

SEED Algorithm

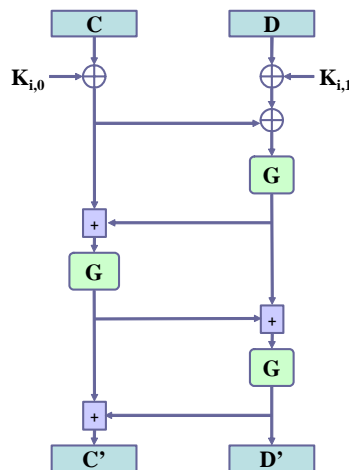
Self Evaluation

1. SECURITY EVALUATION

1.1 DIFFERENTIAL CRYPTANALYSIS CHARACTERISTICS

1.1.1 Introduction

SEED has a 128-bit input data block and a 16 round Feistel structure in which operation is carried out by the divided two 64-bit blocks. The function F, which is the round function has 64 bit input data and is composed of three G functions. This G function has 32-bit input data. The input data of the G function is divided into four 8-bit data. These four 8-bit data go into the four 8×8 S-boxes passing through the diffusion layer.



(Figure 1.1-1) F function in SEED

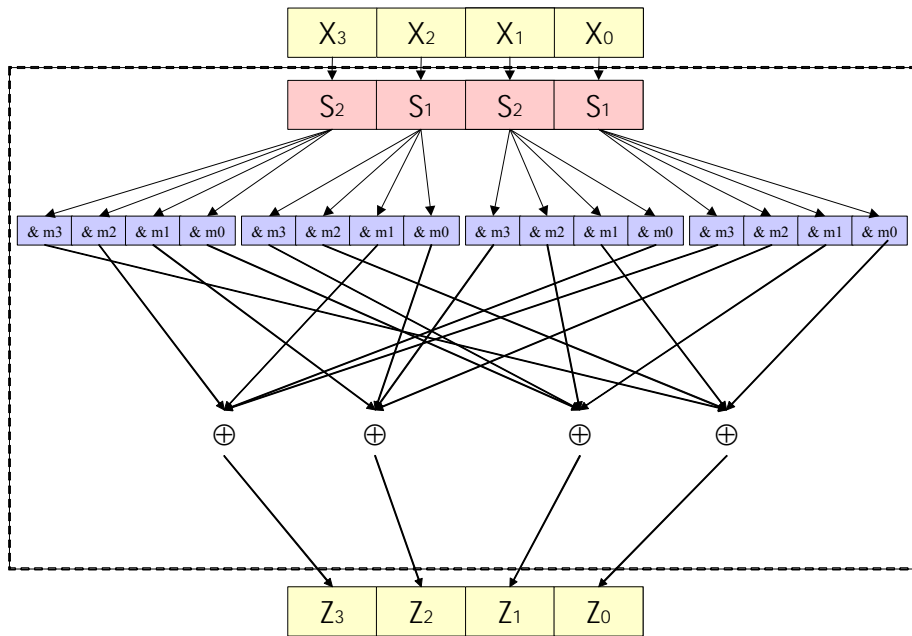
The diffusion of SEED is a linear transformation which has a 32-bit output value with a 32-bit input value. If $m_0 = 0xfc$, $m_1 = 0xf3$, $m_2 = 0xcf$, $m_3 = 0x3f$, and the input value of the diffusion is $X = X_3X_2X_1X_0$, and output value is $Y = Y_3Y_2Y_1Y_0$,

$$Y_3 = (X_0 \& m_3) \oplus (X_1 \& m_0) \oplus (X_2 \& m_1) \oplus (X_3 \& m_2)$$

$$Y_2 = (X_0 \& m_2) \oplus (X_1 \& m_3) \oplus (X_2 \& m_0) \oplus (X_3 \& m_1)$$

$$Y_1 = (X_0 \& m_1) \oplus (X_1 \& m_2) \oplus (X_2 \& m_3) \oplus (X_3 \& m_0)$$

$$Y_0 = (X_0 \& m_0) \oplus (X_1 \& m_1) \oplus (X_2 \& m_2) \oplus (X_3 \& m_3)$$



(Figure 1.1-2) G function in SEED

This linear transformation has at least four active S-boxes in two G functions. When the following four 8-bit input values are selected, (a, b : arbitrary bit string 1 or 0)

$$X_0 = 00000000, X_1 = 00ab0000, X_2 = 00000000, X_3 = 00ab0000$$

$Y = Y_3Y_2Y_1Y_0 = X_3X_2X_1X_0 = X$ will be derived and the number of active S-boxes will become 4.

1.1.2 The Differential Cryptanalysis of the Modified SEED

The modified SEED mentioned here means a version of SEED in which the addition that has the modular of 2^{32} in F function is converted into XOR. Since the combination of the addition and XOR will make the cryptanalysis difficult, this version of SEED will be analyzed first and then, based on this analysis, the analysis of the original version will be attempted. The first stage of this analysis is to examine the S-box. The examination on the DC table for the two S-boxes, S1 and S2 which are used in SEED will show that every column has one 4 and the rest components are 4-uniform which is 0 or 2. Therefore, the probability of the best differential cryptanalysis characteristics is 2^{-6} .

The followings are the input differential cryptanalysis set that will derive the differential characteristics which makes the number of S-boxes become 4:

{ 0x00000000, 0x10001000, 0x20002000, 0x30003000 }

Since the input difference of S1 is always 0, nothing but the consideration on the differential characteristics of S2 is required. The followings are the cryptanalysis of

S2: 0x10 → 0x10 (probability 2^{-7})
S2: 0x10 → 0x20 (probability 2^{-7})
S2: 0x10 → 0x30 (probability 2^{-7})
S2: 0x20 → 0x10 (probability 0)
S2: 0x20 → 0x20 (probability 0)
S2: 0x20 → 0x30 (probability 0)
S2: 0x30 → 0x10 (probability 2^{-6})
S2: 0x30 → 0x20 (probability 0)
S2: 0x30 → 0x30 (probability 2^{-7})

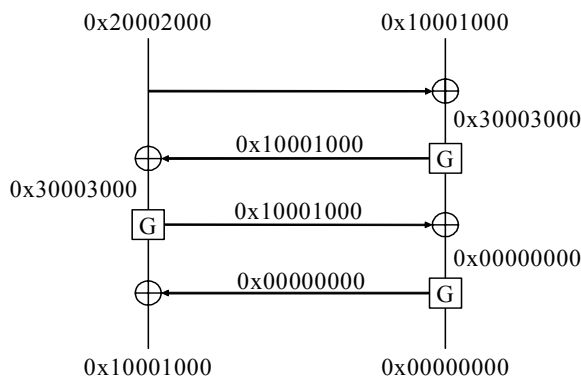
According to this, considering the above set of the input differences and its corresponding output differences, the probability of a useful differential cryptanalysis characteristic of the G function is 2^{-12} and 2^{-14} .

The best method to increase the probability of the differential characteristics of the F function is to make the input difference of any G functions in the F function be 0. This shows the fact that the probability of the differential characteristics of the significant F function is 2^{-24} , 2^{-26} and 2^{-28} .

The first method to attain the differential characteristics is to follow the highest probability found in the DC table. In the above input differences set, the probability of S2: 0x 30 → 0x 10 is the highest. This will provide a clue to follow the highest probability. The fact that the probability of G: 0x30003000 → 0x10001000 is 2^{-12} can be easily confirmed. The practical procedure of seeking the differential characteristics of the F function using this fact is as following:

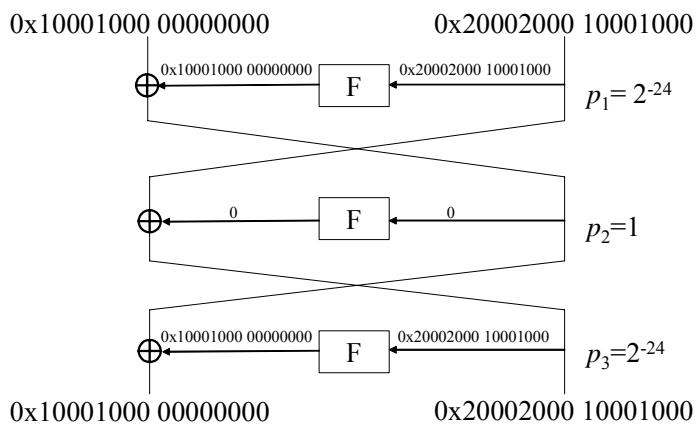
Set the input differences to be 0x20002000 10001000. The input differences of the first G function will be $0x20002000 \oplus 0x10001000 = 0x30003000$ and the output difference is

0x10001000 with the probability of 2^{-12} . Then the input difference will be 0x30003000 and the output difference will be 0x10001000 with the same probability of 2^{-12} . And the input difference of the third G function will be 0. Therefore the probability of F: 0x20002000 10001000 \rightarrow 0x10001000 00000000 will be 2^{-24} . This is the best differential cryptanalysis characteristic of G functions that have been examined so far.



