A proof of P≠NP (New symmetric encryption algorithm against any linear attacks and differential attacks)

Gao Ming*

March 5, 2022

Abstract

P vs NP problem is the most important unresolved problem in the field of computational complexity. Its impact has penetrated into all aspects of algorithm design, especially in the field of cryptography. The security of cryptographic algorithms based on short keys depends on whether P is equal to NP. In fact, Shannon[1] strictly proved that the one-time-pad system meets unconditional security, but because the one-time-pad system requires the length of key to be at least the length of plaintext, how to transfer the key is a troublesome problem that restricts the use of the one-time-pad system in practice. Cryptography algorithms used in practice are all based on short key, and the security of the short key mechanism is ultimately based on one-way assumption. In fact, the existence of one-way function can directly lead to the important conclusion P≠ NP.

In this paper, we originally constructed a short-key block cipher algorithm. The core feature of this algorithm is that for any block, when a plaintext-ciphertext pair is known, any key in the key space is valid, that is, for each block, the plaintext-ciphertext pair and the key are independence, and the independence between blocks is also easy to construct. This feature is completely different from all existing short-key cipher algorithms.

Based on the above feature, we construct a problem and theoretically prove that the problem satisfies the properties of one-way functions, thereby solving the problem of the existence of one-way functions, that is, directly proving that P≠NP.

*20070602094@alu.cqu.edu.cn

1 Introduction

Cryptography is one of the most important applications in the field of communication and computer science. In recent years, with the application of commerce, enterprises, banks and other departments, cryptography has been developed rapidly. Especially after Shannon put forward the mathematical analysis of security in "Communication theory of secrecy systems"[1], various design tools for cipher algorithms and corresponding attack tools have been developed one after another. Among them, the most common attack methods include linear attacks and differential attacks.

Linear attack was first proposed by M. Matsui[2], this is an attack method that is currently applicable to almost all block encryption algorithms. Kaliski BS[3] proposed a multi-linear attack based on the linear attack, but the multi-linear attack has many limitations. And the Biryukov A[4] and Chao, J.Y[5] and others further improved the framework of multi-linear attacks, thus making linear attacks a larger application.

The differential attack method was first proposed by Eli Biham[6]. BIHAM E[7] extended it to a more powerful attack method. TSUNOO[8] further constructed multiple attack methods. These attack methods have extremely high skill in the attack process, which is worthy in-depth study.

In this paper, we first designed a new encoding algorithm, which we named Eagle. Based on the Eagle encoding algorithm, we designed a new block symmetric cipher algorithm, For any block of plaintext-ciphertext pairs, any key in the key space is valid. That is to say, there is no specific mathematical relationship between the plaintext, key, and ciphertext in each block, showing a completely randomly property. It can also be understood that for any plaintext, encrypted with the same key every time, the ciphertext obtained is not uniquely determined, but completely randomly in the possible ciphertext space. And this feature makes the cipher algorithm can resistant all forms of linear attacks and differential attacks.

At the end of this paper, we further construct a new cipher system. Under this cipher system, if any plaintext-ciphertext pair is known, if an attacker wants to guess the possible correct key, he cannot do it by any method other than exhaustive search. We have proved theoretically that this kind of problem satisfies the properties of one-way functions, that is, theoretically prove that one-way functions exist, so that P $\neq$ NP.

2 Introduction to Eagle encoding algorithm

We first introduce two common bit operations. XOR denoted as $\oplus$. Do left cycle shift of D by n bits which can be denoted as D^{+n}, for example $(10011010)^{+2} = (01101010)$.

We select two L-bits parameters w_0 and w_1, have odd number of different bits. For example $w_0 = 10010011$ and $w_1 = 11000111$ have 3 bits (3 is odd) different.

$$w_0 = 10010011$$
$$w_1 = 11000111$$

We set the initial state of L-bit as s_0, we choose one parameter $w \in \{w_0, w_1\}$, without loss of generality, assume that we choose $w = w_0$, then we define the following calculation

$$s_1 = w \oplus (s_0 \oplus s_0^{+1}) = w_0 \oplus (s_0 \oplus s_0^{+1}) \tag{2.1}$$

From (2.1), we can easily know

$$s_0 \oplus s_0^{+1} = s_1 \oplus w \tag{2.2}$$

If we only know s_1, we don 't know whether $w = w_0$ or $w = w_1$ we used in (2.1), we can confirm it through a simple trial-and-error. For example, we guess the parameter $w = w_1$ was used in (2.1), we need to find a certain number s_x to satisfy

$$s_x \oplus s_x^{+1} = s_1 \oplus w_1 \tag{2.3}$$

In fact, since w_0 and w_1 have odd number of different bits, such s_x does not exist. See Theorem 1 for details.

[**Theorem 1**] We arbitrarily choose two L-bit parameters w_0 and w_1 which have odd number of different bits, for arbitrary s_0, we set $s_1 = w_0 \oplus (s_0 \oplus s_0^{+1})$, then there doesn't exists s_x satisfy $s_x \oplus s_x^{+1} = s_1 \oplus w_1$.

Proof. Firstly, by definition we have

$$s_1 \oplus w_1 = w_0 \oplus (s_0 \oplus s_0^{+1}) \oplus w_1 = (s_0 \oplus s_0^{+1}) \oplus (w_0 \oplus w_1) \tag{2.4}$$

Where w_0 and w_1 have odd number of different bits, so there are odd number of 1 in the bit string of $w_0 \oplus w_1$.

Proof by contradiction, we suppose that there exists s_x satisfy $s_x \oplus s_x^{+1} = s_1 \oplus w_1$, then by (2.4), we have

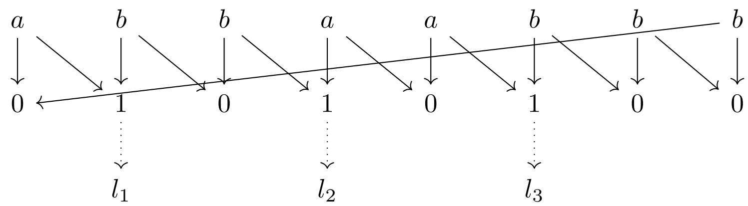
$$s_x \oplus s_x^{+1} = (s_0 \oplus s_0^{+1}) \oplus (w_0 \oplus w_1) \tag{2.5}$$

By simple calculation we have

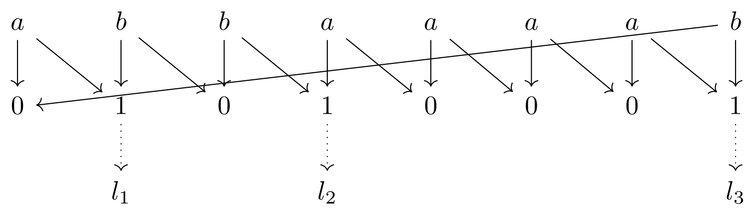
$$(s_0 \oplus s_x) \oplus (s_0 \oplus s_x)^{+1} = w_0 \oplus w_1 \tag{2.6}$$

We set $s_y = s_0 \oplus s_x$, then there are odd number of 1 in the bit string of $s_y \oplus s_y^{+1}$, without of generality, we suppose that the bits with 1 are $l_1, l_2, ..., l_u$ (u is odd). Compare to the first bit of s_y, the $l_1 + 1$ bit of s_y is different from the first bit of s_y , the $l_2 + 1$ bit of s_y is the same with the first bit of s_y, the $l_3 + 1$ bit of s_y is different from the first bit of s_y, the $l_4 + 1$ bit of s_y is the same with the first bit of s_y, and so on, since u is odd, $u-1$ is even, the $l_{u-1}+1$ bit of s_y is the same with the first bit of s_y.

