SHipher: Families of Block Ciphers based on SubSet-Sum Problem

Xiali Hei
Guandong University of Finance & Economics
Temple University

Binheng Song
Tsinghua University

February 11, 2014

Abstract

In this paper, we describe the families of block ciphers named SHipher. We show a symmetric encryption framework based on the SubSet-Sum problem. This framework can provide families of secure, flexible, and any-size block ciphers. We have extensively cryptanalyzed our encryption framework. We can easily control the computational cost by a key selection. Also, this framework offers excellent performance and it is flexible and general enough to admit a variety of implementations on different non-Abelian groups. In this paper, we provide one implementation using a group of matrices whose determinants are 1. This implementation accepts any block size satisfying $3l - 1$. If $l = 21$, the block size is 62 bits, which suits for full spectrum of lightweight applications. While if $l = 341$, the block size is 1022, which provides high security level up to resistant 2^{684} differential-attack effort and 2^{1022} brute-force attack effort.

Keywords: Block cipher; SubSet-Sum problem; Framework; Non-Abelian group

1 Introduction

There are a lot of symmetric encryption schemes such as SIMON and SPECK [6], Advanced Encryption Standard (AES) [1], and Blowfish [10]. Most of them are based on Feistel networks or diffusion or substitution, while our scheme proposed in this paper is inspired by a pure mathematically NP-hard problem (SubSet-Sum problem) with general group operators. The problem we used in this paper is a SubSet-Sum problem (SSSP) on a group. Such problems are not well investigated except the SSSP over addition. Our scheme accepts any block size. We name it as SHipher.

The security level of SHipher depends on the block size and the key K. It is also a key-dependent symmetric cipher. It can be applied in lightweight applications as well as heavyweight applications. In our implementation, when $l = 341$, it can provide an encryption scheme resistant to 2^{684} differential-attack effort and 2^{1022} brute-force attack effort.

In this paper, we first present the framework to generate the SHipher families. After that we get the security requirements and rules for the parameter design through extensive cryptanalysis. We show that the G is supposed to be a non-Abelian group and the size of the conjugate equivalence class will affect the security strength. After that, we describe an implementation of this family based on a group consisting of the matrices whose determinants are 1. Then we analyze the encoding-decoding scheme of the plain texts and cipher texts. At last we conduct the performance evaluation using SHipher-library.

Our contributions are summarized as follows:

- A symmetric encryption framework based on SubSet-Sum problem is proposed. This framework is very flexible and accepts any block size and varied implementations. It is suitable for both lightweight and heavyweight applications.
- We present a complete implementation based on a group of matrices whose determinants are 1. This implementation admits any block size satisfying $3l-1$.

The remainder of this paper is organized as follows: In Section 2 we describe the framework model to design an encryption scheme based on SSSP. We analyze the encryption model and give the security requirements of it in Section 3. In Section 4 we present one detailed implementation based on this framework. We evaluate our implementation in Section 5. We extend our scheme to other groups in Section 6. In Section 7, we review the related work, and we conclude the paper and discuss the future work in Section 8.

2 Construction Framework

Let $G = (\mathbb{S}, \bigoplus)$ be a non-Abelian group. Generally, we use a w-bit binary number to represent an element of $\mathbb{S}$, and use its element of $\mathbb{S}$ to represent a plain text or its cipher text. x^{-1} is the inverse of element x on the group G.

2.1 Global sets

The encryption algorithm is based on a global set $\mathbb{A}$. It is a perfectly chosen random subset of $\mathbb{S}$ and $\|\mathbb{A}\| = n$. The global set is a set passed various randomness tests. We use a binary file having the size $64 * w$ to represent it. For examplein our implementation, if we choose $w = 1022$ and $n = 64 * 1022$, then the size of such a file is about 64MB.

2.2 Key Generation and Encryption Scheme

Since $\|\mathbb{A}\| = n$, we can use a binary number i with the length $log_2 n$ to represent the index of an element of $\mathbb{A}$.

We can randomly choose $2m$ elements from $\mathbb{A}$, whose indices in $\mathbb{A}$ are $i_0, i_1, \ ... \ i_{2m-1}$. Since the elements may be duplicated, we allow identical indices.

A shared key $K = \{i_0 \| i_1 \| ... \| i_{2m-1}\}$, and its length is $2m * log_2 n$.