(Figure 1.1-2) DC of the modified F function

With this, the following 3 round differential cryptanalysis characteristics can be derived.



(Figure 1.1-3) The three round DC characteristics of the modified SEED

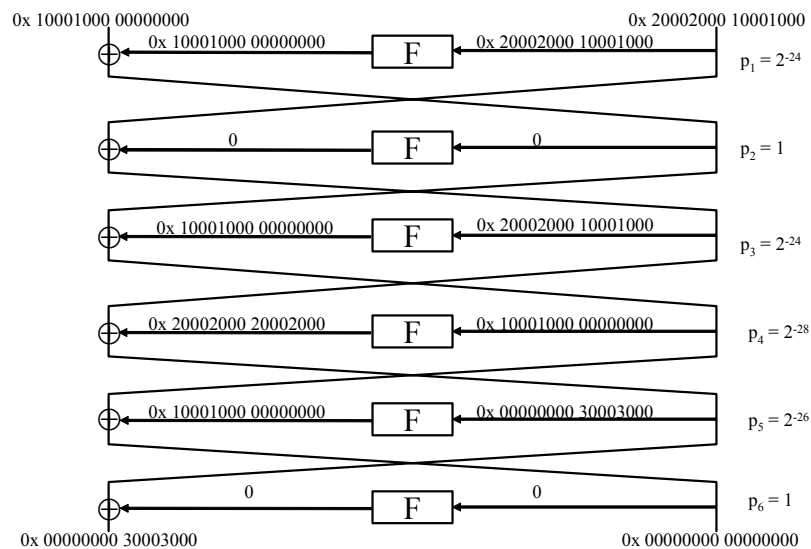
The probability of this differential cryptanalysis is 2^{-48} , which is the highest probability of the 3 round differential cryptanalysis characteristics. But these characteristics cannot be used repeatedly. So another differential cryptanalysis that can be joined together should be found. The following are such differential cryptanalysis characteristics:

$$F : 0x10001000\ 00000000 \rightarrow 0x20002000\ 20002000, \quad p = 2^{-28}$$

$$F : 0x00000000\ 30003000 \rightarrow 0x10001000\ 00000000, \quad p = 2^{-26}$$

Combining these characteristics with the above mentioned differential characteristics, we can derive the following differential cryptanalysis characteristic which has the probability of 2^{-102} :

This differential cryptanalysis characteristic makes it possible to analyze the modified SEED up to the seventh round with 1R(round) attack faster than to analyze it with an exhaustive key search attack. But 2R and 3R attacks are not available because of the diffusion.



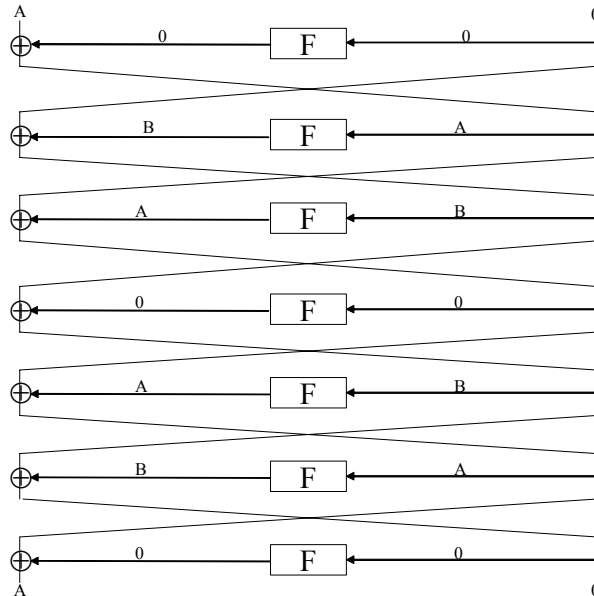
(Figure 1.1-4) The six round differential cryptanalysis characteristics of the modified SEED

The second method to derive the differential cryptanalysis of the modified SEED is to use the recursion of the differential characteristics. Let's assume that the following differential cryptanalysis characteristic recursion is:

$$F : A \rightarrow B, \text{ probability } p$$

$$F : B \rightarrow A, \text{ probability } q$$

Then the following 7 round differential characteristics can be derived.



(Figure 1.1-5) The 7 round characteristics of the modified SEED

The probability of this is p^2q^2 . There is only one pair of these differential cryptanalysis characteristics over the input differential cryptanalysis set we are examining. The following is the pair of the differential characteristics:

$$F : 0x10001000 \ 10001000 \rightarrow 0x30003000 \ 20002000, \text{ probability} = 2^{-28}$$

$$F : 0x30003000 \ 20002000 \rightarrow 0x10001000 \ 10001000, \text{ probability} = 2^{-26}$$

The probability of deriving the above 7 round differential cryptanalysis characteristic with this pair is $(2^{-28})^2(2^{-26})^2 = 2^{-108}$. When we utilize this differential cryptanalysis characteristic, it is possible to analyze the modified SEED up to the eighth round with 1R attack faster than to analyze it with an exhaustive key search attack.

1.1.3 The differential cryptanalysis of SEED

In this section, we discuss the probabilities of the characteristics in the SEED with the original F function, which we found in the modified SEED. When calculating the probability of a characteristic in the modified F function, XOR values were always 1. They get low in the original F function, that is, the probabilities of the output difference to the particular input difference in the additions mod 256 are less than 1.

Let two input differences to addition mod 256 be denoted as A and B, respectively and let C be the output difference from them. This is represented as $A+B \rightarrow C$.

To see the changes in the probabilities of the characteristics in section 1.1.2, we calculate the probabilities of the output difference of the two input differences to addition mod 256 as following:

$$\begin{aligned} 0x20002000 + 0x10001000 &\rightarrow 0x30003000, \text{ probability } 2^{-4} \\ 0x10001000 + 0x10001000 &\rightarrow 0x00000000, \text{ probability } 2^{-2} \\ 0x00000000 + 0x10001000 &\rightarrow 0x10001000, \text{ probability } 2^{-2} \\ 0x00000000 + 0x20002000 &\rightarrow 0x20002000, \text{ probability } 2^{-2} \\ 0x30003000 + 0x30003000 &\rightarrow 0x00000000, \text{ probability } 2^{-4} \\ 0x00000000 + 0x30003000 &\rightarrow 0x30003000, \text{ probability } 2^{-4} \end{aligned}$$

For instance, the probability of $0x10001000 + 0x10001000 \rightarrow 0x00000000$ is calculated in the following way.

We are going to get the probability of $(A+B) \oplus (A'+B') = 0x00000000$, where A, B, A', B' are 32 bit values, $A \oplus A' = 0x10001000$, and $B \oplus B' = 0x10001000$. Let $C = A \oplus B$ and $C' = A' \oplus B'$, and let i-th bits in A, B, C, C' be denoted as a_i, b_i, c_i, c'_i , respectively. The carry bits which occur at the i-th bit positions in C and C' will be represented as Carry_i and Carry'_i . We can see the following expression:

$$c_i = a_i \oplus b_i \oplus \text{Carry}_{i-1}, \quad \text{Carry}_i = a_i b_i \oplus a_i \text{Carry}_{i-1} \oplus b_i \text{Carry}_{i-1}$$

Then, since $c'_{12} = (a_{12} \oplus 1) \oplus (b_{12} \oplus 1) \oplus \text{Carry}_{11} = a_{12} \oplus b_{12} \oplus \text{Carry}_{11} = c_{12}$, $c_{12} \oplus c'_{12}$ is zero with the probability 1, and $\text{Carry}_{12} \oplus \text{Carry}'_{12}$ is $a_{12} \oplus b_{12} \oplus 1$.

$$\begin{aligned} \text{Carry}'_{12} &= (a_{12} \oplus 1)(b_{12} \oplus 1) \oplus (a_{12} \oplus 1)\text{Carry}_{11} \oplus (b_{12} \oplus 1)\text{Carry}_{11} \\ &= a_{12} b_{12} \oplus a_{12} \oplus b_{12} \oplus 1 \oplus a_{12} \text{Carry}_{11} \oplus b_{12} \text{Carry}_{11}, \end{aligned}$$

Thus, $\text{Carry}_{12} \oplus \text{Carry}'_{12} = 0$ with the probability 2^{-1} . Similarly, in the 28th bit position, it is easy to see that $c_{28} \oplus c'_{28} = 0$ and $\text{Carry}_{28} \oplus \text{Carry}'_{28} = 0$ with the probability 2^{-1} . Therefore, the probability of $0x10001000 + 0x10001000 \rightarrow 0x00000000$ is 2^{-2} . The probabilities of others are calculated in a similar way, too.

We can calculate the probabilities of the characteristics in section 1.1.2, using this.