If $l_u < L$, the $l_u + 1$ bit of s_y is different from the first bit of s_y, so the last bit of s_y is different from the first bit of s_y. On the other hand, the last bit of $s_y \oplus s_y^{+1}$ is 0, so the last bit of s_y is the same with the first bit of s_y. This is contradictory.



If $l_u = L$, the last bit of s_y is different from the the first bit of s_y. On the other hand, the last bit of s_y is the same with l_{u-1} bit of s_y which is the same with the first bit of s_y. This is contradictory.



So there doesn't exists s_x satisfy $s_x \oplus s_x^{+1} = s_1 \oplus w_1$. □

Let's go back to the discussion just now, after a trial-and-error, we can accurately confirm which one ($w = w_0$ or $w = w_1$) we just used in (2.1).

Now we suppose that there is a binary sequence $m = b_1 b_2 ... b_L$ with length L. Start with s_0, read each bit of M from left to right sequentially, when the bit $b_i (1 \leqslant i \leqslant L)$ is 0, we set $s_i = w_0 \oplus (s_{i-1} \oplus s_{i-1}^{+1})$, when the bit b_i is 1, we set $s_i = w_1 \oplus (s_{i-1} \oplus s_{i-1}^{+1})$.

According to the above calculation, for every s_i, we can find the only $w_x = w_0$ or $w_x = w_1$ such that there exists s_{i-1} satisfy $s_{i-1} \oplus s_{i-1}^{+1} = s_i \oplus w_x$. According to the properties of XOR and cyclic shift, we can easily see that there are only two s_{i-1} that satisfy $s_{i-1} \oplus s_{i-1}^{+1} = s_i \oplus w_x$, and the two s_{i-1} with each bit different. As long as we know any one bit of s_{i-1}, s_{i-1} can be uniquely determined. So we only need to save one bit of s_i, finally by s_L, we can completely restore the original state s_0 and the binary sequence m.

Based on the above discussion, we can construct a complete Eagle encoding and decoding algorithm. The entire algorithm consists of three processes: generating parameters, encoding, and decoding.

[Parameter generation]

Firstly we choose two L-bit parameters w_0 and w_1 which have odd number of different bits, then we choose L-bit initial state s_0.

[Encoding]

For input data m, we record $m[i] (1 \leqslant i \leqslant L)$ as the i-th bit of m, $m[i] \in$

$\{0,1\}$, L is the length of m, the encoding process is as follows.

[$E1$] Execute E2 to E4 with i from 1 to L.

[$E2$] If $m[i] = 0$, set $s_i = w_0 \oplus (s_{i-1} \oplus s_{i-1}^{+1})$.

[$E3$] If $m[i] = 1$, set $s_i = w_1 \oplus (s_{i-1} \oplus s_{i-1}^{+1})$.

[$E4$] Set the last bit of s_{i-1} as the i-th bit of c , $c[i] = s_{i-1}[L]$.

[$E5$] Use (c, s_L) as the output.

[**Decoding**]

The output (c, s_L) of the above encoding process is used as the input of the decoding process, the decoding process is as follows.

[$D1$] Execute D2 to D4 with i from L to 1.

[$D2$] Do trial-and-error testing with $s_i \oplus w_0$ or $s_i \oplus w_1$, find the unique $w_x (x = 0\, or\, x = 1)$ satisfy $s_x \oplus s_x^{+1} = s_i \oplus w_x$.

[$D3$] After D2, use x as the i-th bit of m, $m[i] = x$.

[$D4$] For the two possible s_x satisfy $s_x \oplus s_x^{+1} = s_i \oplus w_x$ in D2, we set the one which the last bit is equal to $c[i]$ as s_{i-1}.

[$D5$] Use m as the output.

Now we give an example of the above processes.

[**Example 1**] **Eagle encoding**

We choose the parameters as $w_0 = 10010011$, $w_1 = 11000111$, $s_0 = 01011001$, $m = 10010101$.

Since $m[1] = 1$, we set $w = w_1$, then we have

$$s_1 = w \oplus (s_0 \oplus s_0^{+1}) = 11000111 \oplus 11101011 = 00101100$$

Since $m[2] = 0$, we set $w = w_0$, then we have

$$s_2 = w \oplus (s_1 \oplus s_1^{+1}) = 10010011 \oplus 01110100 = 11100111$$

Since $m[3] = 0$, we set $w = w_0$, then we have

$$s_3 = w \oplus (s_2 \oplus s_2^{+1}) = 10010011 \oplus 00101000 = 10111011$$

Since $m[4] = 1$, we set $w = w_1$, then we have

$$s_4 = w \oplus (s_3 \oplus s_3^{+1}) = 11000111 \oplus 11001100 = 00001011$$

Since $m[5] = 0$, we set $w = w_0$, then we have

$$s_5 = w \oplus (s_4 \oplus s_4^{+1}) = 10010011 \oplus 00011101 = 10001110$$

Since $m[6] = 1$, we set $w = w_1$, then we have

$$s_6 = w \oplus (s_5 \oplus s_5^{+1}) = 11000111 \oplus 10010011 = 01010100$$

Since $m[7] = 0$, we set $w = w_0$, then we have

$$s_7 = w \oplus (s_6 \oplus s_6^{+1}) = 10010011 \oplus 11111100 = 01101111$$

Since $m[8] = 1$, we set $w = w_1$, then we have

$$s_8 = w \oplus (s_7 \oplus s_7^{+1}) = 11000111 \oplus 10110001 = 01110110$$

Take the last bit of $s_0, s_1, s_2, s_3, ..., s_7$ as $c = 10111001$.

We use $(c, s_8) = (10111001, 01110110)$ as the results generated by Eagle encoding.

Next, we will provide another example of Eagle decoding.

[Example 2] Eagle decoding

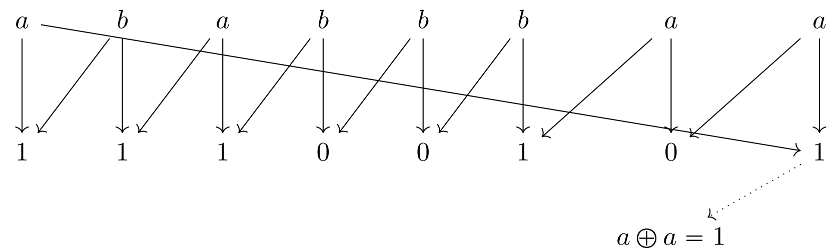
Known $w_0 = 10010011$, $w_1 = 11000111$, $s_8 = 01110110$, $c = 10111001$, how to compute m?

Firstly by $s_8 = w \oplus (s_7 \oplus s_7^{+1}) => s_7 \oplus s_7^{+1} = s8 \oplus w$, we don't know whether $w = w_0$ (the 8-th bit of m is 0) or $w = w_1$ (the 8-th bit of m is 1), we need a trial-and-error.

