

國立交通大學

資訊工程系

碩士論文

替換排列網路之線性攻擊研究



Linear Attacks on
Substitution-Permutation Networks

研究生：劉韋廷

指導教授：陳榮傑 教授

中華民國九十四年六月



替換排列網路之線性攻擊研究

Linear Attacks on
Substitution-Permutation Networks

研究生：劉韋廷 Student：Wei-Ting Liu

指導教授：陳榮傑 Advisor：Dr. Rong-Jaye Chen

國立交通大學
資訊工程學系

碩士論文

A Thesis

Submitted to Department of Computer Science and
Information Engineering
College of Electric Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of Master
in

Computer Science and Information Engineering

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月



致 謝

此篇論文得以完成，要感謝許多人的幫忙，有他們的協助才能順利完成，在這裡向他們表達我的感謝。

首先，感謝陳榮傑老師兩年來的耐心指導，並且提出許多研究方向與意見供我參考，讓我能順利確定研究題目並且獲得一些研究成果。

其次，感謝兩位口試委員，葉義雄老師與特地從台北下來的張仁俊學長，在口試期間所提出的寶貴意見與修正，讓我能減少論文的錯誤使其更加完善。

此外也要感謝密碼理論實驗室的各位學長的幫忙，胡鈞祥學長，黃凱群學長，林志賢學長，吳緯凱學長，鄭文鼎學長，尤其要特別感謝指導我們新生進入密碼學領域，培養我們基礎的黃凱群學長。還有感謝各位實驗室同學與學弟，陳政楷，梁漢璋，蔡志彬，楊葉薰，平日與你們的討論讓我獲益良多，也謝謝你們帶給我許多歡笑。

最後，感謝我的家人全心全意的支持，提供我生活的一切需求，讓我無後顧之憂能專心於課業上，謝謝你們。





摘要

線性攻擊法是對於像 DES 與替換排列網路(SPN)之類的區塊加密系統最重要的攻擊之一。在本篇論文中，我們將著重於討論針對 SPN 的線性攻擊，並且對一個 16 位元區塊長度、32 位元的密鑰長度、4 回合的 SPN 作一個完整的攻擊，我們將詳細的討論所使用的攻擊策略。首先我們由 Piling-up lemma 定義何謂一條路徑的偏差，然後使用有技巧的方式來搜尋可攻擊的路徑，藉此攻擊各路徑所對應的部份密鑰。最後結合 backtracking 的技術提供額外的密鑰候選，以便最後測試發現所取得的密鑰是錯誤的時候有額外的密鑰可供選擇。最後的效能測試則顯示了我們策略的有效性。

接著第二部份我們轉為研究跟 S-box 有關的性質，並且發現非線性度是抵抗線性攻擊的最重要因素。而 AES 在設計時已針對線性攻擊作了有效的預防，所以我們將研究其設計 S-box 的方法有何特殊之處，最後亦得到一些有趣的性質，也許可供未來研究之用。

Keywords: 線性攻擊、替換排列網路、S-box、線性逼近、Backtracking、非線性度、多輸出布林函數

Abstract

Linear cryptanalysis is one of the most important attacks on block cipher systems such as DES and substitution-permutation networks (SPNs). In this thesis, we will focus on linear cryptanalysis on SPNs and give complete linear attacks on 16-bit block and 32-bit key SPNs for 4 rounds. We will discuss in detail over our attack strategies. We first define the bias of a trail by using Piling-up lemma, and then use an intelligent method to search a candidate trail to attack. Thus the corresponding subkeys are derived trail by trail. If the resulting key is tested to be wrong then a backtracking method is used to find the next candidate key to be tested. The performance shows the efficiency of these strategies.

Moreover we will discuss about the properties of S-boxes and show that nonlinearity is the most important property S-boxes should have to resist linear attacks. Since the AES (Advanced Encryption Standard) is designed properly to resist linear attacks, we will study further on its construction method and discover some properties which may be useful for later researches.

Keywords: Linear Cryptanalysis, Substitution-Permutation Networks, S-box, Linear Approximation, Backtracking, Nonlinearity, Multiple-Output Boolean Function

Contents

Chinese Abstract	i
English Abstract	ii
Contents	iii
List of Figures	v
List of Tables	vii
Chapter1 Introduction	1
Chapter2 Block Cipher Systems	3
2.1 Feistel Networks.....	3
2.2 Substitution Permutation Networks.....	4
2.3 Standard Block Cipher Systems.....	7
2.3.1 DES.....	8
2.3.3 AES.....	10
2.4 Other Block Cipher Systems.....	14
2.4.1 RC6.....	14
2.4.2 IDEA.....	16
Chapter3 Linear Cryptanalysis	19
3.1 Matsui's Attack on DES.....	19
3.2 Linear Cryptanalysis on SPNs.....	20
3.2.1 The Piling-up lemma.....	21
3.2.2 Linear approximation of S-boxes.....	21

3.2.3	Linear expression of a trail.....	23
3.2.4	Subkeys attack.....	25
3.3	More on Linear Cryptanalysis.....	25
3.3.1	Linear hull.....	26
3.3.2	Key ranking.....	27
3.3.3	Multiple linear approximations.....	28
3.4	Our Attack Design.....	28
3.4.1	Observations.....	29
3.4.2	Strategies.....	29
3.4.3	Algorithm.....	33
3.4.4	Performance.....	33
Chapter4	Design of S-boxes against Linear Cryptanalysis.....	35
4.1	Boolean Functions.....	35
4.2	Multiple-Output Boolean Functions (S-boxes).....	41
4.3	The Linear Approximation of S-box.....	43
4.4	Construction of S-boxes.....	46
4.4.1	Random generation.....	46
4.4.2	Generation using finite field.....	47
4.5	Design Analysis.....	49
4.5.1	Analysis of random generation.....	49
4.5.2	Analysis of generation using finite field.....	50
Chapter5	Conclusion.....	57
References.....		59

List of Figures

Figure 1	Feistel network structure.....	4
Figure 2	SPN structure.....	6
Figure 3	DES structure.....	8
Figure 4	The DES f function.....	9
Figure 5	Key scheduling of DES.....	10
Figure 6	AES state representation for $N_b=4, 6, 8$	11
Figure 7	AES SubByte function.....	13
Figure 8	AES ShiftRow operation.....	13
Figure 9	AES MixColumn operation.....	13
Figure 10	Encryption algorithm with RC6-w/r/b.....	15
Figure 11	Decryption algorithm with RC6-w/r/b.....	16
Figure 12	The IDEA structure.....	17
Figure 13	A possible attack trail.....	24
Figure 14	Probability distribution.....	30
Figure 15	The backtracking scheme.....	32
Figure 16	AES S-box generation.....	48
Figure 17	Properties of (6,6) S-box generated from random.....	49
Figure 18	Properties of (8,8) S-box generated from random.....	50
Figure 19	S-box and its linear approximation table.....	52
Figure 20	Linear approximation table with different constant vector C.....	53
Figure 21	Linear approximation table with different matrix B.....	55
Figure 22	Linear approximation table with different irreducible polynomial.....	56



List of Tables

Table 1	Shift offsets with different Nb.....	13
Table 2	Linear approximation table of Example 3.1.....	23
Table 3	The success rate of linear cryptanalysis.....	30
Table 4	The trail data.....	34
Table 5	Success rate with different number of candidate keys.....	34
Table 6	3-variable affine and linear functions.....	37
Table 7	Nonlinearity example.....	39
Table 8	Properties of different (n,n) S-box generated from $GF(2^n)$	50





Chapter 1 Introduction

Shannon [32] suggested that secure, practical ciphers should have two properties: *confusion* and *diffusion*. The confusion component aims at concealing any algebraic structure in the system and is often a nonlinear substitution on a small sub-block. The diffusion component aims at spreading out the influence of a minor modification of the input data over all outputs and is often a linear mixing of sub-block connections.

Feistel [5] was the first to introduce such system based on Shannon's concepts. The confusion part consists of several rounds of substitution and referred to as S-boxes. And the diffusion part consists of permutations between rounds. This is why such systems are often called *substitution-permutation networks* (SPNs). DES [22], FEAL [31] and many other modern ciphers are the extended concepts of such system.

Differential cryptanalysis [2] and linear cryptanalysis [18] can both be applied to DES and many other block ciphers. In this thesis, we will emphasize on linear cryptanalysis and use it to break SPNs. The linear cryptanalysis requires that there exists a linear relationship having probability not equal to $1/2$ between several plaintext bits and some data bits into the last round. The further the probability is from $1/2$ the better the attack will succeed.

The reason that linear cryptanalysis can succeed is due to the poor design of the S-boxes. In the past, there were many papers discussing about the properties of S-boxes or how to design good S-boxes [7][12][19][23][25][34]. In the first part of this thesis, we focus on how to attack the SPN with non-proper S-boxes. In the original attack, they just extract a part of the key bits using linear cryptanalysis and the rest of them by exhaustive key search. In this thesis, we want to use linear cryptanalysis recursively and each time to get a different part of the key bits. We will

propose some strategies in deciding many linear expressions all of which should have large bias. And we will experiment on these strategies to see their effects.

In the second main part of this thesis, we will turn to discuss about the properties of S-boxes from the viewpoint of multiple-output Boolean functions. Then we will analyze what properties S-boxes should have to resist linear cryptanalysis. Since the new block cipher AES has proved to be resistant to linear attacks efficiently and its construction is very simple, we will focus on this and try to discover some of its properties for later researches.

The organization of this thesis is as follows: Chapter 2 introduces some basic block cipher systems such as Feistel Networks and SPNs. Chapter 3 describes how linear attacks work and we will use some new strategies to attack more efficiently. Chapter 4 discusses about the relationship between the properties of Boolean functions and the resistance of S-boxes to linear attacks. Further, the constructions of AES S-boxes are studied to find some interesting properties. Finally, our conclusion and future work is given in Chapter 5.

Chapter 2 Block Cipher Systems

In this chapter, we will introduce some block cipher systems that are commonly used including Feistel Networks and Substitution-Permutation Networks (SPNs). Most modern cipher systems are modified from these two ciphers.

2.1 Feistel Networks

Feistel and some other researchers [5][6] proposed the Feistel Networks based on Shannon's [32] theory in 1973. The most famous block cipher, Data Encryption Standard (DES), is an example of Feistel Network. In a Feistel Network, each state (a round input) is divided into two halves of equal length ($N/2$, suppose N is the block length), we call L^i and R^i respectively. In each round, we only modify one half of the round input, which is done by the round function. The round function f takes R^{i-1} and K_i as inputs and output an equal length as R^{i-1} .

$$f : \{0,1\}^{\frac{N}{2}} \rightarrow \{0,1\}^{\frac{N}{2}}$$

Then the output XOR with L^{i-1} to form R^i while R^{i-1} is preserved to form L^i . So we can see that a Feistel Network requires two rounds to change all the input data. Figure 1 depicts the Feistel Network structure.

As for the design of round function, a common way is to incorporate some S-boxes and linear transformations like DES, which uses eight S-boxes and some permutations. Note that the round function in Feistel Network does not need to be invertible. This gives much flexibility to the design issue. Thus the S-boxes used can have n -input and m -output while $n \neq m$ since it does not need to be bijective.

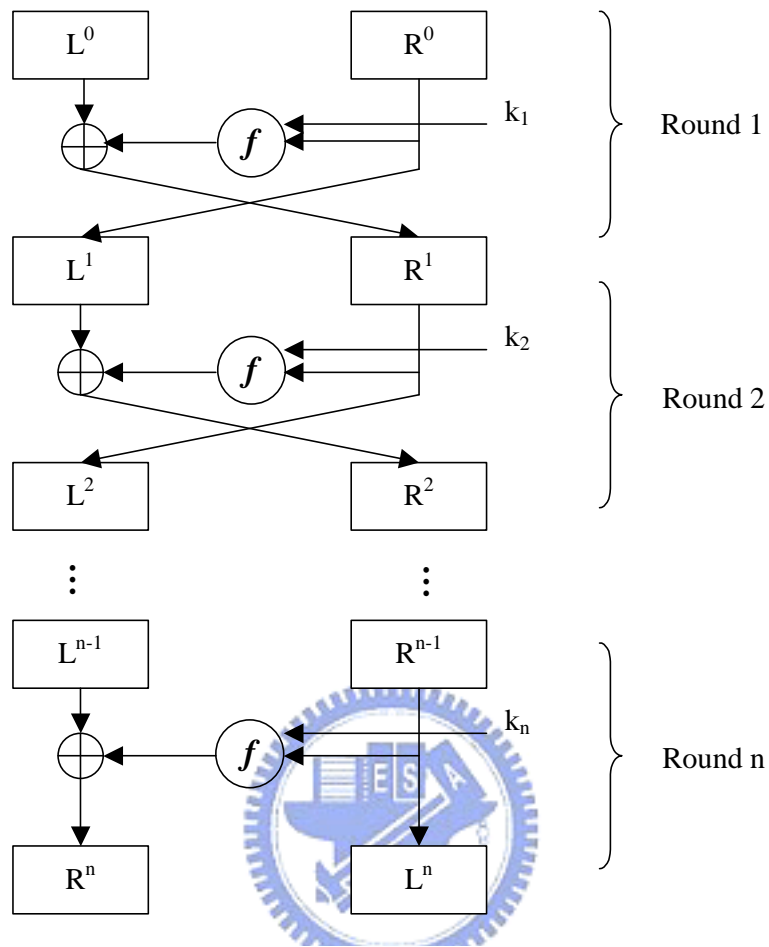


Figure 1: Feistel Network Structure

Another advantage of Feistel Network is that the decryption algorithm is the same as encryption algorithm. That is, we decrypt the ciphertext by processing it through the encryption algorithm except for the reverse key order. This feature is very convenient for the implementation whether in software or hardware.

2.2 Substitution-Permutation Networks

This is the structure we are going to attack in the next chapter. For convenience, the following notations are used:

U_r : the input to S-box in round r (also the result of $W_{r-1} \oplus K_r$)

V_r : the output of S-box in round r

W_r : the permuted result of round r (also the output of round r)

K : the initial key

K_r : the key used in round r

And we represent the i^{th} bit by using the superscript. For example, K_1^4 is the 4th bit of round 1 key bits.

Suppose l and m are positive integers where l is the number of inputs to each S-box and m is the number of S-boxes in each round. So lm is the block length of the cipher. An SPN contains two parts: p_s and p_p .

$$p_s : \{0,1\}^l \rightarrow \{0,1\}^l$$

is a permutation (or substitution), and

$$p_p : \{1,2,\dots,lm\} \rightarrow \{1,2,\dots,lm\}$$

is also a permutation. p_s is the S-box we mentioned above and it is used to replace l bits by another l bits and it should be a bijective operation. p_p is the permutation to permute the outputs of S-boxes which is lm bits in total in a round.

The *key-scheduling algorithm* we use here is a simple shift operation. Thus the first round key bits are the first lm bits of the initial key K , i.e., K^1 to K^{lm} . And the second round key bits shift l bits and become K^{l+1} to K^{2l} . The rest of the round keys are derived in a similar way.

The encryption algorithm consists of the following steps. The SPN consists of N_r rounds. In each round, we will perform m substitutions by p_s , then followed by the permutation p_p (except for the last round, there is no permutation needed since it contributes no security). And before each S-box, we will XOR with round key bits (this is called *round key mixing*). After the substitution of the last round, we perform another round key mixing. So in an N_r -round SPN, there will be N_r+1 round keys needed. The very first and last XOR with round key is called *whitening*, which was

proposed by Schneier [30]. Figure 2 is a pictorial representation of such SPN.

