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, the security requirements for cryptographic keys are much stricter than those for P≠NP, the security of the key must ensure not only a sufficiently high computational complexity to crack it, but also consider the security of each bit of the key, while fully avoiding the effectiveness of various attack methods.

In this paper, we innovatively propose a new encoding mechanism and develop a novel block symmetric encryption algorithm, which be named Eagle, whose encryption and decryption can be completed in linear time. The key consists of 12 variables, for the attacker, in the case when only the plaintext-ciphertext correspondence is known, the problem of cracking the key is equivalent to solving a system of equations about 12 unknown variables $(w_1, w_2, ..., w_{11}, w_{12})$ together with some implicit variables which are generated by the encryption process.

In this paper, we will prove that there is no universal algorithm without brute-force searching at least one variable during the process of verifying five variables (w_1, w_2, w_3, w_4, w_9) whose computational complexity is not greater than solving the 12 variables. After performing elimination the implicit variables, the problem of verifying the five variables will be equivalently transformed into a series of general quintic equations about one variable $w_{11} \oplus w_{12}$ with coefficient given by other six variables (w_5, w_6, w_7, w_8, w_{10}, w_{11}), according to the conclusion that "there is no algebraic solution to quintic equations", we can't establish a system of algebraic equations only containing the six variables (w_5, w_6, w_7, w_8, w_{10}, w_{11})

*20070602094@alu.cqu.edu.cn

without $w_{11} \oplus w_{12}$. Without solving the six variables, the problem of verifying the five variables (w_1, w_2, w_3, w_4, w_9) is equivalent to solve the slope of a line given only the x-coordinates of several points on it, this implies that there is no universal algorithm without brute-force searching at least one variable to verify the five variables (w_1, w_2, w_3, w_4, w_9). Since the computational complexity of brute-force searching any single variable is exponential, thus the complexity of cracking the key is exponential. Thus decryption is a one-way function, and according to "the existence of one-way function means P≠NP", solving the unsolved problem of P vs NP.

In addition, we further constructed another block symmetric encryption algorithm, which be named $Eagle^*$. The key of $Eagle^*$ is composed of three independent variables. Given any plaintext ciphertext correspondences, the problem of verifying any two variables in the key is equivalent to solving a system of quintic equations. Based on the conclusion that there is no algebraic solution to quintic equations, we assert that there is no fast algorithm to verify any two variables, and thus the computational complexity of cracking the key is equal to the computational complexity of completely exhausting the three variables. In addition, we conducted an in-depth analysis that without cracking the key, it is not possible to directly decrypt new ciphertext by simply collecting a large number of ciphertext plaintext correspondences. Due to the fact that ciphertext consists of four parts, three parts are completely random, while the other small part is calculated using plaintexts, keys, and random numbers, and their distribution is also completely random, thus $Eagle^*$ is resistant to all forms of linear and differential attacks.

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.

The P vs NP problem was first proposed by Cook[10] and has become one of the most important unsolved problems in the field of computer science. At the same time, the P vs NP problem has also become the most important theoretical foundation in the field of cryptography, determining the research direction of cryptography. P≠NP is the most basic assumption in cryptography, but this is only a necessary condition. In fact, the security requirements of a key are much stricter than P≠NP. This is because even if P≠NP, there exists an encryption algorithm, the computational complexity of cracking the key is exponential to the length of the key, which does not guarantee that each bit of the key is secure. To ensure that each bit of the key is secure, it is necessary to ensure that exhaustive search is only effective for cracking the key, which is much stricter than the exponential computational complexity to cracking the key.

Among various attack methods, linear attack and differential attack are the two most core attack methods, which have become important analytical tools in the field of cryptography, driving the improvement of encryption algorithm.

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 Chapter 3, we propose a novel encoding mechanism which we named Eagle. The distinguishing feature of Eagle encoding algorithm from any other encoding algorithm is that for any given group of input and output texts, the key parameters are unsolvable.

In Chapter 4, we further delve into the mathematical laws behind the Eagle encoding algorithm and discover an operation that satisfies the allocation law for XOR operations. This operation is very similar to the allocation law of multiplication operations, but does not satisfy the commutative and associative laws, we refer to it as a "right multiplication operation on binary". Then we construct a binary operation of two-dimensional vectors which serves as a key ingredient for the subsequent design of encryption algorithms.

In Chapter 5, we constructed a block symmetric encryption algorithm based on the Eagle encoding algorithm which we named it Eagle. In this encryption algorithm, there are six completely independent sets of random keys, each containing two random variables. For the plaintexts with K groups, $18K+8$ random variables will be randomly generated at the encryption party. After encryption is completed, ciphertext text with $19K + 8$ groups will be generated. After encryption is completed, the encryption party will delete these generated intermediate random variables. For the attacker, knowing the plaintext-ciphertext correspondences, the problem of cracking the key is equivalent to solving a system of equations. In this chapter, we will prove that the computational complexity of cracking the key is exponential to the length of the key, this also means that decryption is a one-way function, thus solving the P vs NP problem based on the conclusion that "one-way functions exist means P$\neq$NP".

In fact the encryption algorithm developed in Chapter 5 does not meet the security requirements for the key. This is because, when knowing the plaintext-ciphertext correspondence, although the complexity of cracking the key is exponential to the length of the key, it does not guarantee that only exhaustive search is effective for cracking the key. There may be some other attack methods that can effectively reduce the difficulty of cracking the key. Such as for a 768 bit key, although it can ensure that the computational complexity of cracking the key is at least $O(2^{64})$, this is still far from the ideal security of $O(2^{768})$. The encryption algorithm constructed in Chapter 5 only analyzes the computational complexity of cracking the key given plaintext ciphertext correspondences, which does not analyzes its ability to resist linear and differential attacks, For attackers, they can construct plaintexts with certain characteristics multiple times, then find some structures that we did not anticipate from the relationship between plaintext and ciphertext.

In Chapter 6, we first construct nonlinear operations of one parameter on another, such as "high-order" operations. Based on this nonlinear construction, we redesigned a block symmetric encryption algorithm and named it $Eagle^*$. For $Eagle^*$, knowing any plaintext ciphertext correspondence, there is no more effective way for attackers to crack the key than exhaustive search. This is because the key is consist of three variables, the problem of verifying any two variables in the key is equivalent to solving a system of quintic equations. Based on the conclusion that there are no algebraic solutions to quintic equations, we assert that there is no fast algorithm to verify any two variables, and thus the computational complexity of cracking the key is equal to the computational complexity of completely exhausting the three variables.

On the other hand, for the $Eagle^*$ encryption algorithm, The ciphertext can be seen as four independent parts. Three parts of the ciphertext is completely determined by the intermediate random number which is completely unrelated to the plaintext. The fourth part of the ciphertext is determined by the intermediate random number together with the plaintext sequence, but it is completely indistinguishable whether the uncertainty of the ciphertext brought by the random number or brought by different plaintexts. This theoretically ensure that $Eagle^*$ can resist all forms of linear attacks and differential attacks.

2 Notations

Some notations

$\oplus$ - XOR operation.

D^{+n} - Do left cycle shift of D by n bits.

$m[i]$ - The $i-$th bit of m.

$\mathbb{E}_{w_0,w_1}(m)$ - Excute Eagle encoding algorithm as shown in Algorithm 1.

$\mathbb{D}_{w_0,w_1}(s)$ - Excute Eagle decoding algorithm as shown in Algorithm 2.

$m * \delta$ - Excute the right multiplication of m with δ as shown in Algorithm 3.

s/δ - Excute the right division of s to δ as shown in Algorithm 4.

$m * \delta^2$ - Excute the right multiplication of m with δ twice as $m * \delta * \delta$ as shown in Algorithm 9.

s/δ^2 - Excute the right division of s to δ twice as $s/\delta/\delta$ as shown in Algorithm 10.

$(m_1, m_2) \bigotimes (k_1, k_2) = (c_1, c_2)$ - A binary operation of two-dimensional vectors defined in chapter 4.2.

$\phi(x)$ - A simple function changes x to an odd number if x contains odd number of bits that are 1, as shown in Algorithm 5.

$\psi(x)$ - A simple function changes x to a number contains odd number of bits that are 1 if x is an odd number, as shown in Algorithm 6.

3 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

$$s_1 = w \oplus (s_0 \oplus s_0^{+1}) = w_0 \oplus (s_0 \oplus s_0^{+1}) \tag{3.1}$$

From (3.1), we can easily know

$$s_0 \oplus s_0^{+1} = s_1 \oplus w \tag{3.2}$$

If we only know s_1 but not s_0, we don 't know whether $w = w_0$ or $w = w_1$ we used in (3.1), we can check it through a simple trial-and-error test. For example, we guess the parameter $w = w_1$ was used in (3.1), we need to find s_x to satisfy

$$s_x \oplus s_x^{+1} = s_1 \oplus w_1 \tag{3.3}$$

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**] Two L-bit parameters w_0 and w_1 which have odd number of different bits, given 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

$$s_1 \oplus w_1 = w_0 \oplus (s_0 \oplus s_0^{+1}) \oplus w_1 = (s_0 \oplus s_0^{+1}) \oplus (w_0 \oplus w_1) \tag{3.4}$$

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 (3.4), we have

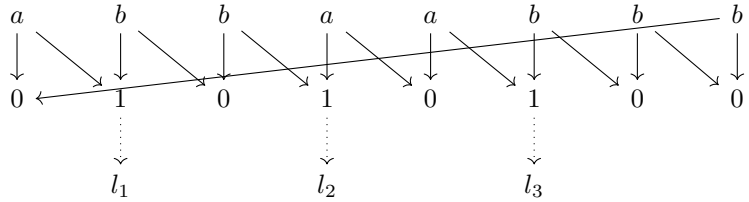
$$s_x \oplus s_x^{+1} = (s_0 \oplus s_0^{+1}) \oplus (w_0 \oplus w_1) \tag{3.5}$$

By simple calculation we have

$$(s_0 \oplus s_x) \oplus (s_0 \oplus s_x)^{+1} = w_0 \oplus w_1 \tag{3.6}$$

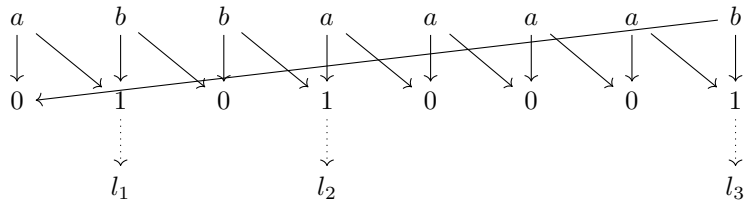
We set $s_y = s_0 \oplus s_x$, then there are odd number of 1 in the bit string of $s_y \oplus s_y^{+1}$, without of generality, we suppose that the bits with 1 are $l_1, l_2, ..., l_u$ (u is odd). Compare to the first bit of s_y, the $l_1 + 1$ bit of s_y is different from the first bit of s_y , the $l_2 + 1$ 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 test, we can accurately check which one ($w = w_0$ or $w = w_1$) we just used in (3.1).

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.

Algorithm 1 *Eagle*-Encoding.

Input:

Parameter: (w_0, w_1).

Input: $m = [m[1], m[2], ..., m[L]]$.

Random number: s_0.

Output:

Output: (c, s_L),

1: Define c as a bit array with length L.
2: **for** $i = 1; i \leqslant L; i++$ **do**
3: $c[i] \leftarrow s_{i-1}[L]$
4: **if** m[i] $= 0$ **then**
5: $s_i \leftarrow w_0 \oplus (s_{i-1} \oplus s_{i-1}^{+1})$
6: **else**
7: $s_i \leftarrow w_1 \oplus (s_{i-1} \oplus s_{i-1}^{+1})$
8: **end if**
9: **end for**

Where the i-th bit of c takes the last bit of state s_{i-1}.

[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.

Algorithm 2 *Eagle*-Decoding.

Input:

Parameter: (w_0, w_1).

Input: $c = [c[1], c[2], ..., c[L]], s_L$.

Output:

Output: m.

1: **for** $i = L; i \geqslant 1; i--$ **do**
2: Define x as a bit. $(x = 0\, or\, x = 1)$
3: 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 $s_x \oplus s_x^{+1} = s_i \oplus w_x$.
4: $m[i] \leftarrow x$
5: For the two possible s_x satisfy $s_x \oplus s_x^{+1} = s_i \oplus w_x$, we select the one which the last bit is equal to $c[i]$ as s_{i-1}.
6: **end for**

Now we give an example of the above algorithms.

[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 the example of Eagle decoding.

[Example 2] Eagle decoding

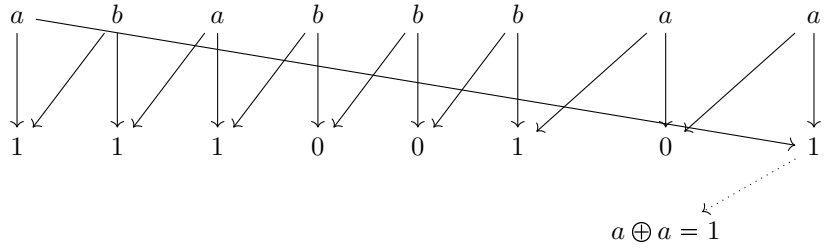
Known $w_0 = 10010011$, $w_1 = 11000111$, $s_8 = 01110110$, $c = 10111001$, how to compute m?

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 test.

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.

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

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

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 unpredictability of the encoded output.

For the convenience of the discussion in the following chapters, here we briefly analyze the effect of unpredictability 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 algorithm, given the parameters w_0 and w_1 that have L bits with odd bits different, 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.

Next we introduce two symbols $\mathbb{E}$ and $\mathbb{D}$.

$\mathbb{E}_{w_0,w_1}(m) = s_L$: use the key (w_0, w_1) to execute Eagle encoding algorithm (Algorithm 1) on any initial state s_0 and the given input m to obtain c and s_L, which s_0 and c will not be analyzed anymore.

$\mathbb{D}_{w_0,w_1}(s_L) = m$: use the key (w_0, w_1) to execute Eagle decoding algorithm (Algorithm 2) on any c and the given s_L to obtain s_0 and m.

$\mathbb{E}$ and $\mathbb{D}$ both represent a complete Eagle encoding process and Eagle decoding process, and their introduction is mainly for the convenience of deriving encryption algorithms later.

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

4 In depth analysis of Eagle encoding algorithm

Back to the Eagle encoding algorithm, we noticed that w_0 differs from w_1 with an odd number of bits, which is equivalent to the binary representation of $w_0 \oplus w_1$ containing an odd number of 1. There are 2^{2L-1} pairs of (w_0, w_1) that satisfy an odd number of different bits, This is because among all (w_0, w_1), w_0 and w_1 contain the same number of odd bits different as even bits different.

Next, let's take a closer look at the features of the Eagle encoding algorithm.

For the parameter (w_0, w_1), we have another completely equivalent representation (δ, w_0), where $\delta = w_0 \oplus w_1$, which means that (w_0, w_1) and (δ, w_0) can be equivalently transformed. Since the selection of s_0 is independent of s_L, without loss of generality, we set $s_0 = 0$, when given (δ, w_0), we only observe the relationship between the final state s_L and the input $m = b_1 b_2 ... b_L$, where $b_i \in \{0, 1\}$. We excute the Eagle encoding algorithm with the first bit to obtain

$$s_1 = w_{b_1} \oplus (s_0 \oplus s_0^{+1}) = w_0 \oplus g_1(b_1, \delta)$$

Where $g_1(b_1, \delta) = 0$ when $b_1 = 0$ and $g_1(b_1, \delta) = \delta$ when $b_1 = 1$. We excute the Eagle encoding algorithm with the second bit to obtain

$$\begin{aligned} s_2 &= w_{b_2} \oplus (s_1 \oplus s_1^{+1}) \\ &= (w_0 \oplus (w_0 \oplus w_0^{+1})) \oplus g_2(b_1, b_2, \delta) \\ &= w_0^{+1} \oplus g_2(b_1, b_2, \delta) \end{aligned}$$

Where $g_2(b_1, b_2, \delta)$ is a function about δ when b_1, b_2 is given. We excute the Eagle encoding algorithm with the third bit to obtain

$$\begin{aligned} s_3 &= w_{b_3} \oplus (s_2 \oplus s_2^{+1}) \\ &= (w_0 \oplus w_0^{+1} \oplus w_0^{+2}) \oplus g_3(b_1, b_2, b_3, \delta) \end{aligned}$$

So far we see that $s_i = h_i(w_0) \oplus g_i(b_1, b_2, ..., b_i, \delta)$, where $h_i(w_0)$ can be regarded as the state obtained by executing i bits Eagle encoding algorithm when $m = 0$, $g_i(b_1, b_2, ..., b_i, \delta)$ is a function about δ when $b_1, b_2, ..., b_i$ are given.

