES and Hierocrypt, we also implemented 10 randomly generated S-boxes with $p_s, q_s \leq 2^{-4}$. In this table, the average cell count is 548 and the average equivalent gate count is 777. Since a small part of cells in the library have more than 2 inputs, the average of gates used for an 8×8 S-box is between 548 and 777 when only gates with 1 or 2 inputs are used. Such a result is quite close to the upper bound derived

Table 3.6: Synthesis Results of 8×8 S-boxes
(RS-1,$\cdots$,10: randomly generated S-boxes with $p_s, q_s \leq 2^{-4}$)

S-box	# cells	Area (# equivalent gates)	Delay (ns)
AES	510	752	18.14
Hierocrypt	555	784	15.85
RS-1	563	785	18.31
RS-2	531	765	17.72
RS-3	567	788	16.43
RS-4	525	759	17.93
RS-5	571	784	17.24
RS-6	557	775	16.32
RS-7	538	780	16.44
RS-8	553	779	17.29
RS-9	552	775	14.83
RS-10	550	793	17.57

from our model when $n = 8$.

When considering the implementation of an S-box with our model, the upper bound of the gate count increases exponentially with the S-box size n, as shown in Figure 3.4. Simultaneously, the upper bound of delay increases linearly, as shown in Figure 3.5. In these two figures, the S-box optimization model described in [34] and presented in Section 2 is used as the reference and the decoder-switch-encoder model is labelled DSE. When the size of an S-box is less than 6, the delay of the two models are similar and the gate count of the reference model is slightly lower. As the size of the S-box increases, the decoder-switch-encoder model costs less in both gate count and delay. The details of gate counts and delays are listed in Table 3.7 and Table 3.8. Given the fact that about half the gates used in the reference model are XOR gates which are typically more expensive in hardware in area and delay than NOT, AND, and OR gates, the decoder-switch-encoder model would appear to be more useful

for hardware design, both as an indication of the upper bound on the optimal S-box complexity and as a general methodology for implementing an invertible S-box.

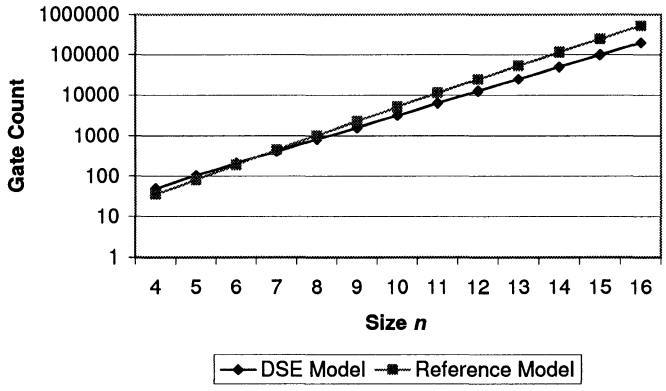


Figure 3.4: Gate Count Upper Bounds of S-boxes

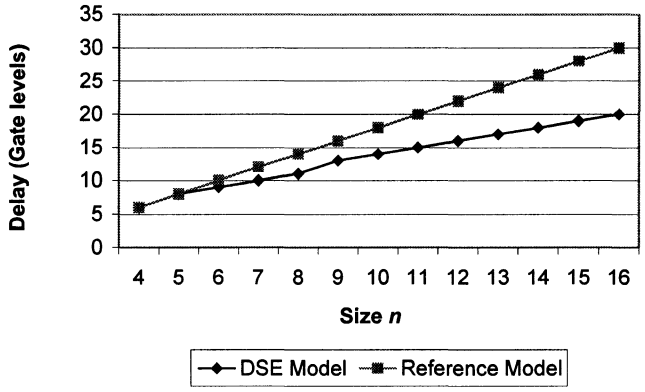


Figure 3.5: Delay Upper Bounds of S-boxes

3.3 Efficient AES Encryption Implementations

Since AES was selected to succeed DES, it is of great significance to characterize the implementation of AES in hardware. As introduced in Section 2.2.4, each round of

Table 3.7: Gate Counts of Invertible S-boxes in the Decoder-Switch-Encoder Model

S-box Size	4×4	6×6	8×8	10×10	12×12	14×14	16×16
NOT #	4	6	8	10	12	14	16
AND #	24	88	304	1120	4272	16712	66144
OR #	22	114	494	2026	8166	32738	131038
Gate Count	50	208	806	3156	12450	49464	197198
Reference Count	36	192	1020	5112	24564	114672	524268

Table 3.8: Gate Delays of Invertible S-boxes in the Decoder-Switch-Encoder Model

S-box Size	4×4	6×6	8×8	10×10	12×12	14×14	16×16
NOT	1	1	1	1	1	1	1
AND	2	3	3	4	4	4	4
OR	3	5	7	9	11	13	15
Delay	6	9	11	14	16	18	20
Reference Delay	6	10	14	18	22	26	30

AES contains the following operations to the State (i.e., the intermediate data stored in a two dimensional array) [32]: (1) a layer of 8×8 S-boxes called *ByteSub*, (2) a byte-wise cyclic shift per row called *ShiftRow*, (3) an MDS mapping based on an (8, 4, 5)-code per column called *MixColumn*, and (4) the round key mixing through XORs. The MDS mapping is defined over GF(2^8) and the S-box performs the equivalent of multiplicative inverse over GF(2^8) followed by a bit-wise affine operation.

With parallel S-boxes implemented through table lookups, a hardware design is proposed in [66]. Adhering to the structure of the algorithm specification of [32] as in Figure 3.6(a), this design achieves a throughput of 1.82 Gbits/s in 0.18 μm CMOS technology, where each S-box costs about 2200 gates. Since some operations over the composite field GF($(2^4)^2$) [22] are more compact than over GF(2^8), an efficient AES design with a low gate count in composite field arithmetic is proposed in [75]. A cryptographic core (i.e., essentially one round mainly consisting of 16 S-boxes and

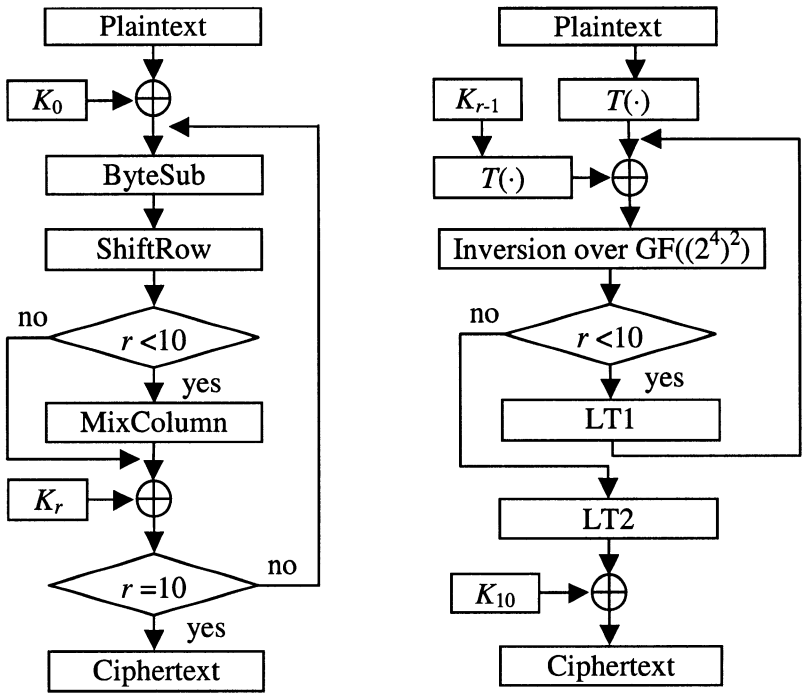
the MDS mapping layer) in [75] only costs about 4000 gates and a delay of 240 gate levels [76] for the full cipher is expected in theory.

3.3.1 Design I

Following the normal encryption dataflow, labelled as Design I in Figure 3.6(a), we apply the discussed S-box model and MDS bit-parallel implementation method to *ByteSub* and *MixColumn*, respectively. After the first round key K_0 is added to the plaintext, the State goes through an iterative round structure. Regardless of its mathematical definition, *ByteSub* is implemented as a layer of 16 parallel 8×8 S-boxes using the decoder-switch-encoder model. Then, the State iteratively proceeds through *ShiftRow*, *MixColumn*, and the addition with round key K_r. *ShiftRow* is implemented through wiring without any gates needed. Four bit-parallel MDS mappings perform *MixColumn* for the 4 columns. As listed in Table 3.9, we get an iterative core circuit of one round which costs 13456 gates and produces a delay of 15 gate levels per round. Because the MDS mappings are omitted in the last round, the AES encryption of 10 rounds produces a delay of 148 gate levels, a significant improvement over the delay of 240 gate levels in the design of [75]. The design needs far fewer gates than that in [66].

Table 3.9: Gate Counts and Delays of Operations in AES Design I

Operations	*ByteSub*	*MixColumn*	Key Addition	Total per Round
Gate Count	12896	432	128	13456
Delay (gate levels)	11	3	1	15



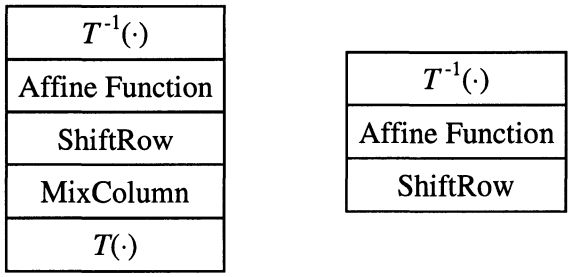
(a) Design I (b) Design II

Figure 3.6: AES Encryption Implementations

3.3.2 Design II

As shown in Figure 3.6(b), labelled as Design II, we get a more compact circuit through hybrid operations over GF(2^8) and its equivalent composite field GF($(2^4)^2$). The polynomial $P_1(y) = y^4 + y + 1$ is used to define GF(2^4) and the polynomial $P_2(x) = x^2 + x + 09\text{H}$ is used to define GF($(2^4)^2$). Such a composite field is the same as in the implementation proposed in [75] for ease of comparison. The conversion from GF(2^8) to GF($(2^4)^2$) is denoted as $T(\cdot)$, and its inverse is $T^{-1}(\cdot)$.

It has been recognized that the multiplicative inverse over GF($(2^m)^n$) can have a much lower complexity than the equivalent inverse over GF(2^{mn}) [73, 77]. As an example, the equivalent *ByteSub* over GF($(2^4)^2$) costs less than one fifth of the gate



(a) LT1 (b) LT2

Figure 3.7: Linear Transformations in AES Design II

count of a general invertible S-box based on the upper bound of 806 in the decoder-switch-encoder S-box model. However, the subfield-based operation is normally slow. In the implementation of Figure 3.6(b), the inverse over the composite field costs a gate delay of 14 (as deduced from [72, 73, 75, 76]). Given additional overhead for field conversion and *ByteSub*'s affine function, the *ByteSub* instance has a much longer delay path than in the implementation of Design I. To mitigate this problem, we can incorporate all linear operations into LT1 in the first nine rounds and LT2 in the last round as shown in Figure 3.7, resulting in a delay of 202 gate levels for encryption. The number of gates used in the iterative core circuit is slightly (about 3%) less than in [75]. The detailed gate counts and delays for Design II components are listed in Table 3.10.

Table 3.10: Gate Counts and Delays of Operations in AES Design II

Operations	16×Inversion over GF($(2^4)^2$) [72, 73, 75, 76]	LT1	LT2	$T(\cdot)$	Key Addition	Total per Round
Gate Count	2384	792	304	208	128	3816
Delay (gate levels)	14	5	3	3	1	20

In order to mathematically represent LT1 and LT2, we denote the input State as

$\{U_{i,j}\}$ and the output State as $\{V_{i,j}\}$, where i denotes the row index and j denotes the column index of an element in the State. The binary coefficients of $U_{i,j}$ and $V_{i,j}$ in their polynomial expressions can be written as two 32-bit tuples $\mathcal{U}_{i,j}$ and $\mathcal{V}_{i,j}$, respectively. LT1 can be expressed as

$$\begin{pmatrix} \mathcal{V}_{0,j} \\ \mathcal{V}_{1,j} \\ \mathcal{V}_{2,j} \\ \mathcal{V}_{3,j} \end{pmatrix} = \begin{pmatrix} \mathcal{F}_{L02} & \mathcal{F}_{L03} & \mathcal{F}_{L01} & \mathcal{F}_{L01} \\ \mathcal{F}_{L01} & \mathcal{F}_{L02} & \mathcal{F}_{L03} & \mathcal{F}_{L01} \\ \mathcal{F}_{L01} & \mathcal{F}_{L01} & \mathcal{F}_{L02} & \mathcal{F}_{L03} \\ \mathcal{F}_{L03} & \mathcal{F}_{L01} & \mathcal{F}_{L01} & \mathcal{F}_{L02} \end{pmatrix} \begin{pmatrix} \mathcal{U}_{0,j} \\ \mathcal{U}_{1,j-1} \\ \mathcal{U}_{2,j-2} \\ \mathcal{U}_{3,j-3} \end{pmatrix} + \begin{pmatrix} T(63\text{H}) \\ T(63\text{H}) \\ T(63\text{H}) \\ T(63\text{H}) \end{pmatrix}. \quad (3.10)$$

In above equation, $\mathcal{F}_{L01}$, $\mathcal{F}_{L02}$, and $\mathcal{F}_{L03}$ are 8×8 submatrices derived from the following expression:

$$\mathcal{F}_{L0i} = \mathcal{F}_T \cdot \mathcal{F}_{0i} \cdot \mathcal{F}_A \cdot \mathcal{F}_T^{-1}, \ i = 1, 2, 3 \quad (3.11)$$

where $\mathcal{F}_{0i}$ is the product matrix associated with 01H, 02H, or 03H in GF(2^8) and matrix $\mathcal{F}_A$ is associated with the affine function $A(\cdot)$ inside *ByteSub* (i.e., $A(\mathcal{X}) = \mathcal{F}_A \cdot \mathcal{X} + 63\text{H}$). $\mathcal{F}_T$ is the 8×8 transformation matrix associated with $T(\cdot)$(i.e., $T(\mathcal{U}_{i,j}) = \mathcal{F}_T \cdot \mathcal{U}_{i,j}$). Its inverse is $\mathcal{F}_T^{-1}$.

Similarly, LT2 is a function defined as

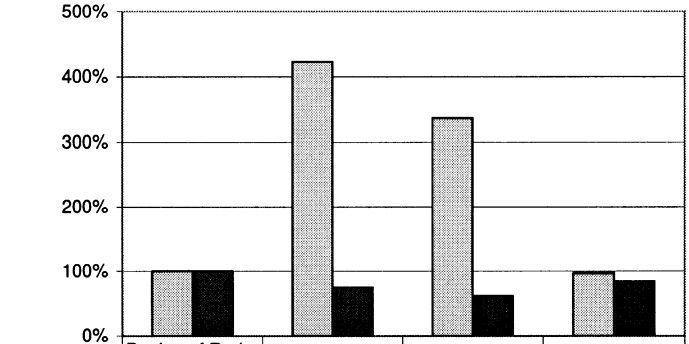
$$\begin{pmatrix} \mathcal{V}_{0,j} \\ \mathcal{V}_{1,j} \\ \mathcal{V}_{2,j} \\ \mathcal{V}_{3,j} \end{pmatrix} = (\mathcal{F}_A \cdot \mathcal{F}_T^{-1}) \begin{pmatrix} \mathcal{U}_{0,j} \\ \mathcal{U}_{1,j-1} \\ \mathcal{U}_{2,j-2} \\ \mathcal{U}_{3,j-3} \end{pmatrix} + \begin{pmatrix} 63\text{H} \\ 63\text{H} \\ 63\text{H} \\ 63\text{H} \end{pmatrix}. \quad (3.12)$$

Once we know the matrices $\mathcal{F}_T$, $\mathcal{F}_{L0i}$, and the result of $\mathcal{F}_A \cdot \mathcal{F}_T^{-1}$ (as listed in the

Appendix), the gate networks consisting of XORs can be straightforwardly derived for LT1 and LT2. The greedy method I described in [73] is used to reduce redundancy in the gate network, where small modifications are made in order to avoid the increase of delay.

3.3.3 Implementation Results

Figure 3.8 compares the estimated performance of the two designs of Figure 3.6 with respect to the implementation in [75]. Design I uses the MDS mapping implementation method and S-box model discussed in Sections 3.1 and 3.2 directly (while "Design I (Ref.)" uses the reference model in [34] for the S-boxes). In Design II, the method discussed in previous section is used to deduce the linear transformations LT1 and LT2. As Figure 3.8 shows, Design II gains a delay reduction of 16% and a slight reduction in the number of gates compared with the implementation of [75]. Design I is a much faster implementation with about three times as many gates.



	Design of Rudra et al.	Design I (Ref.)	Design I	Design II
Gate count	100%	423%	337%	97%
Delay	100%	74%	62%	84%

Figure 3.8: Performance Comparison of AES Designs

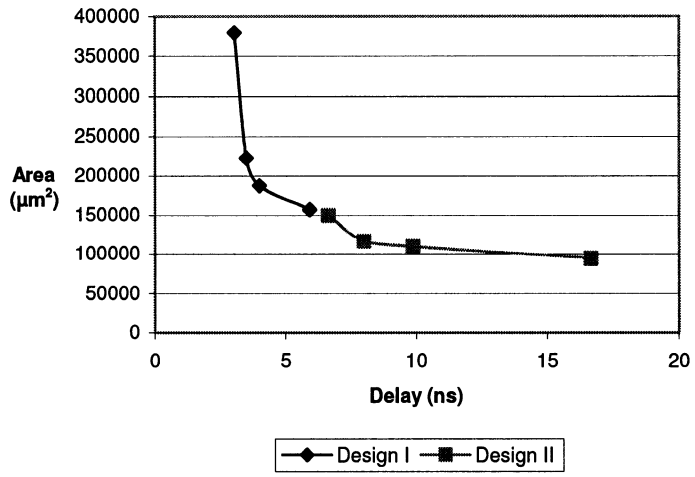


Figure 3.9: Synthesis of AES Round Structure

The round structures of the two AES designs have been coded in VHDL and synthesized using Synopsys Design Compiler and TSMC's 0.18 μm CMOS cell library. Setting constraints to tradeoff area and delay during synthesis, we get the characteristic curves shown in Figure 3.9. The two end points of each curve represent the synthesis results with smallest delay and smallest area. In line with our performance evaluation, Design I can lead to an iterative cipher architecture with a throughput up to 4 Gbits/s (i.e., the smallest round critical path is 3.04 ns). On the other hand, Design II is useful for an area-restricted or pipelined application because of its small area requirement.

3.4 Summary

We have presented a mechanism to select the MDS mappings for optimal hardware implementation of a block cipher. The optimized MDS mapping straightforwardly leads to a compact and fast implementation at the gate level. As well, a general

model of invertible S-boxes was proposed and the upper bounds of the minimal hardware complexity were deduced through systematic logic minimization. Since S-boxes and MDS mappings are both widely used cipher components, the discussed design, optimization, and hardware complexity evaluation provide an analytical basis for studying the hardware performance of block ciphers. As an example, two efficient hardware designs of AES encryption were considered with regards to different tradeoffs between gate count and delay, and their synthesis results were presented.

Chapter 4

Hardware Performance Characterization of Cipher Structures

In this chapter, we present a general framework for evaluating the hardware performance characteristics of block cipher structures composed of S-boxes and Maximum Distance Separable (MDS) mappings. In particular, we examine nested Substitution-Permutation Networks (SPNs) and Feistel networks with round functions composed of S-boxes and MDS mappings. Within each cipher structure, many cases are considered based on two types of S-boxes (i.e., 4×4 and 8×8) and parameterized MDS mappings. In our study of each case, the hardware complexity and performance are analyzed. Cipher security, in the form of resistance to differential, linear, and integral attacks, is used to determine the minimum number of rounds required for a particular parameterized structure. Because the discussed structures are similar to many existing ciphers (e.g., AES, Camellia, Hierocrypt, and Anubis), the analysis provides a meaningful mechanism for seeking efficient ciphers through a wide comparison of performance, complexity, and security. The content of this chapter is also presented in [78].

4.1 Studied Cipher Structures

4.1.1 Nested SPNs

The concept of a nested SPN was first introduced in [37]. In a nested SPN, S-boxes may be viewed at different levels: each S-box at a higher level is actually a small SPN at the lower level. In this chapter, we examine nested SPNs which have the following properties:

- The structure contains just two levels of SPNs. A higher level S-box consists of a lower level SPN; a lower level S-box is an actual 4×4 or 8×8 S-box.
- The linear transformation layers in both levels are based on MDS codes, denoted as MDS_H for the higher level and MDS_L for the lower level.
- The subkey mixture occurs directly before each layer of actual (i.e., lower-level) S-boxes. One additional subkey mixture is used to replace the linear transformation at the end of the cipher structure. The subkey bits are mixed with data bits by XOR operations.
- A "round" refers to the combination of the subkey mixture, lower-level S-box layer, and subsequent MDS_L or MDS_H linear transformation.

As Figure 4.1 shows, MDS_L is an MDS mapping from a $(2m_1, m_1, m_1 + 1)$-code over GF(2^{n_1}), while MDS_H is an MDS mapping from a $(2m_2, m_2, m_2 + 1)$-code over GF(2^{n_2}). The variables m_1, m_2, n_1, and n_2 represent parameter choices for a nested SPN.