Let's assume $w = w_0$, we have

$$s_7 \oplus s_7^{+1} = s_8 \oplus w_0 = 01110110 \oplus 10010011 = 11100101$$

There is no such s_7 satisfies $s_7 \oplus s_7^{+1} = 11100101$.



Let's assume $w = w_1$, we have

$$s_7 \oplus s_7^{+1} = s_8 \oplus w_1 = 01110110 \oplus 11000111 = 10110001$$

There exists two solutions $s_7 = 01101111$ and $s_7 = 10010000$, these two solutions are bit reversed.

Since the 8-th bit of c is $c[8] = 1$, the last bit of s_7 is 1, we have $s_7 = 01101111$.

According to the above, because $w = w_1$, the 8-bit of m is 1.

Similarily we get $m = 10010101$.

It is not difficult to find that the above encoding process and decoding process are correct, that is (c, s_L) generated by encoding from m can be completely restored through the decoding process. In addition, the encoding process is sequential encoding in the order of m's bits, and the decoding process is sequential decoding in the reverse order of c's bits.

$$s_0 \xrightarrow{m[1]} s_1 \xrightarrow{m[2]} s_2 \xrightarrow{m[3]} \dots \xrightarrow{m[L]} s_L$$

$$s_0 \xleftarrow[c[1]]{} s_1 \xleftarrow[c[2]]{} s_2 \xleftarrow[c[3]]{} \dots \xleftarrow[c[L]]{} s_L$$

We also noticed the fact that in the above encoding and decoding process, all inputs and outputs do not need to appear s_0 , This means that the selection of s_0 will not affect the correctness of the encoding process and decoding process. The arbitrary of s_0 will bring the uncertainty of the encoded output.

For the convenience of the discussion in the following chapters, here we briefly analyze the effect of uncertainty of s_0 on the encoded output.

Given the parameters w_0 and w_1 that have odd number of different bits, for a certain input m of L bits, since s_0 is arbitrarily selected, it is obvious that c is uncertain, but is the final state s_L necessarily uncertain?

In fact, the answer is no. In some cases, such as $L = 2^u$ (that is, the parameter length is the power of 2), the final state s_L is determined for different choices of s_0 . The final state variable s_L which is the output of the encoding process is only related to the input m and has nothing to do with the choice of the initial state s_0. See Theorem 2 for details.

[**Theorem 2**] In Eagle encoding, given the parameters w_0 and w_1 that have L bits with different odd bits, for a certain L bit input m, if $L = 2^u$ is satisfied, then for any initial state s_0, after the Eagle encoding process, the final state s_L is only related to the input m, and is unrelated with the choice of the initial state s_0.

Proof. We represent m as binary stream $x_1x_2...x_L$, which $x_i \in \{0,1\}$, $1 \leqslant i \leqslant L$.

We execute the Eagle encoding process to m from x_1 to x_L as follows.

$$s_1 = w_{x_1} \oplus (s_0 \oplus s_0^{+1}) = f_1(w_{x_1}) \oplus (s_0 \oplus s_0^{+1})$$
$$s_2 = w_{x_2} \oplus (s_1 \oplus s_1^{+1}) = f_2(w_{x_1}, w_{x_2}) \oplus (s_0 \oplus s_0^{+2})$$
$$s_3 = w_{x_3} \oplus (s_2 \oplus s_2^{+1}) = f_3(w_{x_1}, w_{x_2}, w_{x_3}) \oplus (s_0 \oplus s_0^{+1} \oplus s_0^{+2} \oplus s_0^{+3})$$
$$s_4 = w_{x_4} \oplus (s_3 \oplus s_3^{+1}) = f_4(w_{x_1}, w_{x_2}, w_{x_3}, w_{x_4}) \oplus (s_0 \oplus s_0^{+4})$$

It is not difficult to find that for any $m = 2^v$, $s_m = f_m(w_{x_1}, ..., w_{x_m}) \oplus (s_0 \oplus s_0^{+m})$ holds, this can be proved by a simple mathematical induction.

In fact, the conclusion is correct for $v = 1$.

We assume that the conclusion is correct for $v - 1$, we have

$$s_{m/2} = f_{m/2}(w_{x_1}, ..., w_{x_{m/2}}) \oplus (s_0 \oplus s_0^{+m/2})$$

Since $s_{m/2}$ to s_m must do calculations with $m/2$ steps, we have

$$\begin{aligned} s_m = f_m(w_{x_1}, ..., w_{x_m}) \oplus (s_0 \oplus s_0^{+m/2}) \oplus (s_0^{+m/2} \oplus s_0^{+m}) \\ = f_m(w_{x_1}, ..., w_{x_m}) \oplus (s_0 \oplus s_0^{+m}) \end{aligned}$$

Since $L = 2^u$, we have $s_L = f_L(w_{x_1}, ..., w_{x_L}) \oplus (s_0 \oplus s_0^{+L})$, where $f_i(...)$ is irrelevant with s_0, by definition of cycle shift, we have $s_0 = s_0^{+L}$, so $s_L = f_L(w_{x_1}, ..., w_{x_L})$ which is irrelevant with s_0.

□

From theorem 2, for any parameters w_0 and w_1 with length $L = 2^u$, for any initial state s_0, execute Eagle encoding on m to obtain s_L which is irrelevant with s_0. In order to facilitate the description in the following chapters, we introduce two symbols ξ and ς.

$\xi_{w_0,w_1} : (s_0, m) \rightarrow (s_L, c)$: use the key (w_0, w_1) to execute Eagle encoding ([E1]-[E5]) on initial state s_0 and input m to obtain c and s_L .

$\varsigma_{w_0,w_1} : (s_L, c) \rightarrow (s_0, m)$: use the key (w_0, w_1) to execute Eagle decoding ([D1]-[D5]) on c and s_L to obtain s_0 and m.

ξ and ς both represent a complete Eagle encoding process and Eagle decoding process, and their introduction is mainly for the convenience of deriving encryption algorithms later. The Eagle encryption algorithm is a block symmetric encryption algorithm, and a complete Eagle encoding process is performed for each block.

In all the following chapters of this paper, we assume the length L is a power of 2.

3 Eagle encryption algorithm

The core idea of Eagle encryption algorithm comes from the Eagle encoding process. If we use the parameters w_0 and w_1 in the Eagle encoding process as encryption keys, the process of encoding input can be regarded as the process of encrypting plaintext input . Output (c, s_L) can be used as ciphertext. In fact, we can introduce uncertainty into the initial state s_0 without affecting the correctness of the decoding process. We will see later that uncertainty allows us to design a more secure encryption system.

Next, we will introduce the Eagle encryption algorithm in detail. The entire Eagle encryption algorithm is divided into three processes: key generator, encryption process, and decryption process.

3.1 Eagle key generator

First, the choice of the key is completely random, and the key needs to be shared between the encryptor and the decryptor. Since w_0 and w_1 must have odd number of different bits, there are only 2^{2L-1} effective keys with bits length of $2L$, one bit will be lost. That is, in the Eagle encryption algorithm, the number of bits for the key is always an odd number.

We randomly generate a number with bits length of $2L$. We take the first L bits as w_0. When the next L bits are different from w_0 with an odd number of "bits", then we directly take the next L bits as w_1; when the next L bits are different from w_0 with an even number of "bits", we set the next L bits as w_1 with the last bit inverted.