When $L = 2^u$, we set $m = 0, s_0 = 0$, after executing Eagle encoding, the final state is $s_L = w_0^{+(L-1)}$. When $m \neq 0$, after excuting Eagle encoding, the final state is

$$s_L = w_0^{+(L-1)} \oplus g_L(b_1, b_2, ..., b_L, \delta) \tag{4.1}$$

Where $g_L(b_1, b_2, ..., b_L, \delta)$ can also be expressed as $g_L(m, \delta)$. It is not difficult to see that $g_L(m, \delta)$ is a series of XOR operations with δ^{+k} given m, this is also equivalent to $g_L(m, \delta) = c_1\delta^{+1} \oplus c_2\delta^{+2} \oplus ... \oplus c_L\delta^{+L}$, where $c_i \in \{0, 1\}$ and $(c_1, c_2, ..., c_L)$ only determined by m. So (4.1) can ultimately be expressed as

$$s_L = w_0^{+(L-1)} \oplus g_L(m, \delta) = w_0^{+(L-1)} \oplus (c_1\delta^{+1} \oplus c_2\delta^{+2} \oplus ... \oplus c_L\delta^{+L}) \tag{4.2}$$

Furthermore, we observe several important facts, as shown in the following theorems.

[Theroem 3] $\forall m, s$, there exists 2^{L-1} different groups (w_0, w_1) satisfy that $\mathbb{E}_{w_0,w_1}(m) = s$. For any δ, there exists a unique w_0 that $\mathbb{E}_{w_0,\delta\oplus w_0}(m) = s$.

Proof. From (4.2), we know $\mathbb{E}_{w_0,w_1}(m) = w_0^{+(L-1)} \oplus g_L(m, \delta)$, through a simple transformation we obtain $w_0^{+(L-1)} = g_L(m, \delta) \oplus s$, it is obvious that for any δ, m, s, the right side of the equation is a difinite result, so there exists a unique

w_0 valid.

Additionally, due to the odd number of bits differ between w_0 and w_1, obviously $\delta = w_0 \oplus w_1$ have an odd number of bits that are 1, that is to say, there are 2^{L-1} different δ.

□

The conclusion of Theorem 3 indicates that, for any $\mathbb{E}_{w_0,w_1}(m) = s$, when only given (m, s), there are exponentially different (w_0, w_1) valid.

[Theorem 4] For any m_1, s_1, m_2, s_2, when the parity between m_1 and m_2 is different, there exists a unique (δ, w_0) that $\mathbb{E}_{w_0,\delta\oplus w_0}(m_1) = s_1$ and $\mathbb{E}_{w_0,\delta\oplus w_0}(m_2) = s_2$.

Proof. The state obtained by excuting the Eagle encoding algorithm with $L-1$ bits on m_1 is $s_{L-1,1} = h_{L-1,1}(w_0) \oplus g_{L-1,1}(m_1, \delta)$, the state obtained by excuting the Eagle encoding algorithm with $L-1$ bits on m_2 is $s_{L-1,2} = h_{L-1,2}(w_0) \oplus g_{L-1,2}(m_2, \delta)$.

Due to the difference in parity between m_1 and m_2, it can be assumed that the last bit of m_1 is 0 and the last bit of m_2 is 1. Further execute the Eagle encoding algorithm with one more bit separately to obtain

$$s_1 = s_{L,1} = w_0^{+(L-1)} \oplus (g_{L-1,1}(m_1, \delta) \oplus g_{L-1,1}(m_1, \delta)^{+1})$$
$$s_2 = s_{L,2} = w_0^{+(L-1)} \oplus \delta \oplus (g_{L-1,2}(m_2, \delta) \oplus g_{L-1,2}(m_2, \delta)^{+1})$$

Perform XOR operation on both sides of the above two equations simultaneously, set $(g_{L-1,1}(m_1, \delta) \oplus (g_{L-1,2}(m_2, \delta)$ as $g(m_1, m_2, \delta)$ to obtain

$$s_1 \oplus s_2 = \delta \oplus (g(m_1, m_2, \delta) \oplus g(m_1, m_2, \delta)^{+1})$$

Obviously, the left side of the equation is a definite number, on the right side of the equation is the result of performing XOR operation on an odd number of δ^{+k}, It is not difficult to verify that the solution to such an equation exists and is unique.

□

The conclusion of Theorem 4 indicates that, for any $\mathbb{E}_{w_0,w_1}(m_1) = s_1$ and $\mathbb{E}_{w_0,w_1}(m_2) = s_2$, when only given m_1, s_1, m_2, s_2, there is a 50% probability that a unique w_0, w_1 valid.

[Theorem 5] For any two groups m_1, m_2 and $m_1^{'}, m_2^{'}$, if $m_1 \oplus m_2 = m_1^{'} \oplus m_2^{'}$, excute the Eagle encoding algorithm as (4.2) on $m_1, m_2, m_1^{'}, m_2^{'}$ to obtain

$$\begin{aligned}
\mathbb{E}_{w_0,\delta\oplus w_0}(m_1) &= w_0^{+(L-1)} \oplus g_L(m_1,\delta)\\
\mathbb{E}_{w_0,\delta\oplus w_0}(m_2) &= w_0^{+(L-1)} \oplus g_L(m_2,\delta)\\
\mathbb{E}_{w_0,\delta\oplus w_0}(m_1^{'}) &= w_0^{+(L-1)} \oplus g_L(m_1^{'},\delta)\\
\mathbb{E}_{w_0,\delta\oplus w_0}(m_2^{'}) &= w_0^{+(L-1)} \oplus g_L(m_2^{'},\delta)
\end{aligned} \tag{4.3}$$

then $g_L(m_1,\delta) \oplus g_L(m_2,\delta) = g_L(m_1^{'},\delta) \oplus g_L(m_2^{'},\delta)$ holds.

Proof. Returning to the calculation process of (4.2), it is not difficult to find that

$$\mathbb{E}_{w_0,\delta\oplus w_0}(m \oplus \Delta m) = w_0^{+(L-1)} \oplus g_L(m,\delta) \oplus j(\delta,\Delta m)$$

Where $j(\delta,\Delta m)$ is a function only related to $\delta,\Delta m$. Without loss of generality we set $m_1 \oplus m_2 = m_1^{'} \oplus m_2^{'} = \Delta m$, (4.2) can be reorganized as

$$\begin{aligned}
\mathbb{E}_{w_0,\delta\oplus w_0}(m_1^{'}) &= w_0^{+(L-1)} \oplus g_L(m_1,\delta) \oplus j(\delta,\Delta m)\\
\mathbb{E}_{w_0,\delta\oplus w_0}(m_2^{'}) &= w_0^{+(L-1)} \oplus g_L(m_2,\delta) \oplus j(\delta,\Delta m)
\end{aligned} \tag{4.4}$$

By simple calculate we can obtain

$$\begin{aligned}
&g_L(m_1^{'},\delta) \oplus g_L(m_2^{'},\delta)\\
&= g_L(m_1,\delta) \oplus g_L(m_2,\delta) \oplus (w_0^{+(L-1)} \oplus w_0^{+(L-1)}) \oplus (j(\delta,\Delta m) \oplus j(\delta,\Delta m))\\
&= g_L(m_1,\delta) \oplus g_L(m_2,\delta)
\end{aligned}$$

□

Theorem 5 indicates that, the function $g_L(m,\delta)$ satisfies the distributive law for both variables m and δ.

[Theorem 6] For the function $g_L(m,\delta)$ in (4.2), fix any δ, where δ contains an odd number of bits that are 1. Fix any u, there exists a unique m that $g_L(m,\delta) = u$.

Proof. Fix any δ, set $w_0 \in \{0,1\}^L, w_1 = w_0 \oplus \delta$, excute Eagle encoding algorithm for m, we obtain $\mathbb{E}_{w_0,w_1}(m) = s$, then $u = s \oplus w_0^{+L}$, $g_L(m,\delta) = u$, thus $m \to u$ is monomorphism.

On the other hand, fix any δ, set $w_0 \in \{0,1\}^L, w_1 = w_0 \oplus \delta$, excute Eagle decoding algorithm for $s = u \oplus w_0^{+L}$, we obtain $\mathbb{D}_{w_0,w_1}(s) = m$, thus $u \to m$ is monomorphism.

Furthermore, since the ranges of m and u are the same, the mapping from m to s is a one-to-one corresponding when δ is fixed.

□

The proof process of Theorem 6 actually describes the process of $g_L(m,\delta) = u$ and its inverse function $g_L^{-1}(u,\delta)$.

4.1 The multiplication operation of binary

According to the conclusion of Theorem 5, the symbol $g_L(m,\delta)$ satisfies the distributive law for both m and δ, that is

$$g_L(m,\delta_1 \oplus \delta_2) = g_L(m,\delta_1) \oplus g_L(m,\delta_2)$$
$$g_L(m_1 \oplus m_2,\delta) = g_L(m_1,\delta) \oplus g_L(m_2,\delta)$$

If we consider the XOR operation $\oplus$ as an addition operation to binary, $g_L(m,\delta)$ can be seen as the multiplication operation between m and δ, as follows

$$g_L(m,\delta) = m * \delta$$

It is not difficult to verify that $m * \delta$ does not satisfy the commutative law or the associative law, so $m * \delta$ can only be seen as the right multiplication of δ by m. It is obvious that this representation is well-defined.

The calculation of $m * \delta$ is shown in Algorithm 3 as follows.

Algorithm 3 calculation of $m * \delta$.

Input:

Input: m, δ.

Output:

Output: $m * \delta$.

1: $w_0 \leftarrow 0$.
2: $w_1 \leftarrow \delta$.
3: $s_L \leftarrow \mathbb{E}_{w_0,w_1}(m)$.
4: **return** s_L.

Similarly, we can also introduce the division operator /, using the notation $S/\delta = R$ to represent $R * \delta = S$. The calculation for division is as follows.

Algorithm 4 calculation of s/δ.

Input:

Input: s, δ.

Output:

Output: s/δ.

1: $w_0 \leftarrow 0$.
2: $w_1 \leftarrow \delta$.
3: $m \leftarrow \mathbb{D}_{w_0,w_1}(s)$.
4: **return** m.

This paper does not intend to delve into the algebraic properties of the right multiplication operation mentioned above, but only simplifies the representation of $g_L(m, \delta)$ using $m * \delta$. Therefore $\mathbb{E}_{w_0,w_1}(m)$ can be expressed as $m * (w_0 \oplus w_1) \oplus w_0^{+(L-1)}$.

At the end of this section, we will provide the following conclusions.

[**Theorem 7**] (i) If m is an odd number, then $m * \delta$ contains odd number of bits that are 1.

(ii) If m is an even number, then $m*\delta$ contains even number of bits that are 1.

Proof. Firstly we prove (i).

By definition, we have $m * \delta = \mathbb{E}_{0,\delta}(m)$. Let s_{L-1} denote the result of the penultimate step of performing Eagle encoding on m, so we have $s_L = \delta \oplus (s_{L-1} \oplus s_{L-1}^{+1})$.

Obviously $(s_{L-1} \oplus s_{L-1}^{+1})$ contains even number of bits that are 1, δ contains odd number of bits that are 1, thus s_L contains odd number of bits that are 1.

Similarly, (ii) also holds.

□

[**Theorem 8**] If m is an odd number, s contains odd number of bits that are 1, then there exists an unique δ that $m * \delta = s$.

Proof. According to theorem 4, set $m_1 = m, m_2 = 0, s_1 = s, s_2 = 0$, because the parity between m_1 and m_2 is different, there exists an unique $(0, \delta)$ that $\mathbb{E}_{0,\delta}(m) = s$ which can be equivalently expressed as $m * \delta = s$.

□

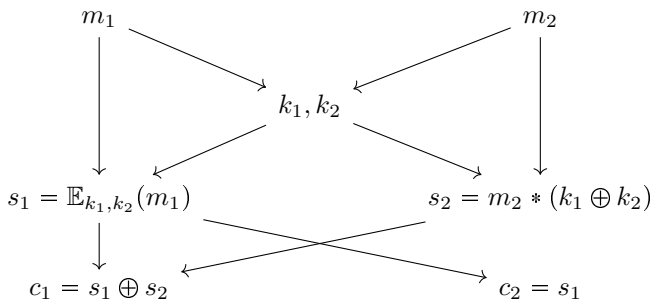
4.2 Introduction of a binary operation of two-dimensional vectors

We define a binary operation of two-dimensional vecotors $(m_1, m_2) \bigotimes (k_1, k_2) = (c_1, c_2)$ as follows.

$$\begin{aligned} c_2 &= \mathbb{E}_{k_1,k_2}(m_1) \\ c_1 &= c_2 \oplus (m_2 * (k_1 \oplus k_2)) \end{aligned} \tag{4.5}$$

Where k_1 contains even number of bits that are 1, k_2 contains odd number of bits that are 1, m_1, m_2 are both odd numbers, c_1 contains even number of bits that are 1, c_2 contains odd number of bits that are 1.

The above operation can be represented by the following figure.



It is not difficult to verify that the binary operation defined above satisfies the following conditions.

(a) Given any (m_1, m_2), the mapping $(k_1, k_2) \to (c_1, c_2)$ is a bijection.

(b) Given any (k_1, k_2), the mapping $(m_1, m_2) \to (c_1, c_2)$ is a bijection.

(c) Given any (c_1, c_2), the mapping $(k_1, k_2) \to (m_1, m_2)$ is a bijection.

(d) Given only m_1 and c_1, any (k_1, k_2) can be used to solve for an unique m_2 and c_2.

(e) Given only m_2 and c_2, any (k_1, k_2) can be used to solve for an unique m_1 and c_1.

(f) Given only m_2 and c_1, any (k_1, k_2) can be used to solve for an unique m_1 and c_2.

(g) Given only m_1 and c_2, any k_1 can be used to solve for an unique k_2, any k_2 can also be used to solve for an unique k_1, any $k_1 \oplus k_2$ can also be used to solve for an unique k_1 or k_2, and it is impossible to solve for m_2 or c_1 given any k_1 or k_2 or $k_1 \oplus k_2$.

(h) Given only m_1, c_1, c_2, any k_1 can be used to solve for an unique k_2, m_2, any k_2 can also be used to solve for an unique k_1, m_2, any $k_1 \oplus k_2$ can also be used to solve for an unique k_1 or k_2.

(i) Given only c_1, k_1, k_2, any m_2 can be used to solve for an unique c_2, any c_2 can also be used to solve for an unique m_2.

For condition (a), It is obvious that $(k_1, k_2) \to (c_1, c_2)$ is an injective, but it is not so obvious that $(c_1, c_2) \to (k_1, k_2)$ be also an injective. Since $c_1 \oplus c_2 = m_2 * (k_1 \oplus k_2)$, from Theorem 8, we can solve an unique $k_1 \oplus k_2$. By $c_2 = \mathbb{E}_{k_1,k_2}(m_1) = m_1 * (k_1 \oplus k_2) \oplus k_0^{+(L-1)}$, we can solve an unique k_1, thus the unique k_2 can be solved.

