SHIPHER: A New Family of Light-weight Block Ciphers based on Dynamic Operators

Xiali Hei
Department of Computer and Information Sciences
Delaware State University
Dover, DE 19901, USA
Email: xhei@temple.edu

Binheng Song
Graduate School at Shenzhen
Tsinghua University
Shenzhen, 518055, China
Email: bsong@tsinghua.edu.cn

Caijin Ling
School of EIE
Heyuan Polytechnic
Heyuan, Guangdong, 517000, China
Email: ling8983@gmail.com

***Abstract*—In this paper, we describe a family of block ciphers named SHIPHER. We present a symmetric encryption framework based on a cryptographic hash function and dynamic operators controlled by small random numbers. This dynamic operator mixes operations from different algebraic groups like IDEA [1]. However, unlike IDEA and extended IDEA ([2], [3]), modular addition is the only calculation in this framework and this makes SHIPHER highly efficient. The round function was chosen to provide confusion and diffusion to facilitate hardware implementations. This framework can provide families of secure, flexible, and variable-key-length block ciphers. *Any block size can be achieved.* We have extensively investigated our encryption framework. We can easily control the computational cost by selecting block size, implementation method, and a hash function. Also, this framework offers excellent performance and it is flexible and generic enough to admit a variety of implementations on different dynamic operators. In this paper, we provide one implementation, show its performance, and discuss possible extensions of similar dynamic operators.**

***Index Terms*—block cipher; symmetric cipher; customized operator; light-weight; embedding devices**

I. Introduction

Many symmetric encryption schemes such as Advanced Encryption Standard (AES) [9] and Blowfish [5]. Most of them are based on Feistel Networks or substitution, while our scheme proposed in this paper is inspired by a customized nonlinear operators, which is a randomized combination of group operations. Our scheme accepts any block size and we name it as SHIPHER, which is different from our previous work [4] using SubSet-Sum problems.

The design goal of SHIPHER is to provide another novel light-weight block cipher family. The security level of SHIPHER depends on the block size, time t', and the key K. So it is also a key-dependent symmetric block cipher. However, it is different from the idea using extra key bits to get strong s-boxes in [6]. (See more comparisons with other ciphers in Section 8).

SHIPHER can be applied in lightweight applications (e.g. 8-bit microcontroller) as well as heavyweight applications. In this paper, we first present the framework to generate the SHIPHER families. Then a novel symmetric block cipher implementation is proposed herein. In the proposed cipher, we choose the block size to be 256, while the length of a secret key varies.

The dynamic operator mixes operations from different algebraic groups and is controlled by small random positive integers with constrains. The required confusion is achieved by successively using a fast cryptographic hash function, two different group operations on pairs of varible-length subblocks (or chunks) and the cipher structure was chosen to provide the necessary diffusion. Our cipher resists differential attacks, linear cryptanalysis, Boomerang attacks, and related-key attacks due to its high randomness. *Its cipher structure was chosen to facilitate hardware implementations, unlike IDEA [1] and extended IDEA [2], it does not have complex operators such as modular multiplication.*

We also extensively analyze and test the avalanche effect. At last we show other extension methods to this framework, and conduct the performance evaluation using SHIPHER-library.

Our contributions are summarized as follows:

- A symmetric encryption framework based on customized nonlinear operators is proposed. This framework is very flexible and accepts any block size and various implementations. It is suitable for both lightweight and heavyweight applications.
- We present a complete implementation of 256-bit block cipher and present potential extensions to this block cipher.

The remainder of this paper is organized as follows: In Section 2 we describe the framework model to design SHIPHER. In Section 3 we present one detailed implementation and several other implementation methods based on this framework. We analyze the encryption model and test the avalanche effect in Section 4. Thorough security analysis is given in Section 5. We evaluate our implementation in Section 6. In Section 7, we discuss extension to this encryption framework. In Section 8, we review the related work, and we conclude the paper and discuss the future work in Section 9.

II. Construction Framework

A. Definition of dynamic operator

A dynamic operator $\star$ is defined by randomly combining several finite group operations together, such as addition modular over 2^l, multiplication over $Z_p \setminus \{0\}$, addition over $GF(2^n)$, etc. Let $G = (\mathbb{S}, \bigoplus)$ be a group. $\bigoplus_l$ means a group operation on a l-bit subblock, which may be any group operation, and $\&$ is a concatenation operation.

Definition 1: Suppose p is the number of subblock (or chunks) in a block, and l_i is the length of a subblock (or

978-1-4673-8999-0/17/$31.00 ©2017 IEEE

chunk), then $\star$=$\bigoplus_{l_0}$ & $\bigoplus'_{l_1}$ & $\cdot$ & $\bigoplus''_{l_i}$ & $\cdot$ & $\bigoplus'''_{l_p}$, where $\bigoplus$, $\bigoplus'$, $\bigoplus''$, and $\bigoplus'''$ are finite group operations.