Encryption algorithm: We assume that $p \in \mathbb{S}$ is a plain text and its cipher text is b. Then the relationship between them is shown as follows,

$$p = \bigoplus_{j=0}^{m-1} a_{i_j} \bigoplus b \bigoplus_{j=m}^{2m-1} a_{i_j}; \tag{2-1}$$

where a_{i_j} is the element in $\mathbb{A}$ whose index is i_j. We can get b by Equation (2-1) as follows:

$$b = \bigoplus_{j=m-1}^{0} a_{i_j}^{-1} \bigoplus p \bigoplus_{j=2m-1}^{m} a_{i_j}^{-1}; \tag{2-2}$$

where $a_{i_j}^{-1}$ is the inverse element of a_{i_j} on the group G.

Definition 1. We call the following problem is a generalized SubSet-Sum Problem (GSSSP): given $\mathbb{S}$ and b to find a subset $\mathbb{B}$ of $\mathbb{S}$ such that the sum of all the elements in $\mathbb{B}$ over operator $\bigoplus$ equals b.

If the set $\mathbb{S}$ is composed of integers and the operator is plus, then the above problem degrades to a SSSP.

Theorem 1. The problem of given b to get p without known the indices of a_{i_j} is a harder problem than a GSSSP.

Proof: In equation (2-2), if one wants to find p with known b and without known the indices of a_{i_j}, the problem can be transferred to a GSSSP as follows:

To find p, we have to find the subset $\mathbb{B}$ whose elements' sum is b, and then guess which element of $\mathbb{B}$ is p. So it is harder than a GSSSP.

Conjecture 1. To most group operator, the GSSSP is a NPC problem. Specifically, it holds when the operator is a matrix operation.

3 Security analysis

3.1 Brute-Force Attacks

If an attacker wants to get the key, he/she needs to get the indices of the elements in $\mathbb{A}$ used by the encryption scheme. The probability of getting them via one-time guess is $n^{-2m} = 2^{-2mlog_2 n}$. If $w = 1022$, $n = 64 * 1022$, then $log_2 n = log_2(64 * 1022)$. According to the requirement of security level, we can choose $m = 32$, the successful rate of one-time brute-force test is $2^{-1022} = 2^{-w}$.

The bigger the m is, the less the probability of being broken is. However, the bigger the m is, the more complex our encryption algorithm is because we need $2m$

group operations of w-bit binary numbers. Hence, there is a tradeoff. In the case of same security level, the greater the n is, the less the m is. This means that the memory overhead increases and the computation overhead decreases.

Additionally, the length of w will affect the computational complexity. The greater the w is, the computation complexity of each group operation increases. When we choose the multiplication, it increases quadratically with the length of w increases.

If the length w is too small, then the randomness strength of each encryption block is not suitable.

If a file with the length T can be split into T/w pieces, we will encrypt each piece. The total computational complexity is $T/w * f(w) * 2m$, where the $f(w)$ is the computational complexity of $\bigoplus$ operator on w-bit numbers.

3.2 Differential Attacks

In our scheme, encryption needs to get the inverse of a_{i_j} and $\bigoplus$ operations on group G, while decryption only needs the $\bigoplus$ operations.

We focus on the differential attacks at $\bigoplus$ operator. They are the left reverse multiplication and right reverse multiplication of a cipher text using the same key.

Suppose the p and p' are different plain texts and b and b' are their cipher texts respectively, then

$$b^{-1} \bigoplus b' = \bigoplus_{j=m}^{2m-1} p_{i_j} \bigoplus p^{-1} \bigoplus p' \bigoplus_{j=2m-1}^{m} p_{i_j}^{-1}; \tag{3-3}$$

If the $\bigoplus$ is abelian, then $b^{-1} \bigoplus b' = p^{-1} \bigoplus p'$. That means the difference of two cipher texts equals to the difference of two plain texts. Then the differential attacks work.

If $\bigoplus$ is non-abelian, then $b^{-1} \bigoplus b'$ and $p^{-1} \bigoplus p'$ are conjugate.

The larger the size of $[b^{-1} \bigoplus b']$ on a group G is, the more verifications an attacker needs to crack the encryption scheme via brute-force attacks. The best case is that $[b^{-1} \bigoplus b'] = \|\mathbb{A}\|$.