For other conditions (b)-(i), we can verify them in a similar way, we won't go into details here.

For the condition (i), As it will be used in the subsequent proof, we will describe it as a theorem here.

[Theorem 9] As (4.5) defined, the following conclusions hold.

(i) For any equation $(*, m_2) \bigotimes (k_1, k_2) = (c_1, *)$, where m_2 is the unknown variable to be solved, k_1, k_2, c_1 are known variables, this equation can be seen as an identity, valid for all m_2.

(ii) For any equation $(*, *) \bigotimes (k_1, k_2) = (c_1, c_2)$, where c_2 is the unknown

variable to be solved, k_1, k_2, c_1 are known variables, this equation can be seen as an identity, valid for all c_2.

If we construct an encryption algorithm where (k_1, k_2) is the key, m_1 is the plaintext, c_1 is the ciphertext, m_2 and c_2 never appear in the plaintext sequences or the ciphertext sequences, according to the condition (d), for a cracker, under the condition of only knowing the plaintext ciphertext correspondence m_1, c_1, any value (k_1, k_2) in the key space is valid and is indistinguishable, this feature allows us to construct the security that meets the requirements of computational complexity.

We summarize it as the following theorem.

[Theorem 10] As (4.5) defined, for any equation $(m_1, *) \bigotimes (k_1, k_2) = (c_1, *)$, which (k_1, k_2) are unknown variables and m_1, c_1 are known variables, this equation can be seen as an identity, valid for all k_1, k_2.

Proof. This theorem requires us to construct an algorithm that, given any (k_1, k_2) to find a valid equation $(m_1, *) \bigotimes (k_1, k_2) = (c_1, *)$.

When given any (k_1, k_2), we can solve c_2 and m_2 through the following method.

$$c_2 = \mathbb{E}_{k_1,k_2}(m_1)$$
$$m_2 = (c_1 \oplus c_2)/(k_1 \oplus k_2)$$

Additionally, due to known m_1, c_1, each m_2, c_2 can solve an unique k_1, k_2, * in $(m_1, *) \bigotimes (k_1, k_2) = (c_1, *)$ can take all values without distinction, thus the equation $(m_1, *) \bigotimes (k_1, k_2) = (c_1, *)$ can be seen as an identity equation to unknown variables k_1 and k_2.

□

5 Construction of one-way functions

5.1 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 vs 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 11**] If one-way function exists, then $P \neq NP$.

5.2 Construction of one-way functions

In this chapter, we constructed a short key symmetric encryption algorithm, which we named it Eagle. The key consists of 12 independent variables, each of which contains at least 2^{L-1} possible values. During the encryption process, several intermediate random numbers are generated at the encryption party, after the encryption process is completed, these intermediate random numbers are directly deleted at the encryption party, and only the ciphertext is sent to the decryption party. Under the condition of knowing the correspondence between any multiple sets of plaintext-ciphertext, the problem of cracking the key is equivalent to solving a system of equations about 12 variables together with some implicit variables. In solving the system of equations, we will prove that there is no universal algorithm without brute-force searching at least one variable, thus the computational complexity of cracking the key is at least $O(2^{L-1})$.

On the other hand, when the key known, both the encryption and decryption processes can be completed in linear time to the length of the key, meaning that the computational complexity for both encryption and decryption is $O(KL)$, where K is a constant.

Furthermore, since the decryption process under the condition of knowing the ciphertext and key is reversible for the problem of cracking the key under the condition of knowing the correspondence between plaintext and ciphertext, the decryption process can be completed in $O(KL)$, while the complexity of cracking the key is exponential, thus satisfying the condition of one-way function.

It should be noted that although the computational complexity of cracking the key is exponentially under the condition of knowing the correspondence between plaintext and ciphertext, this does not mean that this encryption algorithm is safe and practical. Assume the length of the key is $12L - 12$ bits, although the computational complexity of cracking the key is not less than $O(2^{L-1})$, may be much lower than the expected complexity of $O(2^{12L-12})$. A secure and practical encryption algorithm needs to ensure that the cracking key can only exhaustively search for every possible key which is the issue to be considered in the next chapter. In this chapter, our goal is to design a one-way

function where we only need to design and prove that the computational complexity for cracking keys is exponential to the length of the key.

Next, we will describe it in five sections. Section 5.2.1 introduces the key generation of the Eagle encryption algorithm, Section 5.2.2 introduces the process of the Eagle encryption algorithm, Section 5.2.3 introduces the process of the Eagle decryption algorithm, Section 5.2.4 analyzes the correctness and complexity of the Eagle encryption algorithm, and Section 5.2.5 provides the security analysis of the Eagle algorithm which give a construction of one-way function.

5.2.1 *Eagle* key generator

The key is (w_1, w_2), (w_3, w_4), (w_5, w_6), (w_7, w_8), (w_9, w_{10}) and (w_{11}, w_{12}) where w_1, w_3, w_5, w_7, w_9, w_{11} contains even number of bits that are 1, w_2, w_4, w_6, w_8, w_{10}, w_{12} contains odd number of bits that are 1, thus w_1 and w_2 have odd number of bits different, w_3 and w_4 have odd number of bits different, w_5 and w_6 have odd number of bits different, w_7 and w_8 have odd number of bits different, w_9 and w_{10} have odd number of bits different, w_{11} and w_{12} have odd number of bits different.

Assuming $w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}$ are all L bits, the length of the entire key can be considered as $12L - 12$ bits, this is because they have a loss of 12 bit.

5.2.2 *Eagle* encryption process

The plaintext P is grouped by $L-1$ bits, and the last group with less than $L-1$ bits are randomly filled into $L-1$ bits, then fill one bit with 1 to the last bit of each group to ensure the length of each group is L bits. The total number of groups is assumed to be K, denoted as $P = (P_1, P_2, ..., P_K)$. Obviously, P_i are all odd numbers.

Then generate random numbers of $18K + 3$ groups with each have L bits, denoted as $(t_1, t_2, ..., t_{6K}, t_{6K+1})$, $(x_1, x_2, ..., x_{6K}, x_{6K+1})$, $(y_1, y_2, ..., y_{6K}, y_{6K+1})$. where $x_i (1 \leqslant i \leqslant 6K + 1)$, $y_i (1 \leqslant i \leqslant 6K + 1)$, $t_i (1 \leqslant i \leqslant 6K + 1)$ are all odd numbers.

Generate another five random numbers with each have L bits denoted as μ_1, u_1, v_1, u_2 and v_2, where μ_1, u_1, v_1, u_2 and v_2 are all odd numbers.

Next, we need to define two simple functions $\phi(x), \psi(x)$ as follows.

Algorithm 5 calculation of $\phi(x)$.

Input:

Input: x.

Output:

Output: $\phi(x)$.

$y \leftarrow 0$.
$y[1] \leftarrow x[1]$
for $i = 2; i \leqslant L; i++$ **do**
  $y[i] = x[1] \oplus x[2] \oplus ... \oplus x[i]$
end for
return y.

Take the first bit of x as the first bit of $\phi(x)$, take the XOR of the first two bits of x as the second bit of $\phi(x)$, take the XOR of the first three bits of x as the third bit of $\phi(x)$, and so on, to obtain each bit of $\phi(x)$.

It is not difficult to find that $\phi(x)$ changes x to an odd number if x contains odd number of bits that are 1, and $\phi(x)$ changes x to an even number if x contains even number of bits that are 1.

Algorithm 6 calculation of $\psi(x)$.

Input:

Input: x.

Output:

Output: $\psi(x)$.

$y \leftarrow 0$.
$y[1] \leftarrow x[1]$.
for $i = 2; i \leqslant L; i++$ **do**
  $y[i] \leftarrow x[i-1] \oplus x[i]$
end for
return y.

$\psi(x)$ is the inverse function of $\phi(x)$, take the first bit of $\phi(x)$ as the first bit of x, take the XOR of the first bit and the second bit of $\phi(x)$ as the second bit of x, take the XOR of the second bit and the third bit of $\phi(x)$ as the third bit of x, and so on, to obtain each bit of x from $\phi(x)$.

It is not difficult to find that $\psi(x)$ changes x to a number containing odd number of bits that are 1 if x is an odd number, and $\psi(x)$ changes x to a number containing even number of bits that are 1 if x is an even number.

$\phi(x)$ and $\psi(x)$ are inverse functions of each other.

For convenience, we introduce another function $\theta(x)$, which defined as $\theta(x) = x^{+(L-1)}$, do left cycle shift of x by $L-1$ bits.

It is not difficult to verify that $\phi(x), \psi(x), \theta(x)$ are all linear functions, that is $\phi(x_1 \oplus x_2) = \phi(x_1) \oplus \phi(x_2)$, $\psi(x_1 \oplus x_2) = \psi(x_1) \oplus \psi(x_2)$, $\theta(x_1 \oplus x_2) = \theta(x_1) \oplus \theta(x_2)$ hold for any x_1 and x_2.

Additionally, we introduce an operator $\oplus 1$ to change the parity of a number, or changes a number to containing an even number of bits that are 1 if the number contains an odd number of bits that are 1, or changes a number to containing an odd number of bits that are 1 if the number contains an even number of bits that are 1.

The encryption process is described in the following algorithm.

Algorithm 7 *Eagle*-Encrypt.

Input:

Key: (w_1, w_2), (w_3, w_4), (w_5, w_6), (w_7, w_8), (w_9, w_{10}), (w_{11}, w_{12}).

Plaintext sequence: $P = (P_1, P_2, ..., P_K)$.

Random number sequence:

$(t_1, t_2, ..., t_{6K+1})$,
$(x_1, x_2, ..., x_{6K+1})$,
$(y_1, y_2, ..., y_{6K+1})$.

Random number: $\mu_1, u_1, v_1, u_2, v_2$.

Output:

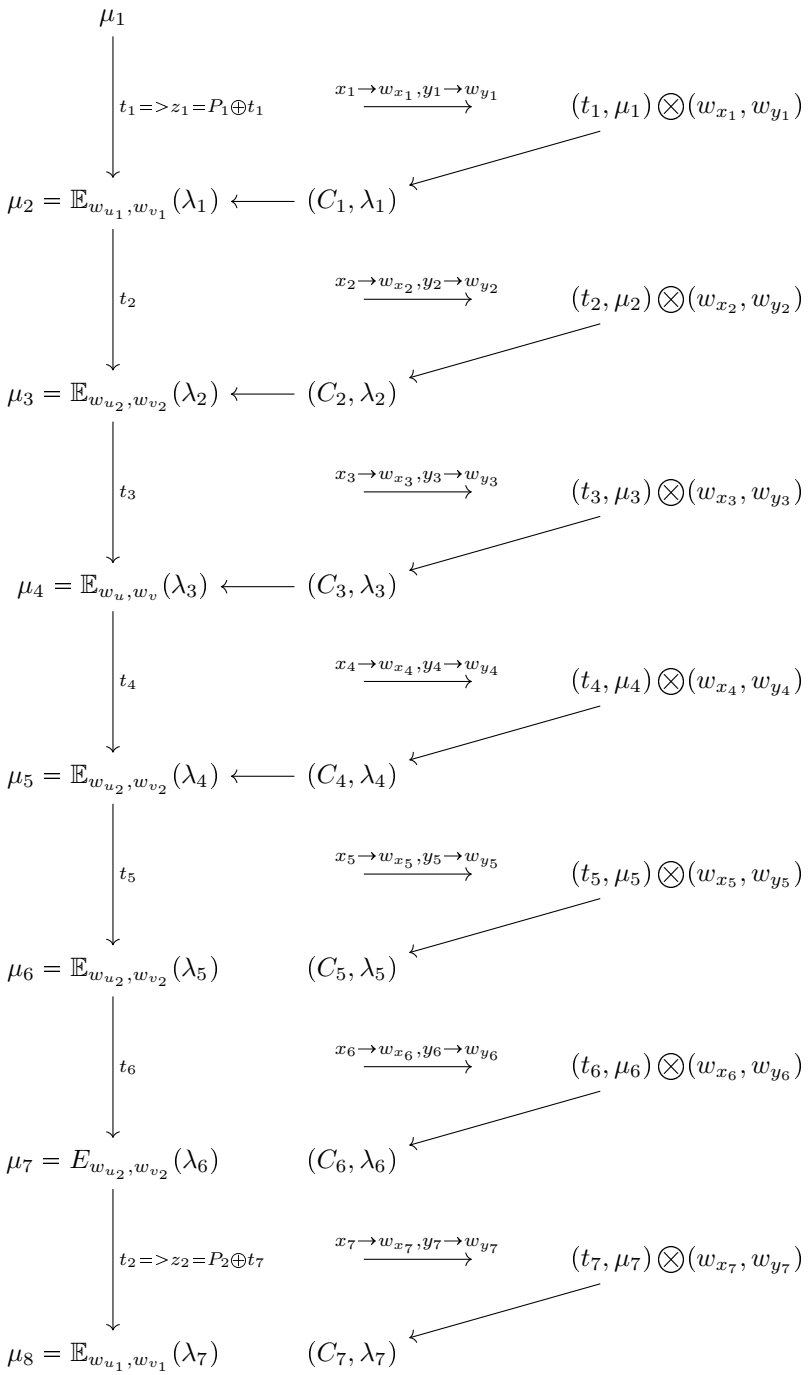
Ciphertext sequence:

$(C_1, C_2, ..., C_{6K+1})$,
$(x_1, x_2, ..., x_{6K+1})$,
$(y_1, y_2, ..., y_{6K+1})$,
$(z_1, z_2, ..., z_K)$,
$u_1, v_1, u_2, v_2, t_{6K+1}$.

$w_{u_1} \leftarrow \mathbb{E}_{w_9,w_{10}}(u_1) \oplus 1$
$w_{v_1} \leftarrow \mathbb{E}_{w_9,w_{10}}(v_1)$
$w_{u_2} \leftarrow \mathbb{E}_{w_{11},w_{12}}(u_2) \oplus 1$
$w_{v_2} \leftarrow \mathbb{E}_{w_{11},w_{12}}(v_2)$
for $j = 1; j \leqslant 6K + 1; j++$ **do**
  if $j \equiv 1 \mod 6$ **then**
    $w_{x_j} \leftarrow \mathbb{E}_{w_1,w_2}(x_j) \oplus 1$
    $w_{y_j} \leftarrow \mathbb{E}_{w_3,w_4}(y_j)$
  else
    $w_{x_j} \leftarrow \mathbb{E}_{w_5,w_6}(x_j) \oplus 1$
    $w_{y_j} \leftarrow \mathbb{E}_{w_7,w_8}(y_j)$
  end if
end for
for $i = 1; i \leqslant K; i++$ **do**
  $z_i \leftarrow P_i \oplus t_{6i-5}$
  $(C_{6i-5}, \lambda_{6i-5}) \leftarrow (t_{6i-5}, \mu_{6i-5}) \bigotimes (w_{x_{6i-5}}, w_{y_{6i-5}})$
  $\lambda_{6i-5} \leftarrow \phi(\lambda_{6i-5})$
  $\mu_{6i-4} \leftarrow \mathbb{E}_{w_{u_1},w_{v_1}}(\lambda_{6i-5})$
  $\mu_{6i-4} \leftarrow \phi(\mu_{6i-4})$
  $(C_{6i-4}, \lambda_{6i-4}) \leftarrow (t_{6i-4}, \mu_{6i-4}) \bigotimes (w_{x_{6i-4}}, w_{y_{6i-4}})$
  $\lambda_{6i-4} \leftarrow \phi(\lambda_{6i-4})$

$\mu_{6i-3} \leftarrow \mathbb{E}_{w_{u_2}, w_{v_2}}(\lambda_{6i-4})$
$\mu_{6i-3} \leftarrow \phi(\mu_{6i-3})$
$(C_{6i-3}, \lambda_{6i-3}) \leftarrow (t_{6i-3}, \mu_{6i-3}) \bigotimes (w_{x_{6i-3}}, w_{y_{6i-3}})$
$\lambda_{6i-3} \leftarrow \phi(\lambda_{6i-3})$
$\mu_{6i-2} \leftarrow \mathbb{E}_{w_{u_2}, w_{v_2}}(\lambda_{6i-3})$
$\mu_{6i-2} \leftarrow \phi(\mu_{6i-2})$
$(C_{6i-2}, \lambda_{6i-2}) \leftarrow (t_{6i-2}, \mu_{6i-2}) \bigotimes (w_{x_{6i-2}}, w_{y_{6i-2}})$
$\lambda_{6i-2} \leftarrow \phi(\lambda_{6i-2})$
$\mu_{6i-1} \leftarrow \mathbb{E}_{w_{u_2}, w_{v_2}}(\lambda_{6i-2})$
$\mu_{6i-1} \leftarrow \phi(\mu_{6i-1})$
$(C_{6i-1}, \lambda_{6i-1}) \leftarrow (t_{6i-1}, \mu_{6i-1}) \bigotimes (w_{x_{6i-1}}, w_{y_{6i-1}})$
$\lambda_{6i-1} \leftarrow \phi(\lambda_{6i-1})$
$\mu_{6i} \leftarrow \mathbb{E}_{w_{u_2}, w_{v_2}}(\lambda_{6i-1})$
$\mu_{6i} \leftarrow \phi(\mu_{6i})$
$(C_{6i}, \lambda_{6i}) \leftarrow (t_{6i}, \mu_{6i}) \bigotimes (w_{x_{6i}}, w_{y_{6i}})$
$\lambda_{6i} \leftarrow \phi(\lambda_{6i})$
$\mu_{6i+1} \leftarrow \mathbb{E}_{w_{u_2}, w_{v_2}}(\lambda_{6i})$
$\mu_{6i+1} \leftarrow \phi(\mu_{6i+1})$
end for
$(C_{6K+1}, \lambda_{6K+1}) \leftarrow (t_{6K+1}, \mu_{6K+1}) \bigotimes (w_{x_{6K+1}}, w_{y_{6K+1}})$

The above algorithm can be represented by the following figure.