For example, we generate a number with 16 bits as 1001001101001000, we set $w_0 = 10010011$ (the first 8 bits) and $w_1 = 01001000$ (the last 8 bits), when

w_0 and w_1 have 6 (even) bits different, we set the last bit of w_1 inverted as $w_1 = 01001001$.

3.2 Eagle encryption process

For the $2L-1$-bit key w_0 and w_1, for the plaintext M, we construct Eagle encryption processes as follows:

$[M1]$ The plaintext M is grouped by L bits, and the last group with less than L bits are randomly filled into L bits. The total number of groups is assumed to be k , the grouped plaintext M is recorded as $M = (M_1, M_2, ..., M_k)$.

$[M2]$ We randomly generate some parameters.

$$S_0 : The\, initial\, state.$$
$$S_1^{'}, S_2^{'}, ..., S_k^{'}, S_{k+1}^{'} : The\, intermediate\, states.$$
$$M_{k+1} : The\, additional\, goup\, to\, M$$

$[M3]$ Encrypt the plaintexts as follows.

For the first group, excute the Eagle encoding as

$$\xi_{w_0,w_1} : (S_0, M_1) \to (S_1, C_1)$$

For the second group, excute the Eagle encoding as

$$\xi_{w_0,w_1} : (S_1 \oplus S_1^{'}, M_2) \to (S_2, C_2)$$

For the third group, excute the Eagle encoding as

$$\xi_{w_0,w_1} : (S_2 \oplus S_2^{'}, M_3) \to (S_3, C_3)$$

For the k-th group, excute the Eagle encoding as

$$\xi_{w_0,w_1} : (S_{k-1} \oplus S_{k-1}^{'}, M_k) \to (S_k, C_k)$$

For the $k+1$-th group, excute the Eagle encoding as

$$\xi_{w_0,w_1} : (S_k \oplus C_k, M_{k+1}) \to (S_{k+1}, C_{k+1})$$

$[M4]$ Encrypt the k intermediate states $S_1^{'}, S_2^{'}, ..., S_k^{'}, S_{k+1}^{'}$ as follows.

For the first intermediate state, excute the Eagle encoding as

$$\xi_{w_0,w_1} : (S_{k+1} \oplus C_{k+1}, S_1^{'}) \to (S_1^{'''}, C_1^{'''})$$

For the second intermediate state, excute the Eagle encoding as

$$\xi_{w_0,w_1} : (S_1^{'''} \oplus C_1^{'''}, S_2^{'}) \to (S_2^{'''}, C_2^{'''})$$

For the third intermediate state, excute the Eagle encoding as

$$\xi_{w_0,w_1} : (S_2^{'''} \oplus C_2^{'''}, S_3^{'}) \to (S_3^{'''}, C_3^{'''})$$

For the $k+1$-th intermediate state, excute the Eagle encoding as

$$\xi_{w_0,w_1} : (S_k^{'''} \oplus C_k^{'''}, S_{k+1}^{'}) \to (S_{k+1}^{'''}, C_{k+1}^{'''})$$

$[M4]$ Output $(C_1, C_2, ..., C_{k+1}, C_1^{'''}, C_2^{'''}, ..., C_{k+1}^{'''}, S_{k+1}^{'''})$ as ciphertext.

3.3 Eagle decryption process

With the same key w_0 and w_1, for ciphertext $(C_1, C_2, ..., C_{k+1}, C_1^{'''}, C_2^{'''}, ..., C_{k+1}^{'''}, S_{k+1}^{'''})$, the Eagle decryption processes are as follows:

$[C1]$ Restore intermediate states $S_1^{'}, S_2^{'}, ..., S_k^{'}$ as follows.

Restore the $k+1$-th intermediate state as

$$\varsigma_{w_0,w_1} : (S_{k+1}^{'''}, C_{k+1}^{'''}) \to (S_k^{*}, S_{k+1}^{'})$$

Restore the k-th intermediate state as

$$\varsigma_{w_0,w_1} : (S_k^{*} \oplus C_k^{'''}, C_k^{'''}) \to (S_{k-1}^{*}, S_k^{'})$$

Restore the $k-1$-th intermediate state as

$$\varsigma_{w_0,w_1} : (S_{k-1}^{*} \oplus C_{k-1}^{'''}, C_{k-1}^{'''}) \to (S_{k-2}^{*}, S_{k-1}^{'})$$

Restore the $k-2$-th intermediate state as

$$\varsigma_{w_0,w_1} : (S_{k-2}^{*} \oplus C_{k-2}^{'''}, C_{k-2}^{'''}) \to (S_{k-3}^{*}, S_{k-2}^{'})$$

Restore the second intermediate state as

$$\varsigma_{w_0,w_1} : (S_2^{*} \oplus C_2^{'''}, C_2^{'''}) \to (S_1^{*}, S_2^{'})$$

Restore the first intermediate state as

$$\varsigma_{w_0,w_1} : (S_1^{*} \oplus C_1^{'''}, C_1^{'''}) \to (S_0^{*}, S_1^{'})$$

$[C2]$ Calculate $M_1, M_2, ..., M_{k+1}$ as follows.

For the $k+1$-th group, calculate M_{K+1} as

$$\varsigma_{w_0,w_1} : (S_0^{*} \oplus C_{k+1}, C_{k+1}) \to (S_k, M_{k+1})$$

For the k-th group, calculate M_k as

$$\varsigma_{w_0,w_1} : (S_k \oplus S_k^{'}, C_k) \to (S_{k-1}, M_k)$$

For the $k-1$-th group, calculate M_{k-1} as

$$\varsigma_{w_0,w_1} : (S_{k-1} \oplus S_{k-1}^{'}, C_{k-1}) \to (S_{k-2}, M_{k-1})$$

For the second group, calculate M_2 as

$$\varsigma_{w_0,w_1} : (S_2 \oplus S_2^{'}, C_2) \to (S_1, M_2)$$

For the first group, calculate M_1 as

$$\varsigma_{w_0,w_1} : (S_1 \oplus S_1^{'}, C_1) \rightarrow (S_0, M_1)$$

$[C3]$ Output $(M_1, M_2, M_3, ..., M_k)$ as the plaintext.

Obviously, the above decryption processes can get the correct plaintext which can be summarized as.