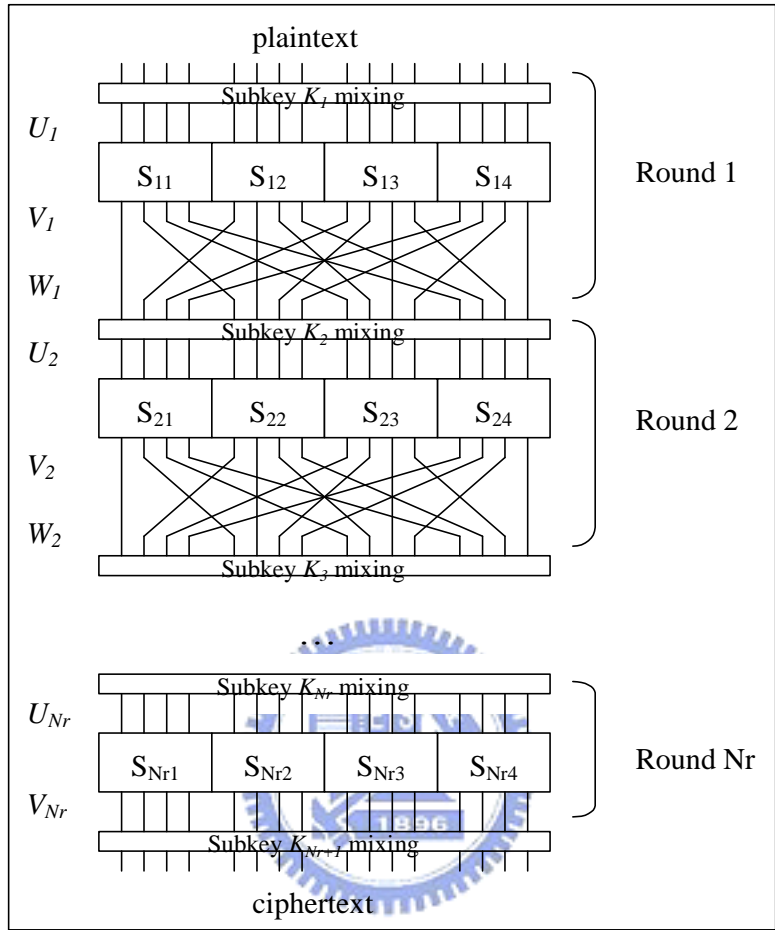


Figure 2: SPN structure

Example 2.1: Suppose $l=m=Nr=4$. Let the S-box p_s be defined as follows, where the input (x) and the output ($p_s(x)$) are written in hexadecimal:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$p_s(x)$	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

And the permutation p_p be defined as follows:

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$p_p(x)$	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

Suppose the key is

$$K = 0011\ 1010\ 1001\ 0100\ 1101\ 0110\ 0011\ 1111$$

Then the round keys according to the key scheduling above are:

$$K_1 = 0011\ 1010\ 1001\ 0100$$

$$K_2 = 1010\ 1001\ 0100\ 1101$$

$$K_3 = 1001\ 0100\ 1101\ 0110$$

$$K_4 = 0100\ 1101\ 0110\ 0011$$

$$K_5 = 1101\ 0110\ 0011\ 1111$$

Suppose the plaintext is

$$x = 0010\ 0110\ 1011\ 0111$$

We see how the encryption processes for the first two rounds:

$$W_0 = x = 0010\ 0110\ 1011\ 0111$$

$$K_1 = 0011\ 1010\ 1001\ 0100$$

$$U_1 = W_0 \oplus K_1 = 0001\ 1100\ 0010\ 0011$$

$$V_1 = 0100\ 0101\ 1101\ 0001$$

$$W_1 = 0010\ 1110\ 0000\ 0111$$

$$K_2 = 1010\ 1001\ 0100\ 1101$$

$$U_2 = W_1 \oplus K_2 = 1000\ 0111\ 0100\ 1010$$

$$V_2 = 0011\ 1000\ 0010\ 0110$$

$$W_2 = 0100\ 0001\ 1011\ 1000$$

The remaining rounds can be processed in a similar way. □

We decrypt the ciphertext in reverse order of the encryption algorithm. That is, the keys are given in reverse order and the S-boxes are bijective so that we can recover the input, which is unlike the Feistel Network.

2.3 Standard Block Cipher Systems

In this section we introduce two standard block cipher systems: DES and AES.

2.3.1 DES

Data Encryption Standard (DES) originates from the Lucifer which was developed by IBM and later modified by NIST (National Institute of Standards and Technology) to become a block cipher standard in 1977. DES is one type of Feistel structure we described earlier. The block length is 64-bit and also 64-bit key length (including 8 parity check bits). Figure 3 is an overview of the DES structure.

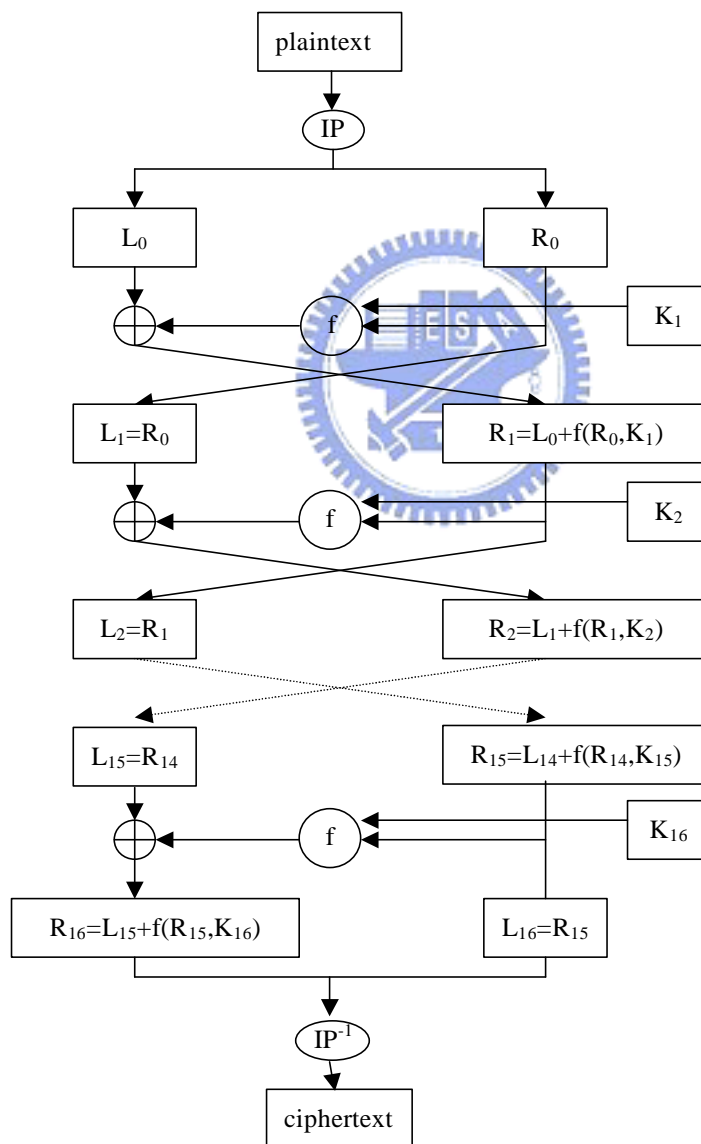


Figure 3: DES structure

The IP and IP^{-1} are initial permutation and inverse permutation, respectively. Each L_i and R_i is 32 bits in length. The f function takes R_{i-1} and K_i as inputs and we show the f function in Figure 4.

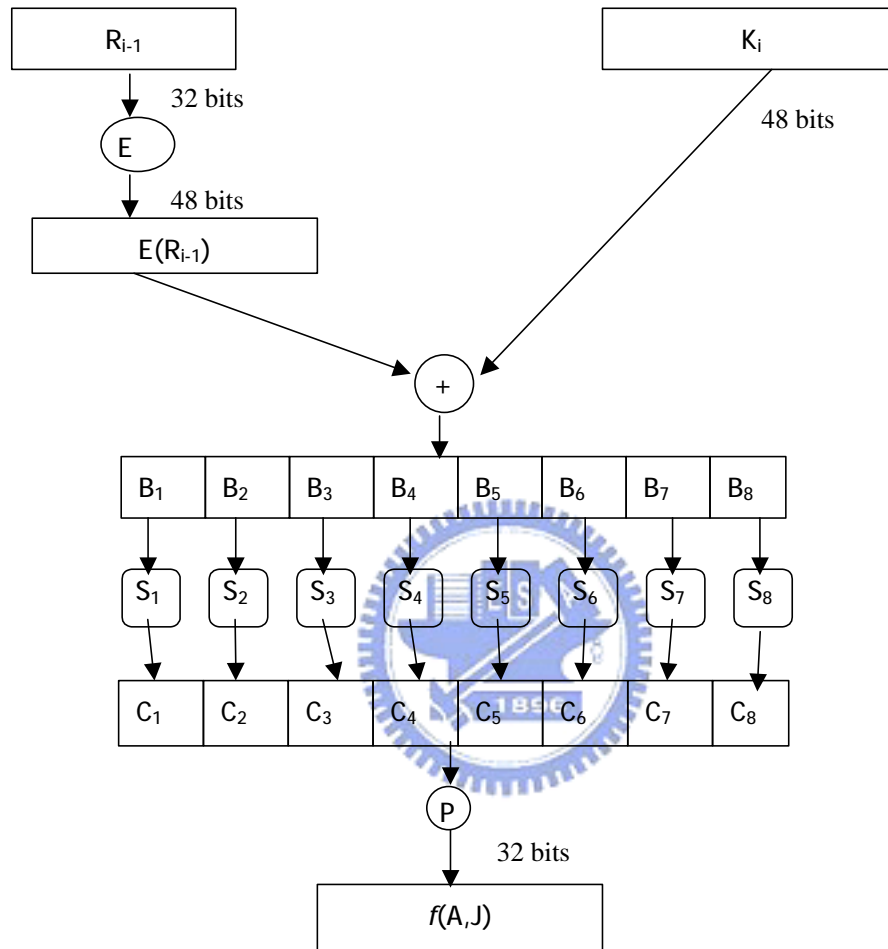


Figure 4: The DES f function

R_{i-1} is first extended to 48 bits and then XOR with round key (48-bit). The result is then divided into 8 blocks each with 6 bits. These 8 blocks are then input to the 8 S-boxes which output 4 bits each. The 8 blocks of C_i are permuted according to P and the output of f function is then the output of permutation P .

The key scheduling will generate 16 subkeys each with 48 bits from the initial 56 (64) bits key. We show the scheduling in Figure 5. The $PC-1$ and $PC-2$ are also permutations.

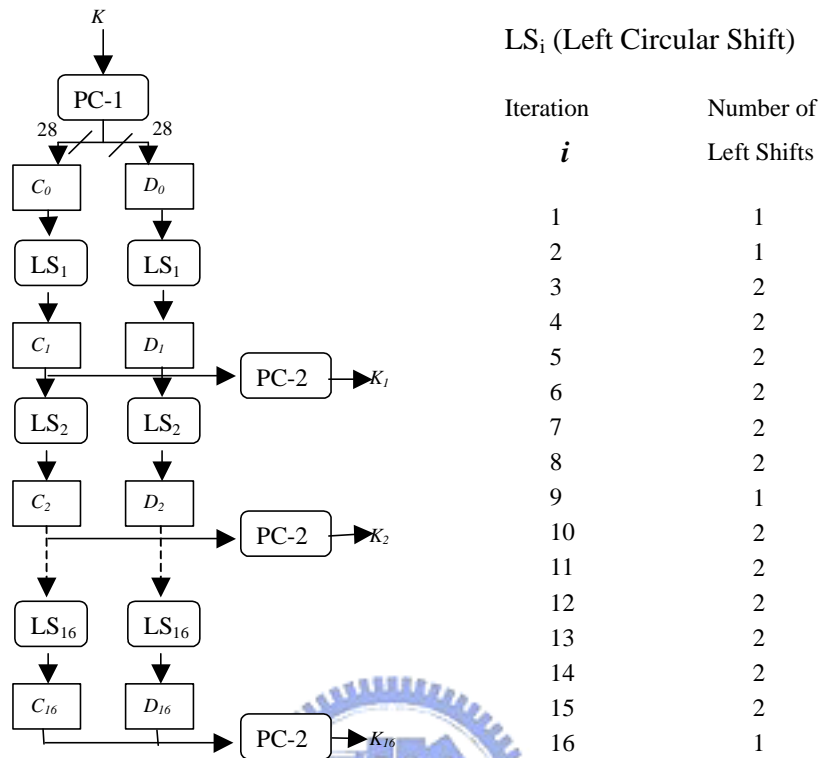


Figure 5: Key scheduling of DES

2.3.2 AES

On January 2, 1997, NIST began the process of choosing a replacement of DES, which is called the Advanced Encryption Standard (AES). AES requires a block length with 128-bit and supporting key length with 128, 192, 256 bits ($N_k=4, 6, 8$). On October 2, 2000, Rijndael [3][4] was selected as the new standard.

AES has block length with 128, 192, 256 ($N_b=4, 6, 8$) bits whose number of rounds N_r , are 10, 12, and 14, respectively. All operations in AES are byte oriented. **State** is the input cut into byte array (Figure 6). AES first generates the subkeys we need using KeyExpansion algorithm from the initial key. Then for the first N_r-1 rounds, it performs the Round function, which contains the ByteSub, ShiftRow, MixColumn and AddRoundKey. Finally we apply the FinalRound, which is the same

as Round except for no MixColumn. The algorithm is given in Algorithm 2.1 in pseudo C language.


S _{0,0}	S _{0,1}	S _{0,2}	S _{0,3}	S _{0,4}	S _{0,5}	S _{0,6}	S _{0,7}
S _{1,0}	S _{1,1}	S _{1,2}	S _{1,3}	S _{1,4}	S _{1,5}	S _{1,6}	S _{1,7}
S _{2,0}	S _{2,1}	S _{2,2}	S _{2,3}	S _{2,4}	S _{2,5}	S _{2,6}	S _{2,7}
S _{3,0}	S _{3,1}	S _{3,2}	S _{3,3}	S _{3,4}	S _{3,5}	S _{3,6}	S _{3,7}

Figure 6: AES **State** representation for Nb=4, 6, 8

```

Rijndael(State, Key)
{
    KeyExpansion(Key);
    AddRoundKey(State, RoundKey);
    For(i=1; i<Nr; i++)
    {
        Round(State, RoundKey);
    }
    FinalRound(State, RoundKey);
}
        
```

Algorithm2.1: AES algorithm



```

Round(State, RoundKey)
{
    ByteSub(State);
    ShiftRow(State);
    MixColumn(State);
    AddRoundKey(State, RoundKey);
}
        
```

```

FinalRound(State, RoundKey)
{
    ByteSub(State);
    ShiftRow(State);
    AddRoundKey(State, RoundKey);
}
        
```

KeyExpansion generates the Nr+1 round subkeys from the initial key. The expanded key is a linear array of 4-byte word. The first Nk words contain the cipher key. All other words are defined recursively in terms of words with smaller indices. The algorithm is given in Algorithm 2.2.

Algorithm 2.2:

```

KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)])
{
    for(i = 0; i < Nk; i++)
        W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
    for(i = Nk; i < Nb * (Nr + 1); i++)
    {
        temp = W[i - 1];
        if (i % Nk == 0)
            temp = SubByte(RotByte(temp)) ⊕ Rcon[i / Nk];
        W[i] = W[i - Nk] ⊕ temp;
    }
}

```

The round constants are independent of Nk and defined by:

$Rcon[i] = (RC[i], '00', '00', '00')$ with $RC[i]$ representing an element in $GF(2^8)$ with a value of $x^{(i-1)}$ so that:

$RC[1] = 1$ (i.e. '01')

$RC[i] = x$ (i.e. '02') $\cdot (RC[i-1]) = x^{(i-1)}$

$RotByte$ is a rotate of the bytes, i.e., $RotByte(B_0, B_1, B_2, B_3) = (B_1, B_2, B_3, B_0)$. Then the RoundKey i is given by the Round Key buffer word $W[Nb*i]$ to $W[Nb*(i+1)]$.

For the first $Nr-1$ rounds, we perform Round function, which contains four sub-function: ByteSub, ShiftRow, MixColumn, and AddRoundKey. The FinalRound is the same as Round except for no MixColumn. Next, we briefly introduce the sub-functions.

ByteSub is the function to replace one byte by another byte, i.e., it acts as a S-box. The detailed algorithm will be given in Chapter 4.