For example, when $\bigoplus$ is $\boxplus_l$, an addition of integers modulo 2^l where the l-bit subblock is treated as the usual radix-two representation of an integer. Then $\star$=$\boxplus_{l_0}$& $\boxplus_{l_1}$ & $\cdot$ & $\boxplus_{l_i}$ & $\cdot$ &$\boxplus_{l_p}$.

Suppose $p = 2$, $l_0 = 3$ and $l_1 = 1$, $\star = \boxplus_3 \& \boxplus_1$ means that adding the first 3 digits (a $\boxplus_3$ operation), and followed by a $\boxplus_1$ operation. Thus, $(0011)\star(1011) = (001+101) mod\ 2^3$ &$(1+1) mod\ 2^1 = 1100$.

An dynamic operator $\star$ is determined by block size w, number of chunk p, length l_i of each chuck, and different group operations on each chunk. Through combining different group operations with &, $\star$ provides nonlinearity. When w and p are the same, a slightly change in l_i will make the $\star$ change.

B. A note of a dynamic operator $\star$

We should notice that for a dynamic operator $\star$,

1) If $\star_1$ and $\star_2$ are different, then

$(n_1 \star_1 n_2) \star_2 n_3 = n_1 \star_1 (n_2 \star_2 n_3)$ is generally not true.

For example, for a 5-bit chunk, assume $\star_1$ is $\boxplus_1 \& \boxplus_2 \& \boxplus_2$, while $\star_2$ is $\boxplus_2 \& \boxplus_3$. $n_1 = 10111$, $n_2 = 11001$ and $n_3 = 01110$;

Then $n_1 \star_1 n_2 = (1)(01)(11) \star_1 (1)(10)(01) = 01100$

$(n_1 \star_1 n_2) \star_2 n_3 = (01)(100) \star_2 (01)(110) = 10010$.

$n_2 \star_2 n_3 = (11)(001) \star_2 (01)(110) = 00111$

$n_1 \star_1 (n_2 \star_2 n_3) = (1)(01)(11) \star_1 (0)(01)(11) = 11010$.

We can see that a pair of different $\star_s$ is not distributive.

C. Confusion

Confusion (see ([7], [8])) means that the ciphertext depends on the plaintext and key in a complicated and involved way. The confusion is achieved by mixing different group operations.

The two operations are incompatible in the sense that:

1. No pair of the two operations satisfies a distributive law. For example,

$$a \star_1 (b \star_2 c) \neq (a \star_1 b) \star_2 (a \star_1 c) \tag{II-1}$$

2. No pair of the 2 operations satisfies an associative law. For example,

$$a \star_1 (b \star_2 c) \neq (a \star_1 b) \star_2 c. \tag{II-2}$$

3. The two group operations in a $\star$ are combined by a &, which inhibits isotopisms as shown in literature [1]. Thus, using any bijections on the operands, it is impossible to realize any one of the two operations by another operation.

D. Round encryption

In the encryption, w is the block size, p is the number of subblock (or chunks) in a block, $\sum_{i=1}^{p} d_i^{(s)} = w$, we use $\star_s$=$\bigoplus_{d_0^{(s)}}$ &$\cdot$& $\bigoplus'_{d_i^{(s)}}$ &$\cdot$& $\bigoplus'''_{d_{p-1}^{(s)}}$, where s is the round label, $\star_s$ is the dynamic operator we will use in each round.

This cipher relies on two factors: (1) shared key K; (2) time t' to generate small random integers, which is transmitted through out-of-band channel, such as text message, email, and personal call. These small random integers are used to determine the $\star_s$.

Suppose that m is a plaintext, and H_{k0} is the hash of shared key K, which is a 256-bit binary digit when we use RC4-256 as the hash function. $H_{k_1}, ..., H_{k_n}$ are generated by H_{k_0}, where the n is determined by security strength. σ_s is a permutation generated by $\{H_{k_1}, ..., H_{k_n}\}$.

Definition 2: We define a mapping F_1, such that $\{H_{k_1}, ..., H_{k_n}\}$=$F_1(K, t')$. F_1 is used to generate $\{H_{k_1}, ..., H_{k_n}\}$ from H_{k_0}, which is a w-bit binary digit. F_1 has different implementations.

Definition 3: We define a mapping F_2, such that σ=$F_2(K, t')$. F_2 is used to generate a permutation from $\{H_{k_1}, ..., H_{k_n}\}$, which are n w-bit binary digits. F_2 has different implementations as well.

Definition 4: We define a mapping F_3, such that $\{d_0, ..., d_i, ..., d_p\} = F_3(K, t', p)$, $\sum_{i=1}^{p} d_i = w$, where w is the block size, time t' is a seed of a Pseudo Random Number Generator $PRNG()$, p is the number of subblock (or chunks) in a block. F_3 also has different implementations.

We use the following round function to encrypt a plaintext m. $e_0 = m$, H_{k_0} and H_{k_1} are seen as n_0 and n_1, permutation σ_s is dependent on key. In the $s+1$ round encryption, H_{k_s} can be seen as n_s;

The round function is:

$$\begin{aligned} e_1 &= \sigma_1(e_0) \star_0 n_0; \\ e_2 &= \sigma_2(e_1) \star_1 n_1; \\ &\cdot \\ e_{s+1} &= \sigma_s(e_s) \star_s n_s. \end{aligned} \tag{II-3}$$

The dynamic operator $\star_s$ is defined in 2.1 and it has various implementations.