F : 0x20002000 10001000 → 0x10001000 00000000, probability 2^{-32}

F : 0x10001000 00000000 → 0x20002000 20002000, probability 2^{-34}

F : 0x00000000 30003000 → 0x10001000 00000000, probability 2^{-32}

F : 0x10001000 10001000 → 0x30003000 20002000, probability 2^{-36}

F : 0x30003000 20002000 → 0x10001000 10001000, probability 2^{-36}

According to these results, the probabilities of 6 round characteristic and 7 round characteristic in section 1.1.2 become 2^{-130} and 2^{-144} . Hence, these characteristics are not available to the differential attack any more.

1.1.4 Remarks on DC

To analyze the differential cryptanalysis of 16 round SEED with a 128 bit secret key, a 15 round differential characteristic is required, whose probability is greater than 2^{-128} . Though we considered the characteristics in the modified SEED and applied them to the original version of SEED, there are at most 8 round characteristics available to the differential cryptanalysis in the modified SEED and the probabilities of those get lower in the original SEED. Hence, according to the result of analysis that has been carried out so far, SEED is secure against the differential cryptanalysis attacks.

1.2 LINEAR CRYPTANALYSIS CHARACTERISTICS

In this section, the Linear Cryptanalysis (LC) for SEED is carried out. We find a 3 round iterative linear approximation with the linear probability $2^{-67.08}$. This linear approximation is predicted to be an optimal one with the highest linear probability.

The LC complexity against the 16 round SEED is $2^{335.4}$. Thus SEED is considered to be quite secure against LC.

1.2.1 Introduction

SEED is a 128-bit block cipher with a 16 round Feistel structure. The round function F has a 64-bit input value and is composed of three G functions (see Figure 1.1-2). G function, the round function of the F function, has a 32-bit input value, and its input value is divided into

four 8-bit values. Then they pass through the diffusion layer after they get through four 8×8 S-boxes. The Diffusion layer is the linear transformation with a 32-bit input value and a 32-bit output value. When we let $m_0 = 0xfc$, $m_1 = 0xf3$, $m_2 = 0xcf$, $m_3 = 0x3f$, and put the input value of diffusion as $X = X_3X_2X_1X_0$, the output value $Y = Y_3Y_2Y_1Y_0$ is as following:

$$\begin{aligned} Y_3 &= (X_0 \& m_3) \oplus (X_1 \& m_0) \oplus (X_2 \& m_1) \oplus (X_3 \& m_2) \\ Y_2 &= (X_0 \& m_2) \oplus (X_1 \& m_3) \oplus (X_2 \& m_0) \oplus (X_3 \& m_1) \\ Y_1 &= (X_0 \& m_1) \oplus (X_1 \& m_2) \oplus (X_2 \& m_3) \oplus (X_3 \& m_0) \\ Y_0 &= (X_0 \& m_0) \oplus (X_1 \& m_1) \oplus (X_2 \& m_2) \oplus (X_3 \& m_3) \end{aligned}$$

In this section, we are going to carry out the LC for SEED. To carry out this analysis, we have to find the optimal linear approximation which has the highest linear probability. We find a 3 round linear approximation with the linear probability $2^{-67.08}$. This linear approximation is not only the optimal one with the highest linear probability but also an iterative one. When we use the three round iterative linear approximation five times, we can get a 15 round linear approximation which has the linear probability of $2^{-335.4}$. The complexity required for the linear cryptanalysis of SEED is $2^{335.4}$. Therefore, SEED is considered to be secure against linear cryptanalysis.

1.2.2 LC

For the linear cryptanalysis of SEED, we have to find the linear approximation that has the highest linear probability. To find this, we have to find the linear approximation of the F function. Since the F function is composed of three G functions, three additions and XORs, at least more than two G functions should be active and at least more than two additions should be active in the linear approximation of the F function. Therefore, to be a linear approximation of the F function that has a higher probability, not only the linear approximation of the G function but also the linear approximation of the addition should have a higher probability.

1) Basic Definition

[Definition]: We assume $S : \{0,1\}^m \rightarrow \{0,1\}^m$ to be the function that has n as its input bit and m as its output bit. For all $a \in \{0,1\}^n$, $b \in \{0,1\}^m$, the linear probability $LP^S(a,b)$ of S is defined as follows:

$$LP^S(a,b) = 4\{\lambda_S(a,b)\}^2 = \left\{ \frac{2^{\#\{x \in \{0,1\}^n \mid a \bullet x = b \bullet S(x)\}}}{2^n} - 1 \right\}^2$$

We denote the linear approximation as $S : a \rightarrow b$ (probability $LP^S(a,b)$). Where, $x \bullet y$ is the

inner product of the two vectors, x, y calculated over $\{0,1\}$. Since the addition defined over $\{0,1\}^n$ can be thought to be the function $S(x, y) = x + y$ ($S : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$), the linear probability of the addition can be defined as the following:

$$LP^+(\alpha, \beta, \gamma) = \left\{ \frac{2^{\#\{(x, y) \in \{0,1\}^{2n} \mid \alpha \bullet x \oplus \beta \bullet y = \gamma \bullet S(x, y)\}}}{2^{2n}} - 1 \right\}^2$$

2) The Linear Approximation of S-Box

Two S-boxes, S1 and S2, are used in SEED. The maximum linear probability of S1 is 2^{-6} ($\max_{a,b \neq 0} |\lambda_{S_1}(a,b)|=16$) and that of S2 is 2^{-6} . Although using the optimal linear approximation of one S-box can derive the optimal linear approximation of G function, we cannot build the optimal linear approximation of the F function by using optimal linear approximations of the S-boxes. This is because of the fact that the linear approximation of the F function can be made of two linear approximations of the S-box and two linear approximations of the addition. The second highest linear probability is $7^2 / 2^{12}$ ($|\lambda_{S_2}(a,b)|=14$). For example, S2 : 0x01 \rightarrow 0x03, S2 : 0x02 \rightarrow 0x03. Where, α_x denotes in hexadecimal number.

3) The Linear Approximation of G function

When we use one optimal linear approximation of the S-box, we can build the optimal linear approximation of the G function. Because the linear approximation of the F function has two active the G functions and two active additions, the optimal linear approximation of the G function may not be used to build the optimal linear approximation of the F function. G function has four 8-bit inputs, which pass through the diffusion layer after they pass through the four S-boxes S2, S1, S2, S1.

Since the linear approximation of the S-box is

$$\begin{aligned} S2 : 0x00 &\rightarrow 0x00 \text{ (probability 1)} \\ S2 : 0x03 &\rightarrow 0x01 \text{ (probability } 7^2/2^{12}) \end{aligned}$$

and that of the diffusion layer is

$$0x01000100 \rightarrow 0x01000100 \text{ (probability 1),}$$

the linear approximation of the G function is as following:

$$G : 0x03000300 \rightarrow 0x01000100 \text{ (probability } 7^4/2^{12})$$

Similarly, since the linear approximation of the S-box is

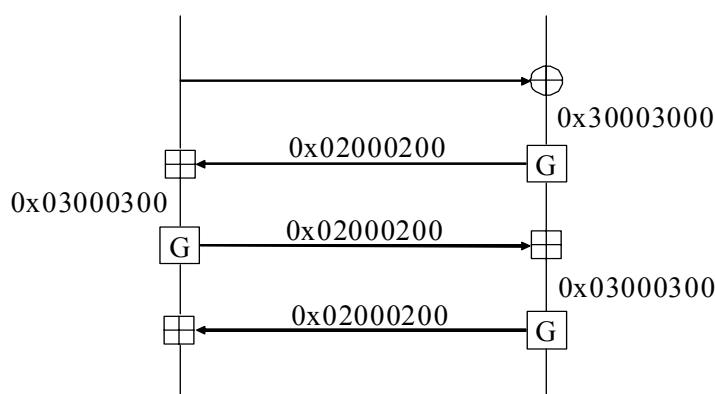
$$\begin{aligned} S_2 : 0x00 &\rightarrow 0x00 \text{ (probability 1)} \\ S_2 : 0x03 &\rightarrow 0x02 \text{ (probability } 7^2/2^{12}\text{)}, \end{aligned}$$

we can find the following linear approximation of the G function:

$$G : 0x03000300 \rightarrow 0x02000200 \text{ (probability } 7^4/2^{12}\text{)}$$

When two S-boxes are active, the optimal linear approximation of the G function has a linear probability $1/2^{12}$. Hence, the linear approximation of the G function derived here is not optimal, but can be considered to be a relatively good linear approximation.

4) The Linear Approximation of F Function



(Figure 1.2-2) The Linear Approximation of F function

Here, we are going to use the following two linear approximations of the G function to derive the linear approximation of the F function:

$$\begin{aligned} G : 0x03000300 &\rightarrow 0x01000100 \text{ (probability } 7^4/2^{24}\text{)} \\ G : 0x03000300 &\rightarrow 0x02000200 \text{ (probability } 7^4/2^{24}\text{)} \end{aligned}$$

We will also use the following linear approximation of the addition twice:

$$0x02000200 \boxplus 0x02000200 \rightarrow 0x03000300 \text{ (probability } 1/2^4\text{)}$$

We can easily calculate the linear probability of the addition when we use the algorithm of Park et al. (1999)[1].