In the most straightforward case, the output of each S-box forms one source symbol for the MDS mapping, and each encoded symbol forms the input of a subsequent

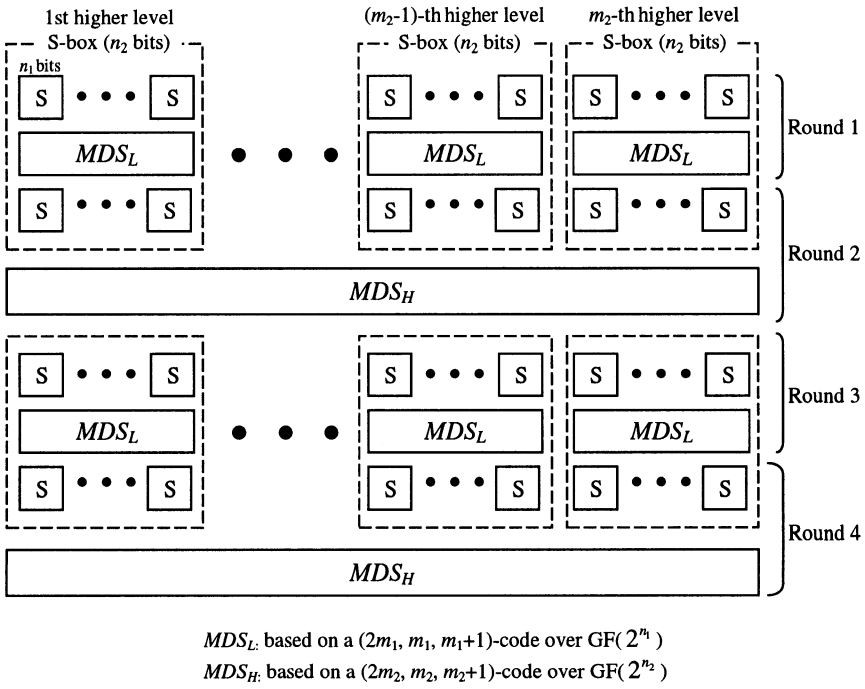


Figure 4.1: Basic 2-level Nested SPN (4 Rounds)

S-box at the same level. So the size of an S-box is n_1 bits at the lower level and n_2 bits at the higher level. This leads to $n_2 = n_1 m_1$. Thus, the block size of the SPN is $n_1 m_1 m_2$. For example, the 128-bit block cipher Hierocrypt (Type I) [37] is described as the iteration of such a 4-round structure where $n_1 = 8$, $n_2 = 32$, and $m_1 = m_2 = 4$.

At each level of a nested SPN, the branch number of the MDS layer determines the minimum number of active S-boxes in differential or linear cryptanalysis. For 4 rounds of a nested SPN, an active S-box at the higher level contains at least $m_1 + 1$ active S-boxes at the lower level. Since there are at least $m_2 + 1$ active S-boxes at the higher level, the minimum number of active lower-level S-boxes is $(m_1 + 1)(m_2 + 1)$. Therefore, the security against differential and linear attacks is evaluated as the following:

Theorem 4.1 (deduced from [5, 21, 36, 37]): *With the assumption that all S-box approximations involved in linear and differential cryptanalysis are independent, for* $4r$ *rounds of a nested SPN the maximum differential characteristic probability (denoted by* P_d*) is upper bounded by* $p_s^{r(m_1+1)(m_2+1)}$ *and the maximum linear characteristic probability (denoted by* P_l*) is upper bounded by* $q_s^{r(m_1+1)(m_2+1)}$.

To attack a cipher using differential cryptanalysis, the number of chosen plaintexts is expected to be in the order of $1/P_d$. Similarly, for linear cryptanalysis, the number of known plaintexts is expected to be in the order of $1/P_l$. Hence, the upper bounds of P_d and P_l provided in Theorem 4.1 indicate the lower bounds of required workload for attacking $4r+1$ rounds of the cipher based on a $4r$ round characteristic.

The basic operations in MDS codes are multiplications and additions in finite fields. When n_2 is large, operations over GF(2^{n_2}) are inefficient and MDS_H can be costly in computation. An alternative method to obtain the same branch number is to concatenate several parallel MDS codes over a smaller finite field. The concatenated codes may be designed to facilitate a bitslice implementation.

Theorem 4.2 [37]: *An MDS mapping defined by a* $(2m, m, m+1)$*-code over the* nl*-bit symbol set can be constructed by concatenating* l *mappings defined by a* $(2m, m, m+1)$*-code over the* n*-bit symbol set, where* l *can be any positive integer.*

For the example illustrated by Figure 4.1, since $n_2 = m_1 n_1$, the mapping MDS_H over GF(2^{n_2}) can be implemented with m_1 parallel MDS mappings over GF(2^{n_1}). In this case, the basic MDS_H layer is denoted as $1 \times (2m_2, m_2, m_2+1)$ over GF($2^{m_1 n_1}$), and its simplified, parallelized MDS_H layer is denoted as $l \times (2m_2, m_2, m_2+1)$ over GF(2^{n_2}) where, for example, we can have $l = m_1$ and $n_2 = n_1$. Since $m_1 n_1$ may be factored in other ways, other simplifications are also possible. Hence, we can

consider that the general relation $n_2 l = m_1 n_1$ can be used to determine different cases of MDS_H defined by the values of the symbol size, n_2, or the number of parallel MDS mappings, l. A similar approach can also be applied to the MDS_L layer. However, restrictions on values of n and m must be considered for designing a $(2m, m, m+1)$-code over GF(2^n) such that $2m \leq 2^n + 1$ in order that it is possible to construct an MDS code [24].

The 128-bit ciphers Square, AES, and Anubis can be regarded as the iterations of 4-round nested SPNs where $n_1 = n_2 = 8$ and $m_1 = m_2 = 4$. The parameters of Hierocrypt (Type II) are selected as $n_1 = 8$, $n_2 = 4$, and $m_1 = m_2 = 4$.

A set of nested SPNs can be generated with appropriate configurations of parameterized MDS_L, MDS_H, and S-boxes. As Theorem 4.2 illustrates, the MDS mapping defined over a large Galois field can be simplified using several mappings in a smaller Galois field. Table 4.1 lists the cases of nested SPNs in 12 categories (labelled as N1 to N12) defined by the S-boxes and MDS_L. Thus, the cases within a category only differ in the simplification of MDS_H. Each case can be regarded as $4r$ rounds of a 128-bit cipher where r is an integer, except that no particular key schedule has been defined. Due to the difficulty of finding optimized MDS mappings, the cases with a Galois field larger than GF(2^8) are not considered. The values of P_d and P_l represent the maximum differential and linear characteristic probabilities for $4r$ rounds evaluated by Theorem 4.1.

In relation to real ciphers, case N4-a includes Square, AES, and Anubis. Type II of Hierocrypt belongs to case N4-b with a simplified MDS_H over GF(2^4). Similar to SHARK and Khazad, case N8 is a one-level SPN. However, SHARK and Khazad are 64-bit ciphers because their MDS mappings are based on a (16, 8, 9)-code over GF(2^8).

Table 4.1: 128-bit Nested SPNs of $4r$ Rounds

Case	S-box size	MDS_L : $l_1 \times (2m_1, m_1, m_1{+}1)$ over $\mathrm{GF}(2^{n_1})$	MDS_H : $l_2 \times (2m_2, m_2, m_2{+}1)$ over $\mathrm{GF}(2^{n_2})$	P_d, P_l
N1-a	8×8	$8\times(4,2,3)$ over $\mathrm{GF}(2^8)$	$2\times(16,8,9)$ over $\mathrm{GF}(2^8)$	2^{-162r}
N1-b			$4\times(16,8,9)$ over $\mathrm{GF}(2^4)$	
N2-a	8×8	$16\times(4,2,3)$ over $\mathrm{GF}(2^4)$	$2\times(16,8,9)$ over $\mathrm{GF}(2^8)$	2^{-162r}
N2-b			$4\times(16,8,9)$ over $\mathrm{GF}(2^4)$	
N3-a	8×8	$32\times(4,2,3)$ over $\mathrm{GF}(2^2)$	$2\times(16,8,9)$ over $\mathrm{GF}(2^8)$	2^{-162r}
N3-b			$4\times(16,8,9)$ over $\mathrm{GF}(2^4)$	
N4-a	8×8	$4\times(8,4,5)$ over $\mathrm{GF}(2^8)$	$4\times(8,4,5)$ over $\mathrm{GF}(2^8)$	2^{-150r}
N4-b			$8\times(8,4,5)$ over $\mathrm{GF}(2^4)$	
N5-a	8×8	$8\times(8,4,5)$ over $\mathrm{GF}(2^4)$	$4\times(8,4,5)$ over $\mathrm{GF}(2^8)$	2^{-150r}
N5-b			$8\times(8,4,5)$ over $\mathrm{GF}(2^4)$	
N6-a	8×8	$2\times(16,8,9)$ over $\mathrm{GF}(2^8)$	$8\times(4,2,3)$ over $\mathrm{GF}(2^8)$	2^{-162r}
N6-b			$16\times(4,2,3)$ over $\mathrm{GF}(2^4)$	
N7-a	8×8	$4\times(16,8,9)$ over $\mathrm{GF}(2^4)$	$8\times(4,2,3)$ over $\mathrm{GF}(2^8)$	2^{-162r}
N7-b			$16\times(4,2,3)$ over $\mathrm{GF}(2^4)$	
N7-c			$32\times(4,2,3)$ over $\mathrm{GF}(2^2)$	
N8	8×8	$1\times(32,16,17)$ over $\mathrm{GF}(2^8)$	same as MDS_L	2^{-204r}
N9	4×4	$16\times(4,2,3)$ over $\mathrm{GF}(2^4)$	$1\times(32,16,17)$ over $\mathrm{GF}(2^8)$	2^{-102r}
N10	4×4	$32\times(4,2,3)$ over $\mathrm{GF}(2^2)$	$1\times(32,16,17)$ over $\mathrm{GF}(2^8)$	2^{-102r}
N11-a	4×4	$8\times(8,4,5)$ over $\mathrm{GF}(2^4)$	$2\times(16,8,9)$ over $\mathrm{GF}(2^8)$	2^{-90r}
N11-b			$4\times(16,8,9)$ over $\mathrm{GF}(2^4)$	
N12-a	4×4	$4\times(16,8,9)$ over $\mathrm{GF}(2^4)$	$4\times(8,4,5)$ over $\mathrm{GF}(2^8)$	2^{-90r}
N12-b			$8\times(8,4,5)$ over $\mathrm{GF}(2^4)$	

In theory, a maximum characteristic probability less than 2^{-128} indicates that the cipher is secure enough when only one characteristic is used for an attack. However, it is possible that several characteristics are combined to improved the attack as discussed in [41, 47]. As a result, it is still desirable that the maximum characteristic probability is much less than 2^{-128}. In the same sense, composite 8×8 S-boxes at the higher level of N9 and N10 cannot gain exactly the same security as 8×8 S-boxes used for N1 to N8, although the two types of ciphers may have the same maximum characteristic probabilities.

4.1.2 A Class of Feistel Networks

Figure 4.2 illustrates one particular class of round function F used for Feistel networks (as shown in Figure 2.5). Such a round function can be regarded as an SPN of one round with a size equal to half of the cipher block size. The round function includes one layer of key mixture with K_i (i.e., bit-wise XOR of X_i and K_i), one layer of invertible[1] S-boxes for substitution, and an MDS mapping layer as a linear transformation. If the MDS mapping layer is constructed through concatenation of several small MDS mappings, it is necessary to include a permutation of MDS symbols in the linear transformation in order to ensure the avalanche effect.

In a Feistel network whose round function has an invertible linear transformation appended to a layer of S-boxes, it is proved in [46] that the number of active S-boxes in any differential or linear characteristic of $4r$ rounds is lower bounded by $r \times B + \lfloor r/2 \rfloor$, where B is the branch number of the linear transformation and r is an integer. For an MDS layer based on m symbols, $B = m + 1$. Therefore, we get:

[1]Invertible S-boxes are used so that a bijective round function can be constructed, which achieves the given upper bounds of maximal differential and linear probabilities faster in rounds than a general round function [79].

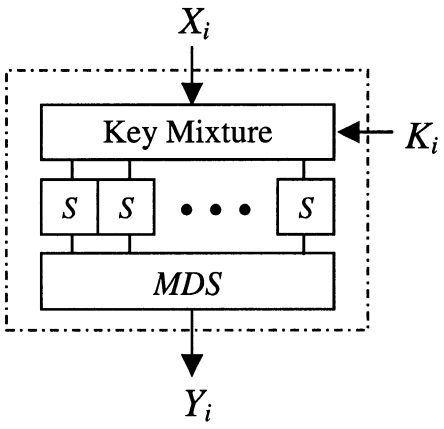


Figure 4.2: A Class of the Round Function

Theorem 4.3 (deduced from [46]): *For 4r rounds of a Feistel cipher with the round function of Figure 4.2, the maximum differential characteristic probability P_d and maximum linear characteristic probability P_l are upper bounded by $p_s^{r\times(m+1)+\lfloor r/2\rfloor}$ and $q_s^{r\times(m+1)+\lfloor r/2\rfloor}$, respectively.*

To construct a typical 128-bit cipher, a Feistel network of this class has a 64-bit round function which contains sixteen 4×4 or eight 8×8 parallel S-boxes followed by an MDS mapping layer. As listed in Table 4.2, six categories (labelled as F1 to F6) of these 128-bit Feistel networks can be generated. To ensure a good avalanche effect, an appropriate fixed permutation of MDS symbols after the MDS mapping is expected, which does not cost any gates.

Table 4.2: 128-bit Feistel Networks of $4r$ Rounds

Case	S-box size	MDS $l\times(2m,m,m{+}1)$ over GF(2^n)	P_d, P_l
F1-a	8×8	$4\times(4,2,3)$ over GF(2^8)	$2^{-6(3r+\lfloor\frac{r}{2}\rfloor)}$
F1-b		$8\times(4,2,3)$ over GF(2^4)	
F1-c		$16\times(4,2,3)$ over GF(2^2)	
F2-a	8×8	$2\times(8,4,5)$ over GF(2^8)	$2^{-6(5r+\lfloor\frac{r}{2}\rfloor)}$
F2-b		$4\times(8,4,5)$ over GF(2^4)	
F3-a	8×8	$1\times(16,8,9)$ over GF(2^8)	$2^{-6(9r+\lfloor\frac{r}{2}\rfloor)}$
F3-b		$2\times(16,8,9)$ over GF(2^4)	
F4-a	4×4	$4\times(4,2,3)$ over GF(2^8)	$2^{-2(3r+\lfloor\frac{r}{2}\rfloor)}$
F4-b		$8\times(4,2,3)$ over GF(2^4)	
F4-c		$16\times(4,2,3)$ over GF(2^2)	
F5-a	4×4	$2\times(8,4,5)$ over GF(2^8)	$2^{-2(5r+\lfloor\frac{r}{2}\rfloor)}$
F5-b		$4\times(8,4,5)$ over GF(2^4)	
F6-a	4×4	$1\times(16,8,9)$ over GF(2^8)	$2^{-2(9r+\lfloor\frac{r}{2}\rfloor)}$
F6-b		$2\times(16,8,9)$ over GF(2^4)	

4.2 Comparison of Hardware Performance

4.2.1 Performance Measures

It is normally hard to compare hardware performance among different block ciphers. The main problems are: (1) each implementation represents a tradeoff between area and delay, (2) the specific hardware cost of a gate network is dependent on the target technology, and (3) ciphers may contain different security margins.

For the first problem, the classical delay-area product is used to evaluate the hardware complexity universally. The typical methods used in the hardware implementation of a block cipher include a round iterated design, a pipelined design, a loop-unrolled design, and a block parallel design [70]. For a given cipher, the delay-area product is kept roughly unchanged across the different design methods (except

for a loop-unrolled design), assuming the control overhead for parallelism can be ignored. If a round iterated design is regarded as a reference, a k-block parallel design using several round iterated implementations will cost about k times the number of gates and result in about $1/k$ of the average time to produce an encrypted block. The same situation occurs in a pipelined design when each stage performs one or several rounds of the cipher. For loop unrolling, when k rounds are unrolled, it can be understood as removing the registers betweens rounds in a pipelined design of k rounds and then laying these rounds out. Using CAD tools to minimize such a large combinational circuit, its gate count is more than an iterative design but possibly much less than a pipelined design. By doing so, the encryption time for one block is reduced. Loop unrolling usually results in low performance in the sense of the delay-area product.

For the second problem, a universal way is to assume that all gates have the same hardware cost [73]. Thus, the gate count and delay of all components are deduced from the upper bound of typical implementations. Such an approach leads to a measure of complexity which is technology-independent. However, in a certain target VLSI technology, the hardware costs of different gates may not be similar. In this case, it is possible to estimate the overall area (respectively, delay) by summing weighted gate counts (respectively, weighted gate layers traversed). The weights are proportional to the size of a gate (respectively, delay) and can be calculated by statistical comparison of hardware among gates based on a target technology. The hardware complexity is then evaluated by weighted area A_W and weighted delay D_W:

$$A_W = \sum_{\text{gate type } u} G(u) \times W_G(u) \tag{4.1}$$

$$D_W = \sum_{\text{gate type } u} D(u) \times W_D(u). \tag{4.2}$$

Associated with gate type u, $G(u)$ and $W_G(u)$ return the gate count and weight of each gate. In the critical path of the circuit, $D(u)$ and $W_D(u)$ return the number of traversed gate layers and weight of each layer associated with gate type u.

For the problem caused by different security margins, we use a rule-of-thumb to determine resistance to differential and linear cryptanalysis. For differential cryptanalysis, the number of chosen plaintext pairs to attack a cipher is expected to be in the order of $1/P_d$, where P_d is the maximum differential characteristic probability determined by Theorems 4.1 and 4.3. Similarly, to attack a cipher using linear cryptanalysis, the number of known plaintexts is expected to be in the order of $1/P_l$, where P_l is the maximum linear characteristic probability.

Based on above considerations, we define three hardware performance metrics η_s, η_t, and η to measure the space, time, and overall performance, respectively. The three metrics integrate security and complexity and are defined as follows:

$$\eta_s = \frac{\log_2 1/P}{\#\text{ of rounds} \times A_W \text{ per round}} \tag{4.3}$$

$$\eta_t = \frac{\log_2 1/P}{\#\text{ of rounds} \times D_W \text{ per round}} \tag{4.4}$$

$$\eta = \frac{\log_2 1/P}{\#\text{ of rounds} \times (A_W \times D_W \text{ per round})} \tag{4.5}$$

where $P = P_d$ for hardware performance in relation to differential attacks and $P = P_l$ in relation to linear attacks. The probability P_d/P_l represents the maximum differential/linear characteristic probability for the number of rounds specified in the denominator. In each expression, the numerator is essentially a security measure in bits and the denominator is a complexity measure. Since we assume that the S-boxes

in the three discussed cipher structures satisfy $p_s = q_s$, the values of $\log_2 1/P_d$ and $\log_2 1/P_l$ are the same. For the nested SPNs and Feistel networks discussed in Section 2, $\log_2 1/P$ is a linear function of the number of rounds. Therefore, the values of η_s, η_t, and η indicate how much security is expected to be obtained for a specific hardware cost, regardless of the number of rounds in a cipher.

Targeted to the same design method, η_s shows the security contribution provided by each area unit; η_t shows the security contribution provided by each delay unit. For a fast implementation such as a pipelined or parallel design, a high η_s means that many independent blocks can be processed simultaneously. For a round iterated design, a high η_t means that the encryption time for a block is small. More generally, using the classical delay-area product as its denominator, η indicates the performance integrating both the delay and area complexities.

The cases that we compare in the following sections are generated as 128-bit block ciphers defined by the nested SPN and Feistel networks. To calculate the gate count and number of gate layers per round, we consider the construction of the combinational circuits of the round structure with S-box and MDS mapping components which can produce high efficiencies in hardware. The hardware design and optimization of these components are described in Chapter 3. The detailed data used in the complexity estimation which is to be used for determining performance will be presented in this chapter.

4.2.2 Hardware Performance of Nested SPNs

From the viewpoint of implementation, a nested SPN follows the iterative dataflow of key addition, an S-box layer, and an MDS mapping layer (either MDS_L or MDS_H). Since S-boxes cost the most hardware complexity, a 128-bit multiplexor

selects MDS_L and MDS_H dynamically such that only one layer of S-boxes is needed in a round iterated design. So assuming a round iterated implementation, the round circuit used for each case in Table 4.1 includes a 128-bit key addition, one layer of S-boxes, MDS_L, MDS_H, and a 128-bit multiplexor[2]. The 128-bit multiplexor can be implemented by 385 NAND gates (i.e., $y = x_1 \cdot c + x_2 \cdot \overline{c}$ where c is the select signal and "+" denotes OR).

In hardware, the complexity of S-boxes are evaluated through the simplification results deduced from an encoder-switch-decoder model as proposed in Section 3.2. In this model, S-boxes are composed of low complexity gates (ANDs, ORs, and NOTs). A 4×4 S-box can be implemented using 50 gates and produces a delay of 6 gate layers; an 8×8 S-box can be implemented using 806 gates and produces a delay of 11 gate layers. Involution MDS codes [30] are found by searching Hadamard matrices and have been optimized for hardware, as has been done in Section 3.1. MDS codes are implemented using XORs. Using these results, the complexity of each 128-bit 2-level nested SPN is evaluated for each round.

When $W_G(u) = W_D(u) = 1$ for any gate type u (i.e., all gates are assumed to have the same hardware complexity), we can sum the number of gates as the universal hardware area of a round structure. The calculation of the universal delay per round assumes the highest delay of MDS_L and MDS_H. Table 4.3 lists the evaluated hardware complexity of S-boxes, MDS mappings, and round structures. The area and delay per round are then used in (4.3), (4.4), and (4.5) and the resultant performance measures η_s, η_t, and η are also listed in Table 4.3.

Although each individual value in Table 4.3 cannot be perfectly accurate, the comparison of the performance measures does enable us to distinguish the cases

[2]MDS Multiplexing is not necessary for N8.