It is not difficult to find that C_i all contain even number of bits that are 1. z_i are all even numbers.

Additionally, it should be noted that all intermediate variables as λ_i and μ_i do not appear in the ciphertext, but are directly deleted by the encrypting party after the encryption algorithm is executed.

5.2.3 *Eagle* decryption process

For the decrypting party, the key is also (w_1, w_2), (w_3, w_4), (w_5, w_6), (w_7, w_8), (w_9, w_{10}), (w_{11}, w_{12}), the ciphertext is $(C_1, C_2, ..., C_{6K+1})$, $(x_1, x_2, ..., x_{6K+1})$, $(y_1, y_2, ..., y_{6K+1})$, $(z_1, z_2, ..., z_K)$, $u_1, v_1, u_2, v_2, t_{6K+1}$. Please refer to the following for details.

Algorithm 8 *Eagle*-Decrypt.

Input:

Key: $(w_1, w_2), (w_3, w_4), (w_5, w_6), (w_7, w_8), (w_9, w_{10}), (w_{11}, w_{12})$.

Ciphertext sequence:

$(C_1, C_2, ..., C_{6K+1})$,
$(x_1, x_2, ..., x_{6K+1})$,
$(y_1, y_2, ..., y_{6K+1})$,
$(z_1, z_2, ..., z_K)$,
$u_1, v_1, u_2, v_2, t_{6K+1}$.

Output:

Plaintext sequence: $P = (P_1, P_2, ..., P_K)$.

$w_{u_1} \leftarrow \mathbb{E}_{w_9, w_{10}}(u_1) \oplus 1$
$w_{v_1} \leftarrow \mathbb{E}_{w_9, w_{10}}(v_1)$
$w_{u_2} \leftarrow \mathbb{E}_{w_{11}, w_{12}}(u_2) \oplus 1$
$w_{v_2} \leftarrow \mathbb{E}_{w_{11}, w_{12}}(v_2)$
for $j = 1; j \leqslant 6K + 1; j++$ **do**
  if $j \equiv 1 \mod 6$ **then**
    $w_{x_j} \leftarrow \mathbb{E}_{w_1, w_2}(x_j) \oplus 1$
    $w_{y_j} \leftarrow \mathbb{E}_{w_3, w_4}(y_j)$
  else
    $w_{x_j} \leftarrow \mathbb{E}_{w_5, w_6}(x_j) \oplus 1$
    $w_{y_j} \leftarrow \mathbb{E}_{w_7, w_8}(y_j)$
  end if
end for
$\lambda_{6K+1} \leftarrow \mathbb{E}_{w_{x_{6K+1}}, w_{y_{6K+1}}}(t_{6K+1})$
$\mu_{6K+1} \leftarrow (\lambda_{6K+1} \oplus C_{6K+1})/(w_{x_{6K+1}} \oplus w_{y_{6K+1}})$
$\mu_{6K+1} \leftarrow \psi(\mu_{6K+1})$
$\lambda_{6K} \leftarrow \mathbb{D}_{w_{u_2}, w_{v_2}}(\mu_{6K+1})$
$\lambda_{6K} \leftarrow \psi(\lambda_{6K})$
for $i = K; i \geqslant 1; i--$ **do**
  $\mu_{6i} \leftarrow (\lambda_{6i} \oplus C_{6i})/(w_{x_{6i}} \oplus w_{y_{6i}})$
  $\mu_{6i} \leftarrow \psi(\mu_{6i})$
  $\lambda_{6i-1} \leftarrow \mathbb{D}_{w_{u_2}, w_{v_2}}(\mu_{6i})$

$\lambda_{6i-1} \leftarrow \psi(\lambda_{6i-1})$
$\mu_{6i-1} \leftarrow (\lambda_{6i-1} \oplus C_{6i-1})/(w_{x_{6i-1}} \oplus w_{y_{6i-1}})$
$\mu_{6i-1} \leftarrow \psi(\mu_{6i-1})$
$\lambda_{6i-2} \leftarrow \mathbb{D}_{w_{u_2}, w_{v_2}}(\mu_{6i-1})$
$\lambda_{6i-2} \leftarrow \psi(\lambda_{6i-2})$
$\mu_{6i-2} \leftarrow (\lambda_{6i-2} \oplus C_{6i-2})/(w_{x_{6i-2}} \oplus w_{y_{6i-2}})$
$\mu_{6i-2} \leftarrow \psi(\mu_{6i-2})$
$\lambda_{6i-3} \leftarrow \mathbb{D}_{w_{u_2}, w_{v_2}}(\mu_{6i-2})$
$\lambda_{6i-3} \leftarrow \psi(\lambda_{6i-3})$
$\mu_{6i-3} \leftarrow (\lambda_{6i-3} \oplus C_{6i-3})/(w_{x_{6i-3}} \oplus w_{y_{6i-3}})$
$\mu_{6i-3} \leftarrow \psi(\mu_{6i-3})$
$\lambda_{6i-4} \leftarrow \mathbb{D}_{w_{u_2}, w_{v_2}}(\mu_{6i-3})$
$\lambda_{6i-4} \leftarrow \psi(\lambda_{6i-4})$
$\mu_{6i-4} \leftarrow (\lambda_{6i-4} \oplus C_{6i-4})/(w_{x_{6i-4}} \oplus w_{y_{6i-4}})$
$\mu_{6i-4} \leftarrow \psi(\mu_{6i-4})$
$\lambda_{6i-5} \leftarrow \mathbb{D}_{w_{u_1}, w_{v_1}}(\mu_{6i-4})$
$\lambda_{6i-5} \leftarrow \psi(\lambda_{6i-5})$
$t_{6i-5} \leftarrow \mathbb{D}_{w_{x_{6i-5}}, w_{y_{6i-5}}}(\lambda_{6i-5})$
$P_i \leftarrow z_i \oplus t_{6i-5}$
$\mu_{6i-5} \leftarrow (\lambda_{6i-5} \oplus C_{6i-5})/(w_{x_{6i-5}} \oplus w_{y_{6i-5}})$
$\mu_{6i-5} \leftarrow \psi(\mu_{6i-5})$
$\lambda_{6i-6} \leftarrow \mathbb{D}_{w_{u_2}, w_{v_2}}(\mu_{6i-5})$
$\lambda_{6i-6} \leftarrow \psi(\lambda_{6i-6})$
end for

5.2.4 Basic analysis of *Eagle*

It is obvious that *Eagle* encryption algorithm 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 parameters are randomly generated. In the encryption stage, the ciphertexts generated by the random parameters and the plaintexts together with the keys. After the encryption process is completed, some intermediate random parameters are directly deleted by the encryption party and do not appear in the ciphertexts.

It should also be noted that after being encrypted by the *Eagle* encryption algorithm, the length of the ciphertext is about 19 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 18 times the plaintext are generated, and then some random numbers are directly packaged into the ciphertexts, and some random numbers binded into the final ciphertexts.

The *Eagle* encryption algorithm performs bit by bit in encryption and decryption process, and the computational complexity of operation for each bit is

also a constant level. Therefore, the encryption and decryption complexity of the *Eagle* encryption algorithm can be regarded as $O(KL)$.

5.2.5 A construction of one-way function

When given the ciphertext (C_1, C_2, ..., C_{6K+1}), (x_1, x_2, ..., x_{6K+1}), (y_1, y_2, ..., y_{6K+1}), (z_1, z_2, ..., z_K), and $(u_1, v_1, u_2, v_2, t_{6K+1})$ of the *Eagle* encryption algorithm, we define the following function.

$$f(w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) = (P_1, P_2, ..., P_K) \tag{5.1}$$

where $(w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12})$ is the short key and (P_1, P_2, ..., P_K) is the plaintext decrypted by the *Eagle* decryption algorithm.

The inverse function of f can be defined as

$$\begin{aligned} f^{-1}(P_1, P_2, ..., P_K) = \{(w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) | \\ f(w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) = (P_1, P_2, ..., P_K)\} \end{aligned} \tag{5.2}$$

Solving f^{-1} is equivalent to finding the key (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) to match the plaintext $(P_1, P_2, ..., P_K)$.

In fact, if we can prove that we can't solve f^{-1} in polynomial time, that is, for any attacker who only knows the plaintext-ciphertext pairs, there is no effective method in polynomial time to find the correct key.

Here we need to clarify that we only need to prove that we can't solve w =(w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) in polynomial time, instead of relying only exhaustive search to find w. In fact,"only exhaustive search can be used to find w" is much stricter semantically than "unable to solve w in polynomial time". In this chapter, we only need to prove that w can't be solved in polynomial time, and this already satisfies the condition of one-way function.

Intuitively, we can't solve (w_1, w_2) when only (x_1, x_7, x_{13}, ..., x_{6K+1}) are given, we can't solve (w_3, w_4) when only (y_1, y_7, y_{13}, ..., y_{6K+1}) are given, we can't solve (w_5, w_6) when only (x_2, x_3, x_4, x_5, x_6, x_8, ..., x_{6K}) are given, we can't solve (w_7, w_8) when only (y_2, y_3, y_4, y_5, y_6, y_8, ..., y_{6K}) are given, we can't solve (w_9, w_{10}) when only u_1, v_1 are given, we can't solve w_{11}, w_{12} when only u_2, v_2 are given, when givenP_i, z_i, C_i, we can obtain the equation $(P_i \oplus z_i, *) \bigotimes (w_{x_i}, w_{y_i}) = (C_i, *)$, according to the theorem 10, this is an identity equation.

Based on the above analysis, given the correspondence between the plaintext $(P_1, P_2, ..., P_K)$ and ciphertext (C_1, C_2, ..., C_{6K+1}), (x_1, x_2,..., x_{6K+1}), (y_1, y_2, ..., y_{6K+1}), (z_1, z_2,..., z_K), $u_1, v_1, u_2, v_2, t_{6K+1}$ of the Eagle encryption algorithm, the problem of solving the key (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) is equivalent to the problem of solving the respective keys in six independent encryption algorithms under the condition that only their plaintext is known. The first encryption algorithm solves its key (w_9, w_{10}) under the condition that its plaintext (u_1, v_1) are known. For the second encryption

algorithm, the key (w_{11}, w_{12}) need to be solved under the condition that only the plaintext (u_2, v_2) is known. For the third encryption algorithm, the key (w_1, w_2) need to be solved under the condition that only the plaintext (x_1, x_7, ..., x_{6K+1}) is known. For the forth encryption algorithm, the key (w_3, w_4) need to be solved under the condition that only the plaintext (y_1, y_7, ..., y_{6K+1}) is known, etc.

In fact, for these six independent encryption algorithms, under the condition of only knowing their respective plaintext, every possible key cannot be reasonably excluded, and every possible key must be exhaustively searched. From the perspective of the entire Eagle encryption algorithm, at least one set of keys from (w_1, w_2) or (w_3, w_4) or (w_5, w_6) or (w_7, w_8) or (w_9, w_{10}) or (w_{11}, w_{12}) needs to be exhaustively searched to find a valid key. The computational complexity of exhausting either (w_1, w_2) or (w_3, w_4) or (w_5, w_6) or (w_7, w_8) or (w_9, w_{10}) or (w_{11}, w_{12}) is exponentially.

During the decryption process, all intermediate variables as λ_i, μ_i are unknown in the ciphertext, and these intermediate variables need to be gradually restored from the last group to the first group.

Without specifying any intermediate variables such as λ_i or μ_i, the problem of solving f^{-1} is equivalently transformed into solving a system of equations about the unknown variables as (w_1, w_2, w_3, w_4, w_5, w_6, w_7. w_8, w_9, w_{10}, w_{11}, w_{12}), any equation in this system of equations can be regarded as an identity equation about the unknown variables, and it is impossible to find an effective solution.

On the other hand, given (P_1, P_2, ... , P_K) and (C_1, C_2, ..., C_{6K+1}), (x_1, x_2,..., x_{6K+1}), (y_1, y_2, ..., y_{6K+1}), (z_1, z_2, ..., z_K), $u_1, v_1, u_2, v_2, t_{6K+1}$, every intermediate variable λ_i or μ_i can take any value, the computational complexity of traversing every intermediate variable is $O(2^{L-1})$.

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

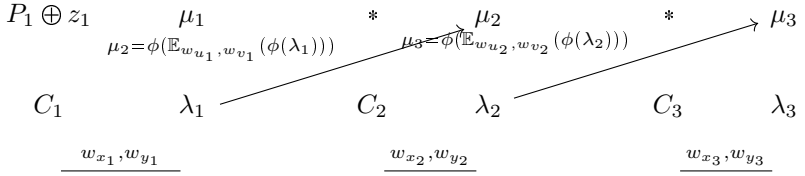
Let's give the following theorem.

[**Theorem 12**] When given the ciphertext (C_1, C_2, ..., C_{6K+1}), (x_1, x_2,..., x_{6K+1}), (y_1, y_2, ..., y_{6K+1}), (z_1, z_2, ..., z_K), $u_1, v_1, u_2, v_2, t_{6K+1}$ and the plaintext (P_1, P_2, ..., P_K) of the *Eagle* encryption algorithm, the computational complexity of the problem of cracking the key (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) is at least $O(2^{L-1})$.

Proof. Firstly, the problem of cracking keys (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) can be equivalently transformed into solving a system of equations.