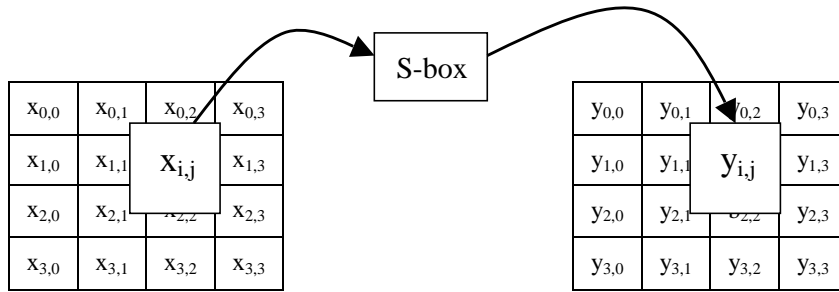


Figure 7: AES ByteSub function

The ShiftRow is a cyclic left shift of the State according to the offsets (Table 1).

Table 1: Shift offsets with different Nb

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

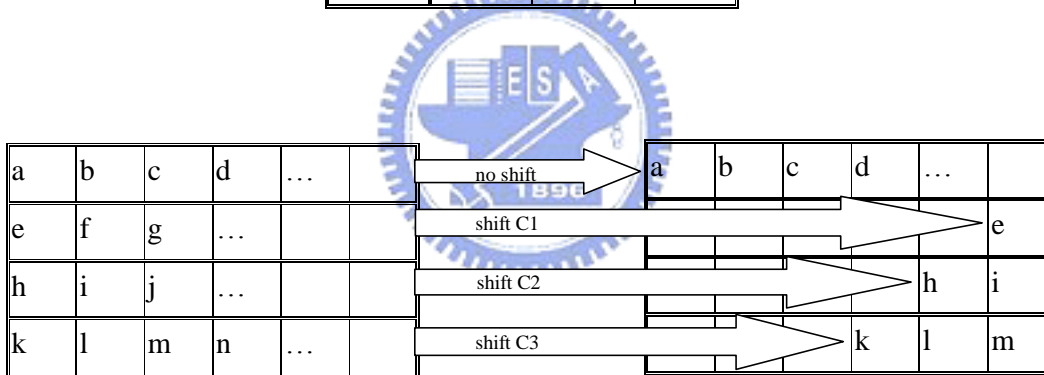


Figure 8: AES ShiftRow operation

MixColumn replaces a column by a new one formed by multiplying the column with a matrix.

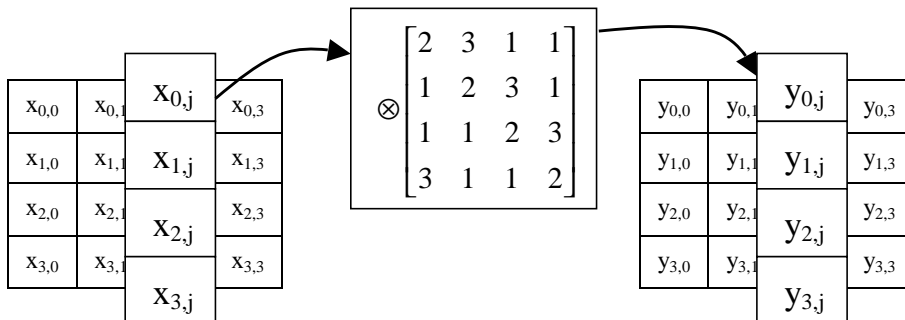


Figure 9: AES MixColumn operation

And the AddRoundKey is simply add the State with the RoundKey.

2.4 Other Block Cipher Systems

Although the previous two ciphers are the most commonly used today, there are still other systems not belonging to these two kinds. However, there exists one common feature in all of them: they use repeated rounds to achieve security requirement.

2.4.1 RC6

RC6 [28] is a block cipher designed to meet the requirements of AES. The design is based on RC5 and modified to increase security and performance. It has block length with 128-bit and can be seen as extending RC5 from 64-bit to 128 bit. However, instead of using two 64-bit registers, they change to use four 32-bit registers since the AES architecture does not support 64-bit operations. Like RC5, RC6 makes an extensive use of data-dependant rotations. The philosophy of RC5 is to exploit operations (such as rotations) that are efficiently implemented on modern processors. RC6 follows the trend and it includes the 32-bit integer multiplication since this operation is now implemented on almost all processors. The advantage of the integer multiplication is to “diffuse” effectively. RC6 uses it to compute the rotation amounts, so that the rotation amounts are dependent on all of the bits of another register. Thus RC6 has much faster diffusion than RC5 and increases security with fewer rounds.

A version of RC6 is more accurately specified as RC6-w/r/b where the word size is w bits, encryption consists of a nonnegative number of rounds r, and b denotes the

length of the encryption key in bytes. RC6 consists of the following six basic operations:

$a + b$: integer addition modulo 2^w

$a - b$: integer subtraction modulo 2^w

$a \oplus b$: bitwise exclusive-or of w -bit words

$a \times b$: integer multiplication modulo 2^w

$a \lll b$: rotate the w -bit word a to the left by the amount given by the least significant $\lg w$ bits of b

$a \ggg b$: rotate the w -bit word a to the right by the amount given by the least significant $\lg w$ bits of b

The key scheduling is as follows. The user supplies a key of b bytes, where $0 \leq b \leq 255$. From this key, $2r + 4$ words (w bits each) are derived and stored in the array $S[0, 1, \dots, 2r + 3]$. This array is used in both encryption and decryption. The encryption and decryption algorithms are shown in the following figures.

Input:	Plaintext stored in four w -bit input registers A, B, C, D Number r of rounds w -bit round keys $S[0, 1, \dots, 2r + 3]$
Output:	Ciphertext stored in A, B, C, D
Steps:	$B = B + S[0]$ $D = D + S[1]$ for $i = 1$ to r do { $t = (B \times (2B + 1)) \lll \lg w$ $u = (D \times (2D + 1)) \ggg \lg w$ $A = ((A \oplus t) \ggg u) + S[2i]$ $C = (C \oplus u) \lll t + S[2i + 1]$ $(A, B, C, D) = (B, C, D, A)$ } $A = A + S[2r + 2]$ $C = C + S[2r + 3]$

Figure 10. Encryption algorithm with RC6- $w/r/b$

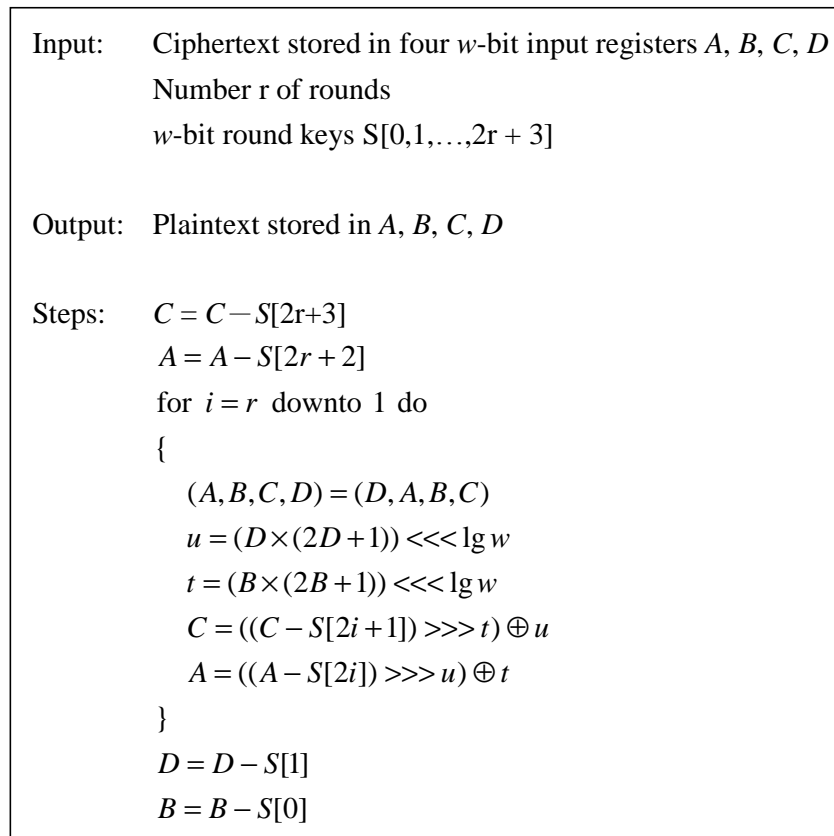


Figure 11. Decryption algorithm with RC6- $w/r/b$

2.4.3 IDEA

IDEA (International Data Encryption Algorithm) [16][17] was developed by Lai in 1991. IDEA is used in PGP (Pretty Good Privacy), the cryptographic system for Internet and E-mail security. IDEA is also 64-bit block length as DES and the round number is 8 and the key size is 128-bit.

The algorithm is illustrated in Figure 12. The 64-bit plaintext is divided into four 16-bit blocks, X_1, X_2, X_3, X_4 . In each round, six 16-bit subkeys are used, denoted by $K_{i,1}, K_{i,2}, \dots, K_{i,6}$ for round i . Since there are 8 rounds, 48 subkeys are used, plus 4 extra subkeys used after the last round to transform the output. And the four output

ciphertext blocks are denoted by Y_1, Y_2, Y_3, Y_4 .

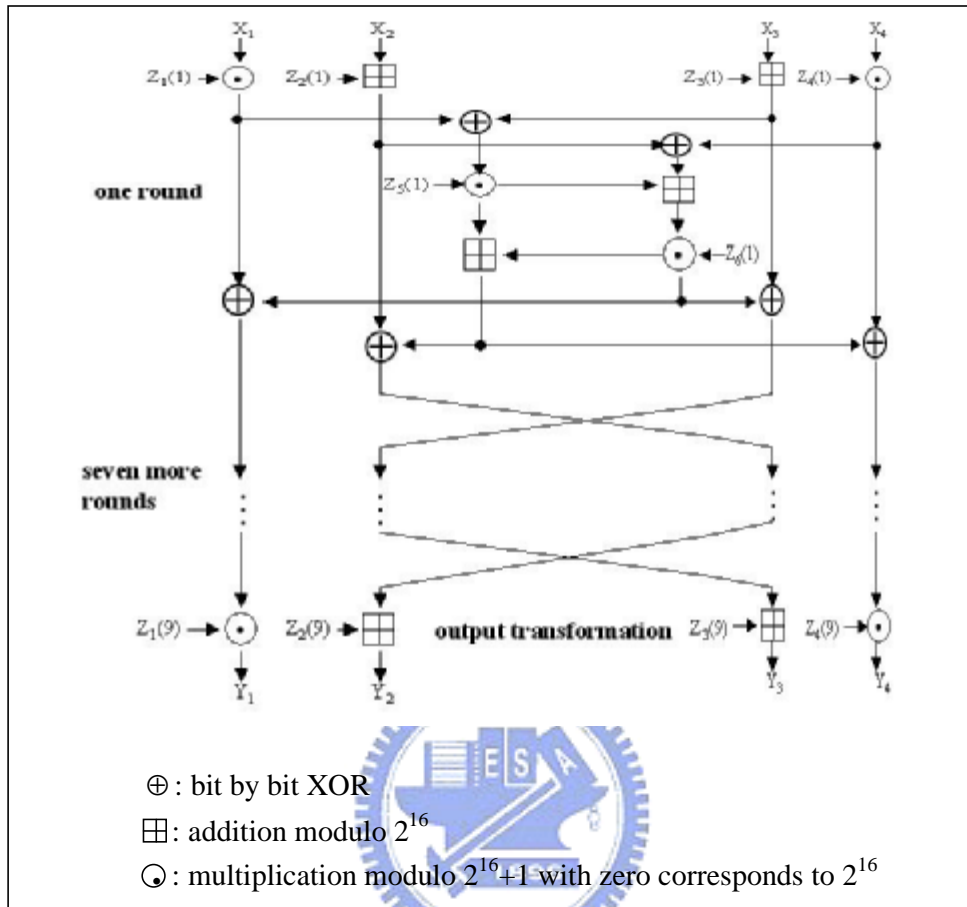


Figure 12. The IDEA structure

In each round, the 16-bit blocks are XORed, added and multiplied as the figure shows. The multiplication modulo $2^{16}+1$ can be regarded as the S-box of IDEA. After the last step, each of the resulting 16-bit blocks is multiplied modulo $2^{16}+1$ by its corresponding subkey.

The key scheduling is very simple as follows. The initial key of 128 bits is divided into 8 blocks of 16 bits and they become $K_{1,1}, \dots, K_{1,6}$, and $K_{2,1}, K_{2,2}$. Then the initial key is shifted 25 bits left and divided into 8 blocks of new subkeys. The procedure continues until 52 subkeys are generated.

The decryption algorithm is the same as encryption. The keys are used in reverse order with some modifications; they are the inverse of the encryption keys for

multiplications as well as addition.

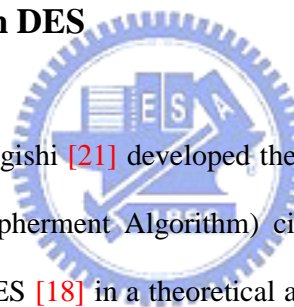
In this chapter, we introduced several block cipher systems from basic schemes, Feistel Networks and SPNs, to standard systems, DES and AES. In the next chapter, we will start to use linear cryptanalysis to attack the SPNs and use our strategies to attack them more efficiently.



Chapter 3 Linear Cryptanalysis

In this chapter, we introduce linear cryptanalysis, which is the most important attack on block cipher systems. Section 1 briefly introduces the Matsui's attack concept on DES. Section 2 gives an entire procedure of the attack on SPNs. Section 3 introduces some other improved techniques proposed by other researchers. Section 4 illustrates our new strategies, which can find trails with good bias to attack and we also show the performance of our new strategies in the end.

3.1 Matsui's Attack on DES



Originally, Matsui and Yamagishi [21] developed the linear cryptanalysis against the FEAL [31] (Fast Data Encipherment Algorithm) cipher in 1992. In 1994, Matsui modified it and used it on DES [18] in a theoretical attack on the full 16-round DES, which requires 2^{47} known plaintext-ciphertext pairs and successfully obtains 14 key bits. Now it has become the most important attack against block ciphers. In Matsui's paper, he introduced two versions of attack algorithms. The first one, called **Algorithm 1**, can only attack one key bit information. The second one, called **Algorithm 2**, can extract more key bits in one attack.

Algorithm 1:

Step 1: Let T be the number of plaintexts such that the left side of equation,

$$P[i_1, i_2, \dots, i_a] \oplus C[j_1, j_2, \dots, j_b] = K[k_1, k_2, \dots, k_c],$$

is equal to zero.

Step 2: If $T > N/2$ (N denotes the number of plaintexts),

then guess $K[k_1, k_2, \dots, k_c] = 0$ (when $p > 1/2$) or 1 (when $p < 1/2$),
else guess $K[k_1, k_2, \dots, k_c] = 1$ (when $p > 1/2$) or 0 (when $p < 1/2$).

Algorithm 2:

Step 1: For each candidate $K_n^{(i)}$ ($i = 1, 2, \dots$) of K_n , let T_i be the number of plaintexts such that the left side of equation

$$P[i_1, i_2, \dots, i_a] \oplus C[j_1, j_2, \dots, j_b] \oplus F_n(C_L, K_n)[l_1, l_2, \dots, l_d] = K[k_1, k_2, \dots, k_c]$$

is equal to zero.

Step 2: Let T_{\max} be the maximal value and T_{\min} be the minimal value of all T_i 's.

- 1 If $|T_{\max} - N/2| > |T_{\min} - N/2|$, then adopt the key candidate corresponding to T_{\max} and guess $K[k_1, k_2, \dots, k_c] = 0$ (when $p > 1/2$) or 1 (when $p < 1/2$).
- 1 If $|T_{\max} - N/2| < |T_{\min} - N/2|$, then adopt the key candidate corresponding to T_{\min} and guess $K[k_1, k_2, \dots, k_c] = 1$ (when $p > 1/2$) or 0 (when $p < 1/2$).

In the remaining parts of this thesis, we focus on Algorithm 2 since it is much more powerful.