4 Implementation

We choose n w-bit random numbers, whose first 1/3 bits are nonzero. They consist of set $\mathbb{A}$ and we publish $\mathbb{A}$. We assume S including all the w-bit binary numbers. Then, we define the $\bigoplus$ as follows:

Let $x, y \in \mathbb{A}$, we split x into three pieces x_1, x_2, x_3 equally. Then, the length of x_t $(t = 1,\ 2,\ 3)$ is $w/3$ bits. We treat y similarly. Then we consider x_1, x_2 and x_3 as elements in $Z_{2^{(w/3)}} = \{0, 1, ...,\ 2^{w/3} - 1\}$. We choose a prime number of length $(w/3 + 1)$ bits as the q. Then we write x as a following matrix:

$$M_x = \begin{pmatrix} x_1 & x_2 \\ x_3 & \frac{x_2 x_3 + 1}{x_1} \end{pmatrix} (mod\ q); \tag{4-4}$$

This design makes sure $det(M_x) = 1$ because $(x_1 \times \frac{x_2 x_3 + 1}{x_1} - x_3 \times x_2)\ mod\ q = 1$.

Then

$$x \bigoplus y = M_x \cdot M_y \ (mod\ q); \tag{4-5}$$

where $M_x \cdot M_y$ is an ordinary matrix multiplication.

4.1 Special Cases of Plain Text and Cipher Text

Special case of plain text: for a plain text p_i, if p_{i_1} is 0, then p_{i_1} is not invertible over modular q. Thus we can not construct a matrix M_{p_i}. To make sure that we can construct such a matrix of any plain text block, we add a redundant bit 1 to the highest bit of p_i. Thus, we can make sure the determinant of corresponding plain text matrix is 1 according to our design for M_{p_i}.

Special case of cipher text: assume the corresponding cipher text matrix is

$$\begin{pmatrix} u & v \\ h & r \end{pmatrix}. \tag{4-6}$$

Since its determinant is 1, then we can represent it as three numbers. a) if $u \neq 0$, we can use (u, v, h) to represent it. b) if $u = 0$, we represent it as (v, h, r). So we need a redundant bit to distinguish these two cases.

How to tune the encoding and decoding methods:

Plain text: we change a message p_i into $1\|p_i$, and split it into $(p_{i_1}, p_{i_2}, p_{i_3})$ equally, where $\|p_{i_1}\| = l$. l represents the length of a binary string. As a result, the length of the every plain text is $w = 3l - 1$. Since we want w to be close to 1024, we choose $l = 341$.

Cipher text: we know that the determinant of the cipher text matrix is 1. In its matrix

$$\begin{pmatrix} u' & v' \\ h' & r' \end{pmatrix}, \tag{4-7}$$

if $u' \neq 0$, it can be presented as $(1, u', v', h')$; otherwise if $u' = 0$, it can be presented as $(0, v', h', r')$. The first bit indicates whether the first element in the ciphertext matrix is 0 or not. If its first element is not 0, we add 1 to the highest bit indicating that the first element is not 0 and use the first three elements to represent this matrix. Otherwise, we add 0 to the highest bit to indicate that the first element is 0 and use the other three elements to represent the matrix. After the encryption, the length of cipher text is $3*(l+1)+1 = 3l+4$ bits. So the memory overhead is $(3l+4)-(3l-1) = 5$ bits for each block.

A given matrix on group G has $2^{2[(w+1)/3+1]} = 2^{2(w+4)/3}$ conjugate matrices, we need $2^{2(w+4)/3}$ verifications to get $b^{-1} \bigoplus b'$ by its conjugate matrices via differential attacks.

4.2 Group Operation Algorithm

Algorithm 1 illustrates the algorithm of a group operator. We first need 2 multiplications, 1 additions, and 1 inverse operation over modular q. Then each group operator needs 2 multiplications and 4 additions.

Algorithm 1 Group Operator Algorithm

1: **Input:** x, y.
2: **Output:** $x \bigoplus y$.
3: Split $1\|x$ into x_1, x_2, and x_3 equally. So does $1\|y$.
4: Using Euclid Algorithm to get $x_4 = \frac{x_2x_3+1}{x_1}$ mod q and $y_4 = \frac{y_2y_3+1}{y_1}$ mod q;
5: Matrix $M_x = (x_1, x_2; x_3, x_4)$ mod q; Matrix $M_y = (y_1, y_2; y_3, y_4)$ mod q;
6: $x \bigoplus y = M_x \cdot M_y$ mod q;

4.3 The Inverse of the Group Operation Algorithm

Since $\|M_x\| = 1$ and $1^{-1} mod\ q$=1, the modular inverse of it is $M_x^{-1} = M_x^* = (x_4, -x_3; -x_2,$
$x_1)$ mod q. Algorithm 2 illustrates the algorithm to get it. The computational complexity of it is $\Theta(logq)$ $logq$-bit integer divisions. This algorithm needs 2 additions.