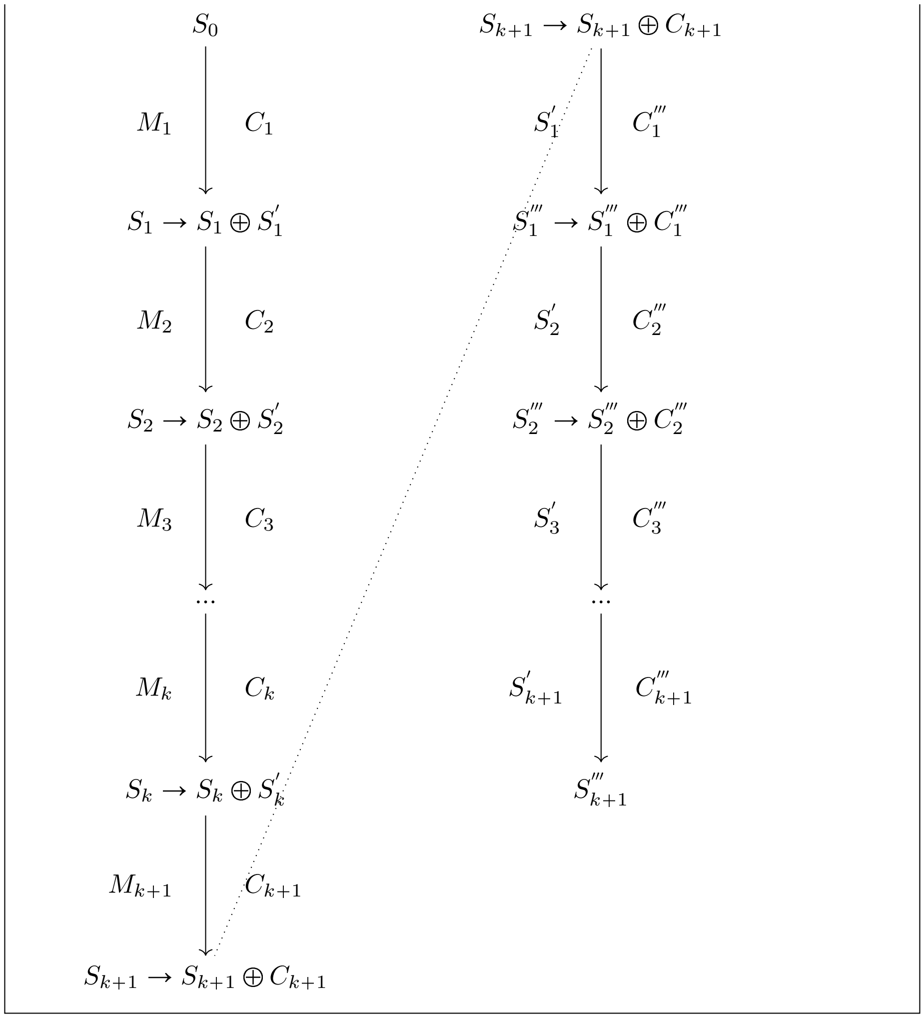
Parameters

$$Key : w_0\,(L - bits),\, w_1\,(L - bits)$$

$$Plaintext : M_1 \mid M_2 \mid ... \mid M_k \mid \boxed{M_{k+1}}$$

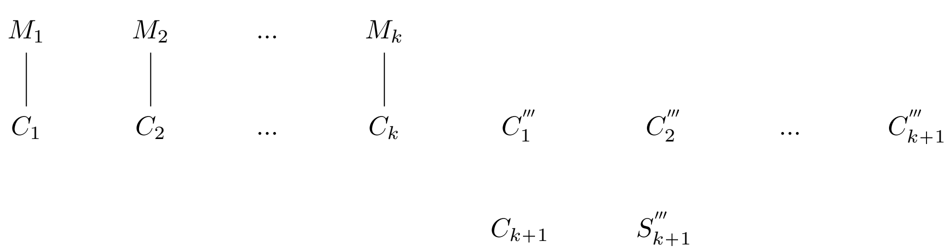
Encryption / Decryption

$$Random\,intermediate\,states : S_1^{'} \mid S_2^{'} \mid ... \mid S_{k+1}^{'}$$



From above, for the plaintext with k groups, in the encryption or decryption process, it is encoded or decoded bit by bit, and the encoding or decoding process of each bit is a certain calculation step, so the encryption process and the decryption process have computational complexity $O(kL)$.

In addition, observing the structure of plaintext and ciphertext.



We can easily find another conclusion that after Eagle encryption, the length of ciphertext is almost twice that of plaintext. This is because the system generates an additional random number of the same length as the plaintext before encryption. This is different from other symmetric block cipher algorithms. It is precisely because of this feature that the Eagle algorithm can resist all forms of linear and differential attacks.

Furthermore, by observing the plaintext ciphertext pairs of each group (M_i, C_i) $(1 \leqslant i \leqslant k)$, we found that any (w_0, w_1) is valid, see the following theorem.

[Theorem 3] In the above algorithm, given the plaintext-ciphertext pair (M_i, C_i) of the i-th group, any key (w_0, w_1) is valid.

Proof. This theorem is equivalent to proving the following conclusion: for a given (M_i, C_i), for any (w_0, w_1), S_x and S_y can be found to satisfy the following.

$$\xi_{w_0,w_1} : (S_x, M_i) \to (S_y, C_i) \tag{3.1}$$

Fixed any (w_0, w_1), by theorem 2, S_y is only related to M_i, every M_i can calculate S_y, every S_y can calculate M_i. Since $M_i, S_i \in \{0,1\}^L$, they are equally numerous, so M_i and S_y correspond one-to-one.

Use S_y and C_i to excute $\varsigma_{w_0,w_1} : (S_y, C_i) \to (S_x, M_i)$, we can get S_x satisfy (3.1). □

This conclusion indicates that, given any known plaintext-ciphertext pair of any group, since (S_x, S_y) is unknown, no matter what algorithm (including exhaustive search) you use, you cannot determine any characteristics of the key (w_0, w_1).

Moreover, since the encryption process between any two groups is independent and there are no common variables between any two groups, any known or constructed plaintext attack is invalid to Eagle encryption algorithm. For more details, please refer to the following chapters.

4 Linear attack analysis to Eagle encryption algorithm

Linear attack is a very effective attack method proposed by M. Matsui[2] at the European Cryptographic Conference in 1993. Later, scholars quickly discovered that the linear attacks are applicable to almost all block encryption algorithms, and linear attacks have became the main attacks for block encryption algorithms. Various new attacks based on linear attacks are constantly being proposed.

The core idea of linear attack is to take the nonlinear transformation in the cryptographic algorithm, such as the linear approximation of the S-box, and then extend the linear approximation to the linear approximation of the round function, and then connect these linear approximations to obtain a linear approximation of the entire cryptographic algorithm, finally a large number of known plaintext-ciphertext pairs encrypted with the same key are used to exhaustively obtain the plaintext and even the key.

We have noticed that the reason why linear attacks have become an effective attack for block encryption algorithms is that when the key is known, there is a certain implicit linear relationship between the ciphertext and the plaintext. By analyzing the known plaintext-ciphertext pairs, some effective linear relations can be obtained, and some bits of the key can be guessed.

In the Eagle encryption processes, for a certain group, suppose that the initial state at the beginning of the group is S_{i-1} , the plaintext of the group is M_i , the keys are w_0 and w_1 , after the [E1]-[E5], we obtain the new state S_i and the encoding result C_i . Only C_i is included in the ciphertext, only M_i is included in the plaintext, S_{i-1} and S_i are not included in the plaintext or ciphertext, that is, S_{i-1} and S_i are invisible to the decryption party and thus invisible to the attacker.

Back to theorem 3, known M_i and C_i, for any w_0 and w_1, there exists (S_x, S_y) satisfy $\xi_{w_0,w_1} : (S_x, M_i) \to (S_y, C_i)$. Here we introduce a stronger conclusion that (S_x, S_y) not only exists, but is also unique, as shown in the following theorem.