If we combine the above four linear approximation expressions, we can derive the linear approximation of the F function :

$$F: 0x03000300 \ 03000300 \rightarrow 0x03000300 \ 03000300 \quad (\text{probability } 7^8/2^{56} \approx 2^{-33.54})$$

On the other hand, the optimal linear approximation of the addition is

$$0x00000001 \oplus 0x00000001 \rightarrow 0x00000001 \quad (\text{probability } 1)$$

The second highest linear probability is 2^{-2} . But it is impossible to derive the optimal linear approximation of the F function by using these linear approximations. The above linear approximation of the F function is predicted to be the expression which has the highest linear probability. Also, it can be used to build a 3 round iterative linear approximation.

5) The r Round Linear Approximation

< Table 1.2-1> The r Round Linear Approximation

Number of Round	Linear Approximation	Linear Probability
4	0-A-A-0	$2^{-67.08}$
5	0-A-A-0-A	$2^{-100.62}$
6	0-A-A-0-A-A	$2^{-134.16}$
7	0-A-A-0-A-A-0	$2^{-134.16}$
8	0-A-A-0-A-A-0-A	$2^{-167.7}$
9	0-A-A-0-A-A-0-A-A	$2^{-201.24}$
10	0-A-A-0-A-A-0-A-A-0	$2^{-201.24}$
11	0-A-A-0-A-A-0-A-A-0-A	$2^{-234.78}$
12	0-A-A-0-A-A-0-A-A-0-A-A	$2^{-268.32}$
13	0-A-A-0-A-A-0-A-A-0-A-A-0	$2^{-268.32}$
14	0-A-A-0-A-A-0-A-A-0-A-A-0-A	$2^{-301.86}$
15	0-A-A-0-A-A-0-A-A-0-A-A-0-A-A	$2^{-335.4}$

It is difficult to find good linear approximations of the r-round SEED when the number of rounds, r is large. Instead, combining the iterative linear approximation several times can be a

good linear approximation of a large round. We can derive the following three-round iterative linear approximation that has the linear probability of $2^{-67.08}$.

F : 0x00000000 00000000 → 0x00000000 00000000 (probability 1)
F : 0x03000300 03000300 → 0x03000300 03000300 (probability $7^8/2^{56} \approx 2^{-33.54}$)
F : 0x03000300 03000300 → 0x03000300 03000300 (probability $7^8/2^{56} \approx 2^{-33.54}$)
F : 0x00000000 00000000 → 0x00000000 00000000 (probability 1)

Combining the above three-round iterative linear approximation several times will derive the linear approximation of r-round SEED (see <Table 1.2-1>). To abbreviate the symbols in the <table 1.2-1>, we will denote

F : 0x00000000 00000000 → 0x00000000 00000000 as 0 and
F : 0x03000300 03000300 → 0x03000300 03000300 as A.

1.2.3 Remarks on LC

For the linear cryptanalysis of 16 round SEED, a 15 round linear approximation is required. We can find a 15 round linear approximation that has the linear probability of $2^{-335.4}$. This linear approximation is predicted to be the optimal 15 round linear approximation with the highest linear probability. Therefore, the complexity required for the linear cryptanalysis of SEED is $2^{-335.4}$, which shows that SEED is quite secure against the linear cryptanalysis.

1.3 THE CRYPTANALYSIS OF THE KEY SCHEDULING ALGORITHM

The encryption key is converted to subkeys required in the rounds of the encryption and the decryption procedure by the key scheduling algorithm. The analysis of the key scheduling algorithm saves efforts to analyze encryption algorithm. We can decrease the complexity of exhaustive key search attack using weak keys, semi-weak keys, equivalent keys, linear factor and complementation property. And we can also reduce the number of the plaintexts needed for DC attacks by using related-keys. Attacks through the key scheduling algorithm are impractical in many cases, but they can be a factor imposing restrictions on the scope of applications of the block cipher. For example, if a block cipher with many related keys and weak keys is used in the construction of a hash function, it is helpful to find the collision pairs

of the hash function easily.

In this section we examine the existence of weak keys, semi-weak keys, complementation property and equivalent keys in the key scheduling algorithm of SEED. And we investigate the applicability of differential related-key attacks and related-key slide attacks to the key scheduling algorithm of SEED.

The key scheduling algorithm of SEED generates sixteen 64-bit subkeys by using the rotations of 64-bits and the G function used in the round function. The G function is composed of S-boxes of high non-linearity and bit-wise operations that give adequate diffusion. And it is so designed that each of the 128 key bits has an influence on the subkeys of all the rounds. By using round constant, S-boxes and modular addition/subtraction with the modulus 2^{32} , the key scheduling algorithm is designed to be secure against the various known attacks. And it can be implemented efficiently, since a subkey is generated with just the same module for the round function after each execution of rotation, addition and subtraction of $B \parallel A$ and $D \parallel C$. Since the methods to generate the round constant and the subkey for each round are the same, it can be implemented in smart cards with limited computing resources.

According to our analysis, none of weak keys, semi-weak keys, complementation property and equivalent keys exists in the key scheduling algorithm of SEED. The Related key slide attack is not applicable to SEED. It is also secure against differential related key attacks if the number of rounds exceeds five.

The input of G function to generate the subkeys is listed in <Table 1.3-1>. Here we represent four 32-bit parameters for the register A, B, C, D used in the key scheduling algorithm of SEED by 16 units of one byte such as $A=a_3a_2a_1a_0$, $B=b_3b_2b_1b_0$, $C=c_3c_2c_1c_0$, $D=d_3d_2d_1d_0$. These A, B, C, D are generated with a rule for each round. The key scheduling algorithm of the SEED generates the subkey of each round with newly generated A, B, C, D for the round. As in <Table 1.3-1>, we can see that, though A, B, C, D of the $(8+i)$ -th round is the same as B, A, D, C of the i -th round, the subkeys for these rounds are different because the combinations for $K_{i,0}$ and $K_{i,1}$ are different. For example, the same register parameters A, B, C, D are used in the rounds 1 and 9, but we have different input values of G for these rounds, say the form of $A \oplus C$, $B \oplus D$ for the first round and $B \oplus D$, $A \oplus C$ for the 9th round. Thus, the subkeys of $K_{1,0}$, $K_{1,1}$, $K_{9,0}$, $K_{9,1}$ come to have different values.

<Table 1.3-1> The input value of G function in each round

	$K_{i,0}$			$K_{i,1}$		
	A	$C \oplus KC_i$	KC_i	B	$D \oplus KC_i$	KC_i
1	$a_3a_2a_1a_0$	$+c_3c_2c_1c_0$	$\oplus KC_1$	$b_3b_2b_1b_0$	$-d_3d_2d_1d_0$	$\oplus KC_1$
2	$b_0a_3a_2a_1$	$+c_3c_2c_1c_0$	$\oplus KC_2$	$a_0b_3b_2b_1$	$-d_3d_2d_1d_0$	$\oplus KC_2$
3	$b_0a_3a_2a_1$	$+c_2c_1c_0d_3$	$\oplus KC_3$	$a_0b_3b_2b_1$	$-d_2d_1d_0c_3$	$\oplus KC_3$
4	$b_1b_0a_3a_2$	$+c_2c_1c_0d_3$	$\oplus KC_4$	$a_1a_0b_3b_2$	$-d_2d_1d_0c_3$	$\oplus KC_4$
5	$b_1b_0a_3a_2$	$+c_1c_0d_3d_2$	$\oplus KC_5$	$a_1a_0b_3b_2$	$-d_1d_0c_3c_2$	$\oplus KC_5$
6	$b_2b_1b_0a_3$	$+c_1c_0d_3d_2$	$\oplus KC_6$	$a_2a_1a_0b_3$	$-d_1d_0c_3c_2$	$\oplus KC_6$
7	$b_2b_1b_0a_3$	$+c_0d_3d_2d_1$	$\oplus KC_7$	$a_2a_1a_0b_3$	$-d_0c_3c_2c_1$	$\oplus KC_7$
8	$b_3b_2b_1b_0$	$+c_0d_3d_2d_1$	$\oplus KC_8$	$a_3a_2a_1a_0$	$-d_0c_3c_2c_1$	$\oplus KC_8$
9	$b_3b_2b_1b_0$	$+d_3d_2d_1d_0$	$\oplus KC_9$	$a_3a_2a_1a_0$	$-c_3c_2c_1c_0$	$\oplus KC_9$
10	$a_0b_3b_2b_1$	$+d_3d_2d_1d_0$	$\oplus KC_{10}$	$b_0a_3a_2a_1$	$-c_3c_2c_1c_0$	$\oplus KC_{10}$
11	$a_0b_3b_2a_1$	$+d_2d_1d_0c_3$	$\oplus KC_{11}$	$b_0a_3a_2a_1$	$-c_2c_1c_0d_3$	$\oplus KC_{11}$
12	$a_1a_0b_3b_2$	$+d_2d_1d_0c_3$	$\oplus KC_{12}$	$b_1b_0a_3a_2$	$-c_2c_1c_0d_3$	$\oplus KC_{12}$
13	$a_1a_0b_3b_2$	$+d_1d_0c_3d_2$	$\oplus KC_{13}$	$b_1b_0a_3a_2$	$-c_1c_0d_3d_2$	$\oplus KC_{13}$
14	$a_2a_1a_0b_3$	$+d_1d_0c_3d_2$	$\oplus KC_{14}$	$b_2b_1b_0a_3$	$-c_1c_0d_3d_2$	$\oplus KC_{14}$
15	$a_2a_1a_0b_3$	$+d_0c_3c_2c_1$	$\oplus KC_{15}$	$b_2b_1b_0a_3$	$-c_0d_3d_2d_1$	$\oplus KC_{15}$
16	$a_3a_2a_1a_0$	$+d_0c_3c_2c_1$	$\oplus KC_{16}$	$b_3b_2b_1b_0$	$-c_0d_3d_2d_1$	$\oplus KC_{16}$