E. Decryption scheme

The encryption and decryption processes are similar. The differences between the encryption and decryption algorithm are: (1) Do the inverse operation of $\star_s$; (2) Use the corresponding inverse of the permutation σ_s.

The decryption formula is as follows,

$$e_s = \sigma_s^{-1}(e_{s+1} \star_s^{-1} n_s); \tag{II-4}$$

where, $\star_s^{-1}$ is the inverse of $\star_s$, σ_s^{-1} is the inverse of σ_s.

The complexity of decryption depends on the complexity of solving the following problem:

Problem 1: Given e_2 and e_0, find (n_0, n_1, $\star_0$), $\star_1$) such that $e_2 = \sigma_2^{-1}(n_1 \star_1 (n_0 \star_0 e_0))$.

The number of the small random numbers used and the number of rounds affects the security level. The larger of them, the more secure the encryption scheme. We will give detailed analysis in Section 4 and 5.

III. Implementation

To construct a dynamic operator, we chop a block into $w/2^h$ 2^h-bit chunks, where h is a small integer such as 3 and 4. The lengths of chunks $d_i^{(s)}$ are related to an order-dependent $(h-1)-$compositions of an integer 2^h. To simplify the implementation and implement it in an 8-bit microcontroller,

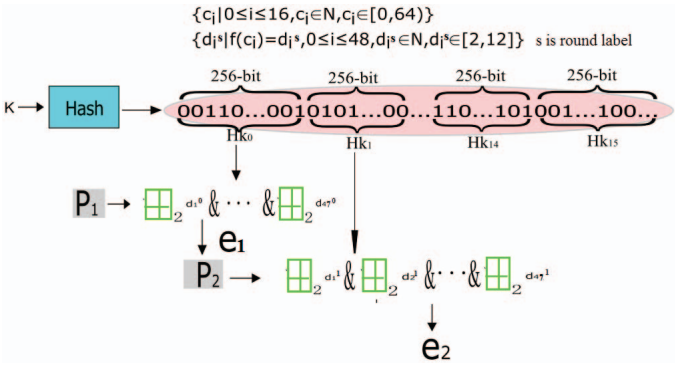

Fig. 1. A 2-round encryption diagraph

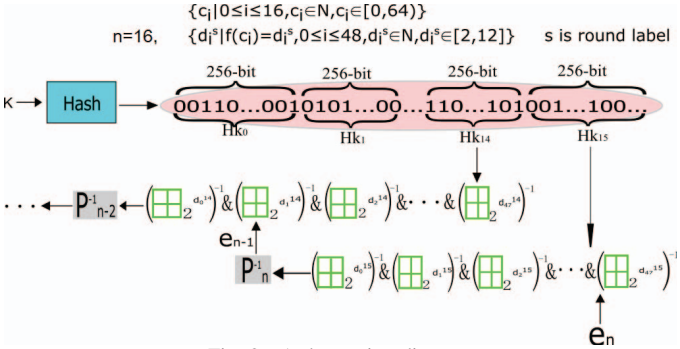

Fig. 2. A decryption diagram

we restrict the $d_i^{(s)}$ within [2, 2^h=8] when $h = 3$. Also, we choose $\boxplus_l$ to replace the finite group operation $\bigoplus_l$ in Section 2.1. $\boxplus_l$ is an addition of integers modulo 2^l where the l-bit subblock is treated as the usual radix-two representation of an integer. $\&$ is a concatenation operation in this paper. Then $\star$=$\boxplus_{l_0}\&\boxplus_{l_1}\&\cdot\&\boxplus_{l_i}\&\cdot\&\boxplus_{l_p}$.

For example, $p = 2$, $l_0 = 3$ and $l_1 = 1$, $\star = \boxplus_3\&\boxplus_1$ means that adding the first 3 digits (a $\boxplus_3$ operation), and followed by a $\boxplus_1$ operation. Thus, $(0011) \star (1011) = (001 + 101) mod\ 2^3\ \&(1 + 1) mod\ 2^1 = 1100$.

A. Encryption diagram

The encryption process consists of a cryptographic hash function, and z-round similar computations according to the block size followed by an output transformation. Averagely, one round of computations can make at least one and a half bit change in the ciphertext. Then a 256-bit block cipher needs 12 rounds of computation in average to make sure that 1-bit change in the plaintext will make 128-bit change in the ciphertext. We can choose z=16 in our design when the block size is 256. σ_s is a permutation (P1 and P2 represent σ_1 and σ_2 in Fig.1) used in kth round and it has different implementations and no invariant.