Algorithm 2 Modular Inverse Algorithm of a 2-2 Matrix

1: **Input:** $M_x = (x_1, x_2; x_3, x_4)$ mod q;
2: **Output:** M_x^{-1}.
3: $M_x^{-1} = (x_4, -x_3; -x_2, x_1)$ mod q;
4: $-x_3 = q - x_3$, $-x_2 = q - x_2$;
5: RETURN M_x^{-1};

4.4 Encryption Algorithm

Algorithm 3 illustrates the encryption algorithm, where k is the number of blocks that plain text p has. Notice that a_{i_j} is an element of $\mathbb{A}$, we will compute the $M_{a_{i_j}}$ and $M_{a_{i_j}}^{-1}$ in advance. So we consider them as part of encoding-decoding phases. Thus, we exclude them when we compute the basis computations needed here. Totally, it needs $16(m-1)+17k$ multiplications, $8(m-1)+9k$ additions, and k inverse operations.

Algorithm 3 Encryption Algorithm

1: **Input:** Plaintext $p = \{p_1, \ ..., \ p_k\}$; $K = \{i_0||i_1||...i_{2m-1}\}$.
2: **Output:**Ciphertext $b = \{b_1, \ ..., \ b_k\}$.
3: $H_1 = \bigoplus_{j=m-1}^{0} a_{i_j}^{-1}$; $H_2 = \bigoplus_{j=2m-1}^{m} a_{i_j}^{-1}$;
4: **for** each s in $[1, 2, ..., k]$ **do**
5: $\quad b_s = H_1 \bigoplus p_s \bigoplus H_2$;

4.5 Decryption Algorithm

Algorithm 4 illustrates the decryption Algorithm, where k is the number of blocks that cipher text b has. Similarly, it needs $16(m-1)+17k$ multiplications, $8(m-1)+9k$ additions, and k inverse operations in total.

Algorithm 4 Decryption Algorithm

1: **Input:** Ciphertext $b = \{b_1, \ ..., \ b_k\}$; $K = \{i_0||i_1||...i_{2m-1}\}$.
2: **Output:**Plaintext $p = \{p_1, \ ..., \ p_k\}$.
3: $H_3 = \bigoplus_{j=0}^{m-1} a_{i_j}$; $H_4 = \bigoplus_{j=m}^{2m-1} a_{i_j}$;
4: **for** each s in $[1, 2, ..., k]$ **do**
5: $\quad p_s = H_3 \bigoplus b_s \bigoplus H_4$;

5 Performance analysis

We now provide some information on the performance achieved by the SHipher-toolkit. We measure the private key generation time, encryption time, and decryption time produced by running SHipher-keygen, SHipher-enc, and SHipher-dec. The measurements were taken on a personal computer with a 64-bit, 2.66 Ghz Intel(R) CPU. The implementation uses the group we discussed in Section 4. Here, assuming $k = 1$ (only one block message), we first choose w=62, $l = 21$, and q=2098081 over a 22-bit finite field. On the test machine, the GMP library can compute the inverse of an element over modular q in less than 1.7ms, and both multiplication and addition take about 0.002ms. Randomly selecting elements for the global set is also a significant operation, requiring about 2ms. So the SHipher-enc needs $(24m + 2) * 0.002 + 1.7$=$(0.048m + 1.704)$ ms and the SHipher-dec needs the same time. $(3.4m + 1.7)$ms are consumed by encoding the elements we selected and plain text or cipher text.

Then we choose w=125 and $l = 42$ over a 43-bit finite field, as well as q = 4466156694371. On the test machine, the GMP library can compute the inverse of a matrix in less than 1.75ms, and multiplication and addition take about 0.002ms respectively. Randomly selecting elements for the global set is also a significant operation, requiring about 2.5ms. So the SHipher-enc needs $(24m + 2) * 0.002 + 1.75$=$(0.048m + 1.754)$ms and the SHipher-dec needs the same time. $(3.5m + 1.75)$ms are consumed by encoding the elements we selected and plain text or cipher text. It is better to encode the elements in $\mathbb{A}$ and publish them. We can see that the time consumed is related to the size of $K = 2mlog_2n$.