3.2 Linear Cryptanalysis on SPNs

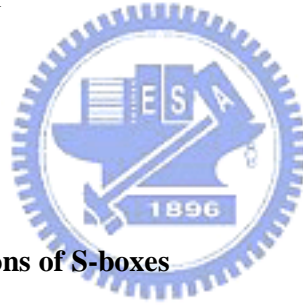
Here we briefly explain how linear cryptanalysis works on SPNs. The detailed introduction is described in [11][33]. Keliher also discussed linear attacks on SPN in [14]. To apply linear attacks, we need to find a subset of bits that their XOR behaves in a non-random way. First, we introduce a useful lemma in linear attacks.

3.2.1 The Piling-up lemma

Suppose $X_1, X_2, \dots \in \{0,1\}$ are independent random variables. p_1, p_2, \dots are real numbers such that $0 \leq p_i \leq 1$, and suppose that $\Pr[X_i=0]=p_i$ and $\Pr[X_i=1]=1-p_i$. Then we define the *bias* of X_i to be $e_i = p_i - \frac{1}{2}$. Let $\mathbf{e}_{i_1, i_2, \dots, i_k}$ denote the bias of the random variable $X_{i_1} \oplus \dots \oplus X_{i_k}$. It is easy to see that $\mathbf{e}_{i_1, i_2} = 2e_{i_1}e_{i_2}$. And we can generalize it in the following lemma.

Lemma 3.1 (Piling-up Lemma) [18]: Let $\mathbf{e}_{i_1, i_2, \dots, i_k}$ denote the bias of the random variable $X_{i_1} \oplus \dots \oplus X_{i_k}$. Then

$$\mathbf{e}_{i_1, i_2, \dots, i_k} = 2^{k-1} \prod_{j=1}^k e_{i_j}.$$



3.2.2 Linear approximations of S-boxes

Next, we need to compute the linear approximation table of an S-box so that we can determine the XOR of which bits is not random.

Example 3.1: Consider the following S-box: $p_S : \{0,1\}^4 \rightarrow \{0,1\}^4$.

X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4
0	0	0	0	1	1	1	0
0	0	0	1	0	1	0	0
0	0	1	0	1	1	0	1
0	0	1	1	0	0	1	0
0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1
0	1	1	0	1	0	1	1
0	1	1	1	1	0	0	0

1	0	0	0	0	0	1	1
1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0
1	0	1	1	1	1	0	0
1	1	0	0	0	1	0	1
1	1	0	1	0	0	0	0
1	1	1	0	1	0	0	1
1	1	1	1	0	1	1	1

If we want to know the probability of $X_2 \oplus Y_2 \oplus Y_3 = 0$, then we count the number of rows in the above table where $X_2 \oplus Y_2 \oplus Y_3 = 0$ and denote this number as N_L value. Then we divide N_L by 2^4 (4 is the number of S-box input) to get the probability of $X_2 \oplus Y_2 \oplus Y_3 = 0$. Here $N_L=4$, thus the probability is $4/16$ and the bias is $-1/4$. \square

In a similar way, we can record all possible input-output XOR in a *linear approximation table* (Table 2). We read the table by using the following notation:

$$\left(\bigoplus_{i=1}^4 a_i X_i \right) \oplus \left(\bigoplus_{i=1}^4 b_i Y_i \right), a_i, b_i \in \{0,1\}.$$

Take (a_1, \dots, a_4) as index of rows and (b_1, \dots, b_4) as index of columns. The values in the table indicate N_L 's-8. Thus, $X_2 \oplus Y_2 \oplus Y_3$ of Example 3.1 is expressed as $a=0100$, $b=0110$ and the corresponding N_L -8 is in the shaded place of the table which is -4 as Example 3.1 counts. This table consists of $2^n \times 2^m$ entries where n and m denote the number of X variables and Y variables respectively (in Example 3.1, $n=m=4$). In the linear cryptanalysis, we are searching for the pattern with a large bias size to attack.

Table 2: Linear approximation table of Example 3.1

X\Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	+8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	-4	0	-4	0	-4	0	+4	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	+2	-2	+6	+2	+2	-2	-2	+2
3	0	0	0	0	0	0	0	0	+2	-6	-2	-2	+2	+2	-2	-2
4	0	+4	-2	-2	-2	-2	-4	0	0	0	-2	+2	+2	-2	0	0
5	0	0	-2	+2	-2	+2	+4	+4	0	0	-2	+2	+2	-2	0	0
6	0	0	-2	+2	+2	-2	0	0	-2	-2	0	+4	-4	0	-2	-2
7	0	0	-2	+2	+2	-2	0	0	-2	+2	0	0	+4	+4	-2	+2
8	0	0	0	0	0	0	0	0	-2	+2	+2	-2	+2	-2	-2	-6
9	0	0	0	0	0	0	0	0	-2	-2	+2	+2	+2	+2	+6	-2
10	0	0	0	0	-4	-4	+4	-4	0	0	0	0	0	0	0	0
11	0	+4	0	-4	+4	0	+4	0	0	0	0	0	0	0	0	0
12	0	0	+2	-2	-2	+2	0	0	+2	+2	0	+4	0	+4	-2	-2
13	0	0	+2	-2	-2	+2	0	0	-6	-2	0	0	0	0	-2	+2
14	0	+4	+2	+2	-2	-2	0	+4	0	0	+2	-2	-2	+2	0	0
15	0	0	-6	-2	-2	+2	0	0	0	0	+2	-2	-2	+2	0	0

3.2.3 Linear expression of a trail

We then use such weakness (large bias) to find a trail through entire SPN to get a linear expression involving only parts of plaintext bits and data bits into the last round (bits of U_{Nr}) and all subkeys encountered in the path. All other intermediate data bits of $U_r \cdot V_r$, where $r < Nr$, will be cancelled. Thus we produce a linear expression in the following:

$$P_I \oplus C_J \oplus K_K = 0, \quad (3.1)$$

where P_I , C_J , and K_K denote the XOR of some plaintext bits, data bits of U_{Nr} and encountered key bits respectively. But what we care is only

$$P_I \oplus C_J = 0. \quad (3.2)$$

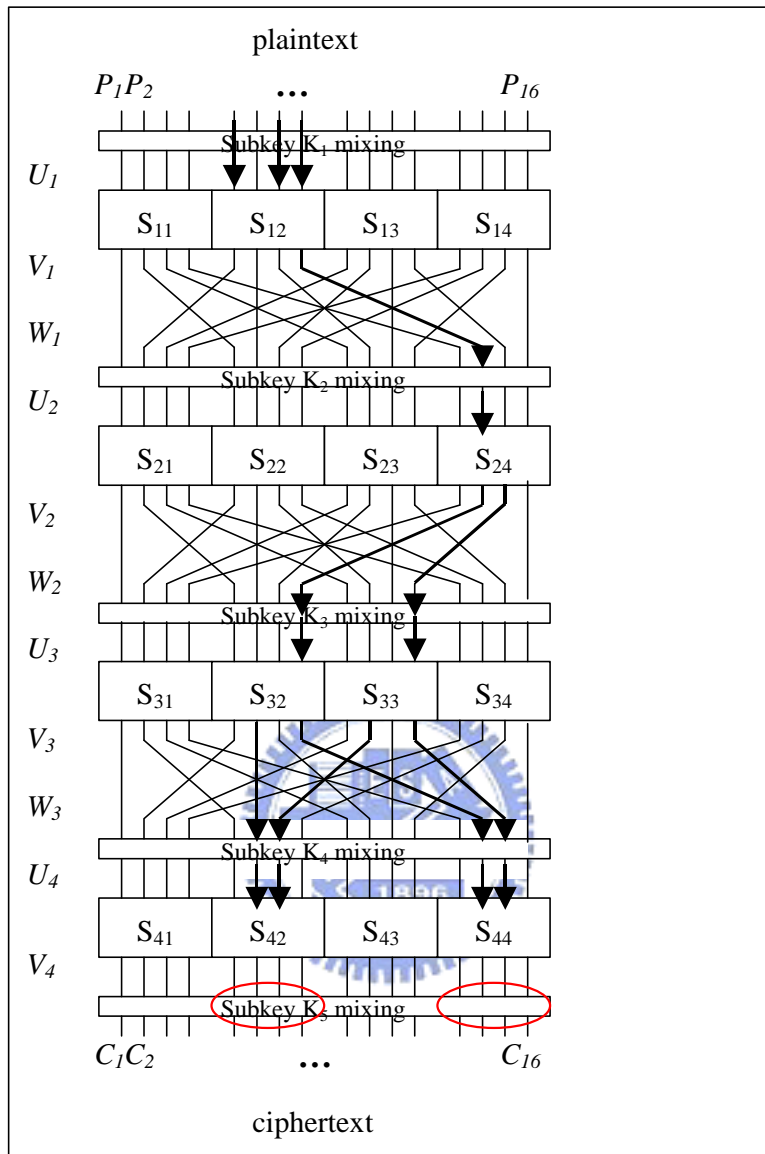


Figure 13: A possible attack trail.

Figure 13 shows a possible attack trail. Here, P_l is $P_5 \oplus P_7 \oplus P_8$ and C_j is $U_4^6 \oplus U_4^7 \oplus U_4^{14} \oplus U_4^{15}$. The trail is formed as follows: In S_{12} , we choose $X_1 \oplus X_3 \oplus X_4 \oplus Y_4$ since it has large bias. Then we follow the output permutation and XOR with K_2 . Now in round 2, they become the input X_2 of S_{24} . So we can look up in the linear approximation table to check what bits X_2 XORing with has large bias (row 4, since X_2 represents 0100_2). As procedure continues we have a trail formed.

After the trail is determined, the overall bias of the entire SPN can be calculated by Piling-up lemma (each S-box encountered viewed as e_{i_j}) and we denote the bias as e .

3.2.4 Subkeys attack

Once we have the trail and the bias, we then begin to extract the subkeys of the last round. It proceeds as follows:

1. The subkeys we are going to extract are those involved in the last part of the trail. For example, in Figure 13, C_J of (3.2) are the bits into the second and fourth S-box. Then the subkeys being extracted are the corresponding position of the output bits of those S-boxes, i.e., the circled part in Figure 13.
2. Since the attack is a known plaintext- ciphertext attack, we have many plaintext- ciphertext pairs and we say we have T pairs. We maintain a counter array for each possible candidate subkeys. Then we partially decrypt the ciphertext for each candidate subkeys. If the linear expression (3.2) holds, then we increment the corresponding counter of that subkey.
3. In the end, we expect the counter, which is closest to $(\frac{1}{2} \pm e)T$, is the most likely subkey.

3.3 More on Linear Cryptanalysis

In this section we introduce some further researches done as the linear cryptanalysis develops. With the help of these techniques, we can increase the success rate and reduce the data pairs we need.

3.3.1 Linear hull

Nyberg [24] proposed the linear hull effect in 1994. The main result shows that the success rate of Algorithm 2 is underestimated in Matsui’s paper. They show this by declaring that the data complexity we need can be reduced. Since we may have many linear expressions with the same input and output mask but different internal subkeys, i.e., P_I and C_J are the same but K_K is different. For input mask \mathbf{a} and output mask \mathbf{b} , he uses $ALH(\mathbf{a},\mathbf{b})$ to denote the *approximation linear hull*. We describe the definition and theorem in a more understandable version by [15].

Definition 3.1: Given nonzero N -bit masks \mathbf{a} , \mathbf{b} , the approximation linear hull, $ALH(\mathbf{a},\mathbf{b})$, is the set of all T -round characteristics, for the T rounds under consideration, having \mathbf{a} as the input mask for round 1 and \mathbf{b} as the output mask for round T , i.e., all characteristics of the form $\Omega = \langle \mathbf{a}, \mathbf{a}^2, \mathbf{a}^3, \dots, \mathbf{a}^T, \mathbf{b} \rangle$.

The characteristic Ω here is like the trail we said before. And we have the following theorem.

Theorem 3.1: Let \mathbf{a} and \mathbf{b} be fixed nonzero N -bit input and output masks, respectively, for T rounds of an SPN. Then

$$E_T[\mathbf{a},\mathbf{b}] = \sum_{\Omega \in ALH(\mathbf{a},\mathbf{b})^*} LCP(\Omega). \quad (3.3)$$

The $E_T[\mathbf{a},\mathbf{b}]$ denotes the expected value of linear probability of mask (\mathbf{a},\mathbf{b}) over all independent keys. And $LCP(\Omega)$ denotes the linear characteristic probability of a characteristic Ω . This theorem shows that under certain masks (\mathbf{a},\mathbf{b}) , we may have many different characteristics and the expected value of masks (\mathbf{a},\mathbf{b}) is the sum of $LCP(\Omega)$ over a large set of characteristics. In other words, under certain P_I and C_J , the expected value of bias is the sum of a large set of different trails with the same P_I

and C_j . Therefore, the linear characteristic probability of best characteristic is strictly less than $E_T[\mathbf{a}, \mathbf{b}]$. This implies that an attacker will overestimate the number of pairs required for a given success rate since the best trail we find is always smaller than $E_T[\mathbf{a}, \mathbf{b}]$.

3.3.2 Key ranking

After the linear cryptanalysis was proposed, Matsui experimented on the attack in 1994 again with some modifications [20]. In his paper, he uses two new linear approximation equations, each of which provides candidates for 13 key bits. Further, he adopts the reliability of key candidates into consideration. The key candidates means that he stores not only the most likely key bits but also the i^{th} likely candidates. That is, he stores the key $\hat{k}_1, \hat{k}_2, \dots$ in order where \hat{k}_i is the i^{th} likely key bits. Then if the most likely key tests to be wrong, he can go back to use the second likely key bits and so on. The test can be done by given a plaintext-ciphertext pair (P, C) , and the rest key bits by exhaustive key search to test if the candidate key bits can generate C from P . To increase accuracy, a few more pairs $\{(P_1, C_1), (P_2, C_2), \dots\}$ can be given since wrong key bits can generate the correct C_i with negligible probability. Thus, if \hat{k}_1 fails the test, then \hat{k}_2 is used and so on until the correct one is found. With this simple improvement, he increased the success rate. In his test, he successfully attacked the 26 key bits of the full 16-round DES with 2^{43} plaintext-ciphertext pairs. The remaining 30 key bits can be found by exhaustive key search. In comparison with his original attack, more key bits are attacked with fewer pairs needed.

3.3.3 Multiple linear approximations

Kaliski and Robshaw [29] proposed a new idea on linear cryptanalysis by using multiple linear approximations in CRYPTO'94. Suppose they have n linear approximations, which involve the same key bits but differ in the plaintext and ciphertext bits that they use. For each linear approximation they assign a different weight a_i (this may be decided by their biases) and $\sum_{i=1}^n a_i = 1$. Then for each candidate key bits $K^{(j)}, j=1,2,\dots$ and each linear approximation i , let T_j^i be the number of the linear equation holds. Then we calculate $U_j = \sum_{i=1}^n a_i T_j^i$ for each j . And the rest parts are just like the original Algorithm 2 in Matsui's attack, i.e., we see which U_j is furthest from $N/2$ (N is number of pairs) and we assume it to be the most likely key bits.

This technique is supposed to increase the success rate and reduce the data complexity. However, in their experiments, the increase of effectiveness on DES is somewhat limited. But, this is still an important skill since it may be generally applicable to other block ciphers and be extremely effective in reducing data complexity.

3.4 Our Attack Design

As we mentioned in the introduction, we want to use linear cryptanalysis many times to get most of the key bits. We use one trail to extract a subset of key bits and another trail to get another subset of key bits. Until the last round keys K_{N_r+1} are all extracted then we go one level up to extract the key bits of K_{N_r} with new trails and so on.

3.4.1 Observations

Before we explain our strategies, there are some observations to be made.

1. The subkeys we are going to attack should not be too many in a single attack, i.e., the S-boxes involved in the last round should not be too many. This is because the more subkeys we want to extract in one attack the more time we need. For example, if we want to get 8 key bits in one time, then we have to test 2^8 candidate key bits for all pairs. But if we get 4 bits and then another 4 bits in two attacks, we only need to test 2×2^4 candidate keys for all pairs.
2. The fewer S-boxes are involved the larger the bias. So, maybe there exists one input-output XOR having the largest bias, but its output spreads to many S-boxes in the permutation. Then we should consider if it is worthwhile to choose such path.
3. It is easy to see that with first N_r-1 round trail we can get bits of K_{N_r+1} . So with N_r-2 round trail we can get bits of K_{N_r} . Continuing the process we can get all key bits up to K_3 . But there is no linear expression for the first two round keys so we can't use linear cryptanalysis to get them. The rest subkeys may be derived by exhaustive search.
4. We may take advantage of the key schedule such as the shift key schedule to know upper round keys from the lower round keys we already get. Thus, we can save time in getting the repeated key bits.