Table 4.3: Complexity and Universal Performance Estimation of One Round of 128-bit Nested Involution SPNs in Hardware

Case	S-boxes	MDS_L	MDS_H	Round Total (universal)	η_s	η_t	η	η_r
	Gate# - - Delay	XOR# - - Delay	XOR# - - Delay	Gate# - - Delay	(10^{-3})		(10^{-4})	(10^{-6})
N1-a	12896 – 11	256 – 3	1728 – 5	15393 – 19	2.63	2.13	1.38	3.42
N1-b	12896 – 11	256 – 3	2048 – 5	15713 – 19	2.58	2.13	1.36	3.35
N2-a	12896 – 11	208 – 2	1728 – 5	15345 – 19	2.64	2.13	1.39	3.43
N2-b	12896 – 11	208 – 2	2048 – 5	15665 – 19	2.59	2.13	1.36	3.36
N3-a	12896 – 11	224 – 2	1728 – 5	15361 – 19	2.64	2.13	1.39	3.43
N3-b	12896 – 11	224 – 2	2048 – 5	15681 – 19	2.58	2.13	1.36	3.36
N4-a	12896 – 11	672 – 4	672 – 4	14753 – 18	2.54	2.08	1.41	3.77
N4-b	12896 – 11	672 – 4	576 – 3	14657 – 18	2.56	2.08	1.42	3.79
N5-a	12896 – 11	576 – 3	672 – 4	14657 – 18	2.56	2.08	1.42	3.79
N5-b	12896 – 11	576 – 3	576 – 3	14561 – 17	2.58	2.21	1.51	4.04
N6-a	12896 – 11	1728 – 5	256 – 3	15393 – 19	2.63	2.13	1.38	3.42
N6-b	12896 – 11	1728 – 5	208 – 2	15345 – 19	2.64	2.13	1.39	3.43
N7-a	12896 – 11	2048 – 5	256 – 3	15713 – 19	2.58	2.13	1.36	3.35
N7-b	12896 – 11	2048 – 5	208 – 2	15665 – 19	2.59	2.13	1.36	3.36
N7-c	12896 – 11	2048 – 5	224 – 2	15681 – 19	2.58	2.13	1.36	3.36
N8	12896 – 11	8064 – 6	8064 – 6	21088 – 18	2.42	2.83	1.34	2.63
N9	1600 – 6	208 – 2	8064 – 6	10257 – 15	2.49	1.70	1.66	6.50
N10	1600 – 6	224 – 2	8064 – 6	10401 – 15	2.45	1.70	1.63	6.41
N11-a	1600 – 6	576 – 3	1728 – 5	4417 – 14	5.09	1.61	3.64	16.2
N11-b	1600 – 6	576 – 3	2048 – 5	4737 – 14	4.75	1.61	3.39	15.1
N12-a	1600 – 6	2048 – 5	672 – 4	4833 – 14	4.66	1.61	3.33	14.8
N12-b	1600 – 6	2048 – 5	576 – 3	4737 – 14	4.75	1.61	3.39	15.1

which are more efficient in hardware.

Figure 4.3 shows the tendency of the universal performance comparison (i.e., $W_G(u) = 1$, $W_D(u) = 1$). In an ASIC design, XOR gates are more expensive than other gates such as NOT, AND, and OR gates. Figure 4.4 shows a weighted performance comparison when $W_G(\text{XOR}) = W_D(\text{XOR}) = 2$ and weight for others is one. The two figures follow the similar tendency in performance comparison:

- The size of the S-box largely determines space and time performances. Using small S-boxes tends to cost less hardware area, but more delay than using large S-boxes. Given fixed chip area, the cipher cases using small S-boxes are more advantageous for parallelism as their higher η_s values show.

- Many SPN structures (N1-N10, N11-N12) are essentially equivalent with respect to their hardware performance. Hence, it is wise for a cipher designer to consider those structures which can facilitate software implementation.

- When the symbol size is 8 bits or less, the simplification of MDS mappings through concatenation does not significantly improve the performance when the MDS mappings have been selected to be optimized for hardware. For example, Case N4-b in Table 4.1 does not gain a much higher improvement in hardware than Case N4-a.

- When m_1 or m_2 are very high, the MDS mapping determined by m_1 or m_2 (e.g., MDS_H in cases of N9 and N10) will cost much more hardware and overwhelm S-box costs, which degrades the cipher performance.

- As a cipher of Case N4-a, AES is very suitable for a round iterated design. However, its suitability for pipelined or parallel implementations is not as high

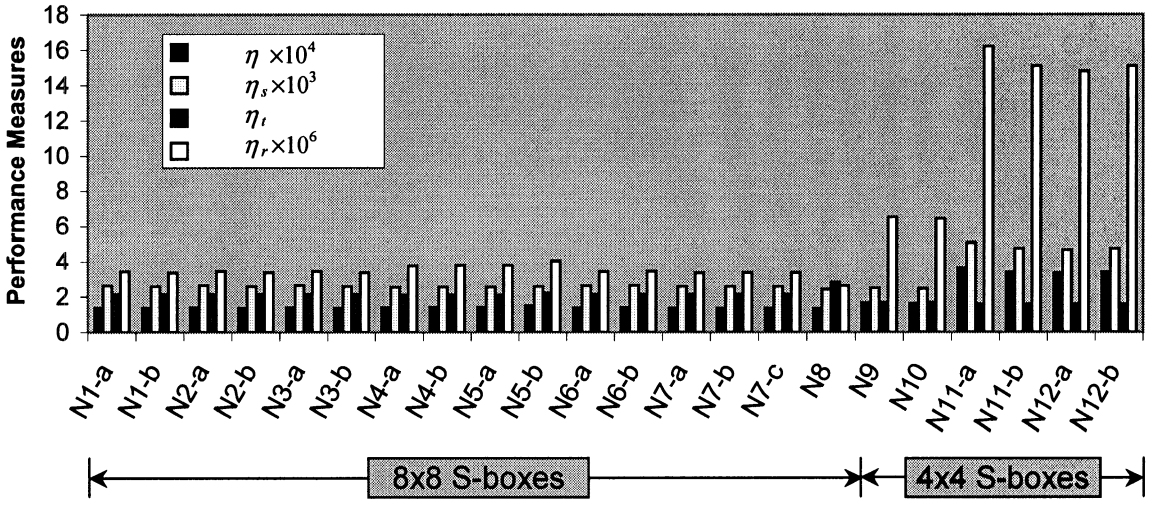


Figure 4.3: Universal Performance Comparison of Nested SPNs

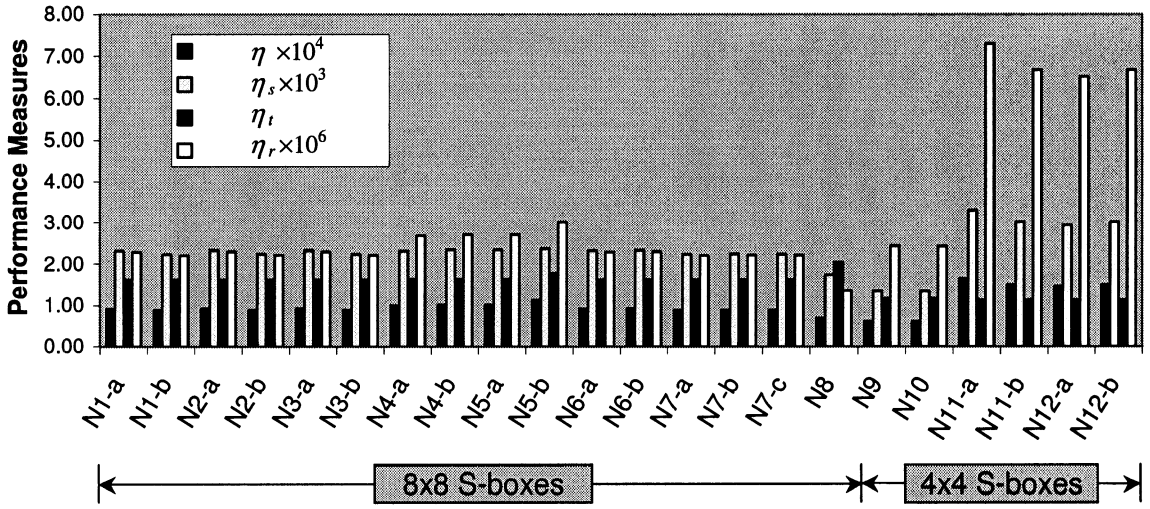


Figure 4.4: Weighted Performance Comparison of Nested SPNs

as cipher cases using 4×4 S-boxes such as cases of N11 and N12.

The above conclusions are based on hardware complexity and security against differential and linear attacks. For some other attacks such as the integral attack, the effectiveness significantly decreases after a certain number of rounds. In this circumstance, a performance metric of the round structure is defined as:

$$\eta_r = \frac{1}{A_W \times D_W \text{ per round}} .$$

Since the security in bits to resist these attacks increases very rapidly in the number of rounds, with a trend much steeper than differential and linear attacks as more rounds are appended, we take a fixed number of rounds (e.g., about 8 for the integral attack on AES) as enough for the security. The comparison of round performance is also included in Figures 4.3 and 4.4. It is obvious that the nested SPNs with small S-boxes and modest sized MDS_L and MDS_H have significantly better performance in relation to the integral attack than other cases.

4.2.3 Hardware Performance of Feistel Networks

The Feistel network discussed in this section is limited to the class described in Section 4.1, which has an SPN-like round function. As listed in Table 4.2, the cases of the same category only differ in the simplification of the MDS mapping. The hardware of one round of the Feistel network includes a 64-bit key addition layer, an S-box layer, an MDS mapping layer, and a 64-bit XOR after the round function (as shown in Figure 4.2). The key addition costs 64 XOR gates and a delay of one gate level. The XOR after the round function has the same hardware complexity as the key addition.

Table 4.4: Complexity and Universal Performance Estimation of One Round of 128-bit Feistel Networks in Hardware

Case	S-boxes	MDS	Round Total (universal)	η_s	η_t	η
	Gate # – Delay	XOR # – Delay	Gate # – Delay	(10^{-3})		(10^{-4})
F1-a	6448 – 11	76 – 3	6652 – 16	0.79	0.33	0.49
F1-b	6448 – 11	72 – 2	6648 – 15	0.79	0.35	0.53
F1-c	6448 – 11	80 – 2	6656 – 15	0.79	0.35	0.53
F2-a	6448 – 11	264 – 3	6840 – 16	1.21	0.52	0.75
F2-b	6448 – 11	240 – 3	6816 – 16	1.21	0.52	0.76
F3-a	6448 – 11	720 – 4	7296 – 17	1.95	0.84	1.15
F3-b	6448 – 11	864 – 4	7440 – 17	1.92	0.84	1.13
F4-a	800 – 6	76 – 3	1004 – 11	1.74	0.16	1.58
F4-b	800 – 6	72 – 2	1000 – 10	1.75	0.18	1.75
F4-c	800 – 6	80 – 2	1008 – 10	1.74	0.18	1.74
F5-a	800 – 6	264 – 3	1192 – 11	2.31	0.25	2.10
F5-b	800 – 6	240 – 3	1168 – 11	2.35	0.25	2.14
F6-a	800 – 6	720 – 4	1648 – 12	2.88	0.40	2.40
F6-b	800 – 6	864 – 4	1792 – 12	2.65	0.40	2.21

As shown in Figures 4.5 and 4.6, both the universal ($W_G(u) = 1$, $W_D(u) = 1$ for any gate type u) and weighted (i.e., $W_G(XOR) = 2$, $W_D(XOR) = 2$ and $W_G = 1$, $W_D = 1$ for all other gate types) performance comparisons indicate:

- It is useful to pick an MDS mapping that has a large branch number (i.e., $m+1$). The cases with such an MDS mapping have significantly higher values in all three performance measures.

- With high η_t values, the cases with 8×8 S-boxes demonstrate high performance in non-pipelined and non-parallel implementations. With high η values, the cases with 4×4 S-boxes demonstrate high performance in pipelined and parallel implementations because many independent blocks can be processed simultaneously.

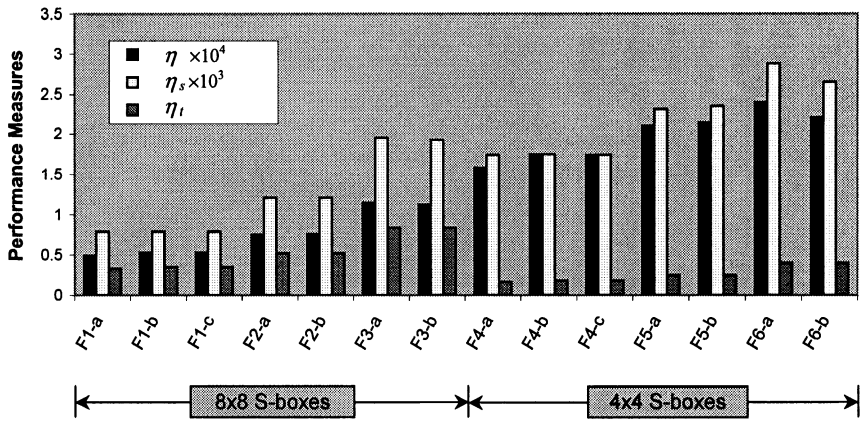


Figure 4.5: Universal Performance Comparison of Feistel Networks

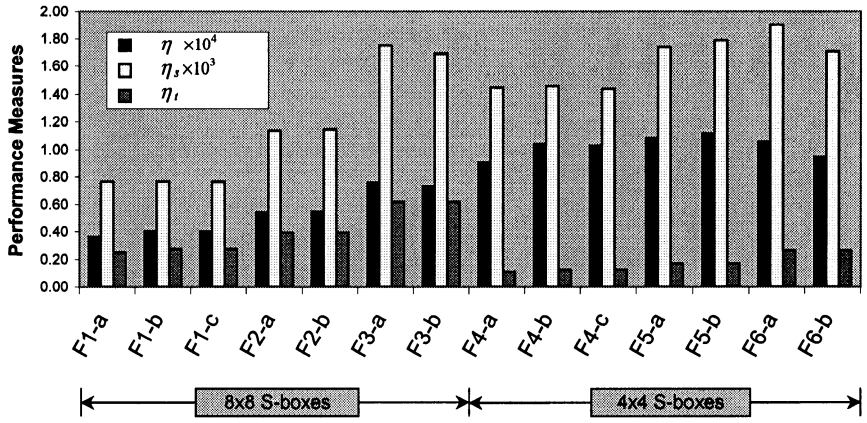


Figure 4.6: Weighted Performance Comparison of Feistel Networks

Camellia is a 128-bit Feistel cipher with a 64-bit round function which consists of eight 8×8 invertible S-boxes and a linear transformation. Hence, Camellia is similar to Feistel networks that we discussed but does not use an MDS mapping. The branch number of the Camellia linear transformation is 5. An efficient implementation of such a linear transformation costs 176 two-input XOR gates and a delay of 3 gate layers in universal comparison. Thus, Camellia has universal performance similar to Case F2-a which has 264 XOR gates and a delay of 3 gate layers (see Table 4.4).

Compared with the case F3-a, Camellia has a slightly more compact round structure (i.e., about 5% less in gate count than Case F3-a). However, each round of Camellia contributes much less to the security. Eleven rounds of F3-a provides equivalent security to nineteen rounds of Camellia. Further calculation shows that the overall hardware universal performance η of F3-a is about 50% higher than that of Camellia. The weighted performance comparison follows a similar trend.

4.2.4 Synthesis Results

The above performance analysis is based on theoretical evaluation of hardware complexity. The usability of these analytical results can be verified when VLSI technology is targeted. To avoid arduous work on synthesizing each cipher case, we did a high level synthesis of each component used in Tables 4.1 and 4.2. The components are coded in VHDL and synthesized with Synopsys Design Compiler. Two CMOS libraries[3] were used where most standard cells have one or two bit-wise inputs.

During synthesis, if the minimum area (respectively, delay) is set as the main constraint[4], the numbers of equivalent gates (respectively, critical delay time) of 8×8 S-boxes are close to their estimates in Tables 4.3 and 4.4. The gates and delays of 4×4 S-boxes are slightly less than their estimates because it is much easier for CAD tools to simplify smaller S-boxes. This effect indicates that the performance advantage of using small S-boxes as shown in Figures 4.3 to 4.6 is significant and slightly understated.

Since the MDS mapping is implemented in XOR gates, the areas and delays

[3]lsi_10k.db and TSMC's 0.18 μm CMOS library are targeted separately.

[4]When other constraints are set, the absolute values of area and delay will vary, but their comparison follows a similar trend.

closely follow the proportional relation of their estimates in Tables 4.3 and 4.4. Because XOR gates are larger and slower than other gate types, synthesis tools may replace them with other gates such as NXORs during optimization. Nevertheless, the delays and numbers of equivalent gates imply that a weight of 2 is reasonable for an XOR gate. This effect makes the cases with large MDS mapping worse in weighted performance, e.g., the cases in N8 to N12, F5, and F6.

This problem is encountered in the realizations where a large percent of XORs are used. The weighted performance shown in Figures 4.4 and 4.6 are thus more useful for a closer comparison than the universal method.

4.3 Summary

In this chapter we have considered two cipher structures composed of S-boxes and MDS mappings. Various cipher cases were generated from these structures with different component configurations. Their security and complexity were examined and integrated into performance metrics.

In hardware, the discussed cipher cases using large S-boxes are suitable for non-pipelined and non-parallel applications where delay is the main design criterion. In pipelined and parallel applications, the cipher cases using small S-boxes produce high performance. Further, appropriate selection of an MDS mapping layer is important for security against differential and linear attacks. With little change in the linear transformation, a suggestion was made to improve Camellia in terms of security and hardware efficiency.

For a Feistel network, more rounds are needed to be secure against differential and linear attacks. Compared with Feistel networks, the nested SPNs generally have

higher hardware performance. When the same S-boxes are used, a nested SPN tends to be more efficient in hardware to resist differential and linear attacks. Considering the threat of integral attacks, nested SPNs with smaller S-boxes are preferred.

Analogous with a nested SPN, MISTY [38] can be regarded as a nested Feistel network. Using provable security as the security measure, it will be interesting future work to compare the hardware performance between these two nested structures with similar performance metrics defined in Section 4.2.

Chapter 5

Software Performance Characterization of Cipher Structures

This chapter analyzes the software performance of cipher structures. The cipher structures studied are still nested SPNs and the class of Feistel networks analyzed in the previous chapter. Similar to the hardware performance metrics previously introduced, a novel performance metric is presented which allows us to consider the performance of a cipher structure as the combination of security and software efficiency. The parameterized cases of 128-bit block ciphers are studied. The software efficiency is mainly evaluated through a table-lookup implementation, where the number of table lookups is used as the time measure and the table size required is the space measure. A table-lookup implementation method is selected because it is usually efficient and such a method makes it possible to compare performance generally across different cipher configurations and different computing platforms. The efficiencies of other implementations (e.g., bitslicing, *xtime* [5], and power-index exchange) are also briefly examined. The content of this chapter is also presented in [80].

5.1 Table Lookup Implementations

The table lookup approach incorporates the S-boxes and the linear transformation into a table that is then accessed to perform both operations. This approach has been used for fast implementations of DES [9], AES [5], and Camellia [10]. Using this approach, the two cipher structures discussed in the former chapter can be implemented efficiently in software through table lookups, logic operations (e.g., XORs), and rotations. This chapter analyzes the efficiency of such fast implementations so that the memory and computational cost for a cipher case can be estimated. Independent of the targeted machine, the space complexity is evaluated as memory used for tables and the time complexity is evaluated by the number of table lookups.

The table lookup approach is chosen for analysis because it is normally faster and more general than other implementation approaches. A table lookup operation involves the reading of data from memory and also encompasses other operations necessary for indexing such as rotation and masking. Although the number of clock cycles to implement different operations is machine dependent, using the number of lookups and the size of the tables is suitable for determining a rough estimate of the time and space complexity of an efficient software implementation.

Larger tables require large data structures, and depending on the memory organization of the computer used, might require longer access times than smaller tables. This connection between the space and time complexities exists but becomes negligible when the tables that we compare are not far different in size. On the other hand, smaller tables may have indices with bit lengths less than 8 bits. In this case, shifting and masking are typically required for each lookup, which costs additional processing time. In this chapter, it is assumed that each table lookup requires the same access

time. The effect on lookup times caused by different table sizes, as will be examined in the experimental results of particular cipher cases, does not significantly affect our performance comparison.

In software, regardless of the implementation approach, the S-box layer is typically done by table lookups. An MDS mapping based on a $(2m, m, m+1)$-code conceptually performs a matrix multiplication over a Galois field, which requires m^2 modular multiplications and $m(m-1)$ XORs on words that are of the size of Galois field elements. To bypass costly multiplications, we enlarge the S-box table such that the MDS mapping work is included in the table lookups. This is the essence of the table lookup implementation. According to the size of the S-boxes and the type of MDS mappings, any cipher case may select appropriate methods as follows to generate lookup tables.

5.1.1 Cases with 8×8 S-boxes

The dataflow of a round in these cases involves the keyed input entering 8×8 S-boxes followed by l mappings based on a $(2m, m, m+1)$-code over GF(2^8). (The case of two concatenated mappings over GF(2^4) will be discussed later.) To represent the operations mathematically, we denote the input, output, subkey, and MDS generation matrix as $\{A_i\}$, $\{E_i\}$, $\{K_i\}$, and $\{C_{i,j}\}$, respectively, each containing 8-bit elements. Thus, the key mixture, S-box layer, and MDS mapping are expressed together as:

$$\begin{pmatrix} E_0 \\ E_1 \\ \vdots \\ E_{m-1} \end{pmatrix} = \begin{pmatrix} C_{0,0} & C_{0,1} & \cdots & C_{0,m-1} \\ C_{1,0} & C_{1,1} & \cdots & C_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{m-1,0} & C_{m-1,1} & \cdots & C_{m-1,m-1} \end{pmatrix} \begin{pmatrix} S(A_0 \oplus K_0) \\ S(A_1 \oplus K_1) \\ \vdots \\ S(A_{m-1} \oplus K_{m-1}) \end{pmatrix}. \quad (5.1)$$

Denoting the keyed input as $B_i = A_i \oplus K_i$, (5.1) is equivalent to:

$$\begin{pmatrix} E_0 \\ E_1 \\ \vdots \\ E_{m-1} \end{pmatrix} = \begin{pmatrix} C_{0,0} \\ C_{1,0} \\ \vdots \\ C_{m-1,0} \end{pmatrix} \times S(B_0) \oplus \cdots \oplus \begin{pmatrix} C_{0,m-1} \\ C_{1,m-1} \\ \vdots \\ C_{m-1,m-1} \end{pmatrix} \times S(B_{m-1}). \quad (5.2)$$

Hence, we may generate m tables as the following:

$$T_j[\cdot] = \begin{pmatrix} C_{0,j} \times S(\cdot) \\ C_{1,j} \times S(\cdot) \\ \vdots \\ C_{m-1,j} \times S(\cdot) \end{pmatrix} \quad (5.3)$$

where $0 \leq j \leq m-1$. The output of several S-boxes followed by the MDS mapping may then be generated using:

$$\begin{pmatrix} E_0 \\ E_1 \\ \vdots \\ E_{m-1} \end{pmatrix} = T_0[B_0] \oplus \cdots \oplus T_{m-1}[B_{m-1}]. \quad (5.4)$$

Each fetch from the table $T_j[\cdot]$ accepts an 8-bit input as the index and produces an $8m$-bit output from the indexed entry. It takes $256m^2$ bytes of memory to store these m tables. Given a processor with a word size of w bits, implementation of (5.4) needs $m\lceil 8m/w \rceil$ lookups and $(m-1)\lceil 8m/w \rceil$ XORs. In cases where the word size w is larger than the size of a table index, the preparation of a table lookup input will

generally need a rotation and masking (bit-wise AND) within a word.