In the encryption, w is the block size, p is the number of subblock (or chunks) in a block, $\sum_{i=1}^{p} d_i^{(s)} = w$, we use $\star_s$=$\boxplus_{d_1^{(s)}}\&\cdot\&\boxplus_{d_i^{(s)}}\&\cdot\&\boxplus_{d_p^{(s)}}$, where s is the round label, $\star_s$ is the dynamic operator we will use in each round. The complete first two rounds are depicted in Fig. 1. In Fig. 1, $H_{k_0} = Hash(K)$, $(H_{k_0}, *H_{k_1}, ..., H_{k_{15}})$=$F_1(H_{k_0})$, $e_1 = \sigma_1(e_0) \star_0 H_{k_0}$; $e_2 = \sigma_2(e_0) \star_1 H_{k_1}$, c_i is a small random number within [0, 64). We will see how to decide the range of it later. f is a mapping between c_i to $d_i^{(s)}$.

B. Decryption scheme

The encryption and decryption processes are similar. The differences between the encryption and decryption algorithm are: (1) Do the modular substraction first instead of the modular addition; (2) Use the corresponding inverse of the permutation. Figure 2 shows the two-round decryption process. P_n and P_{n-2} are permutations and P_n^{-1} and P_{n-2}^{-1} are the corresponding inverse permutation of them. In Fig.2, $n = 16$, $e_{n-1} = P_n^{-1}(e_n \star_{15}^{-1} H_{k_{15}})$; $e_{n-2} = P_{n-2}^{-1}(e_{n-1} \star_{14}^{-1} H_{k_{14}})$, ...

C. Hash function and an implementation of F_1

The hash module in Fig. 1 is a 256-bit cryptographic hash function. We choose RC4-256 in our implementation. After the hash function, an any-length key is converted to H_{k_0}, which is a 256 bit binary integer. Then we choose $w/2^h - 1$ small primes, which are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, ... They last one is dependent on w and h. When $w = 256$, $h = 4$, $w/2^h - 1 = 15$. The last prime is 47. These small primes are used as follows:

H_{k_1} is a 2-bit right circular shift of H_{k_0};

H_{k_2} is a 3-bit right circular shift of H_{k_0}; ... $H_{k_{15}}$ is a 47-bit right circular shift of H_{k_0}.

H_{k_0}, H_{k_1}, ..., $H_{k_{15}}$ are used to generate the permutation σ_s and work as $n_0, n_1, ..., n_{15}$ in the encryption process.

D. An implementation of F_3: The generation of p, $d_i^{(s)}$, and $\star_s$

Suppose w is the block size, p is the number of subblock (or chunks) in a block, $\sum_{r=0}^{p-1} d_i^{(s)} = w$, all the $d_i^{(s)}$ are determined by small random integers, then $\star_s$=$\boxplus_{d_0^{(s)}}\&\boxplus_{d_1^{(s)}}\&\cdot\&\boxplus_{d_i^{(s)}}\&\cdot\&\boxplus_{d_p^{(s)}}$, where s is the round label. We use the following method to determine the p and $d_i^{(s)}$:

A w-bit block was chopped into $w/2^h$ 2^h-bit chunks. $p = (h-1) * w/2^h$. Let $P'([2, 2^h], h-1; 2^h)$ denote the number of order-dependent compositions of integer 2^h with at most $h-1$ parts, each of size within $[2, 2^h]$. Assume that the total number of such compositions is $P'([2, 2^h], h-1; 2^h) = y$, then we use time t' as a seed to generate $w/2^h$ small integers c_g within $[0, y-1]$. If $c_g = v$, then we retrieve the $(v+1)$th ordered partition in the y partitions. We set $[d_{gp}^{(s)}, d_{gp+1}^{(s)}, ..., d_{gp+p}^{(s)}]$ to the vth ordered partition, where s is the round label.

For example, when $h = 3$, $w/2^h = 32$, $h - 1 = 2$, $p = (h-1) * w/2^h = 64$, $2^h = 8$. Recall that we have a constrain that each of size should be within $[2,\ 2^h]$. The satisfied order-dependent compositions of 8 are [3, 5], [5, 3], [4, 4], [2, 6], and [6, 2] and $y = 5$. Thus, random small integers c_g is within [0, 4]. If $c_i = 0$, then we retrieve the 1st ordered partition [3, 5]. We will set $d_{2i}^{(s)} = 3, d_{2i+1}^{(s)} = 5$.

In our implementation, we choose $h = 4$, $w/2^h = 16$, $h - 1 = 3$, $p = (h-1) * w/2^h = 48$, $2^4 = 16$. Recall that we have a constrain that each of size should be within $[2,\ 16]$. The satisfied order-dependent compositions of 16 are [6, 5, 5],