3.4.2 Strategies

In general, if a linear expression of entire cipher has bias ϵ , then it is suggested that we need $c\epsilon^{-2}$ pairs to attack for a constant c . See the following table from [18].

Table 3: The success rate of linear cryptanalysis.

c	2	4	8	16
Success Rate	48.6%	78.5%	96.7%	99.9%

Here, we give a simple proof for this result.

Proof: Let N denote the number of plaintext-ciphertext pairs we need. And let Y_i be the result of equation (3.1) in the i^{th} test.

$$Y_i = P_i^i \oplus C_J^i \oplus K_K^i, i=1,2,\dots,N, Y_i \in \{0,1\} \quad (3.4)$$

Since we have bias equal to e , which means $\Pr[Y_i = 0] = \frac{1}{2} + e$. Then the expected value of Y_i is $E(Y_i) = \frac{1}{2} - e$. And we have

$$m = E(Y_i) = \frac{1}{2} - e,$$

$$s^2 = \text{Var}(Y_i) = (\frac{1}{2} - e)(\frac{1}{2} + e) = \frac{1}{4} - e^2 \approx \frac{1}{4} \text{ since } e \text{ is small.}$$

Then we let $\bar{Y}_N = \frac{Y_1 + Y_2 + \dots + Y_N}{N}$. And we can find it to be a normal distribution

$$\begin{aligned} \bar{Y}_N &\sim N(m, \frac{s^2}{N}) \\ &\approx N(\frac{1}{2} - e, \frac{1}{4N}) \end{aligned}$$

So we want to distribute $\frac{1}{2} \pm e$ from $\frac{1}{2}$ like the following figure.

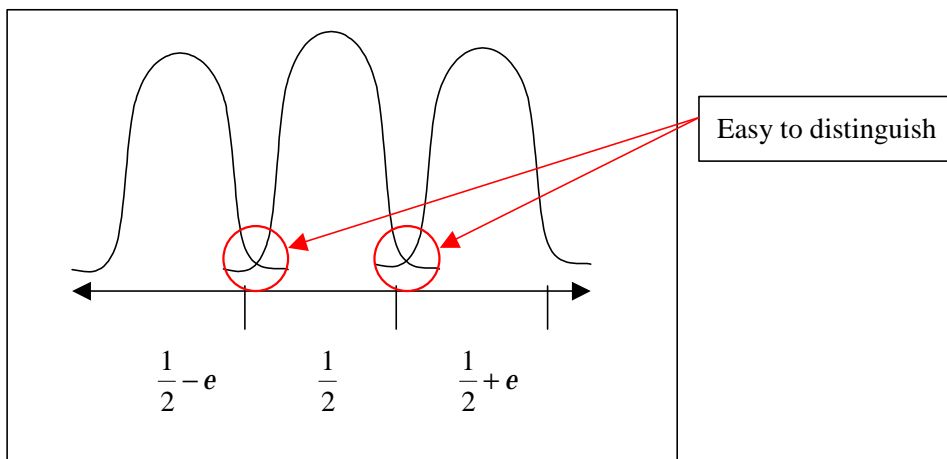


Figure 14: Probability distribution

We hope that taking 3 times the standard deviation is still less than e .

$$\begin{aligned}
& 3\sqrt[3]{\frac{1}{4N}} < e \\
& \Rightarrow 3 < e \times 2\sqrt[3]{N} \\
& \Rightarrow N > ce^{-2}
\end{aligned}$$

So we can see that it is about to take ce^{-2} pairs to test. □

And from observation 1, the computation time is also related to the number of key bits being attacked. So we define the *cost* of a trail with bias e to be $e^{-2} \times 2^k$ where k is the number of key bits being attacked. Then we are ready to introduce our strategies as follows:

(a) Trails finding: We do the process in a “recursive” way. We start it from the first S-box and look up the linear approximation table row by row. Here we set a bias threshold about 1/4. Once we meet a value ≥ 12 or ≤ 4 (bias $\pm 1/4$) we then select the corresponding input-output XOR and follow the output to the next round. The previous output now becomes the input of the S-box, so we only need to look up the corresponding row of the linear approximation table. The process continues and in the end, we can get an expression as (3.2) and the bias is calculated by Piling-up lemma. Every time we finish finding a trail, we record the corresponding data into a queue including the path it walked, the bias e , and the cost ($e^{-2} \times 2^k$). Then we return and going to find another trail until all possible trails are found. Since there won't be too many biases larger than 1/4 or smaller than $-1/4$, the search process won't take too much time.

(b) Cost of trails updating: When the trails are all found, we select the smallest cost to be the first step since it takes the fewest time to solve. After this trail solved, we need to update the cost before we select a next trail to attack since there may be many trails covering the same key bits we already got. And those key bits don't need to be

extracted again, so the cost can divide by 2^s , where s is the number of covered key bits already derived. After all the cost of trails is updated, we can select next trail from the smallest cost again. Until all K_{Nr+1} are all extracted, we then move one level up to attack bits of K_{Nr} using similar method. Continuing the procedure we can solve all round key bits K_3 to K_{Nr+1} as observation 3 stated. The rest two rounds K_1 and K_2 can be solved by exhaustive search.

(c) Backtracking: When we try to linear attack the key bits, we choose the one whose counter is the closest to $(\frac{1}{2} \pm e)T$. In addition, we store r possible candidate keys whose counters are also close to $(\frac{1}{2} \pm e)T$, where r is a flexible parameter and can be modified. In the end, if the key we extracted tests to be wrong, we can go back to choose another candidate subkey systematically. Figure 15 shows this backtracking scheme. If we run out all possible candidate keys, then we declare this attack fails.

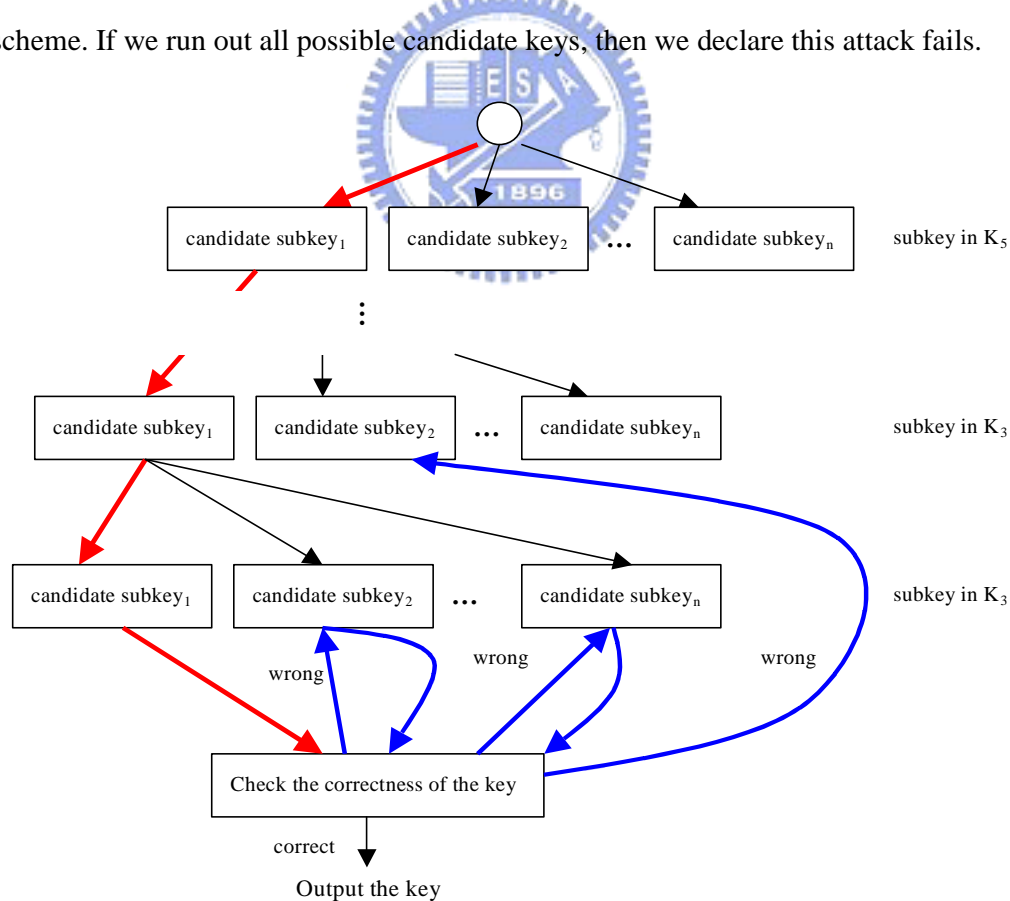
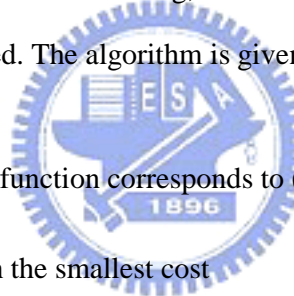


Figure 15: The backtracking scheme

3.4.3 Algorithm

With the three strategies introduced in the previous section, we come up with the algorithm as follows. We first need to find all possible trails with large bias by using the TrailsFinding part. After this process is done, we then have many trails stored in the queue with corresponding bias, costs, and the paths they walk. Thus we can select the smallest cost to attack first. After the trail is attacked, we then do the CostUpdating to update the cost and get the next smallest trail to attack. Once a round key bits are all extracted, we can go up one round to repeat the attack with similar strategies. Finally, we use several plaintext-ciphertext pairs to verify the keys we extracted. If the keys are tested to be wrong, we then apply the BackTracking to use other candidate keys we stored. The algorithm is given below in pseudo C.



```
1  TrailsFinding() // This function corresponds to (a) in Section 3.4.2
   for i = Nr + 1 to 3
     do {
       2  select a trail with the smallest cost
       3  linear attack  $K_i$ 
         3.1 choose the key with counter closest to  $(\frac{1}{2} \pm e)T$ 
         3.2 save some other keys also close to  $(\frac{1}{2} \pm e)T$ 
       4  CostsUpdating() // This function corresponds to (b) in Section 3.4.2
       5  If not all bits in  $K_i$  are extracted
           Then go to 2
     }
6  Exhaustive search ( $K_1, K_2$ )
7  Check the correctness
   7.1 If success then return the key
   7.2 Else BackTracking() // This function corresponds to (c) in Section 3.4.2
       7.2.1 If all candidate keys are failed
           Then return Failure
```

3.4.4 Performance

We use SPN with 16-bit block length, 32-bit initial key and with 4 rounds to do the

experiment. The 5 round keys are derived from initial 32-bit key by simple shift operation.

All S-boxes used in the SPN are the same as follows:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$p_s(x)$	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

We run the experiment on Pentium III 733 CPU with 256MB RAM under FreeBSD OS using C Language

We simulate 100 batches with different keys. Each batch randomly chooses around $8 \times e^{-2}$ plaintext-ciphertext pairs (data complexity). Table 4 shows the subkey length, bias, and data complexity for the two trails used in searching the key bits of the last round. Note that although the 2nd trail seems to have large bias, it is not being attacked first. This is because the 2nd trail actually attacks 12 bits and 4 of them are attacked by trail 1. Thus in the original cost, trail 2 has larger cost than trail 1.

Table 4: The trail data

	Subkey length	Bias of trail	Data complexity
1 st trail	8 bits	0.059326	2272
2 nd trail	8 bits	0.079102	2272

If backtracking strategy is used, Table 5 shows the success rate of these 100 batches in the two trails with different number of candidate subkeys stored. It illustrates the merit of this method.

Table 5: Success rate with different number of candidate keys

	# of candidates, r	1	2	3	4	5	6	7	8	9	10
1 st trail	Success rate (%)	6	12	22	34	44	52	62	70	78	96
2 nd trail	Success rate (%)	8	20	27	36	46	55	65	72	82	97

Chapter 4 Design S-boxes against Linear Cryptanalysis

In this chapter, we will introduce the properties of Boolean functions and multiple-output Boolean functions. Then we discuss about the similarities of S-boxes and multiple output Boolean functions. And we will analyze the linear approximation table to see what properties S-boxes should have to resist linear cryptanalysis. Finally we introduce the construction method of AES S-box and further analyze it.

4.1 Boolean Functions

First, let's see some notations that are commonly used in Boolean functions. Let $\mathbf{F}_2 = GF(2)$. We consider the domain of a Boolean function to be the vector space (\mathbf{F}_2^n, \oplus) over \mathbf{F}_2 , where \oplus is used to denote the addition operator over both \mathbf{F}_2 and \mathbf{F}_2^n (XOR in this case). Suppose $X=(x_1, x_2, \dots, x_n) \in \mathbf{F}_2^n$ is a length n vector. Then an n-variable Boolean function is defined to be a mapping from \mathbf{F}_2^n to \mathbf{F}_2 . We use Ω_n to denote the set of all n-variable Boolean functions. For a Boolean function $f(X) \in \Omega_n$, we can represent it uniquely by the *algebraic normal form* (ANF):

$$f(X) = a_0 + a_1x_1 + \dots + a_nx_n + a_{12}x_1x_2 + a_{13}x_1x_3 + \dots + a_{12\dots n}x_1x_2\dots x_n \quad (4.1)$$

where the coefficient a_i can be 0 or 1. Note we sometimes use + to represent \oplus for simplicity. And all the outputs of $f(X)$ form the 0-1 sequence called *truth table*, denoted by f .

Example 4.1: Suppose the ANF of $f(X) \in \Omega_2$ is $f(X) = x_1 + x_1x_2$, then the truth table f is:

x_1	x_2	x_1x_2	$x_1 + x_1x_2$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	0

f is the output of all possible inputs, i.e., $f=0010$. □

The *inner product* of two vectors $f, g \in \mathbf{F}_2^n$ (f and g can be viewed as the output truth table of $f(X)$ and $g(X)$) will be denoted by $\langle f, g \rangle$. The *Hamming weight* of an n -bit vector u is the number of nonzero elements (number of ones in this case) in f and denoted by $wt(f)$. The *Hamming distance* between two vectors f, g is the number of places where they differ and denoted by $d(f, g)$. And we have $d(f, g) = wt(f+g)$.

Example 4.2: If $f=0110$, $g=1101$, then the Hamming weight of f and g are $wt(f)=2$, $wt(g)=3$ respectively and the Hamming distance $d(f, g)=3=wt(f+g)$. □

Next let's define two important properties of Boolean functions: *balancedness* and *algebraic degree*.

Definition 4.1: For an n -variable Boolean function $f(X)$, if $wt(f)=2^{n-1}$, then $f(X)$ has the property of balancedness.

Definition 4.2: The algebraic degree of an n -variable Boolean function $f(X)$ is the largest number of variables of the terms in its ANF, denoted by $\deg(f)$.

Example 4.3: Suppose $f(X) = x_1 + x_2$ and $g(X) = x_1x_2$ both belong to Ω_2 , then the corresponding truth table output are 0110 and 0001. We can see that $f(X)$ is balancedness while $g(X)$ is not. In addition, $\deg(f)=1$ and $\deg(g)=2$ since $g(X)$ has two variables in one term. □

For those Boolean functions with degree less than or equal to 1, we call them *affine functions* or *linear functions*.

Definition 4.3: For an n-variable Boolean function $f(X)$ and $\deg(f) \leq 1$, we can represent them as

$$f(X) = a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n \quad (4.2)$$

Where $a_i \in GF(2)$, $0 \leq i \leq n$. We call such Boolean function as affine function. If $a_0=0$, then it is also called linear function. $A(n)$ and $L(n)$ denote the set of all n-variable affine functions and linear functions respectively.

For an n-variable linear function $f(X)$, it can be represented in the inner product form $f(x) = a \cdot X$ or $f(x) = \langle a, X \rangle$. Let's see an example to illustrate affine functions and linear functions.