Notations: We put the plaintext as P, the encryption key as K, the ciphertext as C, the encryption algorithm as E and the decryption algorithm as D. Thus we have $C = E(P, K)$ and $P = D(C, K)$.

※ Note : A_i, B_i, C_i, D_i are defined as the register variables A, B, C, D that are used to generate the subkey of the i-th round and they use the master key designated as K

1.3.1 The Influence of Encryption Key on The Round Key

The key scheduling algorithm of SEED is so designed that each byte of the encryption key has even influences on all the round keys. And all the bits of the encryption key are used in generating all the round keys. The rotation is determined so that every byte of the encryption key exerts an even influence. Since every byte of the encryption key is used twice in every byte of the modular operations with the modulus 2^{32} , the degree of correlation in each bit location of the operation is quite even.

1.3.2 Weak Key

Weak key means a key that can regenerate the plaintext by performing the encryption twice with the same key. ($P = E_K(E_K(P))$) That is, the key of the round i should be the same as that of the round $17-i$. If a weak key of SEED exists, the key of the 1st round should be the same as that of the 16th round. And the key of the 8th round should also be the same as that round 9. Since G is an injective function, any weak key satisfies the following conditions:

$$(a_3|a_2|a_1|a_0) \boxplus (c_3|c_2|c_1|c_0) \boxminus \text{KC1} = (a_3|a_2|a_1|a_0) \boxplus (d_0|c_3|c_2|c_1) \boxminus \text{KC16}$$

$$(a_3|a_2|a_1|a_0) \boxminus (d_0|c_3|c_2|c_1) \boxplus \text{KC8} = (a_3|a_2|a_1|a_0) \boxminus (c_3|c_2|c_1|c_0) \boxplus \text{KC9}$$

To summarize these, we get the following:

$$(c_3|c_2|c_1|c_0) \boxminus (d_0|c_3|c_2|c_1) = \text{KC1} \boxminus \text{KC16}$$

$$(c_3|c_2|c_1|c_0) \boxminus (d_0|c_3|c_2|c_1) = \text{KC9} \boxminus \text{KC8}$$

Since $\text{KC1} \boxminus \text{KC16} \neq \text{KC9} \boxminus \text{KC8}$, there are no weak keys.

1.3.3 Semi-weak Key

The semi-weak key means a key that can regenerate the plaintext by performing the encryption twice with different keys K_1 and K_2 i.e., ($P = E_{K_1}(E_{K_2}(P))$). That is, the i -th round key using K_1 should be the same as the $(17-i)$ -th round key using K_2 .

$$(a_3|a_2|a_1|a_0) \boxplus (c_3|c_2|c_1|c_0) \boxminus \text{KC1} = (a_3'|a_2'|a_1'|a_0') \boxplus (d_0'|c_3'|c_2'|c_1') \boxminus \text{KC16}$$

$$(a_3|a_2|a_1|a_0) \boxplus (d_0|c_3|c_2|c_1) \boxminus \text{KC16} = (a_3'|a_2'|a_1'|a_0') \boxplus (c_3'|c_2'|c_1'|c_0') \boxminus \text{KC1}$$

$$(a_3|a_2|a_1|a_0) \boxminus (d_0|c_3|c_2|c_1) \boxplus \text{KC8} = (a_3'|a_2'|a_1'|a_0') \boxminus (c_3'|c_2'|c_1'|c_0') \boxplus \text{KC9}$$

$$(a_3|a_2|a_1|a_0) \boxminus (c_3|c_2|c_1|c_0) \boxplus \text{KC9} = (a_3'|a_2'|a_1'|a_0') \boxminus (d_0'|c_3'|c_2'|c_1') \boxplus \text{KC8}$$

When we compare the 1st round key with the 16th round key and 8th round key and 9th round key, we have the following:

$$(c_3|c_2|c_1|c_0) \boxminus (d_0|c_3|c_2|c_1) \boxplus 2KC16 = (d_0|c_3|c_2|c_1) \boxminus (c_3|c_2|c_1|c_0) \boxplus 2KC1$$

$$(c_3|c_2|c_1|c_0) \boxminus (d_0|c_3|c_2|c_1) \boxplus 2KC8 = (d_0|c_3|c_2|c_1) \boxminus (c_3|c_2|c_1|c_0) \boxplus 2KC9$$

To make these short, we have the following:

$$2KC16 \boxminus 2KC8 = 2KC1 \boxminus 2KC9$$

Since $KC16 \boxminus KC8 \neq KC1 \boxminus KC9$, there are no semi-weak keys.

1.3.4 Complementation Property

The complementation property means that the ciphertext of a plaintext P for the key K has the same value as the bit-wise complement of the ciphertext of \bar{P} for \bar{K} , the bit-wise complement of K. That is, when the encryption algorithm E has complementation property, if $C = E_k(P)$, then $\bar{C} = E_{\bar{K}}(\bar{P})$. In order to have the complementation property in an encryption system having such a structure as SEED, the input of a plaintext and a subkey are XORed, the procedure of scheduling subkeys and the encryption function should be composed of the bit-wise operations such as bit-wise shifting, bit-wise permutation and XOR. On the other hand, the use of such operations as S-boxes and modular addition does not give the complementation property.

To examine the complementation property in SEED, we should analyze the form of the subkey input of the 1st round and the input of the function. Since 64 bit plaintext input is XORed with the 64 bit subkey of the 1st round and the properties of $a \oplus b = \bar{a} \oplus \bar{b}$ and $a \oplus \bar{b} = \bar{a} \oplus b = \overline{a \oplus b}$, the subkey of the 1st round should satisfy the complementation property. Since the subkey uses S-box that has non-linear output, the subkeys generated by K have nothing to do with those generated by \bar{K} . Therefore, they do not have the

complementation property. Even though the key scheduling satisfies complementation property, use of modular addition in the F-function makes it impossible to satisfy complementation property.

1.3.5 Equivalent Key

A pair of Equivalent keys is the two keys that are used when the two ciphertexts that are generated by two different keys are the same. That is, $C = E_K(P) = E_{K^*}(P)$ for the two different keys, K and K^* . To make K and K^* equivalent key, we should see to it that all the subkeys scheduled by each should be the same. If we assume that K and K^* are equivalent key, since $K_1=K^*_1$, $K_9=K^*_9$ and $(A, B, C, D) = (B_9, A_9, D_9, C_9)$, we can have the following expression:

$$\begin{aligned} A \oplus C &= A^* \oplus C^*, & B - D &= B^* \ominus D^* \\ B \oplus D &= B^* \oplus D^*, & A - C &= A^* \ominus C^* \end{aligned}$$

Put the above expressions as $A' = (A \ominus A^*)$, $B' = (B \ominus B^*)$, $C' = (C \ominus C^*)$, $D' = (D \ominus D^*)$, we have the following:

$$\begin{aligned} A' \oplus C' &= 0, & A' \ominus C' &= 0. \\ B' \oplus D' &= 0, & B' \ominus D' &= 0. \end{aligned}$$

Since the solution of the above is $A'=B'=C'=D'=0$ (that is, $K=K^*$), there are no different equivalent key pairs. Since we are using modular operations with the modulus 2^{32} , there can be the solution of such a case as $A'=C'=2^{31}$. But, when we examine the 2nd and 10th rounds, as in the case of A' , all the bits of A_2' , excepting MSB of A_2' , are 0. But, since $A_2' = (b_0' a_3' a_2' a_1')$ and $A' = (a_3' a_2' a_1' a_0')$ (that is, $a_3' = 0$), $A' = 2^{31}$ cannot be derived. Consequently, it is impossible to generate the same subkeys for all the rounds with two different keys. Thus we can say that there are no equivalent keys.