[5, 6, 5], [5, 5, 6], [6, 6, 4], [4, 6, 6], [4, 6, 6], [7, 5, 4], [7, 4, 5], [5, 4, 7], [5, 7, 4], [4, 5, 7], [4, 7, 5], [7, 6, 3], [7, 3, 6], [6, 7, 3], [6, 3, 7], [3, 6, 7], [3, 7, 6], [7, 7, 2], [2, 7, 7], [7, 2, 7], [8, 4, 4], [4, 8, 4], [4, 4, 8], [8, 5, 3], [8, 3, 5], [3, 8, 5], [3, 5, 8], [5, 8, 3], [5, 3, 8], [8, 6, 2], [8, 2, 6], [6, 2, 8], [6, 8, 2], [2, 6, 8], [2, 8, 6], [9, 4, 3], [9, 3, 4], [4, 9, 3], [4, 3, 9], [3, 4, 9], [3, 9, 4], [9, 5, 2], [9, 2, 5], [5, 9, 2], [5, 2, 9], [2, 5, 9], [2, 9, 5], [10, 3, 3], [3, 3, 10], [3, 10, 3], [10, 4, 2], [10, 2, 4], [4, 2, 10], [4, 10, 2], [2, 4, 10], [2, 10, 4], [11, 3, 2], [11, 2, 3], [2, 11, 3], [2, 3, 11], [3, 11, 2], [3, 2, 11], [12, 2, 2], [2, 2, 12], [2, 12, 2] and $y = 66$. For simplification, we delete the compositions [12, 2, 2] and [2, 2, 12], thus, small integers c_g is within [0, 63]. If $c_g = 2$, then we retrieve the 3rd ordered partition [5, 5, 6]. We will set $d_{3i}^{(s)} = 5, d_{3i+1}^{(s)} = 5, d_{3i+2}^{(s)} = 6$, where s is the round label. Once the $d_i^{(s)}$ is determined, the $\star_s$ of each round is determined.

E. An implementation of F_2: Key-dependent bit-level permutation generation algorithm

In our implementation, $w = 256$. We chopped H_{k_0}, H_{k_1}, ..., $H_{k_{15}}$ into 32 8-bit chunks. Let f_r represents rth chunks. If $f_1 = 20$, then the digit on the 1st position will be moved to 20th position after the permutation. If f_r is a number repeated, then we ignore it and continue to test f_{r+1} until we get 256 different integers or test all f_r. If we cannot find 256 different numbers, then we fill the remaining position with an ordered sequence of those numbers not shown up within [0, 255].

In the first round, the following $p[q] = u$ is the permutation σ_1 we used, that means the qth bit will move to uth bit after the permutation. We can see there are no invariant positions.

For the $(s + 1)$th round, the integers we used to generate σ_s are H_{k_s}, $H_{k_{s+1}}$, ..., $H_{k_{15}}, H_{k_0}, H_{k_{s-1}}$.

p[0]=220, p[1]=6, p[2]=7, p[3]=14, p[4]=22, p[5]=33, p[6]=35, p[7]=36, p[8]=37, p[9]=38, p[10]=39, p[11]=51, p[12]=54, p[13]=62, p[14]=64, p[15]=65, p[16]=66, p[17]=67, p[18]=68, p[19]=69, p[20]=70, p[21]=71, p[22]=72, p[23]=73, p[24]=74, p[25]=75, p[26]=76, p[27]=77, p[28]=78, p[29]=79, p[30]=82, p[31]=86, p[32]=87, p[33]=90, p[34]=94, p[35]=95, p[36]=96, p[37]=97, p[38]=98, p[39]=99, p[40]=100, p[41]=101, p[42]=102, p[43]=103, p[44]=104, p[45]=105, p[46]=106, p[47]=107, p[48]=108, p[49]=109, p[50]=110, p[51]=111, p[52]=114, p[53]=115, p[54]=116, p[55]=117, p[56]=118, p[57]=119, p[58]=121, p[59]=122, p[60]=125, p[61]=126, p[62]=127, p[63]=131, p[64]=132, p[65]=134, p[66]=139, [67]=142, p[68]=143, p[69]=150, p[70]=151, p[71]=164, p[72]=166, p[73]=167, p[74]=170, p[75]=193, p[76]=194, p[77]=195, p[78]=196, p[79]=197, p[80]=198, p[81]=199, p[82]=202, p[83]=203, p[84]=204, p[85]=206, p[86]=211, p[87]=212, p[88]=213, p[89]=214, p[90]=215, p[91]=223, p[92]=226, p[93]=227, p[94]=228, p[95]=229, p[96]=230, p[97]=231, p[98]=234, p[99]=236, p[100]=237, p[101]=241, p[102]=242, p[103]=243, p[104]=246, p[105]=254, p[106]=255, p[107]=239, p[108]=172, p[109]=135, p[110]=210, p[111]=133, p[112]=252, p[113]=225, p[114]=12, p[115]=191, p[116]=250, p[117]=92, p[118]=163, p[119]=44, p[120]=190, p[121]=201, p[122]=200, p[123]=244, p[124]=189, p[125]=124, p[126]=141, p[127]=123, p[128]=209, p[129]=45, p[130]=149, p[131]=219, p[132]=42, p[133]=113, p[134]=129, p[135]=224, p[136]=253, p[137]=192, p[138]=235, p[139]=138, p[140]=40, p[141]=176, p[142]=15, p[143]=46, p[144]=89, p[145]=8, p[146]=165, p[147]=63, p[148]=84, p[149]=9, p[150]=188, p[151]=182, p[152]=137, p[153]=218, p[154]=140, p[155]=232, p[156]=145, p[157]=238, p[158]=245, p[159]=83, p[160]=174, p[161]=4, p[162]=11, p[163]=251, p[164]=50, p[165]=61, p[166]=146, p[167]=53, p[168]=13, p[169]=55, p[170]=34, p[171]=5, p[172]=10, p[173]=216, p[174]=171, p[175]=161, p[176]=16, p[177]=162, p[178]=205, p[179]=233, p[180]=3, p[181]=159, p[182]=155, p[183]=0, p[184]=179, p[185]=32, p[186]=112, p[187]=80, p[188]=175, p[189]=23, p[190]=169, p[191]=26, p[192]=222, p[193]=221, p[194]=148, p[195]=187, p[196]=208, p[197]=2, p[198]=43, p[199]=147, p[200]=41, p[201]=157, p[202]=31, p[203]=217, p[204]=91, p[205]=240, p[206]=60, p[207]=57, p[208]=183, p[209]=48, p[210]=177, p[211]=168, p[212]=81, p[213]=130, p[214]=158, p[215]=47, p[216]=85, p[217]=25, p[218]=207, p[219]=88, p[220]=154, p[221]=249, p[222]=59, p[223]=28, p[224]=181, p[225]=248, p[226]=49, p[227]=136, p[228]=128, p[229]=160, p[230]=17, p[231]=30, p[232]=18, p[233]=173, p[234]=19, p[235]=21, p[236]=52, p[237]=20, p[238]=93, p[239]=152, p[240]=27, p[241]=156, p[242]=29, p[243]=24, p[244]=120, p[245]=153, p[246]=178, p[247]=58, p[248]=56, p[249]=186, p[250]=180, p[251]=247, p[252]=184, p[253]=1, p[254]=185, p[255]=144.