When the size of an MDS field is smaller than the size of the S-boxes, we can consider an MDS mapping layer of more than one MDS mapping (i.e., the adjacent S-box output bits may pass through different mappings). The table $T_j[\cdot]$ is then established through concatenation. Each entry of $T_j[\cdot]$ consists of concatenated results from different MDS mappings. The result from one mapping corresponds to a specific subset of the table lookup output. For example, considering 8×8 S-boxes followed by $2\times(2m, m, m+1)$ over GF(2^4), each coefficient C_{ij} in (5.1) can be regarded as concatenation of two 4-bit coefficients C'_{ij} and C''_{ij} from two MDS mappings, so that $C_{ij} = C'_{ij} || \ C''_{ij}$, where " || " denotes concatenation. Then we generate m tables as:

$$T_j[\cdot] = \begin{pmatrix} C'_{0,j}\times S'(\cdot) \ || \ C''_{0,j}\times S''(\cdot) \\ C'_{1,j}\times S'(\cdot) \ || \ C''_{1,j}\times S''(\cdot) \\ \vdots \\ C'_{m-1,j}\times S'(\cdot) \ || \ C''_{m-1,j}\times S''(\cdot) \end{pmatrix} \tag{5.5}$$

where $0 \le j \le m-1$, $S'(\cdot)$ and $S''(\cdot)$ represent 4 output bits of an S-box, and $S(\cdot) = S'(\cdot)|| \ S''(\cdot)$. When these concatenated tables $\{T_j[\cdot]\}$ are used in (5.4), the size of tables and the number of lookups and XORs are the same as for the tables required in (5.3).

5.1.2 Cases with 4×4 S-boxes

In constrained environments such as smart cards, cipher cases using 4×4 S-boxes cost much less memory for table storage than those using 8×8 S-boxes. We can use the same method described by (5.2) and (5.4) to generate a set of small tables.

Since the variables B_i and E_i in (5.2) and (5.4) are now 4 bits, each fetch from the table $T_j[\cdot]$ accepts a 4-bit input as the index and produces a $4m$-bit output from the indexed entry. It takes $8m^2$ bytes of memory to store these m tables since each table requires $16 \cdot m \cdot 4/8 = 8m$ bytes. Such an implementation needs $m\lceil 4m/w \rceil$ lookups and $(m-1)\lceil 4m/w \rceil$ XORs.

When memory is not constrained, a modified method can be used to reduce the number of table lookups by a factor of 2. To implement a cipher case with 4×4 S-boxes, each table $T_j[\cdot]$ in (5.4) has an index of 4 bits. We can combine two tables into one, represented by T'_j, whose index is 8 bits. As a result, (5.4) is transformed to:

$$\begin{pmatrix} E_0 \\ E_1 \\ \vdots \\ E_{m-1} \end{pmatrix} = T'_0[B_0 || \ B_1] \oplus \cdots \oplus T'_{\frac{m}{2}-1}[B_{m-2} || \ B_{m-1}]. \tag{5.6}$$

where B_i and E_i are representing 4-bit values. For each 8-bit input $X || \ Y$ composed of 2 concatenated 4-bit values, X and Y, the table performs:

$$T'_j[X || \ Y] = T_{2j}[X] \oplus T_{2j+1}[Y]$$

where $0 \leq j \leq m/2 - 1$. It takes $64m^2$ bytes of memory to store these $m/2$ tables. The implementation of (5.6) needs $(m/2)\lceil 4m/w \rceil$ lookups and $(m/2-1)\lceil 4m/w \rceil$ XORs.

The method expressed by (5.6) should also be chosen for the cases where the symbol length of an MDS mapping is larger than the S-box size. For example, the inputs of two adjacent 4×4 S-boxes followed by an MDS mapping over GF(2^8) have

to be combined as an 8-bit index to a table of 256 elements.

5.2 Software Performance Comparison

It is normally hard to compare software performance among different block ciphers. The main difficulties are: (1) each cipher has its own security margin, (2) each implementation method represents a tradeoff between memory and speed, (3) the number of clock cycles required by one operation is determined by the platform, and (4) one specific instruction set may facilitate some operation combinations (e.g., DSP processors can do multiplication and accumulation using one single instruction). Considering the above difficulties, in order to get around the last three problems, we select the table lookup approach as a general and efficient method to implement all the cipher cases. Moreover, a meaningful study of the performance of ciphers should make comparisons between ciphers in consideration of a consistent security level.

5.2.1 Time Performance Metric

In software, the memory used for table storage is independent of the number of rounds. Since the memory can be easily allocated in many computers[1], the tradeoff between space and time is not as important as that in hardware. Therefore, instead of defining three metrics for space, time, and overall performance as in the previous chapter, we care more about the time performance in software. To compare the time used for a given cipher to achieve a certain amount of security, we define the time performance measure $\eta_{(w)}$ with respect to differential and linear attacks, where w is

[1]The smart card is an exception where memory is restricted.

the processor word size:

$$\eta_{(w)} = \frac{\log_2 1/P}{(\#\ of\ rounds) \times (\#\ of\ table\ lookups\ per\ round)} . \tag{5.7}$$

The numerator of (5.7) indicates the security of the cipher for the specified number of rounds, where we use a heuristic approach to determine resistance to differential and linear cryptanalysis. For differential cryptanalysis, the number of chosen plaintexts to attack a cipher is expected to be in the order of $1/P$, where P is the maximum differential characteristic probability P_d determined by Theorems 4.1 and 4.3; the number of known plaintexts required by linear cryptanalysis is expected to be in the order of $1/P$, where P is the maximum linear characteristic probability P_l of the cipher. For the nested SPNs and Feistel networks discussed in Chapter 4, $\log_2 1/P$ is a linear function of the number of rounds for both differential and linear cryptanalysis. Therefore, the value of $\eta_{(w)}$ indicates how much security is expected to be obtained within a unit running time (i.e., time for one table lookup), regardless of the number of rounds in a cipher.

Note that one table lookup has associated with it the setup of an index (e.g., one rotation and one masking operation) and a post-lookup XOR. Among these operations, the table lookup would normally require the most clock cycles in most processors. Hence, we use the table lookups as a barometer for the number of operations required to implement the cipher.

5.2.2 Comparison of Nested SPNs

In the previous chapter, Table 4.1 lists the cases of nested SPNs in 12 categories (labelled as N1 to N12) defined by the S-boxes and MDS_L. The values of P_d and P_l

Table 5.1: Software Performance of 128-bit Nested SPNs

Case	Table size (KBytes)	# of table lookups per 4 rounds			$\eta_{(8)}$	$\eta_{(32)}$	$\eta_{(64)}$
		8-bit	32-bit	64-bit			
N1-a,b	17	320	96	64	0.51	1.69	2.53
N2-a,b	17	320	96	64	0.51	1.69	2.53
N3-a,b	17	320	96	64	0.51	1.69	2.53
N4-a,b	8†	256	64	64	0.59	2.34	2.34
N5-a,b	8†	256	64	64	0.59	2.34	2.34
N6-a,b	17	320	96	64	0.51	1.69	2.53
N7-a,b,c	17	320	96	64	0.51	1.69	2.53
N8	64	1024	256	128	0.20	0.80	1.59
N9	64.03125	576	192	128	0.18	0.53	0.80
N10	64.03125	576	192	128	0.18	0.53	0.80
N11-a	16.125	384	128	96	0.23	0.70	0.94
N11-b	0.625	384	128	128	0.23	0.70	0.70
N12-a	4.5	384	96	96	0.23	0.94	0.94
N12-b	0.625	384	128	128	0.23	0.70	0.70

† : By use of the same mapping in MDS_L and MDS_H, half of the table size can be saved.

represent the differential and linear characteristic probabilities for $4r$ rounds evaluated by Theorem 4.1 and used as P in (5.7) to determine $\eta_{(w)}$. Table 5.1 lists the table size (i.e., the sum of sizes of tables required for MDS_L and MDS_H rounds) and the number of table lookups for 4 rounds for each cipher case. The table sizes listed in this table represent a minimum requirement and can only be achieved when an S-box does not differ from the corresponding S-box in another MDS mapping in the same layer (although S-boxes may be different within the domain of one MDS mapping). As a result, only one table as in (5.4) is required for each of the MDS_L and MDS_H layers.

For each case using 4×4 S-boxes, the tables with 4-bit indices are created as shown in (5.4) or (5.5) when the MDS mapping is chosen over GF(2^4) or GF(2^2), respectively. However, as explained in Section 5.1.2, the performance can be improved

by using 8-bit indices in the lookup as in (5.6) at the expense of more memory. For example, case N11-b can be implemented using 8-bit indices thereby doubling the efficiency but requiring 8 times the memory to store the lookup tables for both the MDS_L and MDS_H rounds. When GF(2^8) is chosen for the MDS mapping, the length of table indices has to be 8 bits. The number of table lookups is used in the calculation of the denominator in (5.7).

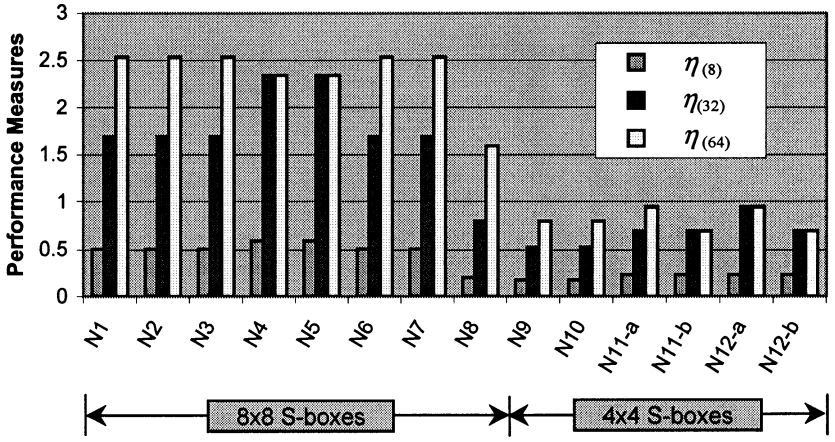


Figure 5.1: Software Performance Comparison of Nested SPNs

The table also includes the performance $\eta_{(w)}$ for each case. The implementation performance on three types of processors (i.e., $w = 8, 32, 64$) is considered. The implementation on an 8-bit processor is suitable for smart cards, where the memory size is constrained. The implementations on 32-bit and 64-bit processors are suitable for applications on general purpose computers and workstations. The values of $\eta_{(w)}$ are also presented in Figure 5.1. By comparing these measures, it is possible to distinguish the cases which are more efficient in software and the following general conclusions can be made:

- The implementation performance is improved when the word size of the processor increases, although in some cases there is no difference in the performance on a 32-bit or 64-bit processor.

- The cases with larger S-boxes (N1, $\cdots$, N8) have better performance but cost more memory to store the lookup tables.

- The cases with the same S-box size (N1, $\cdots$, N8 and N9, $\cdots$, N12) share similar performance although their memory requirements can vary significantly (as shown in Table 5.1).

- Cases N4 and N5 have the best, or close to the best, performance for all word sizes.

- As an example of N4-a, AES has very good performance. MDS_L and MDS_H in AES are based on the same (8,4,5) code. Therefore, half of the table size can be saved. Its byte-wise cyclic shifts (*ShiftRow*), before the MDS mapping, can be easily realized by taking the data from a modified byte location in the State as the index for a table lookup. Since the indirect addressing mode is supported by most processors, such a cyclic shift does not need to be coded separately in a table lookup implementation.

5.2.3 Comparison of Feistel Networks

The Feistel network discussed in this section is limited to the class described in Section 4.1, which has an SPN-like round function. The 128-bit cipher cases of this class have been listed in Table 4.2, which have 64-bit round functions consisting of sixteen 4×4 or eight 8×8 parallel S-boxes followed by an MDS mapping layer. To

ensure a good avalanche effect, an appropriate fixed permutation of MDS symbols after the MDS mapping is expected, which may cost a small amount of additional processing time. The cases of the same category in Table 4.2 only differ in the simplification of the MDS mapping. The performance comparison details are given in Table 5.2, where table lookups use 4-bit indices when 4×4 S-boxes are used in a cipher case. A summary of the performance measure $\eta_{(w)}$ is also illustrated in Figure 5.2. The following conclusions can be drawn:

- An MDS mapping that has a large branch number (i.e., $m+1$) results in good performance for implementations on computers supporting a large word size (e.g., comparing $\eta_{(8)}$ and $\eta_{(64)}$ in cases F3-a and F3-b).

- Although they require more memory, the cases with 8×8 S-boxes demonstrate higher performance.

- For the cases with 4×4 S-boxes, we can trade off memory and time requirements by choosing the element size of the MDS mapping. Using small Galois fields for the MDS codes, cases F4-b, F4-c, and F5-b can be used for some memory-constrained applications. However, their performance is not as high as the counterparts using large Galois fields (e.g., F4-a and F5-a) for a word size larger than 8.

- Compared with nested SPN networks, the Feistel networks discussed here need less memory but result in a lower performance.

Camellia uses 8×8 S-boxes and a linear transformation that is not MDS-based with branch number 5. (An MDS-based linear transformation would have a branch number of 9.) Hence, a simplified Camellia structure (without FL/FL^{-1} functions)

Table 5.2: Software Performance of 128-bit Feistel Networks

Case	Table size (KBytes)	# of table lookups per round			$\eta_{(8)}$	$\eta_{(32)}$	$\eta_{(64)}$
		8-bit	32-bit	64-bit			
F1-a,b,c	1	16	8	8	0.33	0.66	0.66
F2-a,b	4	32	8	8	0.26	1.03	1.03
F3-a,b	16	64	16	8	0.22	0.89	1.78
F4-a	1	16	8	8	0.11	0.22	0.22
F4-b,c	0.03125	16	16	16	0.11	0.11	0.11
F5-a	4	32	8	8	0.09	0.34	0.34
F5-b	0.125	32	16	16	0.09	0.17	0.17
F6-a	16	64	16	8	0.07	0.30	0.59
F6-b	0.5	64	16	16	0.07	0.30	0.30

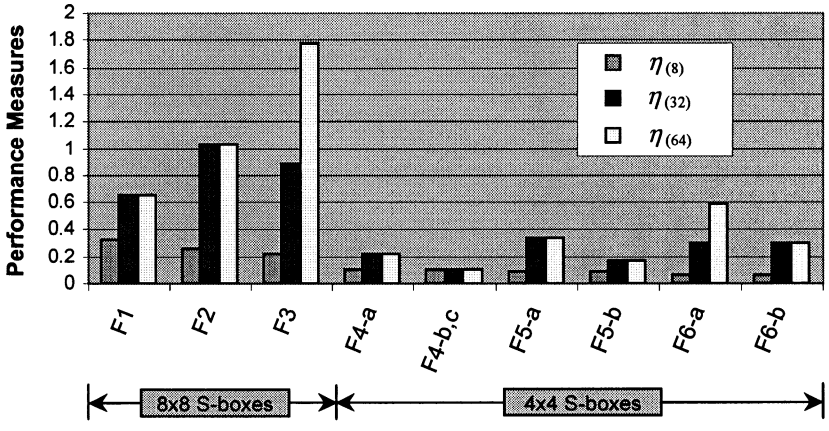


Figure 5.2: Software Performance Comparison of Feistel Networks

produces a security level equivalent to F2-a,b in Table 4.2. A fast Camellia implementation using table lookups was introduced in [10], which incorporates the linear transformation and S-boxes into several tables with 8-bit indices and 64-bit entries. In this method, a simplified Camellia has the equivalent number of table lookups to 18-round F3-a,b. As a result, Camellia uses tables as large as F3-a,b, while its performance is lower than both F2-a,b and F3-a,b on 32-bit processors and equal to F2-a,b but lower than F3-a,b on 64-bit processors.

5.2.4 Experimental Results

The performance comparison above is based on the assumption that the number of lookups is a good time measure for table lookup implementations. We implemented typical SPN cipher cases from Table 4.1 in "C" using the MS Visual C++ 6.0 compiler and determined the throughput for each implementation on an Intel Pentium III 933MHz computer. The throughput is measured by encrypting a data file containing millions of plaintexts. It is expected that the throughput will vary inversely to the number of lookups, considering throughput to be defined as:

$$throughput = \frac{block\ length}{processing\ time\ for\ each\ block}.$$

As shown in Table 5.3, the expected trend in throughput can be observed in the implementations when the number of rounds are set to the same value, especially in the compiler optimized version. When the table index is 8 bits, the byte permutation after the lookup operations (e.g., the concatenation of parallel MDS mappings discussed in Theorem 4.2) can be easily done by reordering the table lookup inputs for next round. When the table index is 4 bits as in N11-b, bit manipulation within

bytes costs more processing time. This cost is compensated by the use of small tables, which can be easily cached during the program run. The bottom two rows of Table 5.3 lists the results of our 10-round AES implementation and reference code in ANSI C2.0 [81], respectively.

Table 5.3: Experimental Results of 32-bit Implementations of Nested SPNs

Case	# rounds	Throughput (Mbits/s)		# lookups (4 rounds)	Comments
		non-optimized	optimized		
N1-a	32	18.16	45.65	96	similar to N2, N3, N6, N7
N4-a	32	22.58	60.86	64	similar to 32-round AES
N8	32	10.49	17.02	256	with uniform round
N11-b	32	16.06	36.21	128	similar to N12-b
N4-a	10	68.38	155.91	64	10-round AES (our code)
N4-a	10	119.51	120.64	64	reference AES code

Table 5.4: Experimental Results of Two Real Ciphers

Cipher	Throughput (Mbits/s)		# lookups
	non-optimized	optimized	
AES (10 rounds, 32-bit)	32.7	55.3	160
Simplified Camellia (18 rounds, 32-bit)	20.1	72.5	288
Simplified Camellia (18 rounds, 64-bit)	35.2	87.3	144

The experimental throughput results for AES and Camellia using the GNU C++ compiler on a 64-bit Alpha machine (COMPAQ AlphaServer DS10) are listed in Table 5.4. The 32-bit implementations of AES and Camellia are tested on this machine by using 32-bit data type arrays to store lookup tables and 32-bit operations for XORs. Before optimization, the throughput and the inverse of estimated number of lookups follow the same trend. After optimization, the 32-bit Camellia implementation is largely improved and close to its 64-bit implementation. With the smallest

number of lookups, the 64-bit simplified Camellia is still the fastest after optimization. It should be noted that such software comparison of Table 5.4 is purely based on software throughput. When security is considered simultaneously as shown in Figures 5.1 and 5.2, AES has a higher performance (as a typical SPN corresponding to N4-a) than Camellia. This reflects the fact that the Camellia cipher has a lower security margin than the AES cipher. Note that it is not meaningful to compare the numbers in Table 5.3 to Table 5.4 as the implementation platforms are different.

5.3 Alternative Implementations

Besides the table lookup approach, a block cipher can be implemented in other ways based on its structure. We briefly discuss these alternatives without a full exposé of their characterization because they apply to specialized circumstances rather than having general application.

5.3.1 Bitslice Implementations

For some cipher cases, a bitslice software implementation derived from the gate level circuit may be more efficient in parallelized applications [34]. A bitslice design is suitable for the cases whose synthesized circuits are compact. A w-bit processor can be regarded as w bit-processors in parallel. The gate level network circuit is described with instructions in software. Each bit in hardware corresponds to a word in software and each word is the concatenation of bits belonging to w separate blocks. Given enough registers in a processor, the memory requirement is negligible since no table lookups are necessary. Typically, the bitslice technique can be applied in three ways:

(a) *Parallel blocks*: This is the classic bitslice implementation. A total of w plaintext blocks are reorganized so that the bits at the same bit positions of different original blocks are now collected in one register. The number of registers required to store these blocks is equal to the block size of the cipher. Then, all registers are used as signals to a gate network deduced from a hardware implementation. Each gate in the network corresponds to a logic operation in software. The output signals, w bits each, are converted to their original format as w ciphertext blocks. Whether a cipher case discussed in this chapter is suitable for bitslice implementation can be determined from its space performance value η_s in hardware, which was investigated in Chapter 4. When η_s is high, a compact gate network can be used. The gate count of the circuit determines the number of instructions used in the bitslice software. Thus, a high η_s indicates a small number of clock cycles in software.

(b) *Bitslice cipher*: Serpent [33] is an example of an internal bitslice implementation. In Serpent, each 128-bit block is expressed as four 32-bit words after a bit permutation. S-boxes in each round can be regarded as 32 sets of parallel and identical 4×4 gate networks. A word is the collection of 32 bit signals, each corresponding to its own set of gate networks at the same locations. The other cipher operations can also be easily expressed by words. At the end of encryption, the bits of the four output words are permuted to form a 128-bit block.