By observing the structure of Algorithm 7 and Algorithm 8, when $i \equiv 1 \pmod 6$, the plaintext ciphertext pair of the i-th group satisfy the equation $(P_i \oplus z_i, \mu_i) \bigotimes (w_{x_i}, w_{y_i}) = (C_i, \lambda_i)$, where μ_i, λ_i are used to represent the intermediate variables generated by the encryption party and $w_{x_i} = \mathbb{E}_{w_1,w_2}(x_i)$ and $w_{y_i} = \mathbb{E}_{w_3,w_4}(y_i)$. When $i \not\equiv 1 \pmod 6$, the plaintext ciphertext pair of the i-th

group satisfy the equation $(*, \mu_i) \bigotimes (w_{x_i}, w_{y_i}) = (C_i, \lambda_i)$ and $w_{x_i} = \mathbb{E}_{w_5,w_6}(x_i)$ and $w_{y_i} = \mathbb{E}_{w_7,w_8}(y_i)$. The structure of the entire parameter can be represented as shown in the figure below.



Where $w_{x_i} = \mathbb{E}_{w_1,w_2}(x_i) \oplus 1$ and $w_{y_i} = \mathbb{E}_{w_3,w_4}(y_i)$ for $i \equiv 1 \mod 6$. $w_{x_i} = \mathbb{E}_{w_5,w_6}(x_i) \oplus 1$ and $w_{y_i} = \mathbb{E}_{w_7,w_8}(y_i)$ for $i \not\equiv 1 \mod 6$. $w_{u_1} = \mathbb{E}_{w_9,w_{10}}(u_1)$, $w_{v_1} = \mathbb{E}_{w_9,w_{10}}(v_1)$, $w_{u_2} = \mathbb{E}_{w_{11},w_{12}}(u_2)$, $w_{v_2} = \mathbb{E}_{w_{11},w_{12}}(v_2)$.

To simplify the analysis, we ignored two simple transformation functions $\phi(x)$ and $\psi(x)$, and also ingored the operation $\oplus 1$, without affecting the correctness of the conclusion, because $\phi(x)$ and $\psi(x)$ are two simple linear functions.

From the above, the problem of cracking the key (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) is equivalent to solving the following system of equations.

$$
\begin{cases}
(P_1 \oplus z_1, \mu_1) \bigotimes (\mathbb{E}_{w_1,w_2}(x_1), \mathbb{E}_{w_3,w_4}(y_1)) = (C_1, \lambda_1) \\
(*, \mu_2) \bigotimes (\mathbb{E}_{w_5,w_6}(x_2), \mathbb{E}_{w_7,w_8}(y_2)) = (C_2, \lambda_2) \\
(*, \mu_3) \bigotimes (\mathbb{E}_{w_5,w_6}(x_3), \mathbb{E}_{w_7,w_8}(y_3)) = (C_3, \lambda_3) \\
(*, \mu_4) \bigotimes (\mathbb{E}_{w_5,w_6}(x_4), \mathbb{E}_{w_7,w_8}(y_4)) = (C_4, \lambda_4) \\
(*, \mu_5) \bigotimes (\mathbb{E}_{w_5,w_6}(x_5), \mathbb{E}_{w_7,w_8}(y_5)) = (C_5, \lambda_5) \\
(*, \mu_6) \bigotimes (\mathbb{E}_{w_5,w_6}(x_6), \mathbb{E}_{w_7,w_8}(y_6)) = (C_6, \lambda_6) \\
(P_2 \oplus z_2, \mu_7) \bigotimes (\mathbb{E}_{w_1,w_2}(x_7), \mathbb{E}_{w_3,w_4}(y_7)) = (C_7, \lambda_7) \\
(*, \mu_8) \bigotimes (\mathbb{E}_{w_5,w_6}(x_8), \mathbb{E}_{w_7,w_8}(y_8)) = (C_8, \lambda_8) \\
\ldots\ldots \\
(P_K \oplus z_K, \mu_{6K-5}) \bigotimes (\mathbb{E}_{w_1,w_2}(x_{6K-5}), \mathbb{E}_{w_3,w_4}(y_{6K-5})) \\
\qquad = (C_{6K-5}, \lambda_{6K-5}) \\
(*, \mu_{6K-4}) \bigotimes (\mathbb{E}_{w_5,w_6}(x_{6K-4}), \mathbb{E}_{w_7,w_8}(y_{6K-4})) \\
\qquad = (C_{6K-4}, \lambda_{6K-4}) \\
\ldots\ldots \\
(*, \mu_{6K-1}) \bigotimes (\mathbb{E}_{w_5,w_6}(x_{6K-1}), \mathbb{E}_{w_7,w_8}(y_{6K-1})) \\
\qquad = (C_{6K-1}, \lambda_{6K-1}) \\
(*, \mu_{6K}) \bigotimes (\mathbb{E}_{w_5,w_6}(x_{6K}), \mathbb{E}_{w_7,w_8}(y_{6K})) \\
\qquad = (C_{6K}, \lambda_{6K}) \\
(t_{6K+1}, \mu_{6K+1}) \bigotimes (\mathbb{E}_{w_1,w_2}(x_{6K+1}), \mathbb{E}_{w_3,w_4}(y_{6K+1})) \\
\qquad = (C_{6K+1}, \lambda_{6K+1})
\end{cases}
\tag{5.3}
$$

$$
\begin{cases}
\mathbb{E}_{w_{u_1},w_{v_1}}(\lambda_1) = \mu_2 \\
\mathbb{E}_{w_{u_2},w_{v_2}}(\lambda_2) = \mu_3 \\
\mathbb{E}_{w_{u_2},w_{v_2}}(\lambda_3) = \mu_4 \\
\mathbb{E}_{w_{u_2},w_{v_2}}(\lambda_4) = \mu_5 \\
\mathbb{E}_{w_{u_2},w_{v_2}}(\lambda_5) = \mu_6 \\
\mathbb{E}_{w_{u_2},w_{v_2}}(\lambda_6) = \mu_7 \\
..... \\
\mathbb{E}_{w_{u_1},w_{v_1}}(\lambda_{6K-5}) = \mu_{6K-4} \\
\mathbb{E}_{w_{u_2},w_{v_2}}(\lambda_{6K-4}) = \mu_{6K-3} \\
\mathbb{E}_{w_{u_2},w_{v_2}}(\lambda_{6K-3}) = \mu_{6K-2} \\
\mathbb{E}_{w_{u_2},w_{v_2}}(\lambda_{6K-2}) = \mu_{6K-1} \\
\mathbb{E}_{w_{u_2},w_{v_2}}(\lambda_{6K-1}) = \mu_{6K} \\
\mathbb{E}_{w_{u_2},w_{v_2}}(\lambda_{6K}) = \mu_{6K+1}
\end{cases} \tag{5.4}
$$

$$
\begin{cases}
\mathbb{E}_{w_9,w_{10}}(u_1) = w_{u_1} \\
\mathbb{E}_{w_9,w_{10}}(v_1) = w_{v_1} \\
\mathbb{E}_{w_{11},w_{12}}(u_2) = w_{u_2} \\
\mathbb{E}_{w_{11},w_{12}}(v_2) = w_{v_2}
\end{cases} \tag{5.5}
$$

Where $(w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12})$ are unknown variables to be solved. (P_1, P_2, ..., P_K), (C_1, C_2, ..., C_{6K+1}), (x_1, x_2,..., x_{6K+1}), (y_1, y_2,..., y_{6K+1}), (z_1, z_2,..., z_K), $u_1, v_1, u_2, v_2, t_{6K+1}$ are known variables. (λ_1, λ_2, ..., λ_{6K+1}), (μ_1, μ_2, ..., μ_{6K+1}) are implicit variables determined by random numbers.

The proof process is divided into the following five steps.

(i) When $K > 12$, the system of equations has an unique solution or no solutions.

(ii) Instead of directly solving for the 12 unknowns variables (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}), we first consider a weaker problem, verifying the correctness of the five variables (w_1, w_2, w_3, w_4, w_9), this is equivalent to solving for the other seven variables (w_5, w_6, w_7, w_8, w_{10}, w_{11}, w_{12}) given specific values for (w_1, w_2, w_3, w_4, w_9). In the following, we will prove that the computational complexity of verifying the five variables (w_1, w_2, w_3, w_4, w_9) is at least $O(2^{L-1})$.

(iii) When $(w_1, w_2, w_3, w_4, w_9)$ given, to solve (w_5, w_6, w_7, w_8, w_{10}, w_{11}, w_{12}), after performing elimination on the implicit variables λ_i, μ_i, the system of equations (5.3) (5.4) (5.5) can be equivalently transformed into a system of general quintic equations about the unknown variables $w_{11} \oplus w_{12}$ with coefficient given by (w_5, w_6, w_7, w_8, w_{10}, w_{11}). According to the common conclusion that there are no algebraic solutions for quintic equations, $w_{11} \oplus w_{12}$ can't be represented by algebraic expressions with the variables (w_5, w_6, w_7, w_8, w_{10}, w_{11}), we cannot establish a system of algebraic equations only containing (w_5, w_6, w_7, w_8, w_{10}, w_{11}) without $w_{11} \oplus w_{12}$, thus we can't solve the variables

$(w_5, w_6, w_7, w_8, w_{10}, w_{11})$ without $w_{11} \oplus w_{12}$.

(iv) When (w_1, w_2, w_3, w_4, w_9) given, without solving for (w_5, w_6, w_7, w_8, w_{11}, w_{12}), we will prove that the problem of solving (w_{10}) is equivalent to the problem of solving the slope determined by $w_9 \oplus w_{10}$ of a line given only the x-coordinates of several points on it. Obviously there is no universal algorithm to solve it without brute-force searching any μ_i or $w_{11} \oplus w_{12}$.

(v) Together with (ii) (iii) and (iv), we will prove that the computational complexity of the problem of solving (5.3), (5.4), (5.5) is at least $O(2^{L-1})$.

Now we prove (i).

When $K > 12$, the number of constraints in the system of equations (5.3), (5.4), (5.5) exceeds the number of unknown variables, which means that the system of equations typically has only one solution or no solutions. This completes the proof of (i).

Now we prove (ii).

In this proof, we do not intend to directly discuss the computational complexity of solving (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}). We only need to discuss the computational complexity of a weaker problem, which is to verify the correctness of the five variables (w_1, w_2, w_3, w_4, w_9). As long as the computational complexity of verifying the correctness of the five variables $(w_1, w_2, w_3, w_4, w_9)$ is at least $O(2^{L-1})$, the computational complexity of solving (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) must not be lower than $O(2^{L-1})$. This will be explained in detail in (v).

From the equations (5.3) (5.4) (5.5), it is not difficult to see that the problem of verifying the correctness of $(w_1, w_2, w_3, w_4, w_9)$ is equivalent to solve the remaining seven variables $(w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12})$ when given a set of specific values for the four variables $(w_1, w_2, w_3, w_4, w_9)$.

Now we prove (iii).

Perform a simple variable substitution for $w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}$ as $\widetilde{\delta_5} = w_5 \oplus w_6$, $\widetilde{\delta_7} = w_7 \oplus w_8$, $\widetilde{\delta_9} = w_9 \oplus w_{10}$, $\widetilde{\delta_{11}} = w_{11} \oplus w_{12}$, $\widetilde{w_5} = \theta(w_5) = w_5^{+(L-1)}$, $\widetilde{w_7} = \theta(w_7) = w_7^{+(L-1)}$, $\widetilde{w_9} = \theta(w_9) = w_9^{+(L-1)}$, $\widetilde{w_{11}} = \theta(w_{11}) = w_{11}^{+(L-1)}$. Additionally we denote $\widetilde{P_i} = P_i \oplus z_i$. Then for any $i \not\equiv 1 \pmod 6$ we have

$$\begin{cases} w_{x_i} = \mathbb{E}_{w_5,w_6}(x_i) = x_i * \widetilde{\delta_5} \oplus \widetilde{w_5} \\ w_{y_i} = \mathbb{E}_{w_7,w_8}(y_i) = y_i * \widetilde{\delta_7} \oplus \widetilde{w_7} \\ w_{u_1} = \mathbb{E}_{w_9,w_{10}}(u_1) = u_1 * \widetilde{\delta_9} \oplus \widetilde{w_9} \\ w_{v_1} = \mathbb{E}_{w_9,w_{10}}(v_1) = v_1 * \widetilde{\delta_9} \oplus \widetilde{w_9} \\ w_{u_2} = \mathbb{E}_{w_{11},w_{12}}(u_2) = u_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}} \\ w_{v_2} = \mathbb{E}_{w_{11},w_{12}}(v_2) = v_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}} \end{cases} \tag{5.6}$$

When give (w_1, w_2, w_3, w_4), $w_{x_i} (i \equiv 1 \pmod 6)$ can be calculated by $w_{x_i} = \mathbb{E}_{w_1,w_2}(x_i)$ and $w_{y_i} (y \equiv 1 \pmod 6)$ can be calculated by $w_{y_i} = \mathbb{E}_{w_3,w_4}(y_i)$, then λ_1 can be directly calculated from the first equation of (5.3) as

$$\lambda_1 = \widetilde{P_1} * (w_{x_1} \oplus w_{y_1}) \oplus \theta(w_{x_1})$$

Since $\mu_2 = \mathbb{E}_{w_{u_1}, w_{v_1}}(\lambda_1)$ from the first equation of (5.4), μ_2 can be expressed as

$$\begin{aligned}\mu_2 &= \lambda_1 * (w_{u_1} \oplus w_{v_1}) \oplus \theta(w_{u_1}) \\ &= \lambda_1 * (u_1 * \widetilde{\delta_9} \oplus v_1 * \widetilde{\delta_9}) \oplus \theta(u_1 * \widetilde{\delta_9} \oplus \widetilde{w_9})\end{aligned}$$

Since $\lambda_2 = \mu_2 * (w_{x_2} \oplus w_{y_2}) \oplus C_2$ from the second equation of (5.3), λ_2 can be expressed as

$$\begin{aligned}\lambda_2 = (\lambda_1 * (u_1 * \widetilde{\delta_9} \oplus v_1 * \widetilde{\delta_9}) \oplus \theta(u_1 * \widetilde{\delta_9} \oplus \widetilde{w_9})) * (\mathbb{E}_{w_5,w_6}(x_2) \oplus \mathbb{E}_{w_7,w_8}(y_2)) \\ \oplus C_2\end{aligned}$$

Similar to the step-by-step process above, we can obtain the expressions for μ_3, λ_3, μ_4, λ_4, μ_5, λ_5, μ_6, λ_6, μ_7, λ_7 separately, which λ_7 is as follow

$$\begin{aligned}\lambda_7 = \\ ((((((((((\lambda_1 * (u_1 * \widetilde{\delta_9} \oplus v_1 * \widetilde{\delta_9}) \oplus \theta(u_1 * \widetilde{\delta_9} \oplus \widetilde{w_9})) * (\mathbb{E}_{w_5,w_6}(x_2) \oplus \mathbb{E}_{w_7,w_8}(y_2)) \\ \oplus C_2) * (u_2 * \widetilde{\delta_{11}} \oplus v_2 * \widetilde{\delta_{11}}) \oplus \theta(u_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}})) * (\mathbb{E}_{w_5,w_6}(x_3) \oplus \mathbb{E}_{w_7,w_8}(y_3)) \\ \oplus C_3) * (u_2 * \widetilde{\delta_{11}} \oplus v_2 * \widetilde{\delta_{11}}) \oplus \theta(u_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}})) * (\mathbb{E}_{w_5,w_6}(x_4) \oplus \mathbb{E}_{w_7,w_8}(y_4)) \\ \oplus C_4) * (u_2 * \widetilde{\delta_{11}} \oplus v_2 * \widetilde{\delta_{11}}) \oplus \theta(u_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}})) * (\mathbb{E}_{w_5,w_6}(x_5) \oplus \mathbb{E}_{w_7,w_8}(y_5)) \\ \oplus C_5) * (u_2 * \widetilde{\delta_{11}} \oplus v_2 * \widetilde{\delta_{11}}) \oplus \theta(u_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}})) * (\mathbb{E}_{w_5,w_6}(x_6) \oplus \mathbb{E}_{w_7,w_8}(y_6)) \\ \oplus C_6) * (u_2 * \widetilde{\delta_{11}} \oplus v_2 * \widetilde{\delta_{11}}) \oplus \theta(u_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}})) * (w_{x_7} \oplus w_{y_7}) \\ \oplus C_7\end{aligned} \tag{5.7}$$