F. Encryption and decryption

Time t' and K are used together as a credential to encrypt and decrypt a block. We transmit t' through an out-of-band channel such as text, email or personal call. We use the formula (2-3) and (2-4) to compute the cipher text and plain text.

IV. Theoretical Analysis and Experiments

A. Diffusion: avalanche effect

The diffusion requirement on a cipher is that each plaintext bit should affect every ciphertext bit and each key bit should influence every cipher bit (see ([7], [8])). Diffusion is provided by the transformation called the addition modulo 2^l, key-dependent bit-level permutation σ_s, and concatenation $\&$.

1) Avalanche effect test: Instead of providing full avalanche, SHIPHER makes two weaker guarantees that together are almost as effective:

- A change to any bit of the input will on average change half the bits of the last 256 bits of the output
- A change to any bit of the last 256 bits of the input will on average change half the bits of the output

We conducted 100 times to test the avalanche effect using one bit change in plaintext. The result is shown in Table 1 when $h = 4$. When $h = 4$, the average of $d_i^{(s)}$ is 5, 12-round encryption is enough for our avalanche requirement. When $h = 3$, 16-round encryption is better because the average of $d_i^{(s)}$ is 4.

TABLE I
AVERAGE AVALANCHE EFFECT RESULTS WHEN $h = 4$

Round	Bits
1	3
2	11
3	32
4	51
5	63
6	79
7	95
8	121
9	129
10	137
11	143
12	157
13	145
14	149
15	154
16	145

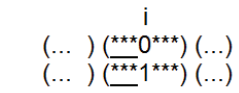

Fig. 3. The effect of a $\boxplus_l$ operation

2) Key sensitive test:: We omitted the key sensitive test because what we use is $H(K)$ instead of Key. The avalanche effect of the cryptographic hash function $H()$ guarantees the key sensitivity of SHIPHER.

3) Theoretical analysis of avalanche effect: **Theorem 1:** Assume n is a binary integer with arbitrary length, then averagely there are two different digits between $n+1$ and n. The probability of the last digit of them is 1; the probability of the second last digit of them is $1/2$; followed by $1/4$, $1/8$, ..., $1/2^i$.

The proof is trivial.

Theorem 2: After a binary integer with length less than k incrementing by 1, there are averagely $2-2^{k-1}$ bits changed.

Theorem 3 If the ith bits of n_1 and n_2 are different, the ith bit needs to do a $\boxplus_l$ operation, then for all n_3, $n_1 \star n_3$ and $n_2 \star n_3$ are the same except the first i bits in the specified subblock with length l as in figure 3.

V. SECURITY ANALYSIS

A. Brute-force attacks and algebraic attacks

Let $P'([2,2^h], h-1; 2^h)$ denote the number of order-dependent compositions of integer 2^h with at most $h-1$ parts, each of size within $[2, 2^h]$. During each round, the number of possible dynamic operators is $P'([2,2^h], h-1; 2^h)^{w/2^h}$. The possible dynamic operators is $(P'([2,2^h], h-1; 2^h)^{w/2^h})^z$ after z-round encryption. When $w = 256$, $z = 16$, $h = 4$, $p = 48$, $P'([2,2^h], h-1; 2^h)^{w/2^h}$=$66^{16}$, $(P'([2,2^h], h-1; 2^h)^{w/2^h})^z = 66^{16*12} = 66^{256}$. For simplification, we choose 64 instead of 66. The number of possible dynamic operators is 64^{256}. It is very hard to conduct the brute-force attacks. Similarly, it is also resistant to algebraic attacks, which tries to find an algebraic expression from input (plain text) to output (cipher text).