(c) *Within special linear operations*: It has been shown that several parallel MDS mappings can be concatenated into one big mapping. When the number of parallel MDS mappings in each round, denoted by w', is at least 8, there is a

more subtle bitslice method within the round structure, as used in Hierocrypt's MDS mapping [37]. The linear expression of each output bit is extended to the expression of words, whose size is w' bits. The input and output bit variables are replaced with word variables, each including w' adjacent bits. Such a method works for any concatenated linear transformation with a convenient number of parallel sets. This parallel structure within a specific operation avoids the overhead caused by the block representation transformation between the standard form and bitslice parallel form as required in (a).

5.3.2 Power Implementations

Although the table lookup method is very efficient, the memory required for table storage is usually too large for a smart card, which has a restricted memory size. Hence, it is desirable to utilize a fast implementation for a smart card application that does not require large tables. Because an S-box just requires a small array, the main concern is then how to perform MDS mappings with low memory cost.

Defined as 1-bit left shift followed by bit-wise XOR with an appropriate irreducible polynomial in [5], the *xtime* operation can be used to perform multiplications for the MDS mapping. The operation *xtime* has no table lookups and the matrix multiplication is easier when all coefficients meet two requirements: (1) low Hamming weights and (2) low value. It is easy to find an MDS mapping satisfying these two requirements. When an SPN uses this mapping, however, the mapping's inverse used for decryption does not necessarily satisfy the two conditions and many more operations are therefore needed, making decryption much slower than encryption.

As we know, any element in a finite field can be expressed by both its power representation and its polynomial representation. As a result, the multiplication of

two elements can be realized by index addition on their power representations [22]. Here we examine the software efficiency when this approach is used for cipher implementations on a processor. Note that this approach cannot be used for AES which has its MDS mapping based on an irreducible but not primitive polynomial.

Suppose $poly(\cdot)$ returns the polynomial representation of a GF(2^n) element from the index of power representation and its inverse function is denoted as $pow(\cdot)$. We know that

$$Y = C \cdot X = poly((pow(C) + pow(X)) \bmod (2^n - 1))$$

when $C \neq 0, X \neq 0$, and where Y is in polynomial representation. If the processor records the carry bit c for n-bit addition $pow(C) + pow(X)$, the modulo operation can be bypassed:

$$Y = C \cdot X = poly((pow(C) + pow(X)) + c) \ .$$

Using this method to perform the MDS mapping after substitution, C indicates one coefficient in MDS generation matrix $\mathcal{C}$. Each coefficient C in $\mathcal{C}$ is constant and nonzero. Denote $pow(C)$ as C_{pow}. If X is the output of an $n \times n$ S-box $S(\cdot)$ with input Z, substitution $S(\cdot)$ can be merged into the above operations:

$$Y = C \cdot X = poly(C_{pow} + pow(S(Z)) + c) \ .$$

Therefore, each multiplication over Galois fields costs two additions and two table lookups. The two tables for $poly(\cdot)$ and $pow(S(Z))$ need 2^{n+1} bytes in total. It can be seen that the nature of the coefficients in generation matrix $\mathcal{C}$ does not affect the speed of multiplication. If the coefficients in $\mathcal{C}$ are randomly selected, this method

is more efficient than the *xtime* method and, hence, does provide better balance between the speeds of encryption and decryption.

5.3.3 General Comparison of Methods

Table 5.5 gives a general comparison of the software implementation methods discussed in this section.

Table 5.5: Comparison of Software Methods Used in MDS Codes

Method	Speed	Memory	Universal	MDS coefficents		S-box/MDS operations merged?
				affect speed	affect memory	
Table lookups	fast	large	yes	no	no	yes
Bitslice	parameter-dependent	none	no†	yes	no	no
Power	slow	small	yes‡	no*	no	yes
xtime	slow	none	yes	yes	no	no

† : the number of parallel sets should be compatible with machine operand sizes.
‡ : the polynomial to define the finite field must be primitive.
* : it can make a small difference depending on how many coefficients are 1s in C.

5.4 Summary

We have considered the software performance of two cipher structures composed of S-boxes and MDS mappings. Various cipher cases were generated from these structures with different component configurations. Table lookup implementations were used to evaluate the software efficiency of the various cases. A performance metric was defined to capture the security and efficiency simultaneously. With the

tendency similar to hardware performance, cases using 8×8 S-boxes are faster than cases using 4×4 S-boxes and nested SPNs are more efficient in obtaining security than Feistel networks. Specifically, AES and Camellia were analyzed in terms of software performance, and some interesting performance features were noted and confirmed through experimental results. Three other software implementation methods that are applicable in special circumstances were also discussed and their general advantages and disadvantages were listed.

Chapter 6

Applicability of XSL Attacks

Many comparisons of security, complexity, and performance have been made in the previous chapters among block ciphers composed of 4×4 and 8×8 S-boxes. The security estimate is based on differential, linear, and integral cryptanalysis. Recently, it has been found that many block ciphers can be described by overdefined systems of quadratic equations. Although solving Multivariate Quadratic (MQ) equations is NP-hard, it is observed in [82, 83] that the complexity could be subexponential if the equations are overdefined. As a result, the complexity evaluations claimed in [11, 84] to break AES and Serpent by solving an overdefined set of quadratic equations are lower than in an exhaustive search. Since the computation is too large, neither the two ciphers nor their simplified variants with a reduced number of rounds have been practically attacked as of now. However, for cipher designers, the potential attack may be a security concern because efficient algorithms to solve overdefined MQ equations could be found in the future.

In this chapter, while we do not study the details of the attack, some properties are discussed based on a toy cipher example presented in [11]. Moreover, a method to evaluate a cipher's susceptibility to this attack is proposed with results

applied to several currently proposed block ciphers and a new S-box design criterion is presented.

6.1 Introduction to XSL Attacks

A block cipher can be sometimes described by a system of MQ equations. The equations involve bit values of input, output, key, and intermediate data. When the unknown bit values are not more than the equations, it is possible to get the key bits as part of the equation system solution. For example, Figure 6.1 shows a very simple partial encryption system[1] from a 2-bit plaintext (x_1, x_0) to a 2-bit ciphertext (z_1, z_0). The S-box performs substitution as shown in Table 6.1. A 2-bit key (k_1, k_0) is mixed with the S-box output (y_1, y_0) by XORs.

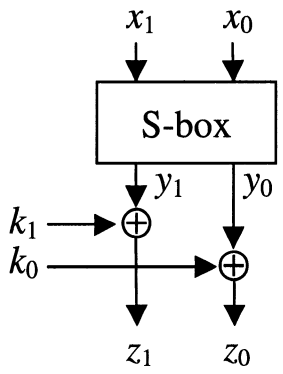


Figure 6.1: A Simple Example

Table 6.1: Mapping Table of a 2×2 S-box

Input x_1x_0	0	1	2	3
Output y_1y_0	3	2	0	1

[1]Such a trivial toy system is, of course, trivial to break and is simply for illustration purposes.

The S-box in Figure 6.1 can be described by the equations over GF(2) as shown in (6.1).

$$\begin{cases} 1 = x_0 + x_1 + y_0 \\ 1 = x_1 + y_1 \\ 0 = x_0x_1 + x_0y_0 \\ 0 = x_0 + x_0x_1 + x_0y_1 \\ 0 = x_0x_1 + x_1y_0 \\ 0 = x_1y_1 \\ 1 = x_0 + x_1 + x_0x_1 + y_0y_1 \end{cases} \tag{6.1}$$

We denote r as the number of possible linearly independent quadratic equations for an S-box and denote t as the number of different terms (including "1") in these equations. In this case, $r = 7$ and $t = 11$. An equation system is *overdefined* if r is much larger than the size of the S-box, e.g., 2 bits in this example. Two other equations can be written for key mixture:

$$\begin{cases} z_0 = x_0 + k_0 \\ z_1 = x_1 + k_1 \end{cases} \tag{6.2}$$

If a plaintext and its ciphertext are obtained, we can easily determine the key (k_1, k_0) by solving the equation system consisting of (6.1) and (6.2), i.e., 4 unknown variables y_1, y_0, k_1, and k_0 with 9 equations. When the quadratic equation system becomes larger and contains many more unknown variables, however, it will be hard to get the solution. In fact, solving systems of MQ equations is an NP-hard problem [82].

It has been observed that the complexity to find the solution drops significantly

if the equations are overdefined. Moreover, if the equations are sparse, the complexity could be further reduced. Three algorithms have been proposed to solve overdefined MQ problems. They are relinearization [82], XL [85], and XSL [11] algorithms. Based on relinearization and XL algorithms, XSL stands for "eXtended Sparse Linearization" or "multiply(X) by Selected monomials and Linearization".

The XSL attack on a block cipher begins with the initial equations for each $n \times n$ S-box, which has r equations and t terms. Based on these equations derived from S-boxes, a system of equations is then written for the whole cipher. Assuming at least one pair of plaintext and ciphertext is known, the intermediate bits and key bits are unknown variables to be solved in these equations. Each equation of an S-box is multiplied by all possible terms for all subsets of $(\beta - 1)$ other S-boxes, where the parameter β is a positive integer selected during the attack. Then, each term of high degree is considered as a new variable and Gaussian elimination is performed. It is assumed that at least one univariant equation (i.e., with only one unknown variable but the equation contains the powers of the variable in this case) can be generated through Gaussian elimination [85]. Such an equation can be solved with Berlekamp's algorithm [86]. With the results of possible univariable equations, the former equation systems can be simplified and a similar process can be repeated until all variables are determined.

To perform a general XSL attack (the first XSL attack in [11]), one working condition has to be satisfied. Denote T as the number of terms in the equations and T' as the number of terms that can be multiplied by one original bit variable and still belong to the set of T terms. Denote $Free$ as the number of linearly independent equations that are newly generated. In [11], the working condition of XSL attacks is

given as

$$\frac{Free}{T - T'} > 1 \tag{6.3}$$

and the complexity of the attack is evaluated by

$$T^{\omega} \approx (\frac{t - r}{n})^{\omega\beta} \cdot (\text{Block Size})^{\omega\beta} \cdot (\text{Number of rounds})^{2\omega\beta} \tag{6.4}$$

where the coefficient ω is the exponent associated with the complexity of Gaussian elimination. Typically, $\omega = 3$. An improved method presented in [87] can lead ω to be no more than 2.376. This upper bound of ω is used in the complexity evaluation of (6.4).

Applying the XSL attack on AES of 128-bit blocks and 256-bit keys, the expected workload is $T^{\omega} \approx 2^{298}$, which is higher than that for an exhaustive search. Further, a new cipher BES is defined in [84] which uses simple algebraic operations over GF(2^8). It has been shown in [84] that AES can be regarded as a special case of BES. As a result, by describing AES in its BES equivalent form, an extremely sparse and overdefined multivariant quadratic systems can be constructed over GF(2^8). In this approach, it is expected that even AES of 128-bit keys may also be vulnerable with the estimated complexity $T^{\omega} \approx 2^{87}$ [11, 84]. The complexity to break Serpent using a key of 256 bits is evaluated as $T^{\omega} \approx 2^{210}$ [11]. For each attack, the value of β must be large enough to ensure that $Free > T - T'$. However, a small increase of β (even 1 or 2) results in a much larger workload.

6.2 Effectiveness of the Attack

Since the XSL attack has not yet been demonstrated to work as claimed by its authors, it is still unclear whether it should be regarded as a serious security threat. Some skepticism of this attack has been raised in the cryptology community [88, 89].

It appears that the XSL attack could be applied in theory to break AES and Serpent. However, two main issues have to be noted when effectiveness is scrutinized. Firstly, the working condition stipulated in (6.3) is necessary but not sufficient for the success of the attack. Secondly, a sub-exponential (or even polynomial) complexity is largely based on the conjecture that β grows very slowly (or is even constant) when the number of rounds or the block size increases.

Due to the difficulty in undertaking the massive computation, the effectiveness of an attack is typically based on the simulation of simplified versions of targeted ciphers. For example, differential and linear attacks are well believed to work because the expected complexity can be demonstrated well on DES with a reduced number of rounds [39, 40]. Until now, the only achievable simulation of the XSL attack on a block cipher came from the Appendix of [90]. A "toy" cipher is targeted for the simulated attack, which is an SPN composed of 3×3 S-boxes. Each round of the cipher contains round key XORing, a layer of parallel S-boxes, and bit permutation. Applying XSL to the toy cipher with different block sizes and numbers of rounds, the working condition is satisfied within a reasonably small number of rounds. The results from [90] are presented[2] in Table 6.2.

For each case in Table 6.2, it is emphasized in [11, 90] that the number of rounds does not cause the complexity to increase exponentially. When the block size of the

[2]The complexity is calculated by T^{ω} where $\omega = 2.376$.

Table 6.2: Maximum Number of Rounds for a Toy Cipher to Satisfy XSL Working Condition

# S-boxes/round	Block size	# rounds	Complexity
2	6	16	$2^{42.2}$
4	12	8	$2^{42.3}$
8	24	4	$2^{42.4}$

toy cipher is fixed (6, 12, or 24 bits), the values of $\frac{Free}{T-T'}$ appear to "either converge to a fixed value, or they decrease very slowly" [90] as the number of rounds increases. However, a trend different from such an optimistic conjecture seems to be overlooked. When the block size of the cipher increases, the maximum number of rounds for which the working condition is satisfied decreases. As a result, it is not surprising that the working condition cannot be satisfied for even one round of the toy cipher with a large block size, e.g., on the order of 128 bits. More extensive simulation will be necessary to understand the effectiveness of the attack when applied to realistically sized block ciphers.

6.3 Applicability to Cipher Structures

More research is required to clearly understand the actual complexity of XSL attacks. Even so, it is wise for a cipher designer to evaluate the potential security threat from this attack. As a necessary requirement to launch an XSL attack, the S-box must be able to be described by overdefined MQ equations. Based on this concept, we propose a new criterion to S-box design which is more straightforward and applicable than the security contribution Γ suggested in [11].

For an $n \times n$ S-box with input $X = (x_{n-1}, \cdots, x_0)$ and output $Y = (y_{n-1}, \cdots, y_0)$, the total number of possible terms $\{1, x_{n-1}, \cdots, y_0, x_{n-1}x_{n-2}, \cdots, y_1y_0\}$ in a quadratic

expression is: $t = n(2n-1)+2n+1 = 2n^2+n+1$. Denote $\{c_0, \cdots, c_{t-1}\}$ as the binary coefficients associated with the t terms in any possible quadratic equation. For each possible X, $0 \leq X \leq 2^n - 1$, we calculate the values of the t terms and denote them by $\{a_{X,0}, \cdots, a_{X,t-1}\}$. Then we can write 2^n equations to form a system with $\{c_0, \cdots, c_{t-1}\}$ as the unknown:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,t-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,t-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{2^n-1,0} & a_{2^n-1,1} & \cdots & a_{2^n-1,t-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{t-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} . \tag{6.5}$$

Each possible nonzero solution of $\{c_0, \cdots, c_{t-1}\}$ will form a quadratic expression of the S-box. Denote $\mathcal{A}$ as the matrix composed of $a_{i,j}$, $0 \leq i \leq 2^n - 1$ and $0 \leq j \leq t-1$, in (6.5). We have the rank of $\mathcal{A}$, denoted as $R(\mathcal{A})$, bounded by

$$\begin{aligned} R(\mathcal{A}) &\leq \min(2^n, t) \\ &\leq \min(2^n, 2n^2 + n + 1) . \end{aligned}$$

When $n \leq 6$, $R(\mathcal{A}) \leq 2^n < t$. Assuming the S-box is such that $R(\mathcal{A}) = 2^n$, we get the expressions of the 2^n terms' coefficients $c_0, \cdots, c_{2^n-1}$ by Gaussian elimination on (6.5). Without loss of generality, we suppose they are the first 2^n terms:

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{2^n-1} \end{pmatrix} = \begin{pmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,t-2^n-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,t-2^n-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{2^n-1,0} & b_{2^n-1,1} & \cdots & b_{2^n-1,t-2^n-1} \end{pmatrix} \begin{pmatrix} c_{2^n} \\ c_{2^n+1} \\ \vdots \\ c_{t-1} \end{pmatrix}$$

where the binary constants $b_{0,0}, \cdots, b_{2^n-1,t-2^n-1}$ are calculated from Gaussian elimination. Thus, $(t-2^n)$ independent vectors are obtained to express all solutions:

$$\begin{pmatrix} c_0 \\ \vdots \\ c_{2^n-1} \\ c_{2^n} \\ c_{2^n+1} \\ \vdots \\ c_{t-1} \end{pmatrix} = c_{2^n} \begin{pmatrix} b_{0,0} \\ \vdots \\ b_{2^n-1,0} \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + c_{2^n+1} \begin{pmatrix} b_{0,1} \\ \vdots \\ b_{2^n-1,1} \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} + \cdots + c_{t-1} \begin{pmatrix} b_{0,t-2^n-1} \\ \vdots \\ b_{2^n-1,t-2^n-1} \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \quad (6.6)$$

Therefore, $(t-2^n)$ linearly independent quadratic equations can be written for the S-box when $R(\mathcal{A}) = 2^n$. More generally, when $R(\mathcal{A}) \leq t$, the number of independent quadratic equations written for the S-box is $t - R(\mathcal{A})$. If $t - R(\mathcal{A}) > n$, the system is overdefined.

When $n > 6$, $R(\mathcal{A}) \leq t < 2^n$. If the S-box is randomly generated, it can be shown that the possibility that $R(\mathcal{A}) = t$ is very high. When $R(\mathcal{A}) = t$, there is no nontrivial solution for (6.5) (i.e., $\{c_0, \ldots, c_{t-1}\}$ must be all zeros). Therefore, the XSL attack cannot be launched. When $R(\mathcal{A}) < t$, the XSL is possible depending on the security contribution [11]:

$$\Gamma = (\frac{t-r}{n})^{\lceil \frac{t-r}{n} \rceil} .$$

In this case, $r = t - R(\mathcal{A})$. We get

$$\Gamma = (\frac{R(\mathcal{A})}{n})^{\lceil \frac{R(\mathcal{A})}{n} \rceil} .$$

It should be noted that $\Gamma = (t/n)^{\lceil t/n \rceil}$ when $R(\mathcal{A}) = t$. In this case, the complexity evaluation is meaningless because the attack definitely fails due to nonexistence of quadratic expressions for S-boxes. Therefore, the security contribution provided in [11] is not a straightforward way to evaluate the susceptibility of a cipher to the attack. Thus, we give an easy approach to evaluate the potential susceptibility of an S-box to the XSL attack as the following:

(a) *Calculate matrix $\mathcal{A}$ given the mapping table of the S-box.*

(b) *Calculate the rank of matrix $\mathcal{A}$, $R(\mathcal{A})$.*

(c) *Compare $R(\mathcal{A})$ with t. If $R(\mathcal{A}) = t$, the cipher is resistant to the XSL attack. Otherwise, the difference between t and $R(\mathcal{A})$ shows how susceptible the cipher may be to the XSL attack.*

The resistance to the XSL attack comes from the difficulty to describe S-boxes using quadratic equations. This approach can be used by cipher designers to ensure that, even if the XSL algorithm becomes practical, the cipher is still immune to this attack. It is important to note that ciphers based on 4×4 S-boxes cannot be easily made immune because $R(\mathcal{A}) \leq 2^n < t$ when $n = 4$, while for 8×8 S-boxes this is possible.

Table 6.3 shows the evaluated resistance of several block ciphers including 10 randomly generated 8×8 S-boxes. The S-boxes of AES, Camellia, Hierocrypt-3, and MISTY have similar algebraic structures which are power operations over finite fields. Such a structure has been proved (e.g., as in [91]) to be able to enhance the resistance to differential and linear attacks. The power operation can also be simplified by the equivalent operations in the composite field although the advantage in circuit synthesis is not as significant as expected (as shown in Chapter 3). For the

Table 6.3: Evaluated Susceptibility to the XSL Attack

Cipher	S-box size n	p_s	q_s	$t-R(\mathcal{A})$	Algebraic structure	Comments
Serpent	4	2^{-2}	2^{-2}	21	random	susceptible
AES Camellia Hierocrypt-3	8	2^{-6}	2^{-6}	39	power over GF	susceptible
Hierocrypt	8	$2^{-5.42}$	$2^{-5.08}$	23	unknown	susceptible
MISTY S7	7	2^{-6}	2^{-6}	21	power over GF	susceptible
MISTY S9	9	2^{-8}	2^{-8}	36	power over GF	susceptible
Anubis Khazad	8	2^{-5}	$2^{-3.83}$	0	random	immune
RS-1,9†	8	$2^{-4.68}$	2^{-4}	0	random	immune
RS-2,3,5,6†	8	$2^{-4.42}$	2^{-4}	0	random	immune
RS-4,7,8,10†	8	$2^{-4.19}$	2^{-4}	0	random	immune

† : RS-1,$\cdots$,10 are randomly generated by computer.

same reason, these S-boxes might be prone to be attacked based on some algebraic methods and the XSL attack encourages more attempts in this direction. On the other hand, S-boxes with this algebraic structure are only a very small subset of all possible bijective mappings. It is very easy to generate S-boxes randomly with the full rank of matrix $\mathcal{A}$. Therefore, it is necessary to consider the full rank of matrix $\mathcal{A}$ as a criterion for future S-box selection, when other criteria have been satisfied.

It should be noted that even if susceptibility is observed by this method, the success of an XSL attack is still based on the reasonableness of the complexity evaluated in [11]. The XSL attack cannot work on the ciphers proved to be immune in this method even if its complexity is implied to be low by the analysis in [11].

6.4 Summary

This chapter briefly discussed the effectiveness of XSL attacks and their applicability to block ciphers. An approach was proposed to check the immunity of an S-box to this attack. It has been shown that ciphers using 8×8 S-boxes can be easily designed to be immune to this attack while it is hard for ciphers using 4×4 S-boxes. The potential susceptibility of several block ciphers to XSL attacks was also analyzed using this approach.