1.3.6 Related Key Attack

The related-key cryptanalysis is a method to attack a cipher with related-keys acquired by key scheduling. This attack is to find keys using the relation between two subkeys when the key scheduling algorithm is simple or regular. This attack is feasible when a definite relation, $K' = f$

(K), between the two keys K and K' is found, and it can be applied regardless of the number of the rounds. The related-key cryptanalysis can be used with other attacks such as Differential Cryptanalysis. In its types of attacks, there are the related-key slide attack (the rotational subkey related-key attack) and the differential related key attack.

1) Related-key slide attack

The related-key slide attack on the structure of SEED is feasible when the encryption for a plaintext (P) executed from the 1st round to the nth round is the same as that for another plaintext (P*) executed from the sth round to the (s+n)th round. Biham et. al. have applied this attack to LOKI89, LOKI91, and Lucifer. And they have shown that it can be applied to DES that is modified so that it can have the same shifting in key scheduling of DES. The related-key slide attack is feasible only when simple operations such as rotation, bit-wise permutation, XOR are used in key scheduling.

In the case of SEED, the subkey of the round i+2 for K should be the same as that of round i for K* to be a rotational related-key since the subkey is generated by D, C, B, A updated by rotating B || A, D || C once for every other round with the same rule. Indeed, SEED generates a subkey for each round with a regular rotation, putting A, B, C, D as the input of the G function. But there is almost no possibility of having related-key since all the round constants are different and the combinations of A, B, C, and D are all different. If related-keys for the related-key slide attack exists in the key scheduling algorithm of the SEED, it must be $K_i^* = K_{i+2}$ (i=1, ..., 14) for the K and K*. Then, since $K_{1,0}^* = K_{3,0}$ and $K_{9,1}^* = K_{11,1}$, we have the following.

$$A^* \oplus B^* \oplus C^* \oplus D^* = A_3 \oplus B_3 \oplus C_3 \oplus D_3 \quad (1)$$

Also combining another two equations, $K_{1,1}^* = K_{3,1}$, $K_{9,0}^* = K_{11,0}$ we obtain the following:

$$A_9^* \oplus B_9^* \oplus C_9^* \oplus D_9^* = A_{11} \oplus B_{11} \oplus C_{11} \oplus D_{11} \quad (2)$$

On the other hand, since (A, B, C, D)=(B₉, A₉, D₉, C₉) and (A₃, B₃, C₃, D₃)=(B₁₁, A₁₁, D₁₁, C₁₁), we have following:

$$A^* \oplus B^* \oplus C^* \oplus D^* = A_3 \oplus B_3 \oplus C_3 \oplus D_3. \quad (2')$$

Then, by (1) and (2') we derive the following:

$$A \oplus B^* = A_3 \oplus B_3, C^* \oplus D^* = C_3 \oplus D_3. \quad (3)$$

By (3), we can assume the following without losing generality:

$$A^* = A_3 \oplus s, B^* = B_3 \oplus s, C^* = C_3 \oplus t, D^* = D_3 \oplus t.$$

Since $K_{1,0}^* = K_{3,0}$, $K_{9,0}^* = K_{11,0}$, s and t must satisfy the following:

$$s \oplus t = KC_1 \oplus KC_3, \quad s \oplus t = KC_{11} \oplus KC_9$$

That is,

$$s = (KC_1 \oplus KC_3 \oplus KC_9 \oplus KC_{11})/2 \text{ or } (KC_1 \oplus KC_3 \oplus KC_9 \oplus KC_{11})/2 \oplus 2^{31}$$

$$t = (KC_1 \oplus KC_3 \oplus KC_9 \oplus KC_{11})/2 \text{ or } (KC_1 \oplus KC_3 \oplus KC_9 \oplus KC_{11})/2 \oplus 2^{31}$$

Since both $(KC_1 \oplus KC_3 \oplus KC_9 \oplus KC_{11})$ and $(KC_1 \oplus KC_3 \oplus KC_9 \oplus KC_{11})$ are odd numbers, there are no s, t that satisfy the above equations. This means that there are no related-keys in SEED. But if all the round constants are the same, we can find related-keys. For example, in the case of $s = t = 0$, $(A, B, C, D) = (0xiiii, 0xjjjj, 0xkkkk, 0xhhhh)$, (i, j, h, k : byte hexacode) are the related-keys. In fact, the related-key slide attack is not applicable to SEED because different round constants are used in the key scheduling of SEED.

2) Differential related-key attack

The differential related-key attack is a method to increase the probability of success of the Differential Cryptanalysis attack, or to attack the block cipher strong against the related-key slide attack through constructing differential characteristic of the difference between the given master keys, and the difference of the plaintext pairs. This attack is applicable when each bit of a master key does not have an even influence on all the subkeys and have an influence on a particular subkey like SAFER-64K, or in the case of block ciphers like IDEA that has a linear factor. In the case of SEED, every bit of the master key has an even influence on the subkeys of all the rounds and uses modular addition/subtraction with the modulus 2^{32} . So it can be said that all the bit combinations of the master key are different for all the subkeys. And an efficient differential related-key attack is difficult to carry out since bit-wise permutation with high diffusion is used.

First, we examine the key scheduling and the input/output differences for XOR operation of the G function and S-boxes to estimate the complexity of the differential related-key attack on SEED.

The distribution of XOR differences for the two most significant consecutive bits of the S-boxes is as follows:

S1:	0x40	→	0x40, 0x80	$p=2^{-7}$
			0xc0	$p=0$
	0x80	→	0x80	$p=2^{-7}$
			0x40, 0xc0	$p=0$
	0xc0	→	0x40, 0x80, 0xc0	$p=2^{-7}$
S2:	0x40	→	0x80	$p=2^{-7}$
			0x40, 0xc0	$p=0$
	0x80	→	0x80, 0xc0	$p=2^{-7}$
			0x40	$p=0$
	0xc0	→	0x40, 0x80, 0xc0	$p=0$

If S_2 has 0xc0 as the input difference, the output difference can be neither of the form 0x40, 0x80, or 0xc0. For simplicity, we only consider the differences of MSBs to analyze the differential characteristics. And for both of two S-boxes, the probabilities for the input difference 0x80 with the output difference 0x80 are the same as 2^{-7} .

The distribution of the XOR differences for the G function is as following. To reduce the number of the active S-boxes, we set the output difference that minimize the diffusion effect of bit-wise permutation.

0x80000000	→	0x80808000,	$p=2^{-7}$
0x00800000	→	0x80800080,	$p=2^{-7}$
0x00008000	→	0x80008080,	$p=2^{-7}$
0x00000080	→	0x00808080,	$p=2^{-7}$
0x80800000	→	0x00008080,	$p=2^{-14}$
0x80008000	→	0x00800080,	$p=2^{-14}$
0x80000080	→	0x80000080,	$p=2^{-14}$
0x00808000	→	0x00808000,	$p=2^{-14}$
0x00800080	→	0x80008000,	$p=2^{-14}$

0x00008080	→ 0x80800000,	$p=2^{-14}$
0x80808000	→ 0x80000000,	$p=2^{-21}$
0x80800080	→ 0x00800000,	$p=2^{-21}$
0x80008080	→ 0x00008000,	$p=2^{-21}$
0x00808080	→ 0x00000080,	$p=2^{-21}$
0x80808080	→ 0x80808080,	$p=2^{-28}$

The probabilities of XOR differences for the modular addition are as following:

(0x00000000, 0x80000000)	→ 0x80000000	: 1
(0x00000000, 0x00800000)	→ 0x00800000	: 1/2
(0x00000000, 0x00008000)	→ 0x00008000	: 1/2
(0x00000000, 0x00000080)	→ 0x00000080	: 1/2
(0x00000000, 0x80800000)	→ 0x80800000	: 1/2
(0x00000000, 0x80008000)	→ 0x80008000	: 1/2
(0x80000000, 0x80000000)	→ 0x00000000	: 1
(0x80000000, 0x00800000)	→ 0x80800000	: 1/2
(0x80000000, 0x00008000)	→ 0x80008000	: 1/2
(0x80000000, 0x00000080)	→ 0x80000080	: 1/2
....		
(0x80008000, 0x80800000)	→ 0x00808000	: 1/4
(0x00808080, 0x80808080)	→ 0x80000000	: 1/8
(0x80808080, 0x80808080)	→ 0x00000000	: 1/8

We can calculate the characteristic of the key scheduling and the F function by utilizing the above listed XOR differences for G and the XOR differences for the addition. First, we define K^* as a key that toggles only the 8 consecutive least significant bits and 64 bits of a randomly chosen key K and denote K' as the XOR difference of K and K^* . That is, K^* means a value which can be attained by toggling the MSBs of a_1 and c_0 of K (see <Table 1.3-1>).