Example 4.4: If $f(X) \in \Omega_3$ and $\deg(f) \leq 1$, then all possible $f(X)$ are listed as follows:

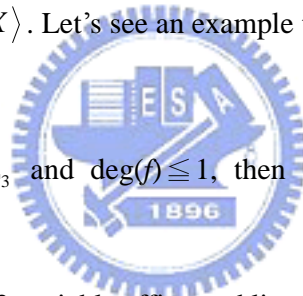


Table 6: 3-variable affine and linear functions

A(3)	
L(3)	
0	1
x_1, x_2, x_3	$1+x_1, 1+x_2, 1+x_3$
$x_1+x_2, x_2+x_3, x_1+x_3$	$1+x_1+x_2, 1+x_2+x_3, 1+x_1+x_3$
$x_1+x_2+x_3$	$1+x_1+x_2+x_3$

□

Next let's define the *Walsh transformation*, which is a very useful tool in the analysis of Boolean functions.

Definition 4.4: For an n-variable Boolean function $f(X)$, the Walsh transform $W_f(u)$

is defined to be:

$$W_f(u) = \sum_{X \in F_2^n} (-1)^{f(X) + u \cdot X} \quad (4.3)$$

Also we let $W_{(f)}(u) = \frac{1}{2^n} W_f(u)$. (4.4)

Here, $u \cdot X$ is a linear function. Since $f(X) + u \cdot X$ is 0 when $f(X) = u \cdot X$ and 1 when $f(X) \neq u \cdot X$. So we can view the Walsh transformation as $W_f(u) = \#\{X \mid f(X) = u \cdot X\} - \#\{X \mid f(x) \neq u \cdot X\}$, i.e., the number of $f(X)$ equal to $u \cdot X$ minus the number of unequals. Then we can derive the relationship between Walsh transformation and the Hamming distance $d(f, u \cdot X)$.

$$\begin{aligned} W_f(u) &= \#\{X \mid f(X) = u \cdot X\} - \#\{X \mid f(x) \neq u \cdot X\} \\ &= 2^n - 2\#\{X \mid f(X) \neq u \cdot X\} \\ &= 2^n - 2d(f, u \cdot X) \\ &\Rightarrow d(f, u \cdot X) = 2^{n-1} - \frac{1}{2}W_f(u) \end{aligned} \quad (4.5)$$

Next, let's define an important property of Boolean functions.

Definition 4.5: The *nonlinearity* $nl(f)$ of an n-variable Boolean function $f(X)$ is defined to be the minimum distance between $f(X)$ and all n-variable affine functions, i.e.,

$$nl(f) = \min_{g \in A(n)} d(f, g) \quad (4.6)$$

In addition, the affine function that has minimum distance with $f(X)$ is called the *best affine approximation* of $f(X)$.

We can deduce the relation between nonlinearity and Walsh transformation.

Theorem 4.1: For an n-variable Boolean function $f(X)$, the nonlinearity of $f(X)$ can be represented as

$$nl(f) = 2^{n-1} - \frac{1}{2} \max_{u \in F_2^n} \{|W_f(u)|\} \quad (4.7)$$

Proof: From Equation (4.5) we know

$$\begin{aligned}
 d(f, u \cdot X) &= 2^{n-1} - \frac{1}{2} W_f(u) \\
 &\Rightarrow \min_{u \cdot X \in A(n)} \{d(f, u \cdot X)\} \\
 &= \min_{u \in F_2^n} \{2^{n-1} - \frac{1}{2} W_f(u)\} \\
 &\Rightarrow nl(f) = 2^{n-1} - \max_{u \in F_2^n} \{|W_f(u)|\} \quad \square
 \end{aligned}$$

Example 4.5: $f(X) \in \Omega_3$ and the truth table output $f=01100011$. Then from the following table we can find that $nl(f)=2$.

Table 7: Nonlinearity example

Function	Truth table (for c=0)								d(f, uX)	
	(000)	(001)	(010)	(011)	(100)	(101)	(110)	(111)	c=0	c=1
f	0	1	1	0	0	0	1	1		
c	0	0	0	0	0	0	0	0	4	4
x_1+c	0	0	0	0	1	1	1	1	4	4
x_2+c	0	0	1	1	0	0	1	1	2	6
x_3+c	0	1	0	1	0	1	0	1	4	4
x_1+x_2+c	0	0	1	1	1	1	0	0	6	2
x_1+x_3+c	0	1	0	1	1	0	1	0	4	4
x_2+x_3+c	0	1	1	0	0	1	1	0	2	6
$x_1+x_2+x_3+c$	0	1	1	0	1	0	0	1	2	6

□

We can also use Walsh transformation to state the balancedness of Boolean functions.

Theorem 4.2: An n-variable Boolean function is balanced if and only if $W_f(0) = 0$.

Proof: From the definition of Walsh transformation, we know when $u=0$,

$$W_f(0) = \sum_{X \in F_2^n} (-1)^{f(X)} = \#\{X \mid f(X) = 0\} - \#\{X \mid f(X) = 1\} \quad (4.8)$$

Case \Rightarrow : Since $f(X)$ is balanced, $\#\{X \mid f(X) = 0\} = \#\{X \mid f(X) = 1\} = 2^{n-1}$, so

$$W_f(0) = 2^{n-1} - 2^{n-1} = 0.$$

Case \Leftarrow : Since $W_f(0) = 0$, we have

$$\begin{aligned} \#\{X \mid f(X) = 0\} - \#\{X \mid f(X) = 1\} &= 0 \\ \Rightarrow \#\{X \mid f(X) = 0\} &= \#\{X \mid f(X) = 1\} = 2^{n-1} \end{aligned}$$

So $f(X)$ is balanced. □

Finally, we introduce a special Boolean function called *Bent* function.

Definition 4.6: Suppose $f(X) \in \Omega_n$ and n is even. If for all $u \in F_2^n$

$$\sum_{X \in F_2^n} (-1)^{f(X) + u \cdot X} = \pm 2^{\frac{n}{2}}$$

Then we call $f(X)$ a Bent function.

Theorem 4.3: For an $f(X) \in \Omega_n$ and n is even, then the following properties describe the same thing:

- (1) $f(X)$ is a Bent function.
- (2) $nl(f) = 2^{n-1} - 2^{\frac{n}{2}-1}$, and $nl(f)$ is the largest nonlinearity of all n -variable Boolean functions.
- (3) $\#\{X \mid f(X) = 1\} = 2^{n-1} \pm 2^{\frac{n}{2}-1}$, $\#\{X \mid f(X) = 0\} = 2^{n-1} \mp 2^{\frac{n}{2}-1}$.
- (4) $f(X)$ is perfect nonlinear.

From properties (3) we can see that Bent functions are not balanced. However, they have the largest nonlinearity, and this makes Bent functions an important component in cipher systems.

4.2 Multiple-Output Boolean Functions (S-boxes)

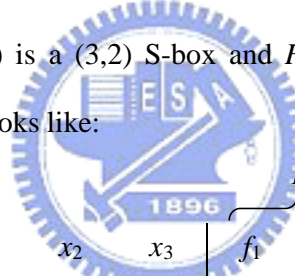
In this section, we turn to introduce the properties of multiple-output Boolean functions. An n -input, m -output Boolean function $F(X)$ is a map

$$F : \{0,1\}^n \rightarrow \{0,1\}^m.$$

It can be viewed as the combination of m single-output Boolean functions, i.e., $F(X)=(f_1(X), f_2(X), \dots, f_m(X))$. The S-boxes we used in SPNs are multiple-output Boolean functions, too. Note that the S-boxes used in SPNs or Feistel Networks require certain properties, not all multiple output Boolean functions are satisfied. This will be described later. And from now on, we call n -input, m -output Boolean functions as (n,m) S-boxes.

Example 4.6: Suppose $F(X)$ is a $(3,2)$ S-box and $F(X)=(f_1(X), f_2(X))=(x_1+x_2, x_2x_3)$.

Then the truth table output looks like:



x_1	x_2	x_3	F	
			f_1	f_2
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

□

Now, let's see some properties of (n,m) S-box.

Definition 4.7: An (n,m) S-box $F(X)=(f_1(X), f_2(X), \dots, f_m(X))$ is said to be balanced (uniformly distributed) if and only if all nonzero linear combinations of f_1, f_2, \dots, f_m are

balanced.

The S-boxes used in SPNs should be balanced.

Definition 4.8: The algebraic degree of an (n,m) S-box $F(X)=(f_1(X),f_2(X),\dots,f_m(X))$ is defined to be the minimum degree of all nonzero linear combinations of f_1, f_2, \dots, f_m , i.e.,

$$\deg(F) = \min_g \{ \deg(g) \mid g = \bigoplus_{i=1}^m a_i f_i, a_i \in F_2, (a_1, a_2, \dots, a_m) \neq 0 \}. \quad (4.9)$$

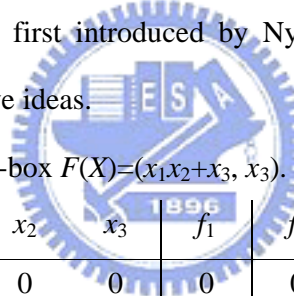
Definition 4.9: The nonlinearity of an (n,m) S-box $F(X)=(f_1(X),f_2(X),\dots,f_m(X))$ is defined to be the minimum nonlinearity among all nonzero linear combinations of f_1, f_2, \dots, f_m , i.e.,

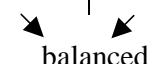
$$nl(F) = \min_g \{ nl(g) \mid g = \bigoplus_{i=1}^m a_i f_i, a_i \in F_2, (a_1, a_2, \dots, a_m) \neq 0 \}. \quad (4.10)$$


The definition of $nl(F)$ was first introduced by Nyberg in [26]. We then take an example to illustrate the above ideas.

Example 4.7: For an (n,m) S-box $F(X)=(x_1x_2+x_3, x_3)$.

x_1	x_2	x_3	f_1	f_2	f_1+f_2
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	0	1	1







All nonzero linear combinations of f_1, f_2 are f_1, f_2, f_1+f_2 . And we can see that f_1+f_2 is not balanced. Although f_1 and f_2 are balanced, $F(X)$ is still not balanced.

The algebraic degree of $F(X)$ is $\min\{\deg(f_1), \deg(f_2), \deg(f_1+f_2)\}=\min\{2,1,2\}=1$.

The nonlinearity $nl(F) = \min\{nl(f_1), nl(f_2), nl(f_1+f_2)\}$. □

In the next section, we will show that nonlinearity is the most important property as far as the linear approximation table is concerned.

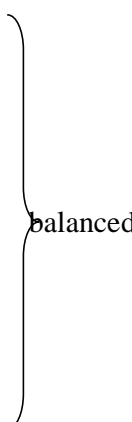
4.3 The Linear Approximation of S-boxes

In this section, we will analyze the linear approximation table and see what is it related to the properties we introduced in the previous section. Also we hope to find out what properties can help to resist the linear cryptanalysis. First, let's see the following lemma.

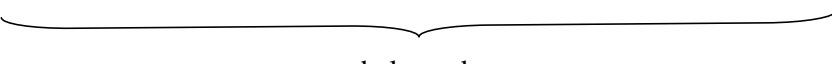
Theorem 4.4: The sum of any row or any column in the linear approximation table will equal to $\pm 2^{n-1}$ for an (n,n) S-box.

Proof: Let's take the sum of row as an example to prove, the sum of column is similar. Since this is a bijective S-box, any nonzero linear combination of the output will be balanced, i.e., there are equal zeros and ones. See the following $n=3$ case.

Y_1	Y_2	Y_3	Y_1+Y_2	Y_2+Y_3	Y_1+Y_3	$Y_1+Y_2+Y_3$	0
0	0	1	0	1	1	1	0
1	0	1	1	1	0	0	0
0	1	1	1	0	1	0	0
1	1	0	0	1	1	0	0
0	1	0	1	1	0	1	0
1	0	0	1	0	1	1	0
1	1	1	0	0	0	1	0
0	0	0	0	0	0	0	0



balanced



balanced

From the above table we see that not only nonzero linear combinations of output truth table are balanced, but also the row they formed except for the all zero one. From the definition of linear approximation table, $\left(\bigoplus_{i=1}^n a_i X_i\right) \oplus \left(\bigoplus_{i=1}^n b_i Y_i\right) = 0$, for some $a_i, b_i \in \{0,1\}$, the sum of one row of linear approximation table means we fix on a certain linear combination of X_i ($1 \leq i \leq n$) and test all linear combination of Y_i ($1 \leq i \leq n$). Let r be the truth table of certain combination of $\left(\bigoplus_{i=1}^n a_i X_i\right)$ and r^z ($z \in \{1,2,\dots,2^n\}$) the bit corresponding to the all zero row (the shaded row). Then there are two cases:

(a) $r^z=0$: Since $0 \oplus 0$ must be 0 and there are 2^n combination of $\left(\bigoplus_{i=1}^n b_i Y_i\right)$ so the sum of row now has at least 2^n .

(b) $r^z=1$: Since $1 \oplus 0 = 1$, this bit contributes no weight to the sum.

Let's see the rest 2^n-1 bits of r . From the above table, we know the combination of $\left(\bigoplus_{i=1}^n b_i Y_i\right)$ form a balanced truth table. Thus no matter the rest bits of r are zeros or ones, they each add 2^{n-1} weight to the sum. So we have the total sum to be $2^n-1(2^{n-1})+2^n$ in case (a) and $2^n-1(2^{n-1})$ in case (b). Since we minus 2^{n-1} for each entry in advance in the linear approximation table, we have to minus $2^{n-1} \times 2^n$ in total for one row. Thus we have the sum of linear approximation table to be $\pm 2^{n-1}$. \square

Next, let's see the relation between the linear approximation table and the nonlinearity. In [12], they have the following equation. The best linear approximation of an S-box occurs with probability p_e where

$$|p_e - \frac{1}{2}| = \frac{2^{n-1} - NL_{min}}{2^n} \quad (4.11)$$

and NL_{min} is the nonlinearity of the S-box. Here, we state this in another way and prove it.

Theorem 4.4: The nonlinearity of an (n,n) S-box is equal to 2^{n-1} minus the maximum absolute value, denoted by $|k|$, of the linear approximation table.

Proof: From Definition 4.8, we know that the nonlinearity of an (n,n) S-box is the minimum nonlinearity of all nonzero linear combination of f_i , where the S-box is $F=(f_1, f_2, \dots, f_n)$. And from the definition of the linear approximation table, each column

of $\bigoplus_{i=1}^n b_i Y_i$ contains the all linear combination of f_i . Thus the smallest nonlinearity among all $\bigoplus_{i=1}^n b_i Y_i$ is the nonlinearity of the S-box. Now we view $\bigoplus_{i=1}^n b_i Y_i$ as one $f(X)$

and prove the theorem by the Walsh transformation. From Theorem 4.1, we know the nonlinearity using Walsh transformation is $nl(f) = 2^{n-1} - \frac{1}{2} \max_{u \in F_2^n} \{|W_f(u)|\}$. And the

Walsh transformation is

$$\begin{aligned} W_f(u) &= \sum_{X \in F_2^n} (-1)^{f(X)+u \cdot X} \\ &= \#\{f(X) = u \cdot X\} - \#\{f(X) \neq u \cdot X\} \\ &= 2^n - 2d(f(X), u \cdot X) \end{aligned}$$

Since each row represents one uX for $u \in F_2^n$, we have the following:

$$\begin{aligned} nl(f) &= 2^{n-1} - \frac{1}{2} \max_{u \in F_2^n} \{|W_f(u)|\} \\ &= 2^{n-1} - \frac{1}{2} \max_{u \in F_2^n} \{|2^n - 2d(f(X), u \cdot X)|\} \end{aligned} \tag{4.12}$$

The maximum u happens in two cases:

Case 1: $2^n - 2d(f(X), uX)$ is positive maximum, thus $d(f(X), uX)$ must be the minimum value. This means that number of $f(X)=uX$ is the largest one; that is to say, when it minus 2^{n-1} , it becomes the largest positive k .