Chapter 7

Simple Power Attacks on Cipher Key Schedules

This chapter explores a potential vulnerability when a block cipher is implemented in an 8-bit smart card environment. Introduced in [56], power analysis exploits the fact that the power consumption of some cryptographic implementations is dependent on the intermediate data values. It is indicated in [92] that many smart card processors demonstrate a roughly linear relation between the Hamming weight of the data and the power consumed at the associated clock cycle. The Hamming weight attacks against the key schedules of DES and AES were discussed in [93, 94]. It was shown in [94] that an AES cipher key could be deduced given accurate leakage information of Hamming weights and a pair of plaintext and ciphertext. The susceptibility of NESSIE candidates to power attacks was theoretically evaluated in [95], which mainly focused on differential power analysis and gave Camellia a high rank among others.

In this chapter, we apply a simple power analysis to Camellia's key schedule as a typical example and demonstrate that the attack works even if leakage information bears noise and distortion. Using the same typical leakage model, our attack on

Camellia runs faster than the attack on AES presented in [94] and does not require any pair of plaintext and ciphertext. More generally, a method is proposed to evaluate how vulnerable a block cipher is toward similar attacks. The countermeasures in terms of both design rationale and implementation are also suggested. The content of this chapter is also presented in [96].

7.1 Camellia's 128-Bit Key Schedule

The attack described in this chapter is focused on Camellia's 128-bit key schedule [6]. The attacking technique to be discussed can be easily modified for the 192- and 256-bit key schedules.

Camellia's 128-bit key schedule expands 26 subkeys of 64 bits from the original key K_L and another derived key K_A of 128 bits. Each subkey can be obtained as one half of K_L or K_A after they are left rotated for a specific number of bits. This number can be 0, 15, 30 (only for K_A), 45, 60, 77 (only for K_L), 94, or 111, depending on the round number. During encryption or decryption, 18 subkeys are used for the round function in the 18 rounds. The other 8 subkeys are used for pre-, post-whitening and the FL-, FL^{-1}-functions.

K_A is derived from the original key K_L through a Feistel network. As shown in Figure 7.1, K_L is the input of such a network. The left half is the input to the same round function as in encryption. The round function can be divided into 3 steps: (1) a 64-bit constant, denoted as Σ_i for round i, is XORed with the input, (2) the S-function performs byte-wise bijective substitution, and (3) the P-function performs a linear transformation. The output of the round function is XORed with the right half of the round input. The two halves are then swapped. This Feistel

structure is iterated for a total of 4 rounds for the 128-bit key schedule. Note that the intermediate result after 2 rounds is XORed with K_L to form the next round input. To ease the description of the attack in later sections, each 64-bit block in Figure 7.1 is labelled as T_i, $0 \leq i \leq 17$.

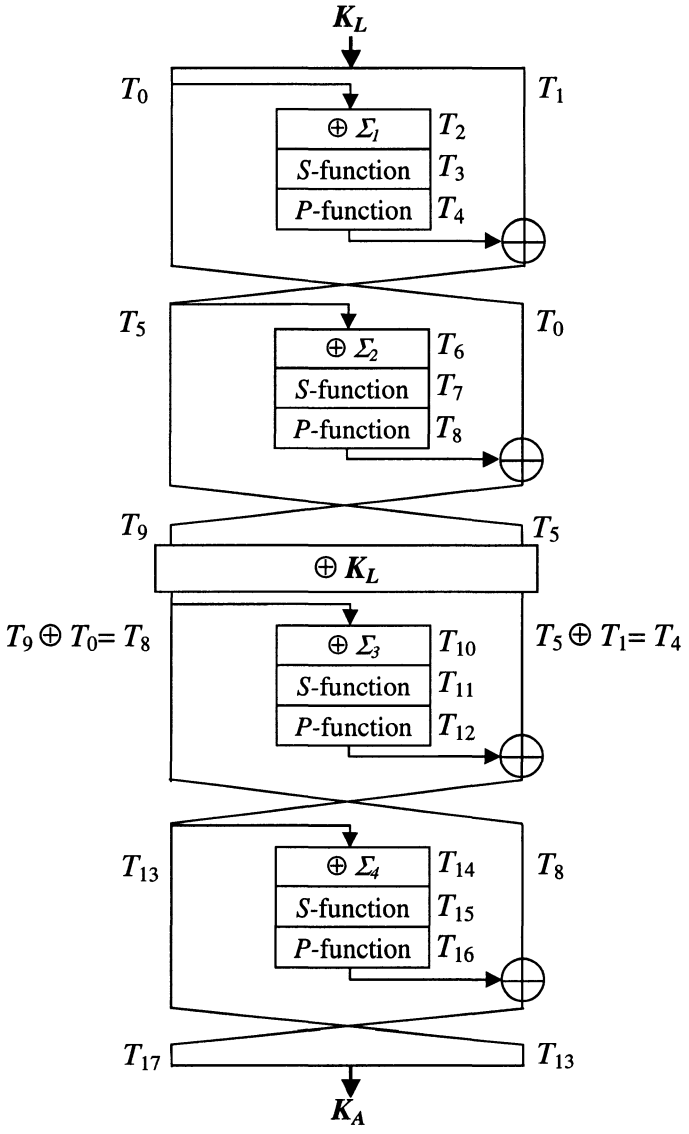


Figure 7.1: Camellia's 128-bit Key Schedule

7.2 Hamming Weight Attack

The Hamming weight attack exploits the relation between data and its Hamming weight derived by examining a power trace. If the Hamming weight can be captured from a poorly designed cryptographic device, we can use it to eliminate those data candidates failing to meet this relation. Given a Hamming weight of h for a particular byte, there are $\binom{8}{h}$ byte values consistent with this weight. Hence, as deduced in [57, 94], the number of byte values consistent with a Hamming weight is expected to be

$$\sum_{h=0}^{8} prob\{H = h\} \begin{pmatrix} 8 \\ h \end{pmatrix} = \sum_{h=0}^{8} \frac{1}{256} \begin{pmatrix} 8 \\ h \end{pmatrix}^2 \approx 50.27 \ . \tag{7.1}$$

Thus, to attack a block cipher with 128-bit key running on an 8-bit processor, the leakage of Hamming weight information for each key byte straightforwardly enables attackers to reduce the possible key space from 2^{128} to $50.27^{16} \approx 2^{90.43}$. However, depending on the nature of a block cipher, the outcome of a Hamming weight attack could be much simpler than this reduced workload if many intermediate values are derived from a small subset of key or subkey bits. For example, the attack presented in this chapter exploits the redundancy in the key schedule of Camellia and is able to determine all key bits without knowledge of any plaintext and ciphertext pair. The complexity of our attack is very low, e.g., a processing time of 5 ms on a PIII computer.

7.2.1 Basic Power Leakage Model

A popular power leakage model was proposed in [92] with two assumptions. One assumption is that the processor leaks the Hamming weights of data being processed.

For example, for an XOR operation on bytes such as $Z = X \oplus Y$, information of the Hamming weights of X, Y, and Z can be derived by examining the processor power consumption. It is also assumed that the power consumed by the processor demonstrates a linear relation to the Hamming weight of the processed data. As defined in [92], the power consumption at a specific time j is

$$Power[j] = \varepsilon \cdot H[j] + L + n \tag{7.2}$$

where $H[j]$ is the Hamming weight at time j, L is the additive constant portion in the power trace, ε represents the incremental amount of power caused by each extra 1 in the Hamming weight, and n is a random variable with zero mean representing noise.

In the basic model of this chapter, we need not restrict the Hamming weight-power relation to be linear. Instead, we simply assume that the power consumption monotonically varies in relation to the change of the Hamming weight of processed data. Hence, the power consumption is

$$Power[j] = f(H[j]) + L + n \tag{7.3}$$

where $f(\cdot)$ is a monotonically increasing or decreasing function. We also assume that the influence of L and n can both be ignored by offsetting and averaging over several similar power traces. Therefore, the Hamming weight can be reliably quantized from $Power[j]$. Our attack discussed in this section is based on this basic model.

The transition count information of Hamming weight is not considered in this attack because the Hamming weight difference possibly measured between two sequential clock cycles is determined by the order of the code, which depends on coding

styles as well as the key schedule specification.

7.2.2 Requirements for the Attack

In general, in order to launch a Hamming weight attack, the following prerequisites have to be satisfied.

- *Access to the power consumption.* The attacker needs to collect the power consumption traces from the cryptographic device. A typical approach is to sample the current through a small resistor, which is inserted between external power or ground and its corresponding pin on the smart card.

- *Ability to identify the clock cycles for individual steps in the key schedule.* For example, if the attacker knows the implementation well (e.g., a former employee), the timing information can be easily determined. Alternatively, a general method is suggested in [93] to distinguish the periods used for the key schedule from periods associated with data processing. The basic idea is to execute the protocol many times on several smart cards, each with different user information. Then, statistical analysis is performed to identify those clock cycles in which the same card behaves similarly with various data to be encrypted but different cards behave differently even if the same data is encrypted. These clock cycles are assumed to be used for the key schedule. Within these periods, the attacker can identify the clock cycles for specific operations based on features of the key schedule.

- *Monotonic relation between power and Hamming weight.* The power consumed by the attacked device has to be at least a monotonic function of the Hamming weight of the processed data as indicated in (7.3), if not a linear function.

- *One pair of plaintext and ciphertext.* The Hamming weight attack is expected to reduce the key space to a small subset. The cipher key is then distinguished by checking which of the remaining keys can be used to encrypt the plaintext to the expected ciphertext. As shown later for Camellia, when enough Hamming weights can be collected, we can deduce all key bits with certainty without requiring a plaintext encryption and in this case, this requirement is not necessary.

7.2.3 Attack Against Camellia Subkey Generation

Our attack on Camellia is implemented through two steps. The first step exploits the rotational relations between K_L and the resultant subkeys; the second step will exploit relations in the derivation of K_A from K_L. Several 64-bit subkeys are derived from K_L through left rotation for a certain number of bit positions (denoted by "<<<"). Since attackers can only check the Hamming weight of each byte, the rotation offsets (15, 45, 60, 77, 94, 111) provide information determined by the equivalent shift in byte-oriented bit positions as given by the remainder when the rotation offset is divided by 8. Hence rotation offsets (15, 45, 60, 77, 94, 111) are equivalent to bit-position shifts of (7, 5, 4, 5, 6, 7). The resulting bit shifts (7, 5, 4, 5, 6, 7) to the left are equivalent to bit shifts (1, 3, 4, 3, 2, 1) to the right.

As shown in Figure 7.2, each rotation of K_L gives a chance to consider bits with a different byte partition due to the shift of bit-positions with bytes. Assuming $8m+4$ adjacent bits of K_L are unknown, up to $5m$ Hamming weights[1] collected through power measurement during subkey generation can be used to validate candidates for

[1] m Hamming weights from K_L and up to $4m$ from 4 rotations of K_L giving bit shifts of 7, 6, 5, and 4. Because the left half of (K_L <<< 60) is not present in subkeys, the number of collected Hamming weights is between $4m$ and $5m$ according to the location of the $(8m+4)$-bit chunk.

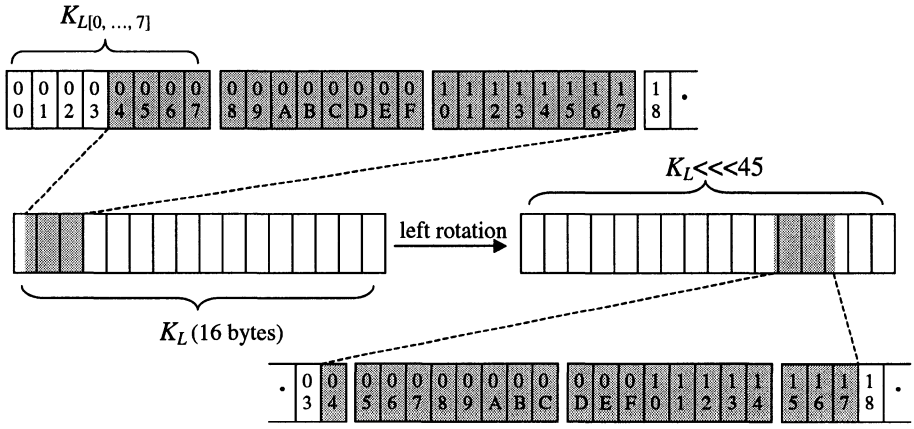


Figure 7.2: An Example of Camellia Subkey Generation
(gray bits: assumed $(8m+4)$-bit chunk where $m=2$)

these key bits. In general, the value of m indicates a tradeoff between speed and effectiveness. A large m causes a long searching time, while a small m may not provide enough Hamming weight checks to significantly reduce the key space (e.g., 5 checks on 12 bits when $m=1$ vs. 10 checks on 20 bits when $m=2$).

In order to increase the attacking speed, a dynamic pruning method is used instead of exhaustive search over all $8m+4$ bits. Firstly, the left-most 12-bit chunk of the $8m+4$ bits under investigation is examined. It is expected that most 12-bit candidates cannot meet the 5 Hamming weights[2] of bytes obtained by different byte partitions of K_L during subkey generation. Since the required Hamming weights are known by an attacker, those invalid candidates are discarded. For each of the resulting candidates for these 12 bits, the next adjacent 8 bits in $(8m+4)$-bit chunk are examined in terms of Hamming weights newly obtained by different byte partitions among the current candidate and these 8 unknown bits. If these hypothetical Hamming weights match values obtained from power measurements, the corresponding

[2]If any byte derived from rotating these 12 bits is located at the left half of (K_L <<< 60), one Hamming weight is not available from measurement.

8 bit candidates are concatenated to the current 12-bit candidate to form a larger partial key candidate of 20 bits. In the same way, the next adjacent 8 bits keep being examined until the whole set of $(8m+4)$-bit partial key candidates are determined.

In our attack, K_L is divided into 4 overlapped parts $K_{L[124\sim127,0\sim31]}$, $K_{L[28\sim63]}$, $K_{L[60\sim95]}$, and $K_{L[92\sim127]}$ so that they can be processed quickly and independently. Each part produces a number of 36-bit candidates (i.e., $m = 4$). Any four candidates from these four parts can be joined into one K_L guess when their overlapped bits are consistent. When this procedure is applied to Camellia's key schedule with 20 randomly generated cipher keys, an average of about 2^{38} candidates of the full K_L pass this step. We could use an exhaustive search on these 2^{38} candidates to find the key, but there is another way to uniquely determine the key as we shall see in the next section.

7.2.4 Attack Against the Derivation of K_A

In this section, we examine the second step in the attack, which gains more key information from the steps involved in the derivation of key K_A. Although the structure to derive K_A has a very good avalanche effect as well as non-linearity, it can be easily broken using a Hamming weight attack.

In the first round illustrated in Figure 7.1, each byte of K_L's left half (denoted as T_0), is XORed with constant Σ_1. The result is denoted as T_2. The following S-function is byte-wise substitution. These two steps are expressed as

$$\begin{aligned} T_2 &= T_0 \oplus \Sigma_1 \\ T_3 &= S(T_2) \ . \end{aligned}$$

If we still use $K_{L[124\sim127,0\sim31]}$ and $K_{L[28\sim63]}$ separately to prune partial key space as described in Section 7.2.3, two more Hamming weight checks for each hypothetical byte can be performed by comparing the Hamming weights in T_2 and T_3 with the corresponding values from measurement. Each output byte of the P-function ($T_4 = P(T_3)$) depends on at least 5 input bytes. To continue the candidate pruning, we combine any two candidates of $K_{L[124\sim127,0\sim31]}$ and $K_{L[28\sim63]}$ with consistent overlapped bit values to form 8 byte guesses of K_L's left half $K_{L[0\sim63]}$, denoted as T_0'. The output of round function with input T_0' is denoted as T_4'. If $T_0' = T_0$, then $T_4' = T_4$. Because the Hamming weight per byte in T_4 is known, another 8 Hamming weight checks can be performed to examine each T_0'. In most cases, only 1 or 2 possible candidates of the left half of K_L can pass this step.

For each T_0' remaining, the right half of K_L (denoted as T_1) is guessed. The second Feistel round in Figure 7.1 is expressed as

$$\begin{aligned} T_5 &= T_4 \oplus T_1 \\ T_6 &= T_5 \oplus \Sigma_2 \\ T_7 &= S(T_6) \\ T_8 &= P(T_7) \ . \end{aligned}$$

Similarly to the left half guess, $K_{L[60\sim95]}$ and $K_{L[92\sim127]}$ can be considered separately to prune the partial key space by using three more Hamming weight checks for each byte in $T_5 \sim T_7$. Then, any two candidates of $K_{L[64\sim99]}$ and $K_{L[96\sim128,0\sim3]}$ with consistent overlapped bit values are combined to form a candidate of K_L's right half $K_{L[64\sim127]}$, denoted as T_1'. The output of the round function with input $T_0' \oplus T_4'$ is denoted as T_8'. If $T_0' = T_0$ and $T_1' = T_1$, then $T_8' = T_8$. Thus, another 8 Hamming

weight checks can be performed to validate each T_1' candidate.

It is unlikely that a wrong guess T_0' of K_L's left half leads to a candidate T_1' that can pass all of the above Hamming weight checks. Even when such a case happens, similar Hamming weight checks can be continued with T_9 to T_{17} so that the original key is identified uniquely. The attack could also stop at any point and a brute force search would be executed on remaining key candidates.

We applied this attack to Camellia's 128-bit key schedule with 10,000 randomly generated sample keys. The experimental results listed in Table 7.1 show that 2 rounds of Hamming weight checks in K_A's derivation is enough for unique key identification in most cases. It takes less than 5 ms to compute the possible key candidate(s) using a PIII 933MHz computer with 512 MB memory.

Table 7.1: Experimental Attack Results with 10^4 Samples of 128-Bit Camellia Cipher Keys

Scope of HW checks in K_A's derivation	$T_0 \sim T_7$	$T_0 \sim T_8$	$T_0 \sim T_9$	$T_0 \sim T_{10}$
Percentage of cases with unique key identification	14.04 %	97.49 %	99.98 %	100 %
Ave. # of spurious keys	5.3588	0.0264	0.0002	0

7.2.5 Extension to 192-Bit and 256-Bit Key Schedules

When the key size is 192 or 256 bits, K_L is the first 128 key bits. The remainder of the key is denoted as K_R, which is also rotated to generate subkeys. For 192-bit keys, K_R's right 64 bits are padded with the complement of its left 64 bits. The input of K_A's derivation is changed from K_L to $K_L \oplus K_R$. Another derived key K_B is obtained through two rounds of Camellia's encryption structure with $K_A \oplus K_R$ as input.

Similar to the attack against the 128-bit key schedule but in the reverse direction, the attack begins with the last round of the Feistel structures used to derive K_A and K_B. Combined with Hamming weight checks during the rotations used in the generation of subkeys, a small number of K_A and K_B candidates are expected to pass the test. For each combination of remaining K_A and K_B candidates, a unique K_R candidate can be determined so that only the 128-bit K_L needs to be deduced using a method similar to what has been shown in the previous section. It is highly unlikely for wrong guesses of K_A and K_B to deduce K_L and K_R to pass Hamming weight checks.

7.3 Two Variants of the Attack with Robustness to Measurement Errors

A Hamming weight attack is normally fast and easy to implement when all required Hamming weights are measured accurately. However, in real circumstances, imperfect measurement cannot be always avoided. An attacker could attempt to mitigate the measurement noise using some statistical methods (e.g., averaging) in order to keep measurement accuracy at a satisfactory level. The attack described in the previous section is not error-tolerant. As a result, a spurious key or no key could be recognized as the correct key when measurement noise is high enough to cause errors in the determination of Hamming weights. Two modified attacks are thus given to tolerate errors. We first present a model which incorporates errors into the determination of Hamming weights.

7.3.1 Noisy Power Leakage Model

Denote $h[j]$ as the Hamming weight quantized from the power trace at time j in this model. Since $f(\cdot)$ in 7.3 is not always linear and averaging over power traces may not be feasible to eliminate the effect of noise, the error during quantization needs to be considered. A number of wrong captures of clock cycles may also occur. This could be caused by imperfect understanding of timing information about the implementation such that the Hamming weight processed by an unrelated instruction may be wrongly recognized. We assume that power measurement noise can result in a Hamming weight quantization error of ± 1. Then, the real Hamming weight obtained from measurement equipment is modelled as

$$h[j] = \begin{cases} \{H[j]+1, H[j]-1\} & with\ prob = P_\alpha \\ rand([0, \cdots, H_{max}]) & with\ prob = P_\beta \\ H[j] & with\ prob = 1-P_\alpha-P_\beta \end{cases} \tag{7.4}$$

where P_α is the probability that $h[j]$ is wrongly quantized as its adjacent Hamming weight and P_β is the probability that the result is uniformly randomly taken due to wrong recognition of clock cycles. When $P_\alpha = P_\beta = 0$, the leakage model is equivalent to the basic model in Section 7.2.1.

It should be noted that the power consumption of some smart cards does not monotonically change with the Hamming weight of the data. For the two types of smart cards examined in [97], the power consumption regions of the data with adjacent Hamming weights are partly overlapping. The quantization errors due to these overlapped regions can be modelled using $P_\alpha \neq 0$ even though the required monotonic relation is not perfectly maintained. Thus, the cryptographic applications

on the smart card types examined in [97] are also susceptible to the attack variants described in this section.

7.3.2 Attack Variant 1 Robust Against Small Noise

The "small" noise mentioned here means that its effect is only able to cause an error no more than 1 on the measured Hamming weight. Such type of noise suits the power leakage model given by (7.4) where $P_\beta = 0$. To tolerate these small errors, the only modification in the attack is to change the method of Hamming weight checks. Instead of considering whether the two Hamming weights from a candidate byte partition and measurement (denoted by h' and h, respectively) are the same in order to determine the viability of the candidate, a candidate byte remains viable if $|h' - h| \leq 1$.