On the other hand, λ_7 can be directly calculated as

$$\lambda_7 = \widetilde{P_2} * (w_{x_7} \oplus w_{y_7}) \oplus \theta(w_{x_7}) \tag{5.8}$$

Together with (5.7) and (5.8), we obtain the following equation

$$\begin{aligned}((((((((((\lambda_1 * (u_1 * \widetilde{\delta_9} \oplus v_1 * \widetilde{\delta_9}) \oplus \theta(u_1 * \widetilde{\delta_9} \oplus \widetilde{w_9})) * (\mathbb{E}_{w_5,w_6}(x_2) \oplus \mathbb{E}_{w_7,w_8}(y_2)) \\ \oplus C_2) * (u_2 * \widetilde{\delta_{11}} \oplus v_2 * \widetilde{\delta_{11}}) \oplus \theta(u_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}})) * (\mathbb{E}_{w_5,w_6}(x_3) \oplus \mathbb{E}_{w_7,w_8}(y_3)) \\ \oplus C_3) * (u_2 * \widetilde{\delta_{11}} \oplus v_2 * \widetilde{\delta_{11}}) \oplus \theta(u_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}})) * (\mathbb{E}_{w_5,w_6}(x_4) \oplus \mathbb{E}_{w_7,w_8}(y_4)) \\ \oplus C_4) * (u_2 * \widetilde{\delta_{11}} \oplus v_2 * \widetilde{\delta_{11}}) \oplus \theta(u_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}})) * (\mathbb{E}_{w_5,w_6}(x_5) \oplus \mathbb{E}_{w_7,w_8}(y_5)) \\ \oplus C_5) * (u_2 * \widetilde{\delta_{11}} \oplus v_2 * \widetilde{\delta_{11}}) \oplus \theta(u_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}})) * (\mathbb{E}_{w_5,w_6}(x_6) \oplus \mathbb{E}_{w_7,w_8}(y_6)) \\ \oplus C_6) * (u_2 * \widetilde{\delta_{11}} \oplus v_2 * \widetilde{\delta_{11}}) \oplus \theta(u_2 * \widetilde{\delta_{11}} \oplus \widetilde{w_{11}})) * (w_{x_7} \oplus w_{y_7}) \\ \oplus C_7 \\ = \widetilde{P_2} * (w_{x_7} \oplus w_{y_7}) \oplus \theta(w_{x_7})\end{aligned} \tag{5.9}$$

Where $\widetilde{\delta_9}, \widetilde{\delta_{11}}, \widetilde{w_{11}}, w_5, w_6, w_7, w_8$ are unknown variables, others are known variables.

It is not difficult to find that (5.9) is a general quintic equation about the variable $\widetilde{\delta_{11}}$ with coefficient as given by $(\widetilde{\delta 9}, \widetilde{w_{11}}, w_5, w_6, w_7, w_8)$. This is because when (5.9) is expanded into an expression connected by $\oplus$, the continuous right multiplication operations on $\widetilde{\delta_{11}}$ are executed at most five times.

In fact, the above explanation is reasonable. This is because the $\oplus$ can be viewed as traditional $+$, the right multiplication can be viewed as traditional $\times$, the repeated right multiplication of the same parameter can be viewed as a power operation with respect to that parameter. Similarly, radical operations based on right multiplication can also be well defined. Thus, the conclusion that the general quintic equation has no algebraic solutions can be entirely extended to the general quintic equation concerning right multiplication. For more details, please refer to Appendix A.

Similar to the elimination process in (5.9), after performing elimination on λ_{6i+1}, μ_{6i+2}, λ_{6i+2}, μ_{6i+3}, λ_{6i+3}, μ_{6i+4}, λ_{6i+4}, μ_{6i+5}, λ_{6i+5}, μ_{6i+6}, λ_{6i+6}, μ_{6i+7}, λ_{6i+7}, more general quintic equaions about the variables $\widetilde{\delta_{11}}$ can be established with coefficient as given by $(\widetilde{\delta 9}, \widetilde{w_{11}}, w_5, w_6, w_7, w_8)$.

That is to say, after performing elimination operations on all implicit variables λ_i, μ_i, the system of equations (5.3) (5.4) (5.5) are equivalently transformed into some quintic equations about the variable $\widetilde{\delta_{11}}$ with coefficient as given by $(\widetilde{\delta 9}, \widetilde{w_{11}}, w_5, w_6, w_7, w_8)$.

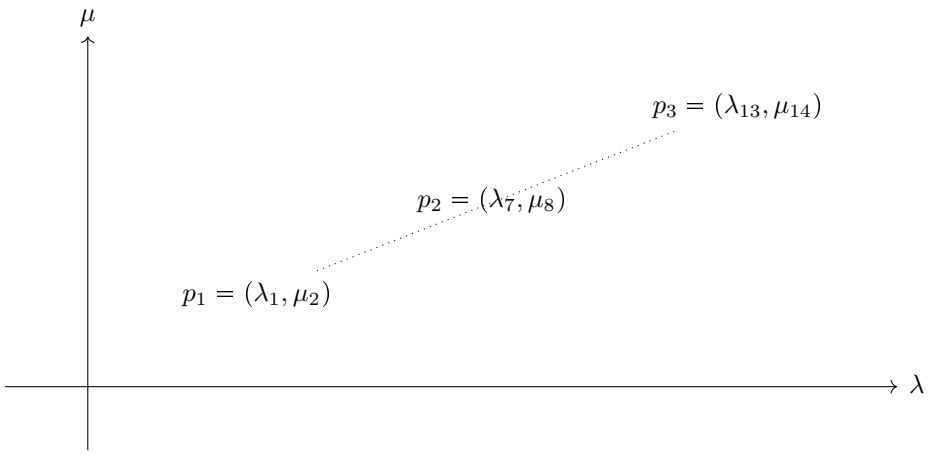
According to the common conclusion that there are no algebraic solutions for quintic equations, $\widetilde{\delta_{11}} = w_{11} \oplus w_{12}$ can't be represented by any algebraic expression with the six variables $(\widetilde{\delta 9}, \widetilde{w_{11}}, w_5, w_6, w_7, w_8)$, thus we can't establish a system of equations only containing $(\widetilde{\delta 9}, \widetilde{w_{11}}, w_5, w_6, w_7, w_8)$ without $\widetilde{\delta_{11}}$. This completes the proof of (iii).

It is noteworthy that the aforementioned quintic equations not only lacks an algebraic solution but also cannot be solved by any approximate numerical method. This is because, based on the right multiplication operation constructed in this paper, it is impossible to establish an "order" relation $\prec$, such that for any $\delta_1 \prec \delta_2$, $m * \delta_1 \prec m * \delta_2$ holds for all m, please refer to Appendix A for details of the proof. Thus it becomes impossible to establish any numerical simulation on the basis of this right multiplication operation.

Now we prove (iv).

When given $(w_1, w_2, w_3, w_4, w_9)$, to solve $\widetilde{\delta_9} = w_9 \oplus w_{10}$, we need to analyze the implicit variables λ_i, μ_i. From (5.5), since $w_{u_1} = \mathbb{E}_{w_9, w_{10}}(u_1)$ and $w_{v_2} = \mathbb{E}_{w_9, w_{10}}(v_2)$, the problem of solving $\widetilde{\delta_9}$ is equivalent to the problem of solving (w_{u_1}, w_{v_1}).

Perform a simple variable substitution for w_{u_1}, w_{v_1} as $\widetilde{u} = w_{u_1} \oplus w_{v_1}$ and $\widetilde{v} = w_{u_1}^{+(L-1)}$, $(\lambda_i, \mu_{i+1})(i \equiv 1 \mod 6)$ is a set of points that satisfy the linear equation $\mu = \lambda * \widetilde{u} \oplus \widetilde{v}$, where $\widetilde{u}, \widetilde{v}$ are parameters, the system of equations (5.4) can be represented by the following figure.



In fact, the straight line in the above figure does not represent a true straight line, it is just an expression of a linear relationship between μ and λ.

When given (w_1, w_2, w_3, w_4), we can calculate λ_{6i+1} and μ_{6i+1} $(1 \leqslant i \leqslant K)$ from the $6i+1$-th equation in (5.3) directly.

According to the (iii), because $(\widetilde{\delta 9}, \widetilde{w_{11}}, w_5, w_6, w_7, w_8)$ can't be solved , must be expressed with at least one free variable $\widetilde{\delta_{11}}$, thus μ_2, μ_8, ..., μ_{6K+2} can't be solved without brute-force searching for $\widetilde{\delta_{11}}$.

The problem of solving $\widetilde{\delta_9}$ when given $(w_1, w_2, w_3, w_4, w_9)$ is equivalent to the problem of solving $(\widetilde{u})$ when only λ_{6i+1} $(1 \leqslant i \leqslant K)$ known, where λ_{6i+1} is the horizontal coordinate of the i-th points of the linear equation $\mu = \lambda * \widetilde{u} \oplus \widetilde{v}$.

Obviously, there is no universal algorithm to solve $\widetilde{u} = (u_1 \oplus v_1) * (w_9 \oplus w_{10})$ without brute-force searching any μ_{6i+2} or $\widetilde{\delta_{11}}$.

This completes the proof of (iv).

Now we prove (v).

Returning to the problem of verifying $(w_1, w_2, w_3, w_4, w_9)$, we have proved that there is no universal algorithm except for brute-force searching at least one variable.

Since the computational complexity of brute-force searching for $\widetilde{\delta_{11}} = w_{11} \oplus w_{12}$ or μ_{6i+2} is $O(2^{L-1})$, thus the complexity of verifying $(w_1, w_2, w_3, w_4, w_9)$ is at least $O(2^{L-1})$.

Obviously, if we solved $(w_1,\ w_2,\ w_3,\ w_4,\ w_5,\ w_6,\ w_7,\ w_8,\ w_9,\ w_{10},\ w_{11},\ w_{12})$, we can immediately solve $(w_1,\ w_2,\ w_3,\ w_4,\ w_9)$, thus the computational complexity of solving $(w_1,\ w_2,\ w_3,\ w_4,\ w_5,\ w_6,\ w_7,\ w_8,\ w_9,\ w_{10},\ w_{11},\ w_{12})$ is not lower than the computational complexity of solving $(w_1,\ w_2,\ w_3,\ w_4,\ w_9)$.

Since the complexity of verifying a solution should not be higher than solving it, the computational complexity of solving $(w_1,\ w_2,\ w_3,\ w_4,\ w_9)$ is not lower than the computational complexity of verifying $(w_1,\ w_2,\ w_3,\ w_4,\ w_9)$. So, the computational complexity of solving the system of equations satisfies the following relationship

$$\begin{aligned} &Complexity(Solve(w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12})) \\ &\geqslant Complexity(Solve(w_1, w_2, w_3, w_4, w_9)) \\ &\geqslant Complexity(Verify(w_1, w_2, w_3, w_4, w_9)) \\ &\geqslant O(2^{L-1}) \end{aligned}$$

In summary, in the process of solving the system of equations, the computational complexity of solving (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) is at least $O(2^{L-1})$.

□

According to the conclusion of Theorem 12, the computational complexity of solving f^{-1} is exponential.

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, we can summarize it to the following theorem.

[**Theorem 13**] The function f defined in (5.1) is a one-way function.

According to Theorem 11 and Theorem 13, we conclude that P≠NP.

6 How to achieve only exhaustive search to crack keys

Without causing ambiguity, we denote the operation of performing two right multiplications by δ on R as $R * \delta^2$.

Similarly, we can use S/δ^2 to represent performing two division operations on S with respect to δ.

Now we give the two algorithms $m * \delta^2$ and s/δ^2 as follows

Algorithm 9 calculation of $m * \delta^2$.

Input:

Input: m, δ.

Output:

Output: $m * \delta^2$.

1: $m \leftarrow \phi(m)$.
2: $s \leftarrow m * \delta$.
3: $s \leftarrow \phi(s)$.
4: $s \leftarrow s * \delta$.
5: **return** s.

Where m and δ both contain odd number of bits that are 1, the output s also contains odd number of bits that are 1.

Algorithm 10 calculation of s/δ^2.

Input:

Input: s, δ.

Output:

Output: s/δ^2.

1: $m \leftarrow s/\delta$.
2: $m \leftarrow \psi(m)$.
3: $m \leftarrow m/\delta$.
4: $m \leftarrow \psi(m)$.
5: **return** m.

Where s and δ both contain odd number of bits that are 1, the output m also contains odd number of bits that are 1.

6.1 Design of encryption algorithm for only exhaustive method to crack the key

Although we found an encryption algorithm in Chapter 5 that, given the known correspondence between plaintext and ciphertext, the computational complexity for cracking the key is at least exponentially to the length of key as $O(2^{L-1})$, this does not mean that this algorithm is practical. Because the length of the key (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}) is $12L-12$ bits, theoretically, the computational complexity for cracking the key must reach $O(2^{12L-12})$ to recognize its secure.

In this chapter, we utilized the Eagle encoding algorithm and its related features to design another short key symmetric block encryption algorithm, which we named the *Eagle** encryption algorithm. The main feature of this algorithm is that it introduce a nonlinear operation, together with the random numbers as designed in Eagle encryption algorithm.

After being encrypted by this encryption algorithm, the problem of cracking the keys given that all plaintext ciphertext correspondences are known, is equivalent to solving a system of nonlinear equations.

In addition, the key is consists of three variables, for verifying any two variables in them needs to solving a system of quintic equations, according to the quintic equation with no algebraic solution, we assert that there is no fast algorithm to verify any two variables, and thus verifying any two variables requires exhaustive enumeration of the other variable. So to crack the key, it is necessary to exhaustively search for all values in these three variables.

The entire *Eagle** encryption algorithm is divided into three parts: key generation, encryption algorithm, and decryption algorithm. Next, we will provide a detailed description.

6.1.1 *Eagle** key generator

The key is divided into three independent variables, denoted as s_x, s_y, s_t, where s_x, s_y, s_t all contain odd number of bits that are 1.

Assuming s_x, s_y, s_t are all L bits, the length of the entire key can be considered as $3L - 3$ bits, this is because each variable has a loss of 1 bit.

6.1.2 *Eagle** encryption algorithm

The plaintext P is grouped by $L-1$ bits, and the last group with less than $L-1$ bits are randomly filled into $L-1$ bits, then fill one bit with 1 to the last bit of each group to ensure the length of each group is L bits. The total number of groups is assumed to be K, denoted as $P = (P_1, P_2, ..., P_K)$. Obviously, P_i are all odd numbers.

We randomly generate $6K+3$ groups number with each have L bits, denoted as $\delta x = (\delta x_1, \delta x_2, ..., \delta x_{2K+1})$, $\delta y = (\delta y_1, \delta y_2, ..., \delta y_{2K+1})$, $\delta t = (\delta t_1, \delta t_2, ..., \delta t_{2K+1})$, where δx_i contains odd number of bits that are 1, δy_i contains odd number of bits that are 1, δt_i contains odd number of bits that are 1.

Then we randomly generate three number as $\delta u, \delta v, M_0$, where $\delta u, \delta v, M_0$ all contain odd number of bits that are 1.

The encryption process is detailed in Algorithm 11 below.

Algorithm 11 *Eagle**-Encrypt.

Input:

Keys: (s_x, s_y, s_t).

Plaintext sequences: $P = (P_1, P_2, ..., P_K)$.

Random number sequences:

$\delta x = (\delta x_1, \delta x_2, ..., \delta x_{2K+1})$,

$\delta y = (\delta y_1, \delta y_2, ..., \delta y_{2K+1})$.

$\delta t = (\delta t_1, \delta t_2, ..., \delta t_{2K+1})$.

Random number: $\delta u, \delta v, M_0$.