Case 2: $2^n - 2d(f(X), uX)$ is negative minimum, thus $d(f(X), uX)$ must be the maximum value. This means that number of $f(X)=uX$ is the smallest one; that is to say, when it minus 2^{n-1} , it becomes negative smallest k .

Combine Case 1 and Case 2 and apply them into the Equation 4.12, we get the nonlinearity of the S-box is $nl(\text{Sbox})=2^{n-1}-|k|$. □

In fact, this lemma is the same as Equation (4.11) as follows.

$$\begin{aligned}
 nl(Sbox) &= 2^{n-1} - k \\
 \Rightarrow \frac{nl(Sbox)}{2^n} &= \frac{2^{n-1} - k}{2^n} \\
 \Rightarrow \frac{k}{2^n} &= \frac{2^{n-1} - nl(Sbox)}{2^n}
 \end{aligned}$$

$\frac{k}{2^n}$ is the bias, which is $p_e - \frac{1}{2}$ and $nl(Sbox)$ is NL_{min} . With this lemma, we know that the larger k is the smaller the nonlinearity of S-box. Since the value will affect the bias of linear expression, we conclude that the smaller the values in the linear approximation table the smaller the biases. In other words, if the nonlinearity of the S-box is larger, then it can resist linear attack more efficiently.

4.4 Construction of S-boxes



In this section, we introduce two common construction methods of S-boxes which can be used in SPNs.

4.4.1 Random Generation

Since the (n,n) S-box we need should be bijective, generating the S-box by randomly permuting the output seems to be the most easy and common way. See the following example.

Example 4.8: We take $(3,3)$ and $(4,4)$ S-boxes as examples and we denote the inputs and outputs in octal and hexadecimal format, respectively.

input	0	1	2	3	4	5	6	7
S-box 1	2	4	7	5	3	1	0	6
S-box 2	6	2	7	1	0	4	5	3

input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
S-box 1	A	3	4	B	E	2	5	C	1	6	0	8	D	7	9
S-box 2	6	A	5	9	0	B	4	D	1	E	7	2	C	3	8

□

The advantage of this method is that it is very simple. However, the corresponding properties are usually not very good. Thus we may take more time in filtering out the properties we need. There are also some other randomly generated methods like [1][10].

4.4.2 Generation using finite field

This method is used in the AES (Rijndael) [3] and it seems to work very well in resisting the linear attack. The AES makes use of the Galois Field over 2^n and we give the algorithm in Algorithm 4.1. This is a very simple algorithm and with good properties, which we will check later. First we denote the S-box input and output in a field element. For example, (00110101) becomes $x^5+x^4+x^2+1$. The S-box input is first represented as field element and then we find the inverse of this element and transfer back to binary form. And we set a constant vector C and a nonsingular matrix B . The final step is to multiply the matrix B with the binary inverse element and add with vector C . The result is then becoming the output of the S-box. The matrix we use comes from choosing the first row and the remains are cyclic shift one bit right from the above row. Notice that the resulting matrix should be nonsingular, otherwise there won't be a bijective mapping. Figure 16 shows the process of generating the output of

the S-box.

Algorithm 4.1:

external FIELDINV, BINARYTOFIELD, FIELDTOBINARY

$z \leftarrow \text{BINARYTOFIELD}(a_7a_6a_5a_4a_3a_2a_1a_0)$

if $z \neq 0$

then $z \leftarrow \text{FIELDINV}(z)$

$(a_7a_6a_5a_4a_3a_2a_1a_0) \leftarrow \text{FIELDTOBINARY}(z)$

$(c_7c_6c_5c_4c_3c_2c_1c_0) \leftarrow (01100011)$

comment: In the following loop, all subscripts are to be reduced modulo 8

for $i \leftarrow 0$ **to** 7

do $b_i \leftarrow (a_i + a_{i+4} + a_{i+5} + a_{i+6} + a_{i+7} + c_i) \bmod 2$

return $(b_7b_6b_5b_4b_3b_2b_1b_0)$

Note: The field elements and their inverses are generated from the irreducible polynomial $h(X) = 1 + X + X^3 + X^4 + X^8$.

FIELDINV transfer a field element to its inverse element.

BINARYTOFIELD transfer a binary sequence to a field element and FIELDTOBINARY does the inverse operation.

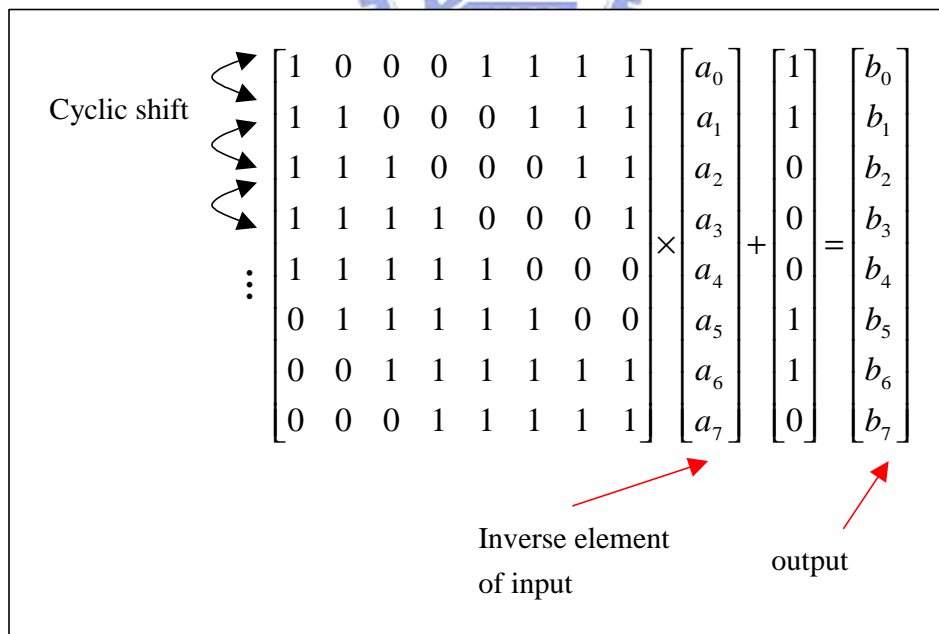


Figure 16: AES S-box Generation

From the construction method, we can see that three parameters could be changed: the

irreducible polynomial $h(X)$, the matrix \mathbf{B} , and the constant vector \mathbf{C} . In the next section, we will discuss over these three parameters and check the difference among them.

4.5 Design Analysis

In this section, we analyze the properties of the construction methods proposed in the previous section.

4.5.1 Analysis of random generation

Generally, the properties of S-boxes generated by random are not very good. We tested about a few hundreds and the maximum absolute value of linear approximation table ranged from 32 to 42 for (8,8) S-boxes and 12 to 18 for (6,6) S-boxes. Interestingly, the average number of appearance will decrease as the maximum absolute value increases. See the following figures.

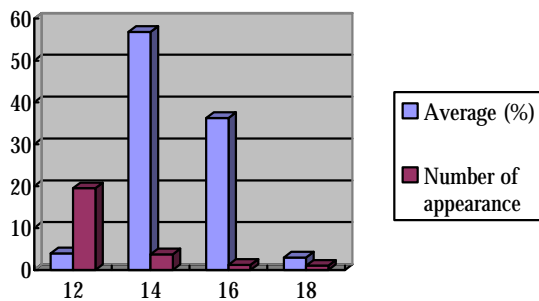


Figure 17: Properties of (6,6) S-box generated from random

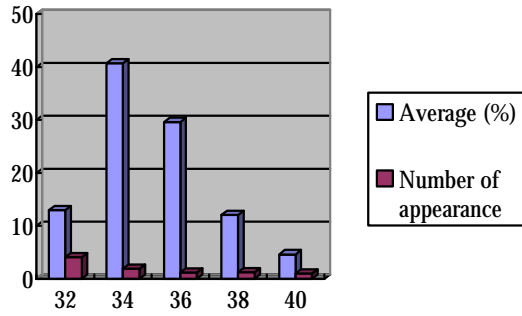


Figure 18: Properties of (8,8) S-box generated from random

4.5.2 Analysis of Generation using finite field

From the end of the previous section, we know three parameters could be changed: the irreducible polynomial, the matrix B, and the constant vector C. Then, we tested these three cases; we find that no matter what we changed, the values are still the same. See the following table (Number of appearance means how many times the maximum value appears in the table).

Table 8: The properties of different (n,n) S-box constructed from GF(2ⁿ)

Number of variables	4	5	6	8
Maximum value	4	6	8	16
Number of appearance	30	31	189	1275

In their original design [3], Daemen and Rijmen declared that the chosen polynomial and the matrix were in simple format. The constant was chosen so that no fixed points exist (no $x=S(x)$) and no opposite points ($\bar{x}=S(x)$), either. And we further analyze this construction in the following parts. Let's first see what the linear approximation table looks like if no matrix and constant vector are used, i.e., the outputs of S-box is the field elements inverse. It appears that this construction will result in a symmetric

linear approximation table.

Proposition 4.1: The S-boxes generated using finite field with no matrix and constant vector applied will result in the linear approximation table with row i equal to column i (a symmetric table).

Proof: Since the only operation now is to calculate the field elements inverse. If element a has inverse b , then element b will have inverse a . Thus the S-box input a will output its inverse b , and in the same way, if the output of the S-box is a , then its input will be b . From the definition of the linear approximation table, row i means that we fix on a certain input pattern (i is represented in $a_1a_2...a_n$ in binary) and compute

$$\left(\bigoplus_{i=1}^n a_i X_i \right) \oplus \left(\bigoplus_{i=1}^n b_i Y_i \right) = 0 \text{ for all output patterns (for all } b_i \text{).}$$

Now since every S-box input-output is a pair, if the bit in $\left(\bigoplus_{i=1}^n a_i X_i \right)$ is different from the bit in $\left(\bigoplus_{i=1}^n b_i Y_i \right)$,

we can find the bit in $\left(\bigoplus_{i=1}^n b_i Y_i \right)$ (for a certain b_i , i is represented in $b_1b_2...b_n$ in binary)

is also different from the bit in $\left(\bigoplus_{i=1}^n a_i X_i \right)$. Thus whenever we find a pair $(a)-(b)$

contributes a distance to row i column j , we can find another pair $(b)-(a)$ contributes the same distance to column i row j . Thus the values of row i in the linear approximation table will equal to column i . □

Example 4.9: See the following (4,4) S-box which is generated from irreducible polynomial $h(X)=1+x+x^4$ and the corresponding linear approximation table.

From the S-box and the linear approximation table, we see that row 1 is equal to column 1 and row 2 is equal to column 2, and...etc. We take row 8 and column 8 to explain more. Row 8 means (1000) in binary, i.e., we see the first column of the truth table of S-box input and it is 00000000 11111111. And we said that the field element and its inverse becomes a pair, i.e., (0000)-(0000), (0001)-(1111), (0010)-(1011), ...,

(1111)-(0001). The first one is the S-box input and the second is the output. The underlined bit becomes the truth table we check in row 8.

0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	1	0	1	1
0	0	1	1	0	1	0	1
0	1	0	0	1	0	0	1
0	1	0	1	0	0	1	1
0	1	1	0	1	1	1	0
0	1	1	1	1	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	0	0	1	0
1	1	0	0	0	1	1	1
1	1	0	1	1	0	1	0
1	1	1	0	0	1	1	0
1	1	1	1	0	0	0	1

+8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0+4+4-2+2-2+2-2+2+2-2
 0 0 0+4+2+2+2-2 0 0 0+4-2-2-2+2
 0 0+4 0+2+2+2-2+2-2-2 0+4 0 0
 0 0+2+2 0+4-2+2 0+4+2-2 0 0-2-2
 0 0+2+2+4 0-2+2-2-2 0 0+2-2+4 0
 0+4+2+2-2-2+4 0 0 0+2-2+2-2 0 0
 0+4-2-2+2+2 0+4+2-2 0 0 0 0-2+2
 0-2 0+2 0-2 0+2-2 0+2 0+2+4-2+4
 0+2 0-2+4-2 0-2 0+2+4+2 0+2 0-2
 0-2 0-2+2 0+2 0+2+4-2 0+4-2 0+2
 0+2+4-2-2 0-2 0 0+2 0+2-2 0+2+4
 0-2-2 0 0+2+2 0+2 0+4-2-2 0+4+2
 0+2-2+4 0-2-2 0+4+2-2 0 0+2+2 0
 0+2-2 0-2+4 0-2-2 0 0+2+4+2+2 0
 0-2+2 0-2 0 0+2+4-2+2+4+2 0 0-2

Figure 19: S-box and its linear approximation table

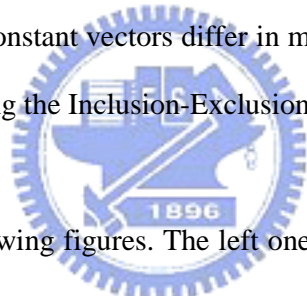
Now if the underlined bit is different from the bold bit, we can find that the same situation in column 8 to row 1. For example, the (0001)-(1111) pair contributes 1 distance to row 8 and column 1. We can find the corresponding pair (1111)-(0001) contributes 1 distance to column 8 and row 1. The (0010)-(1011) pair contributes 1 distance to row 8 and column 1. We can also find the (1011)-(0010) pair contributes 1 distance to column 8 and row 1. Thus whenever we find a pair (*a*)-(*b*) contributes a distance to row 8 column 1, we can find another pair (*b*)-(*a*) contributes the same distance to column 8 and row 1. So we have the values in row 8 equal to the values in column 8. □

Next, let's see what happen when we apply the constant vector. It seems to only affect the sign in the linear approximation table.

Proposition 4.2: The S-boxes generated using finite field with different constant vectors will only affect the sign of the values in the linear approximation table.

Proof: Suppose the two vectors are different in one place k , and then the two corresponding outputs of S-boxes are different in the k th column. And the k th columns are exactly complement of each other due to the construction method of AES S-box.

Thus when we check the equation $\left(\bigoplus_{i=1}^n a_i X_i\right) \oplus \left(\bigoplus_{i=1}^n b_i Y_i\right) = 0$ for the linear approximation table, we find that whenever $b_k=1$, the linear approximation tables of the two S-boxes will differ in their sign. This is because if the equation $\left(\bigoplus_{i=1}^n a_i X_i\right) \oplus \left(\bigoplus_{i=1}^n b_i Y_i\right) = 0$ holds t times for the first S-box, then it holds 2^n-t for the second S-box. Thus after we minus 2^{n-1} to construct the table, they become $\pm(2^{n-1}-t)$. When the two constant vectors differ in more than one place, we still can derive a similar result by using the Inclusion-Exclusion Principle. □



Example 4.10: See the following figures. The left one uses vector 0110 and the right one uses 0101. They differ in b_3 and b_4 , so whenever $b_3=1$ or $b_4=1$ but not both, their sign will be different.

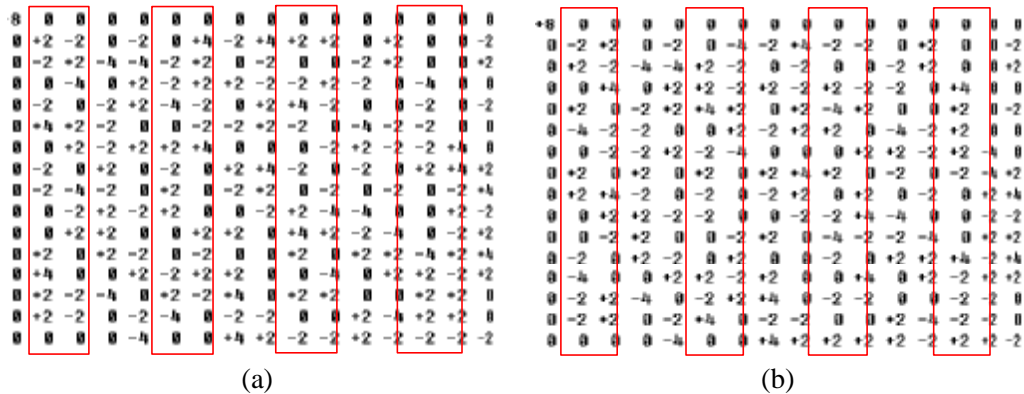


Figure 20: Linear approximation table with different constant vector C □

Then, we check the result when we apply affine matrix B . It shows that the columns of the linear approximation table will be permuted.