Since the current Hamming weight comparison is looser than equality checking, a wrong key guess is more likely to pass the test. This attack variant costs more time and memory because a wrong key guess may need more checks to be eliminated. However, Camellia's K_A derivation provides checks up to T_{17} and these can all be used to eliminate wrong keys. For a randomly generated key K_L[3], the processing times used to perform the attack are listed in Table 7.2. During the listed time periods with different error rates P_α, Hamming weight checks are performed until only the correct key remains. Note that when P_α is high, the processing time is short. This is because when the small measurement errors occur more frequently, it is more likely for candidate Hamming weight h' passing the current Hamming weight comparison to be farther from the Hamming weight of the actual key, thus, more

[3]The first randomly generated key is used as K_L which is {D7, 13, E8, 80, 5F, FD, E3, 9E, 1B, C6, CF, 4D, F4, C7, 66, EF} in hexadecimal.

likely for its associated key guess to fail in next Hamming weight comparison.

Table 7.2: Processing Times of Attack Variant 1 on a PIII 933MHz Computer

Error rate P_α	1	0.8	0.6	0.4	0.2	0
Processing time	13 mins	45 mins	7.2 hours	2.2 days	$\approx$7 days	$\approx$70 days

As shown in Table 7.2 for $P_\alpha = 0$, this attack variant has a processing time that is much longer than the original attack discussed in Section 7.2 which works with the assumption of no measurement errors. Thus, using this variant would not be good unless the effect of small noise is unavoidable (i.e., $P_\alpha \neq 0$).

7.3.3 Attack Variant 2 Robust Against Wide Range of Noise

Attack variant 1 overcomes the effects of small errors in Hamming weight measurement whether frequently happening or not. However, in some systems a wide range of noise may occur due to wrong recognition of clock cycles associated with Hamming weight measurements. When a clock cycle is wrongly recognized, h may be any integer among $[0, H_{max}]$ dependent on the data processed at that moment. The occurrence of this type of error is reflected in a nonzero value for P_β in (7.4). In this circumstance, attack variant 1 could lead to a correct byte failing a check and being eliminated and eventually to determination of an incorrect key. Attack variant 2, however, can be employed to attack the key schedule when a wide range of noise is unavoidable, i.e., $P_\beta > 0$.

Instead of dynamically pruning key guesses through a local Hamming weight comparison, a weighted comparison scheme is applied. Each Hamming weight check

now returns a weight w which measures the difference between $h^{'}$ and h:

$$w = W[|h^{'} - h|] \ .$$

The entry value of array $W[\cdot]$ depends on the error distribution and drops to 0 as the index rises (e.g., in our experiment, $W[0] = 5, W[1] = 2, W[2] = \cdots = W[H_{max}] = 0$). Let S_w denote the sum of returned values from n Hamming weight comparisons for the bytes of a partial key candidate. When a candidate partial key is true, it is expected that

$$S_w = \sum_{i=1}^{n} W[|h_i^{'} - h_i|] \approx (1-P_\alpha-P_\beta) \cdot n \cdot W[0] + P_\alpha \cdot n \cdot W[1] \qquad (7.5)$$

when $W[2] = \cdots = W[H_{max}] = 0$. Thus, the probability of the following inequality being true is quite high:

$$S_w \geq \eta \cdot (1-P_\alpha-P_\beta) \cdot n \cdot W[0] \qquad (7.6)$$

when n is large enough and $0 \leq \eta \leq 1$. A smaller η makes (7.6) more likely to be true, but allows more spurious partial keys to pass the test.

If all of the left half of K_L is hypothesized to calculate S_w, the processing time for an exhaustive search in 2^{64} candidates will be formidable. Therefore, we use a nested approach illustrated in Figure 7.3, which we label as EDST as indicated in the figure caption. The left half of K_L is divided into 3 parts. The weight sum S_w is calculated for each candidate partial key. Given a specific η, the candidates will be discarded if inequity in (7.6) cannot be satisfied. The remaining candidates are sorted according to S_w and only λ candidates with high S_w will be stored to form

larger candidate partial keys. Within affordable computation, the attack prefers a small value of η and a large value of λ so that the correct candidate will not be lost due to errors.

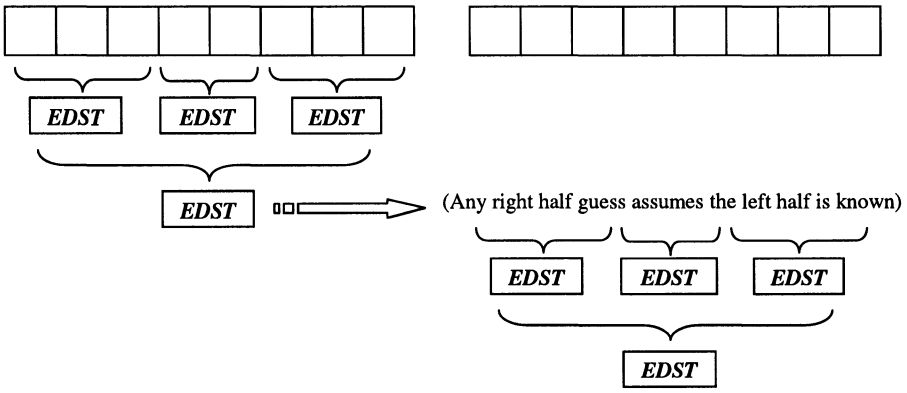


Figure 7.3: A Nested EDST Approach
(**E**: Evaluate S_w for each candidate; **D**: Discard if not satisfying (7.6);
S: Sort remaining candidates by S_w; **T**: Truncate and keep first λ candidates.)

In the experiment to attack Camellia's key schedule with 20 randomly generated keys as samples, the EDST approach has been run for 2, 3, and 8 byte candidate partial keys with $\eta = 0.5, 0.7$, and 0.8, respectively. The percentages of small noise and wide ranging noise are both 10% (i.e., $P_\alpha = P_\beta = 0.1$). When $\lambda = 256$, 30% of keys can be uniquely identified in an average time of 74 hours; 45% of keys will be uniquely identified with more processing time when $\lambda = 512$.

7.4 General Susceptibility Evaluation

The Hamming weight attack and its variants described in this chapter also work for the key schedule of some other ciphers. Two main measures are of interest for this attack: (1) the size of targeted partial key space (denoted by Ω) that the attack

Table 7.3: Susceptibility Evaluation for Several Block Ciphers

Ciphers	$\|\Omega\|$	ξ	Comments
AES	2^{40}	4.4	mainly exploit $\oplus$
Camellia	2^{8m+4}	≈ 6.22	exploiting rotation only
DES	2^{8m+8}	≈ 8	exploit rotation
IDEA	2^{8m+6}	≈ 6.5	exploit rotation
SAFER++	2^{8m+8}	≈ 24	exploit rotation and byte-wise addition
SHACAL-0	2^{32}	3.75	$\oplus$ without rotation hypothesize 1 byte in each word
SHACAL-1	2^{64}	≈ 3.5	$\oplus$ with rotation hypothesize 2 bytes in each word

begins with and (2) the average number of Hamming weight checks per byte in the targeted partial key space, denoted as ξ. An attacker hopes to find a scenario to reduce the candidates in Ω. A small Ω implies a low workload for exhaustive search within Ω. A high ξ leads to a small number of valid candidates left after attacking. Assuming the operations in the key schedule to be independent of each other, the number of candidates left per byte is expected to be $256(\frac{50.27}{256})^{\xi}$. This implies that when $\xi > 3.41$, it is possible to reduce the number of valid partial key candidates close to one. In a real attack, ξ has to be much larger than 3.41 because most operations are correlated (such as the fixed rotations of Camellia). For the attack in Section 7.2, $|\Omega| = 2^{36}$, ξ=6.22.

Table 7.3 shows the susceptibility of DES, IDEA, SAFER++ [98], AES (deduced from [94]), SHACAL-0 and SHACAL-1 [99] toward similar attacks. The values of $|\Omega|$ and ξ listed in this table are based on our assessment of values that can lead to a real attack. It is possible that more efficient attacking scenarios exist with more desirable $|\Omega|$ and ξ. No evident vulnerability to the attack from the key schedules of MISTY1 [38], Khazad [31], SHACAL-2 [99], and RC6 [8] are observed.

For a general Hamming weight attack as described in Section 7.2.3, a low value of $|\Omega|$ is desirable in order to search partial keys promptly (ideally, $m=1$ for ciphers listed in Table 7.3). In some cases, however, it is convenient to choose a larger Ω to facilitate attacking based on the nature of the key schedule as in Section 7.2.4.

7.5 Countermeasures

Hamming weight attacks, like other simple power attacks, work well only on susceptible cryptographic devices. Most countermeasures require additional operations and diminish performance. From the viewpoint of a cipher designer, a key schedule is resistant to a Hamming weight attack in nature if a good avalanche effect exists from the cipher key to subkeys as well as from one subkey to another. As a result, a very large Ω (ideally the whole key space) has to be hypothesized to get a value of ξ high enough for key identification. From the viewpoint of a system designer, a 16- or 32-bit smart card implementation is desirable because a larger word size decreases the number of possible Hamming weight checks and makes measurement harder and less accurate. Alternatively, a more resistant CMOS technology proposed in [100] can be used for smart cards if applicable. The power consumed by these types of circuits does not depend on the data being processed.

To provide resistance to a cipher already designed on 8-bit smart cards, the following countermeasures can be selected during implementation:

- *Data masking.* The approach of masking operations with random content is widely used to frustrate power analysis (e.g., in [101, 102, 103]). For example, $Z = X \oplus Y$ can be implemented with $Z = ((X \oplus R) \oplus Y) \oplus R$. The random data R enlarges $|\Omega|$ to $2^8|\Omega|$. Although the number of Hamming weight checks rises

from 3 to 6 in the operation including masking, the Hamming weight checks per byte in the targeted partial key space only changes to $(\xi \cdot \log_2 |\Omega|/8 + 3)/(\log_2 |\Omega|/8 + 1)$.

- *Operation Randomization.* Some operations in key schedules are commutative and distributive, e.g., $(X \oplus Y) <<< 1 = (Y <<< 1) \oplus (X <<< 1)$. It is hard for attackers to recognize the proper clock cycles from power traces if the program switches these equivalent operations randomly or data-dependently (e.g., reverse order of $\oplus$ and $<<<$ when X is odd). Thus, the measurement Hamming weight h could be unrelated to candidate Hamming weight h' due to wrong clock cycle recognition, which makes P_β larger. However, it is noted in many references (e.g., [102] and [104]) that neither reordering the instruction sequence nor adding delay between instructions guarantees safety if the attacker can run the protocols many times.

7.6 Summary

Camellia has a key schedule with high agility. K_A's derivation brings nonlinear properties into subkeys and gains more resistance to slide and related-key attacks. However, rotations used to generate subkeys provide enough information about K_L to compromise the key if Hamming weight information is available from power measurements. Further, the fact that K_L is used as the input of K_A's derivation structure provides attackers with enough information to launch a Hamming weight attack to uniquely identify the key. The Feistel structure of K_A's derivation gives many chances to verify the hypothesis. The two attack variants in Section 7.3 exploited this redundancy to gain robustness in the presence of errors in the Hamming weight

measurements. Consequently, when Camellia is implemented in the device with Hamming weight leakage, it is very important for implementors to consider appropriate countermeasures as discussed in Section 7.5. Many other ciphers can be shown to be susceptible to similar attacks and we proposed two measures to evaluate the susceptibility of a block cipher to such attacks.

Chapter 8

Conclusions

8.1 Contributions

This thesis analyzes the implementation, performance, and security of block ciphers. The main contributions of this research include the following:

Design and Analysis of Cipher Components

MDS mappings optimized for bit parallel hardware design are searched for in different finite fields (GF(2^2), GF(2^4), GF(2^8)). With different sizes and branch numbers, the resultant involution and non-involution MDS mappings provide many choices for a cipher designer to select. The decoder-switch-encoder model is proposed for invertible S-boxes and its hardware complexity is deduced after circuit simplification. The complexity evaluation is justified with synthesis realization targeted to 0.18 μm and 0.35 μm CMOS technologies.

Two AES hardware designs are implemented in hardware with different tradeoff of area and delay. The shortest round delay is 3.04 ns, which can lead to a very high throughput of up to 4 Gbits/s. By comparing the two designs, the effectiveness

of using composite finite field mathematics to realize the AES S-box (as suggested in [75, 77]) is examined and comments are given based on the ASIC synthesis result. The direct design (Design I) is a fast realization for non-feedback usage of AES. The subfield design (Design II) gives a better result with regards to the tradeoff between area and delay. Both area and delay are better than for the design of Rudra et al. [75] using the same complexity evaluation method.

Performance Characterization of Cipher Structures

The research provides mechanisms to compare cipher structures in terms of hardware and software performance before time-consuming realizations. Since the security is integrated into the performance measures, such mechanisms facilitate good understanding of efficiency and security at an early stage of a block cipher's design. Hence, the connection between cipher design and implementation is enhanced significantly.

The hardware complexity of cipher structures is evaluated on the basis of the complexity of components. Cipher security, in the form of resistance to differential and linear attacks, is used to normalize the performance in the analysis. By defining a set of metrics, the performance comparison is applied to cipher cases with different configurations of parameterized S-boxes and MDS mappings. Because the discussed structures are similar to many existing ciphers such as AES and Camellia, the analysis provides a meaningful mechanism for seeking efficient ciphers through a wide comparison of security, performance, and implementation methods.

Similarly, the software performance is compared for the cipher cases that have been considered. The software complexity is evaluated using a table lookup implementation, which is general and used for many block ciphers as fast implementations. The accuracy of the complexity evaluation is confirmed by coding typical cipher cases

on a PIII PC and an Alpha machine. The alternative software implementation methods are also generally compared.

Alternative Attacks on Block Ciphers

Differential and linear cryptanalysis are the most fundamental methods used by block cipher designers for security evaluation. However, other attacks on block ciphers are also considered in this work.

The XSL attack brings a new concept of security concern into cipher design. Although its complexity was conjectured with a very attractive outcome, an XSL attack has not been practically applied to break AES or any other published block cipher with even one round. Based on simulation results presented by its authors, the effectiveness of XSL attacks is considered. We proposed a method to check and ensure the immunity of a block cipher even if an XSL attack can be practiced.

The vulnerability of Camellia's key schedule to simple power analysis is observed and an attack is implemented for the first time. Further, two attack variants are also developed with robustness in the presence of measurement errors. Countermeasures and a general susceptibility evaluation method are suggested for implementation.

The discussion of the XSL attack in this thesis suggests new directions of S-box design and security evaluation. The Hamming weight attack is illustrated in detail so that the implementers can understand the possible threat and make efforts to avoid it.

8.2 Recommendations for Future Research

According to the results and experience obtained in this work, the following future research can be suggested:

- A model has been proposed to analyze hardware complexity of invertible S-boxes. The dynamic programming method can be used to reduce the redundancy of the decoder because the mapping table is invertible. It would be desirable to relate more cryptographic properties (e.g., nonlinearity, algebraic order, and resistance against attacks) to hardware complexity.

- The performance metrics have been defined as the integration of complexity/efficiency and security. In Chapters 4 and 5, we use the maximum characteristic probabilities P_d and P_l to deduce the resistance of a block cipher against differential and linear attacks, respectively. It is possible to use more advanced tools (such as in [41, 47, 105, 106]) for security evaluation. For example, in practice characteristics are combined to produce a differential used in a differential attack, where a differential probability can be higher than a characteristic probability. Similarly, a linear hull is used in a linear attack and the resultant workload can be expressed by the probability of the linear hull. However, new techniques are required to relate these advanced evaluation methods to the number of rounds for a cipher.

- In Chapter 6, it has been observed through experiment that a randomly generated S-box is very likely to be immune to an XSL attack. However, its probability is hard to calculate due to the complicated structure of matrix $\mathcal{A}$ in (6.5). It would be attractive to develop a theoretical method to determine

this probability as has been done with the probability that a randomly generated S-box gains high resistance to differential and linear attacks [107, 108]. A further step is to design an S-box with optimal properties with respect to the immunity to XSL attacks and security to differential and linear attacks. A more significant and challenging task is to determine the veracity of the XSL attack.

- A simple power analysis attack has been described in Chapter 7 which works well on Camelllia's smart-card solution if the processor leaks Hamming weight information. The simulated results show that the two variants are robust to measurement errors. An actual attack using data measured from a smart-card reader is required for further proof of the practicality of the attack.

References

[1] National Institute of Standards and Technology, "FIPS 46-3 Data Encryption Standard (DES)," Available at `csrc.nist.gov/publications/fips`.

[2] "IEEE standard specifications for public-key cryptography," in *IEEE Standard Documents - 1363*, 2000.

[3] Electronic Frontier Foundation Press Release. Available at `www.eff.org/Privacy/Crypto_misc/DESCracker`.

[4] RSA's DES Challenge III. Available at `www.rsasecurity.com/rsalabs/challenges`.

[5] J. Daemen and V. Rijmen, "AES proposal: Rijndael," 1999. Available at `csrc.nist.gov/encryption/aes/rijndael`.

[6] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: a 128-bit block cipher suitable for multiple platforms - design and analysis," in *Proceedings of Selected Areas in Cryptography - SAC 2000*, vol. 2012 of *Lecture Notes in Computer Science*, pp. 39–56, Springer-Verlag, 2001.

[7] New European Schemes for Signatures Integrity and Encryption (NESSIE) website. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[8] R. L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin, "The RC6 block cipher," August 2002. Available at `www.rsasecurity.com/rsalabs/rc6`.

[9] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C (2nd Edition)*. John Wiley & Sons, 1995.

[10] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: A 128-bit block cipher suitable for multiple platforms (NESSIE submission)," 2000. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[11] N. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations," in *Proceedings of Advances in Cryptology - ASIACRYPT 2002*, vol. 2501 of *Lecture Notes in Computer Science*, pp. 267–287, Springer-Verlag, 2002.

[12] D. R. Stinson, *Cryptography Theory and Practice (2nd Edition)*. Chapman & Hall/CRC, 2002.

[13] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *The Handbook of Applied Cryptography*. CRC Press, 1996.

[14] W. Stallings, *Cryptography and Network Security: Principles and Practice (2nd Edition)*. Prentice Hall, 1998.

[15] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital

signatures and public-key cryptosystems," in *CACM*, vol. 21, pp. 120–126, 1978.

[16] RSA Laboratories' Frequently Asked Questions About Today's Cryptography (Version 4.1). Available at `http://www.rsasecurity.com/rsalabs/faq/3-1-2.html`.

[17] W. Diffie and M. E. Hellman, "New directions in cryptography," in *IEEE Transactions on Information Theory*, vol. IT-22, pp. 644–654, November 1976.

[18] A. J. Menezes, *Elliptic Curve Public Key Cryptosystems.* Kluwer, 1993.

[19] C. E. Shannon, "Communication theory of secrecy systems," in *Bell System Technical Journal*, vol. 28, pp. 656–715, 1949.

[20] H. M. Heys and S. E. Tavares, "Substitution-permutation networks resistant to differential and linear cryptanalysis," in *Journal of Cryptology*, vol. 9, pp. 1–19, 1996.

[21] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. D. Win, "The cipher SHARK," in *Proceedings of Fast Software Encryption - FSE'96*, vol. 1039 of *Lecture Notes in Computer Science*, pp. 99–112, Springer-Verlag, 1997.

[22] R. Lidl and H. Niederreiter, *Finite Fields, Volume 20 of Encyclopaedia of Mathematics and its Applications.* Reading, Massachusetts: Addison-Wesley, 1983.

[23] J. Daemen, *Cipher and Hash Function Design Strategies Based on Linear and Differential Cryptanalysis.* PhD thesis, Katholieke Universiteit Leuven, 1995.

[24] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes.* Amsterdam: North-Holland, 1977.

[25] C. M. Adams and S. E. Tavares, "The structured design of cryptographically good S-boxes," in *Journal of Cryptology*, vol. 3, pp. 27–41, 1990.

[26] W. Meier and O. Staffelbach, "Nonlinearity criteria for cryptographic functions," in *Proceedings of Advances in Cryptology - EUROCRYPT'89*, vol. 434 of *Lecture Notes in Computer Science*, pp. 549–562, Springer-Verlag, 1990.

[27] K. Nyberg, "Perfect nonlinear S-boxes," in *Proceedings of Advances in Cryptology - EUROCRYPT'91*, vol. 547 of *Lecture Notes in Computer Science*, pp. 378–386, Springer-Verlag, 1991.

[28] J. Daemen, R. Govaerts, and J. Vandewalle, "Correlation matrices," in *Proceedings of Fast Software Encryption - FSE'94*, vol. 1008 of *Lecture Notes in Computer Science*, pp. 275–285, Springer-Verlag, 1995.

[29] A. Youssef, S. Mister, and S. Tavares, "On the design of linear transformations for substitution-permutation encryption networks," in *Proceedings of Selected Areas in Cryptography - SAC'97*, 1997.

[30] P. Barreto and V. Rijmen, "The Anubis block cipher," in *First Open NESSIE Workshop*, Leuven, November 2000. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[31] P. Barreto and V. Rijmen, "The Khazad legacy-level block cipher," in *First Open NESSIE Workshop*, Leuven, November 2000. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[32] National Institute of Standards and Technology, "FIPS 197 Advanced Encryption Standard (AES)," Available at `csrc.nist.gov/publications/fips`.

[33] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A proposal for the Advanced Encryption Standard." Available at `www.cl.cam.ac.uk/~rja14/serpent.html`.

[34] E. Biham, "A fast new DES implementation in software," in *Proceedings of Fast Software Encryption - FSE'97*, vol. 1267 of *Lecture Notes in Computer Science*, pp. 260–272, Springer-Verlag, 1997.

[35] R. L. Rivest, "The RC5 encryption algorithm," in *Proceedings of Fast Software Encryption - FSE'94*, vol. 1008 of *Lecture Notes in Computer Science*, pp. 86–96, Springer-Verlag, 1995.

[36] J. Daemen, L. Knudsen, and V. Rijmen, "The block cipher Square," in *Proceedings of Fast Software Encryption - FSE'97*, vol. 1267 of *Lecture Notes in Computer Science*, pp. 54–68, Springer-Verlag, 1997.