Output:

Ciphertext sequences:

$(\delta x_1, \delta x_2, ..., \delta x_{2K+1})$,

$(\delta y_1, \delta y_2, ..., \delta y_{2K+1})$,

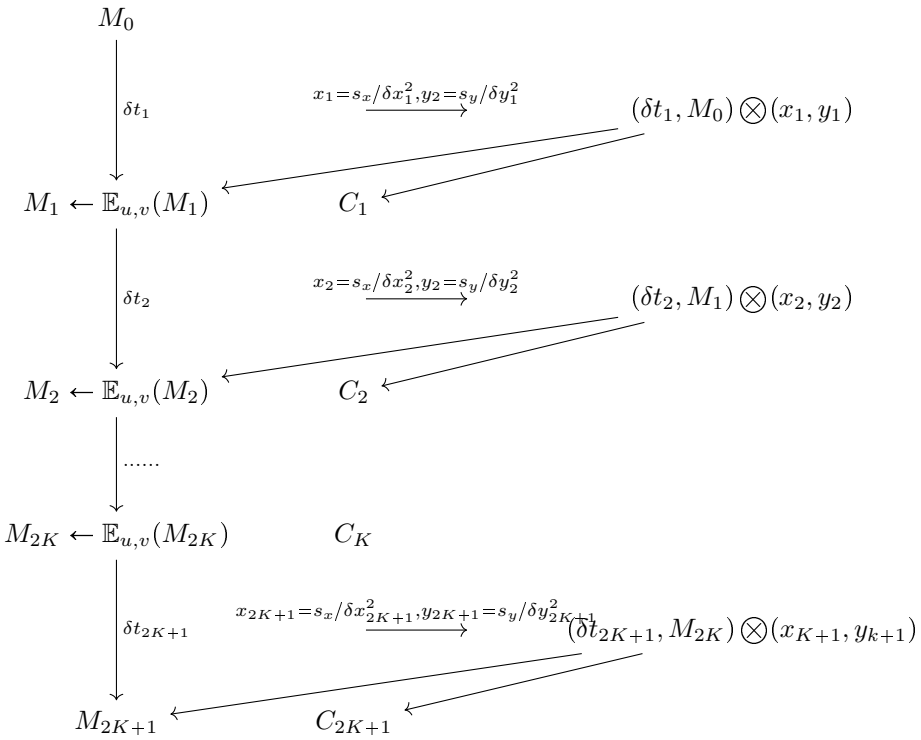
$(C_1, C_2, ..., C_{2K+1})$,

$(z_1, z_2, ..., z_K)$,

$\delta u, \delta v, \delta t_{2K+1}$

1: $u \leftarrow (s_x \oplus s_y \oplus 1)/\delta_u^2$.
2: $v \leftarrow s_t/\delta_v^2$.
3: **for** $i = 1; i \leqslant 2K + 1; i + +$ **do**
4: $\quad x_i \leftarrow s_x/\delta x_i^2$.
5: $\quad y_i \leftarrow s_y/\delta y_i^2$.
6: **end for**
7: **for** $j = 1; j \leqslant K; j + +$ **do**
8: $\quad z_j \leftarrow P_j \oplus (s_t/\delta t_{2j-1}^2)$.
9: **end for**
10: $M \leftarrow M_0$
11: $C \leftarrow 0$
12: **for** $k = 1; k \leqslant 2K + 1; k + +$ **do**
13: $\quad (C_k, C) \leftarrow (\phi(\delta t_k), \phi(M)) \bigotimes (x_k \oplus 1, y_k)$
14: $\quad C \leftarrow \phi(C)$
15: $\quad M \leftarrow \mathbb{E}_{u\oplus 1,v}(C)$
16: **end for**

From the above algorithm, it can be seen that the *Eagle** encryption algorithm is divided into two independent parts, as shown in the following figure, which ignored two functions $\phi(x), \psi(x)$ and the operation $\oplus 1$ in the figure.



As shown from the above figure, the ciphertext length is about three times the plaintext length.

6.1.3 *Eagle** decryption algorithm

For *Eagle** decryptors, the ciphertext sequences are $C = (C_1, C_2, ..., C_{2K+1}, \delta x_1, \delta x_2, ..., \delta x_{2K+1}, \delta y_1, \delta y_2, ..., \delta y_{2K+1}, z_1, z_2, ..., z_K)$, δu, δv and δt_{2K+1}. Here, the ciphertext sequences can be divided into three independent parts (C_1, C_2, ..., C_{2K+1}, z_1, z_2, ..., z_K, δt_{2K+1}) and (δx_1, δx_2, ..., δx_{2K+1}) and (δy_1, δy_2, ..., δy_{2K+1}), First we use (s_x) and (δx_1, δx_2, ..., δx_{2K+1}) to calculate the random number sequences $x = (x_1, x_2, ..., x_{2K+1})$, then we use ($s_y$) and ($\delta y_1$, δy_2, ..., δy_{2K+1}) to calculate the random number sequences $y = (y_1, y_2, ..., y_{2K+1})$, then we use the random number $u = (s_x \oplus s_y \oplus 1)/\delta_u^2$, $v = s_t/\delta_v^2$ and the ciphertexts (C_1, C_2, ..., C_{2K+1}, $z_1, z_2, ..., z_K, \delta t_{2K+1}$) and the random number sequences $x = (x_1, x_2, ..., x_{2K+1})$ and $y = (y_1, y_2, ..., y_{2K+1})$ to decrypt $P = (P_1, P_2, ..., P_K)$. The specific process is detailed in the following algorithm.

Algorithm 12 *Eagle**-Decrypt.

Input:

Key: (s_x, s_y, s_t).

Ciphertext sequences:

$(\delta x_1, \delta x_2, ..., \delta x_{2K+1})$,
$(\delta y_1, \delta y_2, ..., \delta y_{2K+1})$,
$(C_1, C_2, ..., C_{2K+1})$,
$(z_1, z_2, ..., z_K)$,
$\delta u, \delta v, \delta t_{2K+1}$.

Output:

Plaintext sequences: $P = (P_1, P_2, ..., P_K)$.

$u \leftarrow (s_x \oplus s_y \oplus 1)/\delta_u^2$.
$v \leftarrow s_t/\delta_v^2$.
for $i = 1; i \leqslant 2K+1; i++$ **do**
  $x_i \leftarrow s_x/\delta x_i^2$.
  $y_i \leftarrow s_y/\delta y_i^2$.
end for
$M \leftarrow 0, C \leftarrow 0$
$C \leftarrow \mathbb{E}_{x_{2K+1} \oplus 1, y_{2K+1}}(\phi(\delta t_{2K+1}))$.
$M \leftarrow (C_{2K+1} \oplus C)/(x_{2K+1} \oplus y_{2K+1} \oplus 1)$
for $j = 2K; j \geqslant 1; j--$ **do**
  $M \leftarrow \psi(M)$
  $C \leftarrow \mathbb{D}_{u \oplus 1, v}(M)$
  $C \leftarrow \psi(C)$
  $\delta t_j \leftarrow \mathbb{D}_{x_j \oplus 1, y_j}(C)$
  $\delta t_j \leftarrow \psi(\delta t_j)$.
  $M \leftarrow (C_j \oplus C)/(x_j \oplus y_j \oplus 1)$.
end for
for $k = 1; k \leqslant K; k++$ **do**
  $P_k \leftarrow z_k \oplus (s_t/\delta t_{2k-1}^2)$.
end for

From the processes of Algorithm 11 and Algorithm 12, it can be seen that the *Eagle** encryption and *Eagle** decryption are mutually inverse, which means that as an encryption algorithm, *Eagle** is correct.

In addition, it is easy to see that the calculations are all completed in linear time, so the computational complexity of Algorithm 11 and Algorithm 12 are all $O(KL)$.

6.1.4 Basic Security Analysis

In this chapter, we mainly discuss the security of *Eagle** from the following three aspects.

(i) Under the condition of knowing any number of plaintext ciphertext correspondences, cracking the key is equivalent to solving three independent vari-

ables s_x, s_y, s_t. Through our analysis, to verify the correctness of any two of these three variables, it is necessary to solve a 'quintic equation', and we assert that there is no fast solution for this type of equation. Thus, the computational complexity of verifying any two variables is $O(2^{3L-3})$, which is equivalent to the computational complexity that solved the three variables by exhaustive search.

(ii) For attackers who collect a large number of ciphertext plaintext correspondences without cracking the key, and then use statistical or mathematical relationships between plaintext and ciphertext to directly decrypt the new ciphertext, this attack mode often has greater danger. We note that the ciphertexts in the $Eagle^*$ encryption algorithm can be viewed as completely independent and random, without any statistical relationship with the plaintext. In addition, theoretically, attackers cannot obtain the algebraic relationship between a single group of plaintext and ciphertext, they can only obtain the nonlinear algebraic expression with ciphertexts as a parameter that satisfies the relationship between adjacent groups of plaintext. Obviously, this directly prevents decrypting new ciphertext from the ciphertext samples.

(iii) The global relationship between the ciphertexts and plaintexts can be seen as constructed through nonlinear operations by the local operation as $s_x/\delta x^2$. In fact, the local operation $x = s_x/\delta x^2$ does not satisfy any specific linear relationship between different independent variables $\delta x_1, \delta x_2$ and different dependent variables x_1, x_2. This means that there is no effecient local analysis technique for $Eagle^*$.

To simplify the analysis, in all the following equations, we directly ignore the functions $\phi(x)$ and $\psi(x)$. In fact, these two functions are linear and inverse, and their function is only to make a simple conversion between odd numbers and numbers containing odd numbers of bits with 1. Ignoring them will not have any substantial impact.

In addition, to simplify the unnecessary redundancy in equation expression without affecting the essence of the equation, we can directly ignore the cyclic left shift operation $x^{+(L-1)}$. $\mathbb{E}_(w_0, w_1)(m) = m * (w_0 \oplus w_1) \oplus w_0^{+(L-1)}$ is represented as $\mathbb{E}_(w_0, w_1)(m) = m * (w_0 \oplus w_1) \oplus w_0$.

Knowing the plaintexts $P = (P_1, P_2, ..., P_K)$ and ciphertexts $C = (C_1, C_2, ..., C_{2K+1}, \delta x_1, \delta x_2, ..., \delta x_{2K+1}, \delta y_1, \delta y_2, ..., \delta y_{2K+1}, z_1, z_2, ..., z_K)$, the problem of cracking the keys s_x, s_y, s_t is equivalent to solving the following system of equations.

$$\begin{cases} ((((\delta t_1 * (x_1 \oplus y_1) \oplus x_1) * (u \oplus v) \oplus u) * (x_2 \oplus y_2) \oplus C_2) \\ \qquad *(u \oplus v) \oplus u) * (x_3 \oplus y_3) \oplus (\delta t_3 * (x_3 \oplus y_3) \oplus x_3) \\ \qquad = C_3 \\ ((((\delta t_3 * (x_3 \oplus y_3) \oplus x_3) * (u \oplus v) \oplus u) * (x_4 \oplus y_4) \oplus C_4) \\ \qquad *(u \oplus v) \oplus u) * (x_5 \oplus y_5) \oplus (\delta t_5 * (x_5 \oplus y_5) \oplus x_5) \\ \qquad = C_5 \\ ..... \end{cases} \tag{6.1}$$

Where $s_t/\delta t_{2i-1}^2 = P_i \oplus z_i$, $x_i = s_x/\delta x_i^2$, $y_i = s_y/\delta y_i^2$, $u = (s_x \oplus s_y \oplus 1)/\delta u^2$, $v = s_t/\delta v^2$.

Next, we will demonstrate the security described in (i) by analyzing the following three scenarios.

[**Case 1**] Verify s_x, s_t.

In order to verify a certain s_x, s_t, each equation in the equation system (6.1) can be regarded as an equation about the unsolvable variable s_y, After expanding the right multiplication operation in the equations according to the distributive law, we will find that s_y in each equation is combined in the form of $*(s_y/\delta^2)$ or $(s_y/\delta^2)*$, where the most frequent term contains 5 right multiplication operations.

So any equation in the system of equations (6.1) can be regarded as a "quintic equation" about s_y. According to the general conclusion that there is no algebraic solution for quintic equations, we assert that there is no fast solution to the equation about s_y mentioned above, which means that the correctness of each equation can only be verified through exhaustive enumeration of s_y. So the computational complexity for verifying s_x, s_t can be considered as $O(2^{L-1})$.

[**Case 2**] Verify s_y, s_t.

Similar to the analysis in Case 1, in order to verify a certains_y, s_t, any equation in the system of equations (6.1) can be regarded as a "quintic equation" about s_x. We assert that there is no fast solution to the equation about s_x mentioned above, which means that the correctness of each equation can only be verified through exhaustive enumeration of s_x. So the computational complexity for verifying s_y, s_t can be considered as $O(2^{L-1})$.

[**Case 3**] Verify s_x, s_y.

To verify a certain s_x, s_y, the system of equations (6.1) cannot directly express for s_t explicitly. Firstly, we can directly calculate $x_i = s_x/\delta x_i^2$ and $y_i = s_y/\delta y^2$ and $u = (s_x \oplus s_y \oplus 1)/\delta u^2$, denote $t_i = P_i \oplus z_i$, the problem of verifying s_x, s_y can be equivalently transformed into a problem of solving δt_1, thus v can be expressed as $v = t_1 * \delta t_1^2/\delta v^2$. $\delta t_3, \delta t_5$... can be expressed by an implicit equation of the form as $t_3 = t_1 * \delta t_1^2/\delta t_3^2$, $t_5 = t_1 * \delta t_1^2/\delta t_5^2$.

Any equation in the system of equations (6.1) can be regarded as a "quintic equation" about δt_1. We assert that the above equations about δt_1 have no fast solutions, meaning that the correctness of each equation can only be verified by exhaustive enumeration of δt_1. So the computational complexity for verifying s_x, s_y can be considered as $O(2^{L-1})$.

Based on the discussion of Case 1, Case 2, and Case 3 above, under the condition of knowing the correspondence between plaintexts and ciphertexts, the key is composed of three independent unknown variables. The computational complexity of verifying any two variables is $O(2^{L-1})$, thus the computaional complexity of solving the three variables is $O(2^{3L-3})$, which is equivalent to the computational complexity of completely exhaustively search for s_x, s_y, s_t.

Next, we demonstrate the security described in (ii).

Attackers who collect a large number of ciphertext plaintext correspondences to directly decrypt new ciphertext without cracking the key essentially exploit some algebraic relationship between plaintext and ciphertext. After appropriate deduction, we can obtain the relationship between plaintexts and ciphertexts as follows.

$$\begin{cases} (((((\delta t_{2i-1} * ((s_x/\delta x_{2i-1}^2) \oplus (s_y/\delta y_{2i-1}^2)) \oplus (s_x/\delta x_{2i-1}^2)) * (((s_x \oplus s_y \oplus 1)/\delta u^2) \\ \quad \oplus (s_t/\delta v^2)) \oplus ((s_x \oplus s_y \oplus 1)/\delta u^2)) * ((s_x/\delta x_{2i}^2) \oplus (s_y/\delta y_{2i}^2)) \\ \quad \oplus C_2) * (((s_x \oplus s_y \oplus 1)/\delta u^2) \oplus (s_t/\delta v^2)) \oplus ((s_x \oplus s_y \oplus 1)/\delta u^2)) \\ \quad *((s_x/\delta x_{2i+1}^2) \oplus (s_y/\delta y_{2i+1}^2)) \oplus (\delta t_3 * ((s_x/\delta x_{2i+1}^2) \oplus (s_y/\delta y_{2i+1}^2))) \\ \quad = C_3 \\ s_t/\delta t_{2i-1}^2 = P_i \oplus z_i \\ s_t/\delta t_{2i+1}^2 = P_{i+1} \oplus z_{i+1} \end{cases} \tag{6.2}$$

It is obvious that through a large number of ciphertext plaintext correspondences, attackers can only obtain algebraic equations satisfied between adjacent groups of plaintext, whose parameters are completely composed of ciphertexts in a nonlinear relationship. That is to say, without cracking the key, attackers can only obtain some non-linear relationship between adjacent groups of plaintext in the most extreme case, but cannot obtain a specific group of plaintext.