When $w = 256$, $z = 16$, $h = 3$, $p = 64$, $P'([2,2^h], h-1; 2^h)^{w/2^h}$=$5^{32}$, $(P'([2,2^h], h-1; 2^h)^{w/2^h})^z = 5^{32*16} = 5^{512}$. It is also resistant to the brute-force attacks and algebraic attacks.

B. Linear and differential attacks

Linear and differential attacks rely on the known structure of encryption scheme. For key-dependent ciphers, both attacks are inefficient because the attacker does not know the details of $\star$. Thus, it is also resistant to other attacks related to differential attacks such as Boomerang attacks [13], statistical saturation attack [14].

C. Related-key attacks

After a key getting through a cryptographic hash function such as RC4-256 or SHA2-256, even one-bit change in the key will cause a huge change in SHIPHER. Thus, SHIPHER is resistant to relate-key attacks [20].

D. Side channel attacks

Most ciphers are vulnerable to side channel attacks such as a timing attack [15] in which the attacker attempts to compromise a cryptosystem by analyzing the time taken to execute cryptographic algorithms. SHIPHER has this weakness as other ciphers, however, it can mitigate this attack because it is key-dependent and uses time as the second secret to generate 100-200 small random integers. The randomness lowers the risk to timing attacks.

E. Other attacks

It it not clear whether our cipher provide key-dependent message security [19] now because there is no existing testing methodology for this property. We believe our cipher is resistant to the key-dependent attacks [26] because the key was hashed by RC4-256 or SHA2-256 in this cipher.

F. Special cases of plain text and cipher text

If the plain text is 0x0 or 0xFF...F, the permutation turns out useless in the first round. This case will not generate weak encryption blocks. We verified this.

VI. PERFORMANCE ANALYSIS

There are many lightweight block ciphers such as SIMON and SPECK [29], AES [9], KLEIN [11], and PRESENT [27]. We have restricted our comparisons exclusively to AES. Comparing with the above block ciphers, the algebraic expression of our scheme is simple and straightforward. And the block size and the key length of our scheme is more flexible. Also, the users can choose different implementations according to their security strength requirement.

We implement SHIPHER in C language and attach the encryption and decryption codes in the Appendix. The measurements were taken on a personal computer with a 64-bit, 2.66 Ghz Intel(R) Core (TM) 2 Quad CPU Q8400. We now provide some information on the performance achieved by the SHIPHER-toolkit.

SHIPHER-enc and SHIPHER-dec run in a predictable amount of time based on the length of the key and the

TABLE II
PERFORMANCE COMPARISONS WITH OTHER LIGHT-WEIGHT CIPHERS

Algorithm	Key length (bit)	Block size (bit)	RAM (byte)	ROM (byte)	Block processing speed (ms)
AES	256	256	14028.8	38563.84	2860
SHIPHER	any	256	19251	42782	2738

number of rounds. The performance of them depends on the selected hash function and random number generation algorithm. *Short private keys are as secure as long private keys while maintaining reasonable running times*. It is as fast as AES-256 and slower than KLEIN-96 and PRESENT-80. Table 2 shows our results.

VII. EXTENSION

Our block cipher family use the cryptographic hash value of key and small random integers to construct the dynamic operators. It is a key-dependent encryption framework. A cryptosystem designer can have a different encryption scheme after changing the cryptographic hash function and the dynamic operators. This encryption framework admits a variety of implementations on random number generator.

There are several ways to extend this scheme: (1) Add other finite group operations such as multiplication over $Z_p \setminus \{0\}$, addition over $GF(2^n)$, etc; (2) Interleave these group operations with the dynamic operator we proposed in this paper; (3) Make the permutation more uniformly distribution over $[1, n]$. We will test these schemes one by one and compare them in the future.

VIII. RELATED WORK

We mention lightweight block ciphers such as SIMON and SPECK [29], AES [9], KLEIN [11], and PRESENT [27] in last section. Consequently, we don't consider other interesting lightweight stream ciphers like HUMMINGBIRD-2 [28], GRAIN [30], KATAN [10], TRIVIUM [31], and SALSA20 [32].

Our cipher is a key-dependent block cipher. There are similar key-dependent ciphers such as a modification of Hill cipher using a key dependent permutation [16], in which the key matrix is 384 bits, while our cipher accepts any-length key. Key dependent block cipher SAFER [22] uses log_{45} and 45^x arithmetic operators. Lucifer [24] is a key dependent block cipher which essentially uses 16-round Feistel network, also on 128-bit blocks and 128-bit keys. Cipher ICE [23] is a Feistel network with a block size of 64 bits, which also uses key-dependent bit permutation in the round function as in our design. It round function is different from our design.