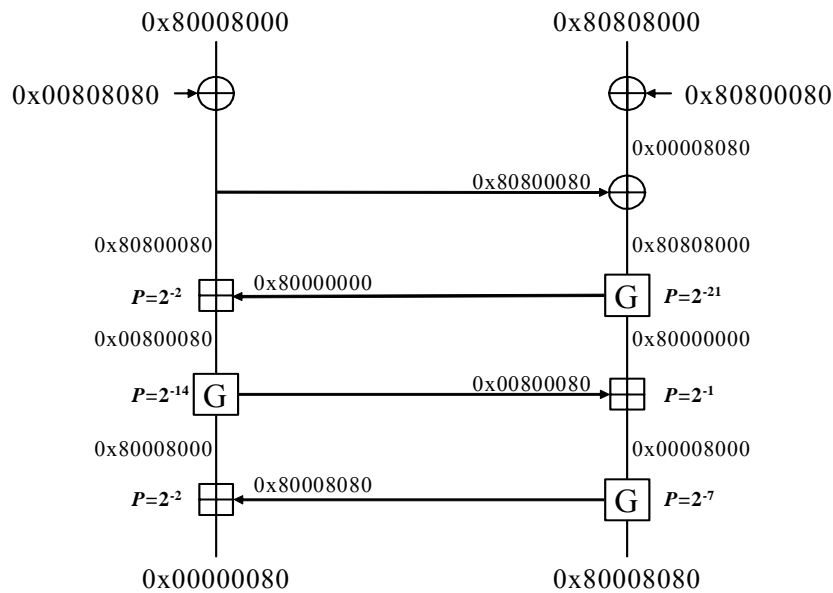
The input/output differences of G in the process of finding each subkey for the given K and K^* , and their probabilities are given in <Table 1.3-2>.

<Table 1.3-2> The input/output differences for each subkey with $K'=(2^8 + 2^{64})$

I	Difference of $K_{i,0}$			Difference of $K_{i,1}$		
	Input difference of G (p_1)	Output difference of G (p_2)	$p=p_1p_2$	Input difference of G (p_1)	Output difference of G (p_1)	$p=p_1p_2$
1	0 0 80 80 (1/4)	80 80 00 (2^{-14})	2^{-16}	0(1)	0(1)	1
2	0(1/2)	0(1)	1/2	0(1)	0(1)	1
3	0 0 80 80 (1/4)	80 80 00 (2^{-14})	2^{-16}	0(1)	0(1)	1
4	0 0 80 0 (1/2)	80 0 80 80 (2^{-7})	2^{-8}	80 0 0 0 (1)	80 80 80 0 (2^{-7})	2^{-7}
5	0 80 0 0 (1/2)	80 80 0 80 (2^{-7})	2^{-8}	80 0 0 0 (1)	80 80 80 0 (2^{-7})	2^{-7}
6	0 80 0 0 (1/2)	80 80 0 80 (2^{-7})	2^{-8}	0 80 0 0 (1/2)	80 80 0 80 (2^{-7})	2^{-8}
7	80 0 0 0 (1)	80 80 80 0 (2^{-7})	2^{-7}	0 80 0 0 (1/2)	80 80 0 80 (2^{-7})	2^{-8}
8	80 0 0 0 (1)	80 80 80 0 (2^{-7})	2^{-7}	0 80 0 0 (1/2)	80 80 0 80 (2^{-7})	2^{-8}
9	0(1)	0(1)	1	0 0 80 80 (1/4)	80 80 0 0 (2^{-14})	2^{-16}
10	0(1)	0(1)	1	0(1/2)	0(1)	1/2

Now, we examine the process of finding $K'_{1,1}$ by means of calculating the difference of a subkey in the key scheduling. Since the MSBs of a_1 and c_0 (see <Table 1.3-1>) are changed, when we assume the input difference of G in $K'_{1,0}$ does not have carry at the lower bit 8 and bit 16 (probability $p_1=1/4$), the difference of $A \oplus C \oplus KC_1$ is $0x00008080$. By function G, if the input difference is $0x00008080$ then the output difference is $0x80800000$ with the probability of $p_2=2^{-14}$ (by the two S-boxes and the property of bit-wise permutation). On the other hand, since the input difference in $K'_{1,1}$ is 0, the output difference is also 0. Therefore, we have $K'_{1,1}=0$ (probability=1).

And we can show as following the process of finding the differential of F for the input difference of the plaintext= $0x80008000 \parallel 0x80808000$ and for the input difference of the subkey= $0x00808080 \parallel 0x80800080$, when both the plaintext used as the input of the function F and the subkey are changed. Then the probability of differential is 2^{-47} .



Now, we are going to calculate the differential characteristic of K^* which is the difference of the above two keys K and K' . To take the advantage of the characteristic of the Feistel structure, we set the difference of the plaintexts as (K'_2, K'_1) . Then, in 1st and 2nd round, all the differences of the G function are 0 since the input of function F offsets the key input. So, the output difference of F function is 0, too. And we can see the same effect in 3rd round as in 1st and 2nd round since $K'_3 = K'_1$. The probability of the characteristic that is composed of 1st, 2nd and 3rd round is decided only by the probability of the occurrence of K'_1 , K'_2 and K'_3 for a given key difference. When the difference between the two plaintexts is (K'_2, K'_1) and the difference of the two keys is the K' defined above, we can represent the characteristic of the first 5 rounds, according to their order, as following:

<Table 1.3-3> The difference of the plaintext : (K'_2, K'_1)

	Input difference of F	Output difference of F	Probability
1 round	(K'_1, K'_1)	0	2^{-16}
2 round	(K'_2, K'_2)	0	1/2
3 round	(K'_1, K'_3)	0	2^{-16}
4 round	(K'_2, K_4)	0x8080800080	$2^{-15-49} = 2^{-64}$
5 round	$(K'_1 \oplus 0x8080800080, K'_5)$	0x80008080 00008000	$2^{-15-34} = 2^{-49}$

Since the probability of the characteristic of 4 round is 2^{-97} , and that of 5 round is 2^{-146} , the related-key differential attack is almost impossible, if the number of the rounds exceeds 5. The above characteristic is $K'_3=K'_1$. So it has a relatively high probability in 3 round. But as we can see in 4 and 5 round, since the diffusion effect of the function G is great, it is difficult to find a characteristic that has adequately high probability only with the change of a particular bit in the plaintext and the key. For K we put K^* as the key obtained by toggling the MSBs of b_0 and c_3 of K, or toggling the MSBs of a_3 and c_2 of K to satisfy $K'_3=K'_1$. But we cannot find any characteristic that has much higher probability than the above characteristic.

On the other hand, we examine the probability of characteristic by making K and K' have one bit of difference so that they can have a reduced diffusion from the bit-wise permutation (to minimize the carry effect, we considered just the MSB of each byte) though the probability of the characteristic of 3 round is low in the case that K'_3 and K'_1 are not equal.

The probability of the characteristic when we toggled the MSB of a_1 of K : (2^{-8} , 2^{-8} , 2^{-72} , 2^{-42})

The probability of the characteristic when we toggled the MSB of a_3 of K : (2^{-7} , 2^{-8} , 2^{-70} , 2^{-57})

The probability of the characteristic when we toggled the MSB of c_3 of K : (2^{-7} , 2^{-7} , 2^{-55} , 2^{-70}).

But, in this case also, the probability gets remarkably low from 3 round that has other than 0 as the input difference of the G function. Therefore, the characteristic is more efficient when we set K' as $K'_3=K'_1$.

So far we have shown that the differential related-key attack is almost impossible when SEED has more than 5 rounds. When we consider both the effect of carry and the probability of another input difference/output difference including 0x80, and the diffusion of the bit-wise permutation, we can find a characteristic that has relatively high probability. But even if we find it, a comprehensive analysis considering every case is quite difficult to perform, and because of the diffusion characteristic of this kind of bit-wise permutation, it is also difficult to find a characteristic that has much higher probability than the above characteristic.

2. STATISTICAL TEST

To analyze the randomness characteristics of SEED, we gathered the data with the methods shown in 4.1. We used four kinds of methods to gather the data and the data were composed of 384 sequences (each sequence is 2068480/12800 bits) for each method. We tested this data with the following 13 random test methods, and the total number of tests was $4 * 384 * (13 + 8) = 32256$.

- 1) frequency test
- 2) frequency-m test with $m = 4, 6$
- 3) binary derivative test
- 4) 2nd binary derivative test
- 5) change point test
- 6) poker test with $m = 4, 8, 12$
- 7) runs test
- 8) runs distribution test
- 9) autocorrelation test with $d = 2, 4, 8, 12$
- 10) serial test
- 11) sequence complexity test
- 12) universal test with $L = 4, 6, 8$
- 13) linear complexity test

2.1. DATA ACQUISITION METHODS

To examine the randomness characteristics, we acquired sample sequences using the following four types of methods. We decided the size of each sequence as 2068480 bits, based on Maurer's universal test ($L=8$). But in the cases of the change point test and the sequence complexity test, we decided the size of the sample sequences to be 12800 bits because of its long testing time. We decided the number of the sample sequences in each acquisition method to be 384 units, based on the key/plaintext avalanche method.