[Theorem 4] Known M and C, fixed any w_0 and w_1, there exists unique (S_x, S_y) satisfy $\xi_{w_0,w_1} : (S_x, M) \to (S_y, C)$.

Proof. Fixed any w_0 and w_1, in the proof process of theorem 3, we know that M and S_y correspond one to one, that is to say, S_y is unique.

Now we prove S_x is also unique by contradiction. Let's assume that there are two different S_{x_1} and S_{x_2} that both satisfy.

$$\xi_{w_0,w_1} : (S_{x_1}, M) \to (S_y, C)$$
$$\xi_{w_0,w_1} : (S_{x_2}, M) \to (S_y, C)$$

However, according to $\varsigma_{w_0,w_1} : (S_y, C) \to (S_x, C)$, only one S_x can be satisfied, this is contradict.

□

The conclusion of Theorem 4 also indicates that, known M_i and C_i, any key (w_0, w_1) in the key space have the same probability, which can be denoted as

$$Pr(W = (w_0, w_1)|(M = M_i, C = C_i)) = 1/(2^{2L-1}).$$

For a single group, known plaintext-ciphertext pair, any key is valid and have the same probability to give a solution.

For two adjacent groups, the initial state of the next group differs from the final state of the previous group by a random number. The calculation process between two adjacent groups can be regarded as completely independent.

5 Differential attack analysis to Eagle encryption algorithm

Differential attack was proposed by Biham and Shamir[6] in 1990, it is a chosen-plaintext attack. Its core idea is to obtain key information by analyzing specific plaintext and ciphertext differences.

The essence of a differential attack is to track the "difference" of the plaintext pair, where the "difference" is defined by the attacker according to the target, which can be an exclusive XOR operation or other target values. For example, if you choose the plaintext M and the difference δ, the other plaintext is $M + \delta$. The attacker mainly analyzes the possible keys by analyzing the difference between the ciphertext C and $C + \varepsilon$.

For the Eagle encryption algorithm, suppose the differential attacker chooses two specific plaintexts M_1 and M_2 , their difference is δ, that is $M_2 = M_1 + \delta$, the corresponding ciphertexts are C_1 and C_2 , and the difference between the ciphertexts is ε , and That is $C_2 = C_1 + \varepsilon$. Since in the encryption processes of Eagle algorithm, C_1 and C_2 are completely random, it is completely uncertain whether the difference ε of the ciphertext is caused by randomness or the spread of the plaintext. Furthermore, for any key (w_0, w_1), $\xi_{w_0,w_1}(?, M_1)$ and $\xi_{w_0,w_1}(?, M_2)$ subject to the same probability distribution, which can be denoted as

$$Pr(C_1 = c_1, C_2 = c_2|(M_1 = m_1, M_2 = m_2, W = (w_0, w_1))) = 1/2^{2L}.$$

That is to say, for any specific plaintext M_1 and M_2 selected by the attacker, after being encrypted with the same key, the corresponding block ciphertexts C_1 and C_2 are completely random, and any possible value in the ciphertext space appears with equal probability. The attacker has no way to capture the propagation characteristics of the "difference" in the plaintext.

6 One-way function design

6.1 Introduction to one-way functions

Before constructing the one-way function, we briefly introduce the properties of one-way function and the relationships with the $P \neq NP$ problem.

[Definition 1] A function is a one-way function means that the function satisfies the following properties:

a) For a given x, there exists a polynomial-time algorithm that output $f(x)$.

b) Given y, it is difficult to find an x that satisfies $y = f(x)$, that is, there does not exists a polynomial-time algorithm that finding the x.

The NP-complete problem refers to a set of problems that are verifiable in polynomial-time algorithm. For all NP-complete problems, whether there exists algorithms that are solvable in polynomial-time, this is the P vs NP problem. If $P \neq NP$, then for some NP problems, there is no algorithm that is solvable in polynomial-time. If $P = NP$, then for all NP problems, there exists algorithms that are solvable in polynomial-time.

If one-way function exists, it means that there exists such an NP problem, which has no deterministic polynomial time solvable algorithm, that is, $P \neq NP$. This is a direct inference, which can be directly described as the following theorem , See [9] for details.

[**Theorem 5**] If one-way function exists, then $P \neq NP$.

We then introduce an additional simple algorithmic problem, which we describe as the following theorem.

[**Theorem 6**] For two sets selected completely independently, the number of elements is l_1, l_2 , then the average algorithm complexity of finding the common elements of the two sets (there may be only one common element at most) is at least $c * min(O(l_1), O(l_2))$, where c is a certain constant.

This is because the remaining unvisited elements in the two sets will be visited at least once with equal probability before no common element is found.

6.2 Construction of one-way functions

For short key encryption algorithms, the security of the algorithm depends on the computational complexity of cracking the key when given the known plaintext-ciphertext pairs. In theory, if the key can only be cracked through exhaustive search, this algorithm is considered computationally secure. However, currently all short key encryption algorithms, with known plaintext-ciphertext pairs, have no evidence to suggest that attackers can only crack the key through exhaustive search.

In this chapter, we will further upgrade the above encryption algorithm and construct a new encryption algorithm called *Eagle**, which is still a short key encryption algorithm, and its encryption and decryption processes are completed within polynomial time. Given any known plaintext-ciphertext pairs, we will prove that the problem of cracking its key can be equivalently reduced to the problem of cracking plaintext using only ciphertext in another encryption algorithm, which can only be tested through exhaustive search for every possible key.

6.2.1 Introduction to *Eagle**

The key is $W = (w_0, w_1)$, w_0 and w_1 have odd number of different bits.

The plaintext is $M = (M_1, M_2, ..., M_k)$ which is gouped by L-bits.

Next, we will provide a detailed introduction to the design of the *Eagle** encryption system, which consists of three processes: Random parameters generator, encryption process, and decryption process.

[Random parameters generator]
First we randomly generate $k+1$ group keys.

$$(w_i^{(0)}, w_i^{(1)}), 1 \leqslant i \leqslant k+1$$

where $w_i^{(0)}$ and $w_i^{(1)}$ have odd number of different bits.
Then we randomly generate k intermediate states.

$$S_i^{'}, 1 \leqslant i \leqslant k$$

Then we randomly genrate the $k+1$ group inserted plaintext and the initial state.

$$M_{k+1}: \, the \, k+1 \, group \, inserted \, plaintext.$$
$$S_0: \, the \, initial \, state.$$

[encryption process]
The encryption process is as follows.
$[E1^*]$ Encrypt $(M_1, M_2, ..., M_k, M_{k+1})$.
For the first group M_1, excute the following

$$\xi_{w_1^{(0)}, w_1^{(1)}} : (S_0, M_1) \to (S_1, C_1)$$

For the second group M_2, excute the following

$$\xi_{w_2^{(0)}, w_2^{(1)}} : (S_1 \oplus S_1^{'}, M_2) \to (S_2, C_2)$$

For the i-th group M_i, excute the following

$$\xi_{w_i^{(0)}, w_i^{(1)}} : (S_{i-1} \oplus S_{i-1}^{'}, M_i) \to (S_i, C_i)$$

For the $k+1$-th group M_{k+1}, excute the following

$$\xi_{w_{k+1}^{(0)}, w_{k+1}^{(1)}} : (S_k \oplus S_k^{'}, M_{k+1}) \to (S_{k+1}, C_{k+1})$$

$[E2^*]$ Use (w_0, w_1) to encrypt $(w_1^{(0)}, w_1^{(1)}, ..., w_i^{(0)}, w_i^{(1)}, ..., w_{k+1}^{(0)}, w_{k+1}^{(1)})$ and $(S_1^{'}, ..., S_i^{'}, ..., S_k^{'})$.