[37] K. Ohkuma, H. Muratani, F. Sano, and S. Kawamura, "The block cipher Hierocrypt," in *Proceedings of Selected Areas in Cryptography - SAC 2000*, vol. 2012 of *Lecture Notes in Computer Science*, pp. 72–88, Springer-Verlag, 2001.

[38] M. Matsui, "New block encryption algorithm MISTY," in *Proceedings of Fast Software Encryption - FSE'97*, vol. 1267 of *Lecture Notes in Computer Science*, pp. 54–68, Springer-Verlag, 1997.

[39] E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems," in *Proceedings of Advances in Cryptology - CRYPTO'90*, vol. 537 of *Lecture Notes in Computer Science*, pp. 2–21, Springer-Verlag, 1991.

[40] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Proceedings of Advances in Cryptology - EUROCRYPT'93*, vol. 765 of *Lecture Notes in Computer Science*, pp. 386–397, Springer-Verlag, 1994.

[41] X. Lai, J. L. Massey, and S. Murphy, "Markov ciphers and differential cryptanalysis," in *Proceedings of Advances in Cryptology - CRYPTO'91*, vol. 547 of *Lecture Notes in Computer Science*, pp. 17–38, Springer-Verlag, 1991.

[42] H. M. Heys, "A tutorial on linear and differential cryptanalysis," in *Cryptologia*, vol. XXVI, pp. 189–221, 2002.

[43] X. Lai, "Higher order derivatives and differential dryptanalysis," in *Proceedings of Symposium on Communication, Coding and Cryptography in honour of James L. Massey on the occasion of his 60-th birthday*, Monte-Verita, Ascona, Switzerland, 1994.

[44] L. R. Knudsen, "Truncated and higher order differentials," in *Proceedings of Fast Software Encryption - FSE'94*, vol. 1008 of *Lecture Notes in Computer Science*, pp. 196–211, Springer-Verlag, 1995.

[45] E. Biham, A. Biryukov, and A. Shamir, "Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials," in *Proceedings of Advances in Cryptology - EUROCRYPT'99*, vol. 1592 of *Lecture Notes in Computer Science*, pp. 12–23, Springer-Verlag, 1999.

[46] M. Kanda, "Practical security evaluation against differential and linear attacks for Feistel ciphers with SPN round function," in *Proceedings of Selected Areas in Cryptography - SAC 2000*, vol. 2012 of *Lecture Notes in Computer Science*, pp. 324–338, Springer-Verlag, 2001.

[47] K. Nyberg, "Linear approximation of block ciphers," in *Proceedings of Advances in Cryptology - EUROCRYPT'94*, vol. 950 of *Lecture Notes in Computer Science*, pp. 439–444, Springer-Verlag, 1995.

[48] L. Keliher, H. Meijer, and S. Tavares, "New method for upper bounding the maximum average linear hull probability for SPNs," in *Proceedings of Advances in Cryptology - EUROCRYPT 2001*, vol. 2045 of *Lecture Notes in Computer Science*, pp. 420–436, Springer-Verlag, 2001.

[49] B. S. Kaliski Jr. and M. Robshaw, "Linear cryptanalysis using multiple approximations," in *Proceedings of Advances in Cryptology - CRYPTO'94*, vol. 839 of *Lecture Notes in Computer Science*, pp. 26–39, Springer-Verlag, 1994.

[50] L. R. Knudsen and M. Robshaw, "Non-linear approximations in linear cryptanalysis," in *Proceedings of Advances in Cryptology - EUROCRYPT'96*, vol. 1070 of *Lecture Notes in Computer Science*, pp. 224–236, Springer-Verlag, 1996.

[51] E. Biham, "On matsui's linear cryptanalysis," in *Proceedings of Advances in Cryptology - EUROCRYPT'94*, vol. 950 of *Lecture Notes in Computer Science*, pp. 398–412, Springer-Verlag, 1995.

[52] F. Chabaud and S. Vaudenay, "Links between differential and linear cryptanalysis," in *Proceedings of Advances in Cryptology - EUROCRYPT'94*, vol. 950 of *Lecture Notes in Computer Science*, pp. 356–365, Springer-Verlag, 1995.

[53] M. Matsui, "On correlation between the order of S-boxes and the strength of DES," in *Proceedings of Advances in Cryptology - EUROCRYPT'94*, vol. 950 of *Lecture Notes in Computer Science*, pp. 366–375, Springer-Verlag, 1995.

[54] Toshiba Corporation, "Security evaluation: Hierocrypt-3." NESSIE Algorithm Submission, 2000. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[55] L. R. Knudsen and D. Wagner, "Integral cryptanalysis (extended abstract)," in *Proceedings of Fast Software Encryption - FSE 2002*, vol. 2365 of *Lecture Notes in Computer Science*, pp. 112–127, Springer-Verlag, 2002.

[56] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proceedings of Advances in Cryptology - CRYPTO'99*, vol. 1666 of *Lecture Notes in Computer Science*, pp. 388–397, Springer-Verlag, 1999.

[57] T. Messerges, E. Dabbish, and R. Sloan, "Examining smart-card security under the threat of power analysis attacks," in *IEEE Transactions on Computers*, vol. 51, pp. 541–552, April 2002.

[58] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Proceedings of Advances in Cryptology - CRYPTO'96*, vol. 1109 of *Lecture Notes in Computer Science*, pp. 104–113, Springer-Verlag, 1996.

[59] H. Handschuh and H. M. Heys, "A timing attack on RC5," in *Proceedings*

of Selected Areas in Cryptography - SAC'98, vol. 1556 of *Lecture Notes in Computer Science*, pp. 306–318, Springer-Verlag, 1999.

[60] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Proceedings of Advances in Cryptology - CRYPTO'97*, vol. 1294 of *Lecture Notes in Computer Science*, pp. 513–525, Springer-Verlag, 1997.

[61] E. Biham, "New types of cryptanalytic attacks using related keys," in *Journal of Cryptology*, vol. 7, pp. 229–246, 1994.

[62] A. Biryukov and D. Wagner, "Slide attacks," in *Proceedings of Fast Software Encryption - FSE'99*, vol. 1636 of *Lecture Notes in Computer Science*, pp. 245–259, Springer-Verlag, 1999.

[63] Helion Technology Technical Report, 2003. Available at `www.heliontech.com`.

[64] B. Weeks, M. Bean, T. Rozylowicz, and C. Ficke, "Hardware performance simulations of round 2 Advanced Encryption Standard algorithms," in *National Security Agency white paper*, 2000. Available at `www.nist.gov/aes`.

[65] T. Ichikawa, T. Kasuya, and M. Matsui, "Hardware evaluation of the aes finalists," in *Proceedings of 3rd AES conference*, pp. 279–285, 2000.

[66] H. Kuo and I. Verbauwhede, "Architectural optimization for a 1.82gbits/sec VLSI implementation of the AES Rijndael algorithm," in *Proceedings of Cryptographic Hardware and Embedded Systems - CHES 2001*, vol. 2162 of *Lecture Notes in Computer Science*, pp. 51–64, Springer-Verlag, 2001.

[67] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalists," in *Proceedings of 3rd AES conference*, pp. 13–27, 2000.

[68] K. Gaj and P. Chodowiec, "Comparison of the hardware performance of the AES candidates using reconfigurable hardware," in *Proceedings of 3rd AES conference*, pp. 40–56, 2000.

[69] B. Preneel, B. V. Rompay, S. Örs, A. Biryukov, L. Granboulan, E. Dottax, M. Dichtl, M. Schafheutle, P. Serf, S. Pyka, E. Biham, E. Barkan, Dunkelman, J. Stolin, M. Ciet, J.-J. Quisquater, F. Sica, H.Raddum, and M. Parker, "Performance of optimized implementations of the NESSIE primitives," tech. rep., NESSIE, February 2003. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[70] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback, "Report on the development of the Advanced Encryption Standard (AES)," tech. rep., U.S. National Institute of Standards and Technology (NIST), October 2000. Available at `csrc.nist.gov/encryption/aes`.

[71] L. Xiao and H. M. Heys, "Hardware design and analysis of block cipher components," in *Proceedings of the 5th International Conference on Information Security and Cryptology - ICISC 2002*, vol. 2587 of *Lecture Notes in Computer Science*, pp. 164–181, Springer-Verlag, 2003.

[72] E. Mastrovito, "VLSI design for multiplication over finite fields GF(2^m)," in *Proceedings of Applied Algebra, Algebraic Algorithms and Error-Correcting*

Codes - AAECC-6, vol. 357 of *Lecture Notes in Computer Science*, pp. 297–309, Springer-Verlag, 1989.

[73] C. Paar, *Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields.* PhD thesis, University of Essen, Germany, 1994.

[74] *Online Documentation on Synopsys Design Compiler*, v2000.05 ed.

[75] A. Rudra, P. Dubey, C. Jutla, V. Kumar, J. Rao, and P. Rohatgi, "Efficient Rijndael encryption implementation with composite field arithmetic," in *Proceedings of Cryptographic Hardware and Embedded Systems - CHES 2001*, vol. 2162 of *Lecture Notes in Computer Science*, pp. 171–184, Springer-Verlag, 2001.

[76] A. Rudra. Personal Communication.

[77] V. Rijmen, "Efficient implementation of the Rijndael S-box." Available at `www.esat.kuleuven.ac.be/~rijmen/rijndael`.

[78] L. Xiao and H. M. Heys, "Hardware performance characterization of block cipher structures," in *Proceedings of Cryptographers' Track RSA Conference 2003*, vol. 2612 of *Lecture Notes in Computer Science*, pp. 176–192, Springer-Verlag, 2003.

[79] M. Kanda, Y. Takashima, T. Matsumoto, K. Aoki, and K. Ohta, "Strategy for constructing fast round functions with practical security against differential and linear cryptanalysis," in *Proceedings of Selected Areas in Cryptography - SAC'98*, vol. 1556 of *Lecture Notes in Computer Science*, pp. 264–279, Springer-Verlag, 1999.

[80] L. Xiao and H. M. Heys, "Software performance characterization of block cipher structures," 2003. Submitted to IEEE Transactions on Computers.

[81] P. Barreto and V. Rijmen, "AES reference code in ANSI C." Available at `www.esat.kuleuven.ac.be/~rijmen/rijndael/`.

[82] A. Kipnis and A. Shamir, "Cryptanalysis of the HFE public key cryptosystem by relinearization," in *Proceedings of Advances in Cryptology - CRYPTO'99*, vol. 1666 of *Lecture Notes in Computer Science*, pp. 19–30, Springer-Verlag, 1999.

[83] A. K. Lenstra and A. Shamir, "Analysis and optimization of the TWINKLE factoring device," in *Proceedings of Advances in Cryptology - EUROCRYPT 2000*, vol. 1807 of *Lecture Notes in Computer Science*, pp. 35–52, Springer-Verlag, 2000.

[84] S. Murphy and M. J. Robshaw, "Essential algebraic structure within the AES," in *Proceedings of Advances in Cryptology - CRYPTO 2002*, vol. 2442 of *Lecture Notes in Computer Science*, pp. 1–16, Springer-Verlag, 2002.

[85] N. Courtois, A. Klimov, J. Patarin, and A. Shamir, "Efficient algorithms for solving overdefined systems of multivariate polynomial equations," in *Proceedings of Advances in Cryptology - EUROCRYPT 2000*, vol. 1807 of *Lecture Notes in Computer Science*, pp. 392–407, Springer-Verlag, 2000.

[86] E. R. Berlekamp, *Algebraic Coding Theory*, ch. Factoring Polynomials over Fnite Fields. New York: McGraw-Hill, 1968.

[87] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," in *Journal of Symbolic Computation*, vol. 9, pp. 251–280, 1990.

[88] B. Schneier, "More on AES cryptanalysis." Crypto-Gram Newsletter, available at `www.counterpane.com/crypto-gram-0210.html`, October 2002.

[89] T. Moh, "On the Courtois-Pieprzyk's attack on Rijndael." Available at `www.usdsi.com/aes.html`, September 2002.

[90] N. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations (preliminary version)," 2002. Available at `eprint.iacr.org/2002/044`.

[91] K. Nyberg, "Differential uniform mappings for cryptography," in *Proceedings of Advances in Cryptology - EUROCRYPT'93*, vol. 765 of *Lecture Notes in Computer Science*, pp. 55–64, Springer-Verlag, 1994.

[92] T. S. Messerges, "Using second-order power analysis to attack DPA resistant software," in *Proceedings of Cryptographic Hardware and Embedded Systems - CHES 2000*, vol. 1965 of *Lecture Notes in Computer Science*, pp. 238–251, Springer-Verlag, 2000.

[93] E. Biham and A. Shamir, "Power analysis of the key scheduling of the AES candidates," in *Second Advanced Encryption Standard (AES) Candidate Conference*, Rome, Italy, 1999.

[94] S. Mangard, "A Simple Power-Analysis (SPA) attack on implementations of the AES key expansion," in *Proceedings of the 5th International Conference on*

Information Security and Cryptology - ICISC2002, vol. 2587 of *Lecture Notes in Computer Science*, pp. 343–358, Springer-Verlag, 2002.

[95] E. Oswald and B. Preneel, "A theoretical evaluation of some NESSIE candidates regarding their susceptibility towards power analysis attacks," Technical report, Katholieke Universiteit Leuven, Dept. ESAT, October 2002.

[96] L. Xiao and H. M. Heys, "A simple power analysis attack against the key schedule of the Camellia block cipher," 2003. Submitted to Information Processing Letters.

[97] M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart, "Power analysis, what is now possible...," in *Proceedings of Advances in Cryptology - ASIACRYPT 2000*, vol. 1976 of *Lecture Notes in Computer Science*, pp. 489–502, Springer-Verlag, 2000.

[98] NESSIE Archive, "SAFER++ submission," 2000. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[99] H. Handschuh and D. Naccache, "SHACAL," 2000. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[100] K. Tiri, M. Akmal, and I. Verbauwhede, "A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards," in *Proceedings of the 28th European Solid-State Circuits Conference*, Florence, Italy, September 2002.

[101] L. Goubin and J. Patarin, "DES and differential power analysis," in *Proceedings*

of Cryptographic Hardware and Embedded Systems - CHES'99, vol. 1717 of *Lecture Notes in Computer Science*, pp. 158–172, Springer-Verlag, 1999.

[102] S. Chari, C. Jutla, , J. R. Rao, and P. Rohatgi, "A cautionary note regarding evaluation of AES candidates on smart-cards," in *Proceedings of the 2nd AES Candidate Conference*, March 1999.

[103] T. S. Messerges, "Securing the AES finalists against power analysis attacks," in *Proceedings of Fast Software Encryption - FSE 2000*, vol. 1978 of *Lecture Notes in Computer Science*, pp. 150–164, Springer-Verlag, 2000.

[104] R. Anderson and M. Kuhn, "Tamper resistance - a cautionary note," in *Proceedings of the 2nd USENIX Workshop Electronic Commerce*, pp. 1–11, 1996.

[105] K. Nyberg and L. R. Knudsen, "Provable security against differential cryptanalysis," in *Journal of Cryptology*, vol. 8, pp. 27–37, 1995.

[106] M. Matsui, "New structure of block ciphers with provable security against differential and linear cryptanalysis," in *Proceedings of Fast Software Encryption - FSE'96*, vol. 1039 of *Lecture Notes in Computer Science*, pp. 205–218, Springer-Verlag, 1996.

[107] L. O'Connor, "On the distribution of characteristics in bijective mappings," in *Proceedings of Advances in Cryptology - EUROCRYPT'93*, vol. 765 of *Lecture Notes in Computer Science*, pp. 360–370, Springer-Verlag, 1994.

[108] L. O'Connor, "Properties of linear approximation tables," in *Proceedings of Fast Software Encryption - FSE'94*, vol. 1008 of *Lecture Notes in Computer Science*, pp. 131–136, Springer-Verlag, 1995.

Appendix A

MDS Searching Results

This appendix contains the search results for the most efficient MDS in hardware for specific parameters. The hardware complexities are measured by the Hamming weights of the product matrices associated with the MDS generation matrices. The maximum delay is determined from the row with the maximum Hamming weight.

To simplify the expression of circulant, Hadamard, and Cauchy matrices, the following three functions are defined for the representation from tuples to square matrices $\{A_{i,j}\}$, $0 \leq i, j \leq k-1$:

$$\begin{aligned}
\{A_{i,j}\} = &\quad cir(\alpha_0, \cdots, \alpha_{k-1}) && \text{if } A_{i,j} = \alpha_{(i+j) \bmod k} \\
\{A_{i,j}\} = &\quad had(\alpha_0, \cdots, \alpha_{k-1}) && \text{if } A_{i,j} = \alpha_{i \oplus j} \\
\{A_{i,j}\} = &\quad cauchy((\alpha_0, \cdots, \alpha_{k-1}), (\beta_0, \cdots, \beta_{k-1})) && \text{if } A_{i,j} = 1/(\alpha_i \oplus \beta_j).
\end{aligned}$$

A normal matrix can be written as $\{\{A_{0,1}, \cdots, A_{0,k-1}\}, \cdots, \{A_{k-1,1}, \cdots, A_{k-1,k-1}\}\}$. By doing so, the MDS searching results can be straightforwardly presented in Tables A.1 and A.2.

Table A.1: Search Results of MDS Codes Optimized For Encryption

MDS	Galois Field	$w(\mathcal{F}_C)$	Delay $\mathcal{F}_C$	$w(\mathcal{F}_C^{-1})$	Delay $\mathcal{F}_C^{-1}$	$\mathcal{F}_C$ Example (may not be unique)
$(4,2,3)$	GF(2^2)	9	2	12	2	{{1,1},{1,2}}
$(4,2,3)$	GF(2^4)	17	2	46	3	{{1,1},{1,2}}
$(4,2,3)$	GF(2^8)	35	3	182	4	{{1,1},{1,2}}
$(8,4,5)$	GF(2^4)	76	3	168	4	$cir(1,1,4,9)$
$(8,4,5)$	GF(2^8)	164	3	460	5	$cir(1,1,2,71)$
$(16,8,9)$	GF(2^4)	464	4	600	5	$had(1,9,13,6,2,12,10,8)^\dagger$
$(16,8,9)$	GF(2^8)	784	4	2256	6	$cir(1,1,2,142,71,16,1,70)$

† : equivalent to $cauchy((0,3,5,6,8,11,13,14),(1,2,4,7,9,10,12,15))$

Table A.2: Search Results of Involution MDS Codes

MDS	Galois Field	$w(\mathcal{F}_C)$	Delay	$\mathcal{F}_C$ Example (may not be unique)
$(4,2,3)$	GF(2^2)	11	2	{{2,1},{2,2}}
$(4,2,3)$	GF(2^4)	21	2	{{4,1},{2,4}}
$(4,2,3)$	GF(2^8)	48	3	{{142,1},{70,142}}
$(8,4,5)$	GF(2^4)	88	3	$had(1,4,9,13)$
$(8,4,5)$	GF(2^8)	200	4	$had(1,2,140,142)$
$(16,8,9)$	GF(2^4)	544	5	$had(2,4,8,5,12,3,10,15)$
$(16,8,9)$	GF(2^8)	928	5	$had(1,2,142,70,71,4,143,6)$

Appendix B

Matrices Used for AES Design II

The following five matrices are derived from the mathematical representation of the two linear transformations, LT1 and LT2, in the AES Design II of Section 3.3. A network of XOR gates can be generated easily with knowledge of these matrices.

$$\mathcal{F}_T = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix} \qquad \mathcal{F}_{L01} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$\mathcal{F}_{L02} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \qquad \mathcal{F}_{L03} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\mathcal{F}_A \cdot \mathcal{F}_T^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

consumption, a small resistor is usually inserted in series with the power or ground input of the device. Simple power analysis directly examines the relation between the measured power trace and cryptography operations. As a more sophisticated attack, differential power analysis uses the statistics over a long period of power consumption trace to distinguish the correct key guess.

Two types of power leakage have been observed and analyzed in [57], which exist in smart cards due to different internal circuits. The Hamming weight information may be leaked when the majority of the current is used to discharge the equivalent capacitor associated with the data bus. The transition count information (i.e., Hamming weight difference between the previous and current data values) may be leaked when the majority of the current is used to switch the gates driven by the data bus.

In addition to power analysis attack, attacks based on timing information leakage were introduced in [58] to break Diffle-Hellman, RSA, DSS, and other asymmetric key cipher implementations. A timing attack exploits the fact that different inputs require slightly different amounts of processing time. Some block ciphers may also be liable to this attack if the data dependent operations are used without protection. For example, the data-dependent rotations of block cipher RC5 have been exploited for a timing attack in [59]. As another type of attack suggested in [60], differential fault analysis recovers the hidden secret key stored in a smart card by investigating the malfunction induced by the attacker.

It should be noted that appropriate countermeasures can be selected to frustrate many potential implementation attacks. However, these countermeasures typically cause system performance to be lowered.

2.3.5 Other Attacks

Although most security evaluations are focused on linear, differential, and sometimes integral cryptanalysis, other statistical attacks still need to be considered by a cipher designer. For example, both the related key attack [61] and the slide attack [62] exploit the regularity of a key schedule, and their complexities are independent of the number of rounds.

2.4 Block Cipher Implementations

When a cipher is implemented, the complexity is embodied by the area and delay measured in hardware and by the memory and clock cycles measured in software. One block cipher can be implemented using different approaches due to a variety of development platforms, technologies, and implementors' experience. This section lists some published implementations of DES, AES, and Camellia as specific examples.

2.4.1 Hardware Implementations

In hardware, ciphers are normally implemented into two major VLSI devices: *Field-Programmable Gate Arrays (FPGAs)* and *Application-Specific Integrated Circuits (ASICs)*. An FPGA is an integrated circuit chip consisting of a large two-dimensional array of small function units, which can be programmed. The circuits can be reconfigured within the array by changing the connection status between the units. Therefore, FPGAs are flexible and easy to develop. An FPGA implementation is considerably faster than software implementations and its product cost is low for small volume manufacture. An ASIC is a full- or semi-custom chip and cannot be