Proposition 4.3: The S-boxes generated using finite field with different matrix B will permute the columns of the linear approximation table.

Proof: Let's first assume the constant vector is 0 since we proved that it only affects the sign. Now we are using the same irreducible polynomial but different affine matrix. From the construction method, we see the outputs of S-box come from $B \times A + C$ for all 2^n possible A (inverse of inputs), where B is affine matrix and C is 0 now. Then, let's see what is the relation between two linear approximation tables of different S-boxes using different B . Recall that the columns of linear approximation table is the condition $\left(\bigoplus_{i=1}^n a_i X_i\right) \oplus \left(\bigoplus_{i=1}^n b_i Y_i\right) = 0$ for all possible a_i and certain b_i .

Thus we can view it as a vector $[b_1, b_2, \dots, b_n]$ to multiply the outputs

$$\begin{bmatrix} B \end{bmatrix} \times \begin{bmatrix} A \end{bmatrix} + \begin{bmatrix} C \end{bmatrix}, \text{ i.e., } [b_1, b_2, \dots, b_n] \times \left(\begin{bmatrix} B \end{bmatrix} \times \begin{bmatrix} A \end{bmatrix} + \begin{bmatrix} C \end{bmatrix} \right). \text{ And we focus}$$

on b_i and B , $[b_1, b_2, \dots, b_n] \times \begin{bmatrix} B \end{bmatrix}$ since A and C are fixed.

Suppose the two S-boxes use B_1 and B_2 , respectively. We can always find

$$[b_1, b_2, \dots, b_n] \times \begin{bmatrix} B_1 \end{bmatrix} = [b_1', b_2', \dots, b_n'] \times \begin{bmatrix} B_2 \end{bmatrix}.$$

So the columns $[b_1', b_2', \dots, b_n']$ of second S-box is identical to $[b_1, b_2, \dots, b_n]$ of first S-box. □

Example 4.11: See the following figure. The left one is constructed using

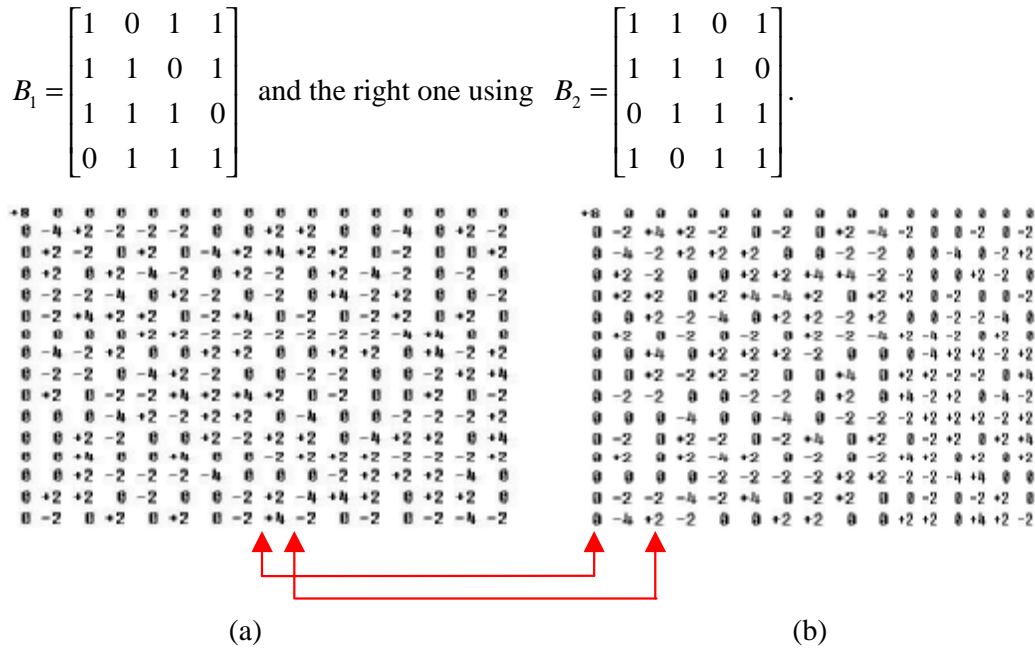


Figure 21: Linear approximation table with different matrix B

We can see that column 1 of (b) is identical to column 8 of (a) (regardless of sign)

since $[1000] \times \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} = [0001] \times \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} = [1011]$. And column 3 of (b)

is identical to column 9 of (a) (ignoring the sign) since

$$[1001] \times \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} = [0011] \times \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} = [1100]. \text{ The rest cases are similar}$$

and we omit them here. □

Finally we also tested different irreducible polynomials and it seems that they still have the same maximum absolute value and the value appears in the table the same times.

Example 4.12: See the following figure. The left one is constructed with irreducible polynomial $h_1(X)=1+x+x^4$ and the right one is $h_2(X)=1+x^3+x^4$. Though they look very

different, they still have the same maximum absolute value 4 and appear in the table the same 30 times.

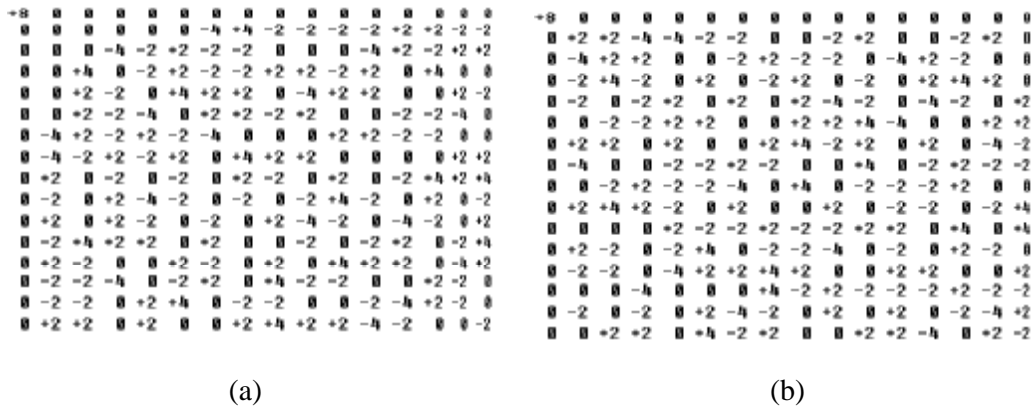


Figure 22: Linear approximation table with different irreducible polynomial □

From the above propositions, we know that changing any one of the parameters won't affect the nonlinearity of the S-box. However, from Proposition 4.1 and Proposition 4.3, we know that with matrix B applied will permute the columns of the linear approximation table so that it won't be too regular (symmetric if no matrix). Besides, the Rijndael designers said the constant vector C can avoid S-box with $x=S(x)$ or $\bar{x}=S(x)$. We found in Proposition 4.2 that the constant vector will affect the sign in the linear approximation table.

Chapter 5 Conclusion

In this thesis, we discussed about the linear cryptanalysis over small substitution-permutation networks. To apply linear cryptanalysis, we first analyzed the linear approximation table of the S-box. With this table we know which bits of input-output XOR of the S-box act in a nonrandom fashion. Then we combined several such input-output patterns to form a trail through the entire SPN. This trail finally becomes an equation $P_i \oplus C_j \oplus K_k = 0$ and indicates what plaintext bits and data bits into last round XOR with extra high or low probability. We calculated the bias of this trail by Piling-up lemma and denote it as e . Then we started to attack the key bits by partially decrypting the plaintext-ciphertext pairs and we had to check whether the equation holds or not. In the end, the key counter closest to $(\frac{1}{2} \pm e)T$ is supposed to be the most likely key bits, where T is the number of plaintext-ciphertext we have.

We thus proposed a simple strategy to determine the trails efficiently by the definition of the **cost**, $e^{-2} \times 2^k$, where e is the bias of the trail and k is the number of bits to attack. The cost tells us how much effort we need to succeed in the attack. By scanning the linear approximation table, we derived many trails and stored their corresponding bias, cost, etc., in a queue. Then we selected the one with the smallest cost to attack first. By using the CostUpdating we introduced in Chapter 3 we updated the cost in the queue and decided which trail to attack next. Finally we combined the Backtracking technique by storing extra candidate keys and we thus improved our success rate.

In the second part of this thesis, we turned to study the resistance of the S-box to linear attacks. We first introduced many properties of Boolean functions since S-box can be viewed as a kind of multiple-output Boolean function. And we found that

nonlinearity is the most important property S-box should have to resist the attack. With larger nonlinearity the values in the linear approximation table will be smaller and thus becomes hard to find a trail with a large bias to attack. Then we turned to discuss about the construction method of AES S-box since the design of AES S-box is proved to be resistant efficiently to linear attacks. And we also found several interesting properties when we changed the three parameters: the constant vector, the affine matrix and the irreducible polynomial. Though there are still several things we don't understand, we believe our result may be useful for the future research.

In the future, we think there are two directions worthy of further study. First, to further analyze the linear attack and try to find new attack strategies. This includes designing strategies to find good linear trails with higher bias so that we can attack with less cost. And we can try to design a brand new strategy to attack or try to combine several techniques we already know, such as key ranking, multiple linear approximations, etc. Second, we can design the construction method of S-boxes with high nonlinearity to resist linear attacks. There are many papers about constructions of multiple output Boolean functions [8][9][13][27], but not all of them satisfying the requirements of S-box. So we can try to modify them without destroying their high nonlinearity property.

References

- [1] C. Adams and S. Tavares, "Good S-boxes are Easy to Find," *Advances in Cryptology-CRYPTO '89*, pp. 612-615, 1989.
- [2] E. Biham and A. Shmir, "Differential cryptanalysis of DES-like cryptosystems," *Journal of Cryptology*, 4(1):3-72, 1991.
- [3] J. Daemen and V. Rijndael, "The Block Cipher Rijndael," *Lecture Notes in Computer Science*, 1820 (2000), 288-296. (Smart Card Research and Applications.)
- [4] J. Daemen and V. Rijndael, "The design of Rijndael.AES," *The Advanced Encryption Standard*, Springer-Verlag, 2002.
- [5] H. Feistel, Cryptography and Computer Privacy, *Scientific American*, 228(5):15-23, 1973.
- [6] H. Feistel, W. A. Notz, and J. L. Smith, "Some Cryptographic Techniques for Machine-to-Machine Data Communications," *Proceedings of the IEEE*, 63(11): 1545-1554, 1975.
- [7] J. Fuller and W. Millan, "Linear Redundancy in S-boxes," *FSE 2003 (Lecture Notes in Computer Science no. 2887)*, Springer-Verlag, pp. 74-86, 2003.
- [8] K. C. Gupta and P. Sarkar, "Construction of Perfect Nonlinear and Maximally Nonlinear multiple-Output Boolean Functions Satisfying Higher Order Strict Avalanche Criteria," *IEEE Trans. Info. Theory*, vol. 50, no. 11, pp. 2886-2893. Nov. 2004.
- [9] K. C. Gupta and P. Sarkar, "Improved Construction of Nonlinear Resilient S-boxes," *IEEE Trans. Info. Theory*, vol. 51, no. 1, pp. 339-348. Jan. 2005.
- [10] F. Hendessi and T. A. Gulliver, A. U. H. Sheikh, "Large S-Box Design Using a Converging Method," *Information Theory 1997*, Proceedings, 1997 IEEE

Symposium on Information Theory, pp.177.

- [11] H. M. Heys, "A tutorial on Linear and Differential Cryptanalysis," *Technical report CORR 2001-17*, Dep. of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada, 2001.
- [12] H. M. Heys and S. E. Tavares, "Substitution-Permutation Networks Resistant to Differential and Linear Cryptanalysis," *Journal of Cryptology*, 9(1996), 1-19.
- [13] T. Johansson and E. Pasalic, "A Construction of Resilient Functions with High Nonlinearity," *IEEE Trans. Info. Theory*, vol. 49, no. 2, pp. 494-501. Feb. 2003.
- [14] L. Keliher, "Linear Cryptanalysis of Substitution-Permutation Networks," PhD. Thesis, 2003.
- [15] L. Keliher, H. Meijer and S. Tavares, "New Method for Upper Bounding the Maximum Average Linear Hull Probability for SPNs," *EUROCRYPT 2001*, LNCS 2045, pp. 420-436, 2001.
- [16] X. Lai and J. Massey, "A proposal for a new block encryption standard," *Advances in Cryptology-EUROCRYPT '90*, LNCS 473, pp. 389-404, Springer-Verlag, 1991.
- [17] X. Lai, J. Massey, and S. Murphy, "Markov ciphers and differential cryptanalysis," *Advances in Cryptology-EUROCRYPT '91*, LNCS 547, pp. 17-38, Springer-Verlag, 1991.
- [18] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," *Advances in Cryptology: Proceedings of EUROCRYPT '93*, Springer-Verlag, Berlin, pages 386-397, 1994.
- [19] M. Matsui, "On Correlation Between the Order of S-boxes and the Strength of DES," *Advances in Cryptology- EUROCRYPT '94 (Lecture Notes in Computer Science no. 950)*, Springer-Verlag, pp. 366-375, 1995.
- [20] M. Matsui, "The First Experimental Cryptanalysis of the Data Encryption

- Standard,” *Advances in Cryptology-CRYPTO '94*, LNCS 839, pp. 1-11, 1994.
- [21] M. Matsui and A. Yamagishi, “A New method for Known Plaintext Attack of FEAL Cipher,” *Advances in Cryptology-EUROCRYPT '92*, Lecture Notes in Computer Science, Vol. 658, pp. 81-91, 1992.
- [22] National Bureau of Standards, Data Encryption Standard (DES), *Federal Information Processing Standard Publication 46*, U. S. Department of Commerce, January 1977.
- [23] K. Nyberg, “Differentially Uniform Mappings for Cryptography,” *Advances in Cryptology: Proceedings of EUROCRYPT '93*, Springer-Verlag, Berlin, pages 55-64, 1994.
- [24] K. Nyberg, “Linear Approximation of Block Ciphers,” *Advances in Cryptology-EUROCRYPT '94*, LNCS 950, pp. 439-444, 1995.
- [25] K. Nyberg, “Perfect Nonlinear S-boxes,” *Advances in Cryptology: Proceedings of EUROCRYPT '91*, Springer-Verlag, Berlin, pages 378-386. 1991.
- [26] K. Nyberg, “On the Construction of Highly Nonlinear Permutations,” in *Advances in Cryptology-EUROCRYPT '92 (Lecture Notes in Computer Science, vol. 658)*. Berlin/Heidelberg/New York: Springer-Verlag, 1993, pp. 92-98.
- [27] E. Pasalic and S. Maitra, “Linear Codes in Generalized Construction of Resilient Functions with Very High Nonlinearity,” *IEEE Trans. Info. Theory*, vol.48, bo. 8, pp. 2182-2191, Aug. 2002.
- [28] R. Rivest, M. Robshaw, R. Sidney, and Y. Lin, “The RC6 block cipher,” *The First Advanced Encryption Standard Candidate Conference*, Proceedings, Ventura, California, August 1998.
- [29] M. J. B. Robshaw and B. S. Kaliski, “Linear Cryptanalysis Using Multiple Approximations,” *Advances in Cryptology-CRYPTO '94 (Lecture Notes in Computer Science no. 839)*, Springer-Verlag, pp. 1-11, 1994.

- [30] B. Schneier, *Applied Cryptography*, Second Edition, John Wiley and Sons, 1996.
- [31] A. Shamir and S. Miyaguchi, "Fast Data Encipherment algorithm: FEAL," *Advances in Cryptology: Proceedings of EUROCRYPT '87*, Springer-Verlag, Berlin, pages 267-278, 1988.
- [32] C. E. Shannon, *Communication Theory of Secure systems*, *Bell System Technical Journal*, 28:656-715, 1949.
- [33] D. R. Stinson, *Cryptography Theory and Practice*, 2nd edition, *Chapman & Hall/CRC*, 2002
- [34] A. F. Webster and S. E. Tavares, "On the Design of S-boxes," *Advances in Cryptology: Proceedings of CRYPTO '85*, Springer-Verlag, Berlin, pages 523-534, 1986.