2.1.1 Output Only

In order to examine the randomness of the ciphertext (based on random plaintext and random key), we randomly generated 384 sequences of 128-bit key and encrypted 2068480/12800 bit

random plaintext for each sequence. The resulting ciphertexts were our data.

2.1.2 Plaintext/Ciphertext Correlation

In order to examine the correlation of plaintext/ciphertext pairs, 384 sequences were generated. Given a random 128 bit key and a random 2068480/12800 bit plaintext, each sequence was constructed as the result of applying the XOR operator on the plaintext block and its corresponding ciphertext block computed in ECB mode.

2.1.3 Key Avalanche

In order to examine the influence of the change of one bit in a key for a fixed plaintext on ciphertext, 384 sequences were constructed. Given a random 128 bit key and a fixed plaintext, the sample data is the XOR of the “ciphertext formed using the fixed plaintext and a random 128 bit key” and the “ciphertext formed using the fixed plaintext and the perturbed random 128 bit key with the i^{th} bit changed, for $1 \leq i \leq 128$ ”.

2.1.4 Plaintext Avalanche

In order to examine the influence of the change of one bit in a plaintext for a fixed key on ciphertext, 384 sequences were constructed. Given a fixed 128 bit key and a random 2068480/12800 bit plaintext, the sample data is the XOR of the “ciphertext formed using the fixed key and a random plaintext” and the “ciphertext formed using the fixed key and the perturbed random plaintext with the i^{th} bit changed, for $1 \leq i \leq 128$ ”.

2.2 ANALYSIS OF THE TEST RESULTS

For each experiment, we tested 384 sample sequences based on the significant level of 1 % ($\alpha = 0.01$). This implies, that ideally, no more than a 3.84 sequence should be rejected for each sample of 384 sequences evaluated by a statistical testing.

However, in all likelihood, any given data set will deviate from this ideal case. A more realistic interpretation is to use a *confidence interval* (CI) for the “proportion of binary sequences” that should pass at the 0.01 level. When we put the total test number as s and put the failed number as n , then n/s will have a normal distribution of $N(\alpha, \frac{\alpha(1-\alpha)}{s})$. Thus, when

$\alpha = 0.01$, $s = 384$ we can think of $s(\alpha + 2.58\sqrt{\frac{\alpha(1-\alpha)}{s}}) = 8.87$ as our boundary value.

Therefore, if more than 9 units of the 384 units of sample sequences do not pass this test, the sample sequence is thought not to pass the test.

The test results on SEED are shown in Table 2.1. The value of each item means the number of the sample sequences among the 384 sample sequences that failed the test. The test result shows that the number of sample sequences that failed the test does not exceed 9 units in every test. Therefore, it can be said that, in statistical test, the sample sequence scheduled by SEED is not so different from the random sequence.

<Table 2.1> The result of the statistical test on SEED

	output only	plaintext/cipherte xt. correlation	key avalanche	plaintext avalanche
Frequency	0	6	3	3
Frequency - m test	m = 4	4	1	3
	m = 6	3	5	6
Binary derivative	1	4	3	3
2 nd binary derivative	4	4	3	2
Change point	4	3	4	2
Poker	m = 4	5	4	5
	m = 8	3	4	4
	m = 12	2	6	2
Runs	1	2	4	3
Runs distribution	3	4	8	8
Autocorrela- tion	d = 2	0	0	0
	d = 4	0	0	0
	d = 8	3	5	4
	d = 12	0	2	1
Serial	1	2	2	4
Sequence complexity	0	0	0	0
Universal	L = 4	2	8	8
	L = 6	7	4	4
	L = 8	2	3	4
Linear complexity	5	5	5	7

3. PERFORMANCE AND IMPLEMENTATION COST

This clause covers implementation issues in aspects of software and hardware: C and Java implementation, 8-bit microprocessor, and integration performance of gates.

3.1 S/W IMPLEMENTATION

3.1.1 Implementation in C

We are going to discuss the result we achieve after implementing SEED in MS Visual C++ 6.0 Enterprise with Pentium III @800MHz. It is convenient to denote the speed as the number of cycles in the measurement of the data processing speed since we can easily get the speed which is proportional to clock speed under the same architecture. Under the above mentioned environment, the number of cycles required for key expansion is 354 cycles for encryption and decryption respectively. And the processing speed of SEED is approximately 131Mbps.

<Table 3.1-1> Number of cycles required for key expansion (128-bit key, 128-bit block)

	AES (Rijndael *)	SEED
Encryption	305	354
Decryption	1389	354
Total	1694	708

<Table 3.1-2> Cipher Performance in C (128-bit key, 128-bit block)

	AES (Rijndael *)	SEED
#Cycles/Block	363	783
Speed (Mbit/sec)	282	131

*Joan Daemen, Vincent Rijmen, "AES proposal: Rijndael", 1999

3.1.2 Implementation in Java

In addition, implementing SEED in Java VM of Sun Microsystems Inc., version 1.2.2(OS is Windows NT/x86, version 4.0 with Pentium III @866MHz) is executed in CBC mode. We are able to obtain 3806.9 kbyte/sec and 4030.3 kbyte/sec, at 16 bytes and 256 bytes length in block size, respectively.

<Table3.1-3> Cipher Performance in Java (kbyte/sec)

Cipher \ Size of block	16 bytes	64 bytes	128 bytes	256 bytes
Rijndael	17583.1	21352.7	22092.1	22476.9
SEED	3806.9	3989.1	4004.7	4030.3

3.2 HARDWARE IMPLEMENTATION

3.2.1 Performance on the 8-bit processor

SEED is implemented on the 8-bit processor chip as follows:

- Model : ST19RF08 (ST Microelectronics),
- MCU : 8 bit ST processor @ 3.5MHz,
- RAM : 960 bytes, and
- ROM : 32 kbytes .

In this implementation, we modify size of s-box table specified in SEED algorithm from 2 kbytes to 576 bytes for memory efficiency. And we have coded SEED algorithm in 1,773 bytes including the s-box table size of 576 bytes.

SEED is shown to require 10,560 clocks and 13,168 clocks for round key generation and data round function of 16 rounds, respectively, as below. In this table, “encrypt (or decrypt)” is the resources required to encrypt (or decrypt) a single 128-bit block with a 128-bit key. “schedule” is the resources required to schedule a 128-bit key.

<Table 3.2-1> Memory requirements and execution time of the algorithms.

Cipher	RAM(bytes)			ROM(bytes)		Time (clocks)	
	scheduled key	encrypt	schedule	encrypt	encrypt+ schedule	encrypt	schedule
SEED	16	32	26	1,240	1,773	13,168	10,560
Rijndael-e *	16	34	0	876	879	9,464	0
Rijndael-d*	16	37	1	976	1,049	13,538	2,278

* The encryption(or decryption) process in Rijndael-e(or -d) is listed as Rijndael-e(or -d).

Geoffrey Keating, “Performance Analysis of Candidates on the 6805 CPU core”, 15 April, 1999

3.2.2 Hardware Complexity

In this section, we estimate the hardware complexity of SEED targeting the cell-based ASIC implementation, and we compare it with the AES (Advanced Encryption Standard) algorithm. There are three main factors to consider in the hardware complexity analysis: area, performance, and power consumption. In this document, we only describe area and performance measures due to the difficulty in estimating power consumption in the architectural level. Although these estimates may differ from real implementations, they give good abstraction levels of comparative analysis for hardware complexity. Our estimate is based on Samsung 0.5 um CMOS standard cell library.

<Table 3.2-2> Comparison of Hardware Complexity for SEED and AES

		SEED	AES
Algorithm specification	block length	128-bit	128-bit
	key length	128-bit	128, 129, 256-bit
	# of rounds	16	10 (for 128-bit mode)
Area		3,893 gates 8 k-bit ROM	9,917 gates 32 k-bit ROM
Implemental hardware complexity	critical path delay (clock frequency)	10.9 ns (90 MHz)	9.9 ns (100 MHz)
	# of clocks	48	11
	total time for a 128-bit encryption/decryption	533 ns	110 ns
	Performance	240 bps	1.16 Gbps
Notes	(*) The module implements only 1/3 of each iteration round (one G-function and one 32-bit Adder), hence needs 3 clock cycles to finish each iteration.	(*) The module implements a whole iteration round., hence needs only 10 clock cycles to perform each iteration. (*) It uses a modified decryption flow specified in the standard document to reduce the area. (*) The ByteSub & InvByteSub block is implemented using ROM.	