In this step, $(w_1^{(0)}, w_1^{(1)}, ..., w_i^{(0)}, w_i^{(1)}, ..., w_{k+1}^{(0)}, w_{k+1}^{(1)})$ and $(S_1^{'}, ..., S_i^{'}, ..., S_{k+1}^{'})$ can be seen as another $2(k+1)+k = 3k+2$ grouped plaintext, we can write it as $M^* = (M_1^*, M_2^*, ..., M_{3k+2}^*)$, where

$$\begin{cases} M_i^* = w_i^{(0)}, \, 1 \leqslant i \leqslant k+1 \\ M_i^* = w_{i-k-1}^{(1)}, \, k+2 \leqslant i \leqslant 2k+2 \\ M_i^* = S_{i-2k-2}^{'}, \, 2k+3 \leqslant i \leqslant 3k+2 \end{cases}$$

For the first group of M^*, the initial state can be set as S_{k+1}, excute the following

$$\xi_{w_0,w_1} : (S_{k+1}, M_1^*) \rightarrow (S_1^*, C_1^*)$$

For the second group of M^*, excute the following

$$\xi_{w_0,w_1} : (S_1^* \oplus C_1^*, M_2^*) \rightarrow (S_2^*, C_2^*)$$

For the i-th group of M^*, excute the following

$$\xi_{w_0,w_1} : (S_{i-1}^* \oplus C_{i-1}^*, M_i^*) \rightarrow (S_i^*, C_i^*)$$

For the last group of M^*, excute the following

$$\xi_{w_0,w_1} : (S_{3k+1}^* \oplus C_{3k+1}^*, M_{3k+2}^*) \rightarrow (S_{3k+2}^*, C_{3k+2}^*)$$

[$E3^*$] Output $(C_1, C_2, ..., C_{k+1}, C_1^*, C_2^*, ..., C_{3k+2}^*, S_{3k+2}^*)$ as the ciphertext.

[decryption process]

The process of decryption are as the following

[$D1^*$] Use (w_0, w_1) to decrypt $(C_1^*, C_2^*, ..., C_{3k+2}^*, S_{3k+2}^*)$ to obtain $M^* = (M_1^*, M_2^*, ..., M_{3(k+1)}^*)$.

For the last group $(3k+2)$, excute the following

$$\varsigma_{w_0,w_1} : (S_{3k+2}^*, C_{3k+2}^*) \rightarrow (S_{3k+1}^{**}, M_{3k+2})$$

For the $3k+1$ group, excute the following

$$\varsigma_{w_0,w_1} : (S_{3k+1}^{**} \oplus C_{3k+1}^*, C_{3k+1}^*) \rightarrow (S_{3k}^{**}, M_{3k+1})$$

For the i-th group, excute the following

$$\varsigma_{w_0,w_1} : (S_i^{**} \oplus C_i^*, C_i^*) \rightarrow (S_{i-1}^{**}, M_i)$$

For the first group, excute the following

$$\varsigma_{w_0,w_1} : (S_1^{**} \oplus C_1^*, C_1^*) \rightarrow (S_0^{**}, M_1)$$

[$D2$]* Translate $M^* = (M_1^*, M_2^*, ..., M_{3k+2}^*)$ to ($w_1^{(0)}$, $w_1^{(1)}$, ..., $w_i^{(0)}$, $w_i^{(1)}$, ..., $w_{k+1}^{(0)}$, $w_{k+1}^{(1)}$) and ($S_1^{'}$, ..., $S_i^{'}$, ... , $S_k^{'}$).

$$\begin{cases} w_i^{(0)} = M_i^*,\ 1 \leqslant i \leqslant k+1 \\ w_i^{(1)} = M_{i+k+1},\ 1 \leqslant i \leqslant k+1 \\ S_i^{'} = M_{i+2k+2},\ 1 \leqslant i \leqslant k \end{cases}$$

[$D3$]* Decrypt $(C_1, C_2, ..., C_{k+1})$ to obtain $(M_1, M_2, ..., M_k, M_{k+1})$.

For the last group, excute the following

$$\varsigma_{w_{k+1}^{(0)},w_{k+1}^{(1)}} : (S_0^{**}, C_{k+1}) \rightarrow (S_k, M_{k+1})$$

For the $k-$th group, excute the following

$$\varsigma_{w_k^{(0)},w_k^{(1)}}:(S_k\oplus S_k^{'},C_k)\rightarrow(S_{k-1},M_k)$$

For the $i-$th group, excute the following

$$\varsigma_{w_i^{(0)},w_i^{(1)}}:(S_i\oplus S_i^{'},C_i)\rightarrow(S_{i-1},M_i)$$

For the first group, excute the following

$$\varsigma_{w_1^{(0)},w_1^{(1)}}:(S_1\oplus S_1^{'},C_1)\rightarrow(S_0,M_1)$$

$[D4]^*$ Output $(M_1, M_2, ..., M_k)$ as the plaintext.

6.2.2 Basic analysis of *Eagle**

It is obvious that Algorithm *Eagle** is correct because the decryption process and encryption process are mutually inverse.

From the encryption process above, it can be seen that the *Eagle** encryption algorithm is divided into two stages: the preparation stage and the encryption stage. In the preparation stage, different keys and initial states are randomly generated for each group's plaintext. In the encryption stage, the entire process is divided into two independent processes. The first process encrypts the plaintext of each group using independent random keys and states. The second process encrypts all intermediate keys and states generated during the preparation phase using the given short key.

It should also be noted that after being encrypted by the *Eagle** encryption algorithm, the length of the ciphertext is about 4 times the length of the plaintext. This is because in the preparation stage of the *Eagle** encryption algorithm, independent random numbers with a length of about 3 times the plaintext are generated, and then these random numbers are encrypted using the known short key. The encrypted ciphertext formed by these random numbers is also bound to the final ciphertext.

The *Eagle** encryption algorithm performs bit by bit in encryption and decryption process, and the operation for each bit is also a constant level of computational complexity. Therefore, the encryption and decryption complexity of the *Eagle** encryption algorithm can be regarded as $O(kL)$.

6.2.3 Safety analysis of *Eagle**

When given the ciphertext (C_1, C_2, ..., C_{k+1}, C_1^*, C_2^*, ... , C_{3k+2}^*, S_{3k+2}^*) of the *Eagle** encryption algorithm, we define the following function.

$$f(w_0, w_1) = (M_1, M_2, ..., M_k)$$

where w_0, w_1 is the short key and $(M_1, M_2, ..., M_k)$ is the plaintext decrypted by the *Eagle** decryption algorithm.