Next, we demonstrate the security described in (iii).

Furthermore, we observe that the global relationship between ciphertexts and plaintexts is expressed locally by operation as s/δ^2, as shown by $s_x/\delta x_1^2 = x_1$, $s_x/\delta x_2 = x_2$. For a binary function $F(x, y)$, set $\delta x_3 = F(\delta x_1, \delta x_2)$, thus $x_3 = s_x/\delta x_3^2$ can be seen as a binary function about x_1, x_2 as $x_3 = G(x_1, x_2)$. Intuitively, $F(x, y)$ and $G(x, y)$ can't be linear functions simultaneously.

Due to the fact that the right multiplication operation defined in this paper does not satisfy the commutative and associative laws, it is obvious that this operation rule can only be regarded as a subset of the operation rules satisfied by traditional multiplication operations. If it can be proven that $F(x, y)$ and $G(x, y)$ cannot both be linear functions for traditional numerical multiplication operations, then for the right multiplication operation described in this paper, it is even more impossible for $F(x, y)$ and $G(x, y)$ to both be linear functions.

Next, we will analyze from the perspective of traditional multiplication operations, assuming$F(x, y) = u_f.x + v_f.y$ and $G(x, y) = u_g.x + v_g.y$, where $u_f \neq 0$, $v_f \neq 0$, $u_g \neq 0$, $v_g \neq 0$, that is to say, both $F(x, y)$ and $G(x, y)$ are linear functions, after substitution, we obtain $\delta x_3 = u_f.\delta x_1 + v_f.\delta x_2$, $x_3 = u_g.x_1 + v_g.x_2$, by simple merge we obtain.

$$s_x/(u_f.\delta x_1 + v_f.\delta x_2)^2 = u_g.(s_x/\delta x_1^2) + v_g.(s_x/\delta x_2^2) \tag{6.3}$$

By simple algebraic deduction, we obtain

$$1/(u_f + v_f.(\delta x_1/\delta x_2)^2) = u_g + v_g.(\delta x_1/\delta x_2)^2 \tag{6.4}$$

Obviously, $\delta x_1/\delta x_2$ is a variable, there is no fixed parameters (u_f, v_f, u_g, v_g) that make (6.4) hold for any $\delta x_1/\delta x_2$. That is to say, from the perspective of traditional multiplication operations, there is no binary function $F(x, y), G(x, y)$ that both satisfies a linear relationship, therefore, from the perspective of the right multiplication operation defined in this paper, there is no binary function $F(x, y), G(x, y)$ that both satisfies a linear relationship too.

The absence of the linear binary functions $F(x, y)$ and $G(x, y)$ mentioned above also indicates that without cracking the key, attackers cannot derive the linear relationship of local variables x_i, x_j from any linear relationship between two ciphertexts $\delta x_i, \delta x_j$. Similarly, attackers are unable to derive the linear relationship of local variables y_i, y_j from any linear relationship between two ciphertexts $\delta y_i, \delta y_j$. Due to the fact that C_i can be mapped to δt_i, attackers are also unable to derive the linear relationships of local variables t_i, t_j ($t_i = s_t/\delta t_i^2$, $t_j = s_t/\delta t_j^2$) from any linear relationship between any two ciphertexts Ci, Cj.

6.1.5 Linear attack analysis to *Eagle** encryption algorithm

Linear attack is a very effective attack method proposed by M. Matsui[2] at the European Cryptology 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 algorithm, the entire encryption system is divided into three independent parts, each with its own independent key and parameters. The ciphertext of the encryption system that includes the original plaintext is generated by mixing the original plaintext with random parameters, without knowing the random parameters, ciphertext can be regarded as completely independent of the original plaintext, which means that with random parameter changes, ciphertext can equally probability traverse any text in the ciphertext space. And the two other parts of the ciphertext is completely generated by

random parameters, completely unrelated to the original plaintext. This can be expressed as

$$
\begin{aligned}
Pr(C_i = c_i | P_i = p_i) &= \frac{1}{2^{L-1}} \\
Pr(\delta x_i = sx_i | P_i = p_i) &= \frac{1}{2^{L-1}} \\
Pr(\delta y_i = sy_i | P_i = p_i) &= \frac{1}{2^{L-1}} \\
Pr(z_i = sz_i | P_i = p_i) &= \frac{1}{2^{L}} \\
Pr(\delta u = du | P_i = p_i) &= \frac{1}{2^{L-1}} \\
Pr(\delta v = dv | P_i = p_i) &= \frac{1}{2^{L-1}} \\
Pr(\delta t_{2K+1} = dt_{2K+1} | P_i = p_i) &= \frac{1}{2^{L-1}}
\end{aligned}
$$

6.1.6 Differential attack analysis to *Eagle** encryption algorithm

Differential attack was proposed by Biham and Shamir[6] in 1993, 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 the *Eagle** encryption algorithm, suppose the differential attacker chooses two specific plaintexts P_1 and P_2, their difference is Δ, that is $P_2 = P_1 + \Delta$, Since the ciphertexts in the second part and the third part are completely unrelated to the plaintext, only the ciphertext in the first part will be analyzed here, the corresponding ciphertexts are C_1 and C_2 , and the difference between the ciphertexts is ε as $C_2 = C_1 + \varepsilon$. Since C_1 and C_2 are completely random, it is uncertain whether the difference ε of the ciphertext is caused by randomness of (x_1, y_1, x_2, y_2, u) or the uncertainty of the plaintext (P_1, P_2). Furthermore, for any key (δ_1, δ_2), $x_1 * \delta_1 * \delta_1$, $y_1 * \delta_2 * \delta_2$ subject to the uniform distribution, which can be denoted as

$$
\begin{aligned}
&Pr(C_1 = c_1, ..., C_{2K+1} = c_{2K+1} | \\
&(P_1 = p_1, ..., P_K = p_K, W = (s_x, s_y, s_t)) \\
&= 1/2^{(2K+1)(L-1)}
\end{aligned}
$$

$$
\begin{aligned}
&Pr(\delta x_1 = dx_1, ..., \delta x_{2K+1} = dx_{2K+1}) | \\
&(P_1 = p_1, ..., P_K = p_K, W = (s_x, s_y, s_t)) \\
&= 1/2^{(2K+1)(L-1)}
\end{aligned}
$$

$$Pr(\delta y_1 = dy_1, ..., \delta y_{2K+1} = dy_{2K+1} | (P_1 = p_1, ..., P_K = p_K, W = (s_x, s_y, s_t)) = 1/2^{(2K+1)(L-1)}$$

$$Pr(z_1 = sz_1, ..., z_K = sz_K | (P_1 = p_1, ..., P_K = p_K, W = (s_x, s_y, s_t)) = 1/2^{K(L-1)}$$

$$Pr(\delta u = du, \delta v = dv, \delta t_{2K+1} = dt_{2K+1} | (P_1 = p_1, ..., P_K = p_K, W = (s_x, s_y, s_t)) = 1/2^{3(L-1)}$$

That is to say, for any specific plaintext P_1 and P_2 selected by the attacker, after being encrypted with the same key, the corresponding block ciphertexts (C_1, C_2, ..., C_{2K+1}), (δx_1, δx_2, ..., δx_{2K+1}), (δy_1, δy_2, ..., δy_{2K+1}), (z_1, z_2, ..., z_K), $\delta u, \delta v, \delta t_{2K+1}$ are completely random, and every 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.

Appendix A General quintic equation with Eagle right multiplication

As introduced in chapter 4.1, We have constructed a right multiplication operation, denoted as $m * \delta$. and also constructed a corresponding right division operation, denoted as s/δ.

Similar to the traditional definition of power operations, we can also define right power operations as follows

$$(m_1 \circ m_2 \circ ... \circ m_n) * \delta^n = m_1 * \delta * m_2 * \delta * ... * m_n * \delta \tag{A.1}$$

(A.1) represents performing n consecutive right multiplication operations on the parameter δ using $(m_1, m_2, ..., m_n)$.

Similarly, right root operations can also be defined.

$$(m_1 \circ m_2 \circ ... \circ m_n) * \delta^n = s \tag{A.2}$$

The root of equation (A.2) can be expressed as

$$\delta = \sqrt[n]{s[m_1 \circ m_2 \circ ... \circ m_n]} \tag{A.3}$$

So, the general quintic equation for Eagle right multiplication operation can be expressed as.

$$\begin{aligned}(a_1 \circ a_2 \circ a_3 \circ a_4 \circ a_5) * \delta^5 \oplus (b_1 \circ b_2 \circ b_3 \circ a_4) * \delta^4 \\ \oplus (c_1 \circ c_2 \circ c_3) * \delta^3 \oplus (d_1 \circ d_2) * \delta^2 \oplus e * \delta = f\end{aligned} \tag{A.4}$$

Similar to the conclusion that traditional general quintic equations have no algebraic solutions, the quintic equation defined as (A.4) also has a similar conclusion, as follows

[**Theorem A.1**] The equation defined in (A.4) has no algebraic solution.

Proof. Proof by contradiction, we suppose that there exists an algebraic solution for (A.4) expressed as h, such that $\delta = h(a_1, ..., a_5, b_1, ..., b_4, c_1, ..., c_3, d_1, d_2, e, f)$. Where f is composed of parameters $(a_1, ..., a_5, b_1, ..., b_4, c_1, ..., c_3, d_1, d_2, e, f)$ that undergo a finite number of $\oplus$ operations, right multiplication operations, right division operations, right power operations, right root operations.

Now consider another traditional quintic equation as follows.

$$\begin{aligned} &\bar{a_1}.\bar{a_2}.\bar{a_3}.\bar{a_4}.\bar{a_5}.\bar{\delta}^5 + \bar{b_1}.\bar{b_2}.\bar{b_3}.\bar{b_4}.\bar{\delta}^4 \\ &+\bar{c_1}.\bar{c_2}.\bar{c_3}.\bar{\delta}^3 + \bar{d_1}.\bar{d_2}.\bar{\delta}^2 + \bar{e}.\bar{\delta} = \bar{f} \end{aligned} \tag{A.5}$$

We perform the following regular transformations on all symbols in the expression $h(a_1, ..., a_5, b_1, ..., b_4, c_1, ..., c_3, d_1, d_2, e, f)$.

(i) Change the symbol a_i to $\bar{a_i}$, change the symbol b_i to $\bar{b_i}$, change the symbol c_i to $\bar{c_i}$, change the symbol d_i to $\bar{d_i}$, change the symbol e to $\bar{e}$, change the symbol f to $\bar{f}$.

(ii) Change the symbol $\oplus$ to $\pm$.

(iii) Change the symbol $a * b$ to $a.b$.

(iv) Change the symbol a/b to a/b.

(v) Change the symbol $(m_1 \circ m_2 \circ ... \circ m_n) * \delta^n$ to $m_1.m_2.....m_n.\delta^n$.

(vi) Change the symbol $\sqrt[n]{s[m_1 \circ m_2 \circ ... \circ m_n]}$ to $\sqrt[n]{s/(m_1.m_2.....m_n)}$.

(vii) Change the constant to a universal constant symbol (after constant replacement, although the specific constant value after replacement is currently unknown, such a constant must be a certain determined value after replacement).

The expression after transformation is $\bar{h}(\bar{a_1}, ..., \bar{a_5}, \bar{b_1}, ..., \bar{b_4}, \bar{c_1}, ..., \bar{c_3}, \bar{d_1}, \bar{d_2}, \bar{e}, \bar{f})$.

Since replacing δ in (A.4) with $h(a_1, ..., a_5, b_1, ..., b_4, c_1, ..., c_3, d_1, d_2, e, f)$ makes the equation hold, it can be seen from the homogeneity of rules (i)-(vii) that replacing $\bar{\delta}$ in (A.5) with $\bar{h}(\bar{a_1}, ..., \bar{a_5}, \bar{b_1}, ..., \bar{b_4}, \bar{c_1}, ..., \bar{c_3}, \bar{d_1}, \bar{d_2}, \bar{e}, \bar{f})$ also makes the equation hold.

It should be pointed out that the $\pm$ symbol in $\bar{h}$ does not mean that replacing $\pm$ with $+$ or $-$ will make the equation hold, but rather that each $\pm$ can only be replaced with a specific $+$ or $-$ to make the equation hold, such replacement is definitely feasible.

According to the conclusion that there are no algebraic solutions for general quintic equations, it can be concluded that such $\bar{h}$ does not exist, and therefore h satisfying (A.4) also does not exist.

□

Theorem A.1 states that although the general quintic equation defined by Eagle right multiplication operation has no algebraic solution, it cannot be ruled

out that it has some approximate numerical expression, which requires the following theorem.

[**Theorem A.2**] The right multiplication $m * \delta$ can't establish an "order" relationship $\prec$, such that for $\delta_1 \prec \delta_2$, the relationship $m * \delta_1 \prec m * \delta_2$ always holds for any m.

Proof. Proof by contradiction, we assume that there exists an order relationship $\prec$, such that for $\delta_1 \prec \delta_2$, the relationship $m * \delta_1 \prec m * \delta_2$ holds for any m.

After sorting all δ in the order of $\prec$, the sequence is denoted as $\delta_1 \prec \delta_2 \prec ... \prec \delta_{2^L-1}$.

For any m, $m * \delta_1 \prec m * \delta_2 \prec ... \prec m * \delta_{2^L-1}$ holds.

Since the sequence $\delta_1 \prec \delta_2 \prec ... \prec \delta_{2^L-1}$ and the sequence $m * \delta_1 \prec m * \delta_2 \prec ... \prec m * \delta_{2^L-1}$ are the same sequence, it can be deduced that $m * \delta_i = \delta_i$, which is obviously incorrect.

□

Due to the inability to establish an effective order relationship based on Eagle right multiplication operation, it is impossible to establish some numerical correspondence, and thus it's unable to measure the approximation of a solution to the equation (A.4). So we can conclude that there is no approximate solution for equations of type (A.4).

Appendix B More about brute-force searching

In the computational complexity theory, a problem is deemed difficult if it cannot be solved within polynomial time. Furthermore, if a certain internal variable with a problem can only be solved through brute-force searching, we can always construct the domain size of this variable to be exponential in scale, resulting in the problem having exponential complexity.

Can the internal variable of a problem only be solved through brute-force searching? We summarize this problem into the following three levels semantically:

(i) When a variable cannot be expressed with symbols in finite length for any constraint condition, its value cannot be solved using effective general algorithms other than brute-force searching.

(ii) When a variable needs to be expressed in any constraint condition which the length of the symbols used must exceed the size of its domain, the computational complexity for solving this variable should not be lower than that of brute-force searching.

(iii) When there are inherent obstacles in the constraints of a variable itself, there is no better way to solve this variable than brute-force searching.

In fact, in the Eagle encryption algorithm constructed in this paper, the problem of cracking the key is equivalently transformed into a general quintic equation with one variable as the unknown variable and the other variables as

coefficients. According to the conclusion that general quintic equations have no algebraic solutions, and the right multiplication algorithm constructed in this paper does not have a numerical algorithm for solving approximate values, thus this proves that any constraint condition of the variable in the equation cannot be expressed by finite length symbols of other variables as coefficients. This can also be equivalently understood as the inability to establish equations that satisfy the constraint conditions between variables as coefficients through finite length expressions, so naturally there is no universal algorithm other than brute-force searching to solve these variables.

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: EUROCRYPT́93, LNCS765, springer verlag*, 1994:386-397.

[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. 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.

[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] Tsunoo 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. pp. 1–6.

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

[10] Cook, Stephen A. The complexity of theorem-proving procedures. [J].ACM, 1971:151-158.