Compare them with the results in [3], when block size =125 bits, our implementation is a little faster than KATAN [2], and a little slower than AES [1], KLEIN [3], and PRESENT [4]. We have restricted our comparisons exclusively to the above ciphers. However, our framework is based pure mathematical problem, which can be thoroughly analyzed and optimized.

As expected, SHipher-keygen runs in time precisely linear in the length of the key it is issuing. The running time of SHipher-enc and SHipher-dec are also almost perfectly linear with respect to the length of the key K. The matrix inverse operations significantly contribute to the running time as well as the multiplication operations over group G.

In summary, SHipher-keygen, SHipher-enc, and SHipher-dec run in a predictable amount of time based on the value of w and K. The performance of them depends on

the group operation. In all cases, the toolkit consumes almost no overhead beyond the cost of the underlying group operations and random selection of elements. If we choose suitable group, large private keys are possible in practice while maintaining reasonable running times.

From Section 4.1, we know that the memory overhead of SHipher is 5 bits for each block in our implementation. And it can resist $2^{2(w+4)/3}$ differential attack effort.

6 Extension

Our encryption framework uses the key to decide which elements to be added or computed. It is a key-dependent encryption framework. A crypto system designer can have a different encryption scheme after changing the non-Abelian group. This encryption framework admits a variety of implementations on non-Abelian groups.

7 Related Work

There are a lot of lightweight block ciphers such as SIMON and SPECK [6], AES [1], KATAN [2], KLEIN [3], and PRESENT [4]. We have restricted our comparisons exclusively to the above ciphers. Consequently, we don't consider other interesting lightweight stream ciphers like HUMMINGBIRD-2 [5], GRAIN [7], TRIVIUM [8], and SALSA20 [9]. Comparing with the above block ciphers, the expression of our scheme is simple. And the block size of our scheme is more flexible. Also, the users can choose different implementations according to their security requirements. Besides, our framework provides a new way to design symmetric block ciphers.

8 Conclusions and Future Work

We created a symmetric crypto framework based on SubSet-Sum problem. Our symmetric encryption framework is easy to implement and give us another way to design crypto systems. At the same time, it provides flexible security strength. It also can be applied to many applications. We provided an implementation of our framework.

In the future, it would be interesting to consider these new encryption families as one-time pad encryption schemes. The primary challenge in this line of work is to find an efficient non-Abelian group. We believe an important endeavor would be to find a more suitable non-Abelian group for this framework. Furthermore, it is an important issue to investigate properties which the groups should satisfy to implement a good cipher system under this framework.

References

[1] J. Daemen and V. Rijmen, "The design of Rijndeal", Springer, Berlin, 2002.

[2] C. D. Cannière, o. Dunkelman, and M. Knežević, "KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers", in CHES 2009, Lecture Notes in Computer Science, no. 5747, pp. 272 - 288, Springer-Verlag, 2009.

[3] Z. Gong, S. Nikova, and Y. W. Law, "KLEIN: a new family of lightweight block ciphers", in RFIDsec '11 Workshop Proceedings, Cryptology and Information Security Series, no. 6, pp. 1 - 18, IOS Press, 2011.

[4] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra- Lightweight Block Cipher", in CHES 2007, Lecture Notes in Computer Science, no. 4727, pp. 450 C 66, Springer-Verlag, 2007.

[5] D. Engels, M. Saarinen, and E. Smith, "The Hummingbird-2 lightweight authenticated encryption algorithm", in Cryptology ePrint Archive, Report 2011/126, 2011.

[6] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK Families of Lightweight Block Ciphers", in Cryptology ePrint Archive, Report 2013/404, 2013.

[7] M. Hell, T. Johansson, A. Maximov, and W. Meier, "The Grain Family of Stream Ciphers", in New Stream Cipher DesignsThe eSTREAM Finalists, Lecture Notes in Compter Science, no. 4986, pp. 179 C 90, Springer-Verlag, 2008.

[8] C. D. Canniere and B. Preneel, "TRIVIUM Specifications", in ECRYPT Stream Cipher Project Report 2005/030, 2005.

[9] D. J. Bernstein, "The Salsa20 Family of Stream Ciphers", in New Stream Cipher DesignsThe eSTREAM Finalists, Lecture Notes in Compter Science, no. 4986, pp. 84 C 97, Springer-Verlag, 2008.

[10] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Twofish: A 128-Bit Block Cipher", https://www.schneier.com/paper-twofish-paper.pdf.