The inverse function of f can be defined as

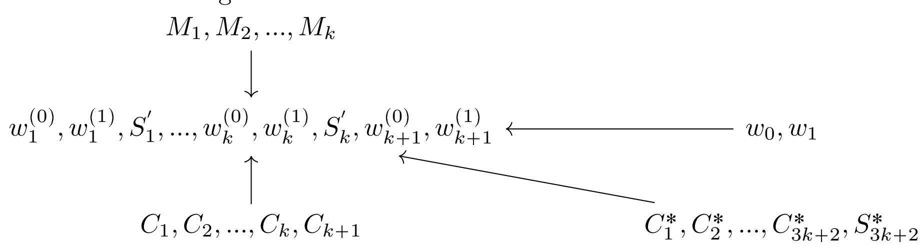
$$f^{-1}(M_1, M_2, ..., M_k) = \{(w_0, w_1)|f(w_0, w_1) = (M_1, M_2, ..., M_k)\}$$

Solving f^{-1} is equivalent to finding the key (w_0, w_1) to obtain the plaintext $(M_1, M_2, ..., M_k)$.

In fact, if we can prove that only exhaustive search can solve f^{-1}, that is, for any attacker who only knows the plaintext-ciphertext pairs, there is no more effective method other than exhaustive search to find the correct key, then the *Eagle** algorithm can be considered secure in the sense of computational complexity.

Next, we will argue that there is indeed no effective method other than exhaustive search to solve f^{-1}.

We observe the process of *Eagle** encryption algorithm and it is not difficult to find the following structure.



The ciphertext $(C_1, C_2, ..., C_{k+1}, C_1^*, C_2^*, ..., C_{3k+2}^*, S_{3k+2}^*)$ can be divided into two independent parts $(C_1, C_2, ..., C_{k+1})$ and $(C_1^*, C_2^*, ..., C_{3k+2}^*, S_{3k+2}^*)$. The plaintext $(M_1, M_2, ..., M_k)$ is only related to the first part of the ciphertext $(C_1, C_2, ..., C_{k+1})$, and has no association with the other part of the ciphertext $(C_1^*, C_2^*, ..., C_{3k+2}^*, S_{3k+2}^*)$.

How to find $(w_i^{(0)}, w_i^{(1)}, S_j^{'})$? For the left part of the diagram, given any (M_i, C_i), according to Theorem 3, any $(w_i^{(0)}, w_i^{(1)})$ is valid. For the right part of the diagram, solving $(w_i^{(0)}, w_i^{(1)}, S_j^{'})$ is equivalent to obtain the plaintext with the ciphertext $(C_1^*, C_2^*, ..., C_{3k+2}^*, S_{3k+2}^*)$ and the key (w_0, w_1). In fact, the key (w_0, w_1) in the right part is exactly the solution that f^{-1} is looking for.

The above process can be equivalently understood as follows: the key stream on the left is the plaintext on the right. For the left part, in the case where the plaintext ciphertext pair is known, finding $(w_i^{(0)}, w_i^{(1)}, S_j^{'})$ is valid is equivalent to finding the key in OTP encryption when only the ciphertext is known. For the right part, finding $(w_i^{(0)}, w_i^{(1)}, S_j^{'})$ that is valid is equivalent to solving the plaintext when only the ciphertext is known.

Since we don't know $(w_i^{(0)}, w_i^{(1)}, S_j^{'})$, the only way to find a valid (w_0, w_1) is to use exhaustive search to test every (w_0, w_1).

The formal description is as follows.

[Theorem 7] The computational complexity of solving f^{-1} is at least $O(2^{2L-1})$.

Proof. We first consider solving $(w_i^{(0)}, w_i^{(1)})$, formally defined the following function

$$f_1(w_i^{(0)}, w_i^{(1)}) = (M_1, M_2, ..., M_k)$$

where $1 \leqslant i \leqslant k+1$.

f_1^{-1} can be written as

$$f_1^{-1}(M_1, M_2, ..., M_k) = \{(w_i^{(0)}, w_i^{(1)}) | f_1(w_i^{(0)}, w_i^{(1)}) = (M_1, M_2, ..., M_k)\}$$

where $1 \leqslant i \leqslant k+1$.

Solving f_1^{-1} is equivalent to solving the following equations

$$\begin{cases} F((C_1, C_2, ..., C_{k+1}), (M_1, M_2, ..., M_k)) \\ \quad = (w_1^{(0)}, w_1^{(1)}, w_2^{(0)}, w_2^{(1)}, ..., w_{k+1}^{(0)}, w_{k+1}^{(1)}) \\ \\ G((C_1^*, C_2^*, ..., C_{3k+2}^*, S_{3k+2}^*)) \\ \quad = (w_1^{(0)}, w_1^{(1)}, w_2^{(0)}, w_2^{(1)}, ..., w_{k+1}^{(0)}, w_{k+1}^{(1)}) \end{cases}$$

where F solve $(w_i^{(0)}, w_i^{(1)})$ in the left part and G solve $(w_i^{(0)}, w_i^{(1)})$ in the right part. Obviously F and G are two independent processes.

For F, according to Theorem 3, all $(w_i^{(0)}, w_i^{(1)})$ are valid, which means there are $2^{(k+1)(2L-1)}$ feasible solutions.

For G, since all (w_0, w_1) are valid, there are 2^{2L-1} feasible solutions.

According to theorem 6, the computation complexity of solving f_1^{-1} is at least $O(2^{2L-1})$.

Moreover, solving f^{-1} can directly solve f_1^{-1} in polynomial time, because (w_0, w_1) can directly decrypt using *Eagle** decryption algorithm to obtain $(w_1^{(0)}, w_1^{(1)}, w_2^{(0)}, w_2^{(1)}, ..., w_{k+1}^{(0)}, w_{k+1}^{(1)})$ in polynomial time.

So the computational complexity of solving f^{-1} is at least $O(2^{2L-1})$. □

Due to the fact that solving f can be completed in polynomial time and the computational complexity of solving f^{-1} is exponential, f is a one-way function.

According to Theorem 5, we conclude that $P \neq NP$.

References

[1] Shannon C E. Communication theory of secrecy systems. *Bell System Technical Journal.*, 28(4):656 –715, 1949.

[2] M. Matsui. Linear cryptanalysis method for DES cipher. *Advances in cryptology: EUROCRYPT9́3, LNCS765, springer verlag*, 1993:386-397.

[3] Kaliski B S, Robshaw M J B. Linear Cryptanalysis Using Multiple Approximations. *Annual International Cryptology Conference. Springer, Berlin, Heidelberg*, 1994.

[4] Biryukov, A., De Cannière, C., Quisquater, M. On Multiple Linear Approximations. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 1–22. Springer, Heidelberg, 2004.

[5] Cho, J.Y., Hermelin, M., Nyberg, K. A new technique for multidimensional linear cryptanalysis with applications on reduced round serpent. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 383–398. Springer, Heidelberg, 2009.

[6] Eli Biham, Adi Shamir. Differential Cryptanalysis of the Data Encryption Standard. Springer-Verlag, 1993.

[7] Biham E, Biryukov A, Shamir A. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In: Advances in Cryptology—EUROCRYPT 1999. Springer Berlin Heidelberg, 1999: 12–23.

[8] Tsumoo Y, Tsujihara E, Shigeri M, et al. Cryptanalysis of CLEFIA using multiple impossible differentials. In: 2008 International Symposium on Information Theory and Its Applications—ISITA 2008. IEEE, 2008: 1–6.

[9] Sanjeev Arora, Boaz Barak. Computational Complexity: A Modern Approach. 2009.