Literature [17] proposed five key-dependent s-box using a matrix, which is different from our scheme. Literature [18] proposed a key-dependent s-box for AES using a permutation of all possible 256 8-bit elements of $GF(2^8)$. Literature [25] proposed a DNA based key-dependent ShiftRows transformation approach for AES. Another key dependent S-boxes based on 2D logistic map and 2D cross map was proposed in literature [21]. SHIPHER is different from all of them because it does not use S-boxes and the chunk length varies while the length of S-boxes is fixed in other block ciphers.

IX. CONCLUSIONS AND FUTURE WORK

We created a symmetric block cipher framework based on the design concept of a customized operator –"mixing 2 different group operations and random number generation operations"– to achieve the required confusion and diffusion. Confusion is achieved by arranging the operations in a way that no pair of successive operations are of the same type and by the fact that operations of different types are bit-level. The diffusion can be achieved using 16 different bit-level key-dependent permutations. Encryption and decryption are essentially the same process with different customized operators. Since some implementation of SHIPHER uses of an 8-bit-less regular modular addition structure, the SHIPHER can be implemented efficiently in both memory-restrained hardware (e. g. microcontroller) and software. The security of the proposed cipher needs further intensive investigation. We hereby invite interested partied to attack this proposed cipher and will be grateful ro receive the results of any such attacks. In the future, it would be interesting to consider these new encryption families as encryption schemes for big data and disk section encryption because it is highly efficient.

REFERENCES

[1] X. Lai and J. L. Massey, "A Proposal for a New Block Encryption Standard", EUROCRYPT 1990, pp. 389 - 404, 1990.
[2] S. Su, S. Lü, and D. Dong, "A 128-bit block cipher based on three group arithmetics", http://eprint.iacr.org/2014/704.pdf.
[3] S. Patil, "An enhancement in international data encryption algorithm for increasing security", Intl. J. of Application or Innovation in Engineering & Management, vol.3, issue 8, pp. 64 - 70, 2014.
[4] X Hei, B Song, "SHipher: Families of Block Ciphers based on SubSet-Sum Problem", IACR Cryptology ePrint Archive 103, 2014.
[5] 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.
[6] S. Harris and C. Adams, "Key-dependent S-Box Manipulations", in: S. Tavares and H. Meijer (Eds.): SAC'98, LNCS vol. 1556, pp. 15 - 26, 1999.
[7] C. E. Shannon, "Communication Theory of Secrecy Systems", B. S. T. J., Vol. 28, pp. 656 - 715, Oct. 1949.
[8] J. L. Massey, "An Introduction to Contemporary Cryptology", Proc. IEEE, Vol. 76, No. 5, pp. 533 - 549, May 1988.
[9] J. Daemen and V. Rijmen, "The design of Rijndeal", Springer, Berlin, 2002.
[10] 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.
[11] 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.
[12] A. F. Webster and S. E. Tavares, "On the design of s-boxes", in Advances in Cryptology: Proc. of CRYPTO'85, Springer-Verlag, Berlin, pp. 523 - 534, 1986.
[13] D. Wagner, "The Boomerang Attack", presented at 6th International Workshop on Fast Software Encryption (FSE'99), Springer-Verlag, pp. 156 - 170, Rome, 1999.
[14] B. Collard and F.-X. Standaert, "A Statistical Saturation Attack against the Block Cipher PRESENT", in: Topics in Cryptology - CT-RSA 2009, vol. 5473 of LNCS, pp. 195 - 210, 2009.
[15] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", in Proc. of CRYPTO'96, Springer-Verlag, pp. 104 - 113, 1996.

[16] V. U. K. Sastry, N. R. Shankar, and S. D. Bhavani, "a large block cipher involving key dependent permutation, interlacing and iteration", Bulgarian academy of sciences - Cybernetics and information technologies, vol. 13, no.3, pp. 50 - 63, 2013.
[17] S. S. M. Aldabbagh, I. F. T. Shaikhli, and M. R. Zaba, "Key-dependent s-box in lightweight block ciphers", J. of Theoretical and Applied Information Technology, vol. 62, no. 2, pp. 554 - 559, April 2014.
[18] A. Fahmy, M. Shaarawy, K. El-Hadad, G. Salama, and K. Hassanain, "A proposal for a key-dependent AES", in Proc. of 3rd International Conference: Sceiences of Electronic, Technologies of Information and Telecommunications, Tunisia, March 27 - 31, 2005.
[19] B. Barak, I. Haitner, D. Hofheinz, and Y. Ishai, "Bounded key-dependent message security", in: Advances in Cryptology C EUROCRYPT 2010, pp. 423 - 444, 2010.
[20] F. Böhl, G. T. Davis, and D. Hofheinz, "Encryption schemes secure under related-key and key-depedent message attacks", in: Public-Key Cryptography C PKC 2014, vol. 8383 of LNCS, pp. 483 - 500, 2014.
[21] F. J. Lumal, H. S. Hilal, and A. Ekhlas, "New Dynamical Key Dependent S-Box based on chaotic maps", in: IOSR J. of Computer Engineering (IOSR-JCE), vol. 17, issue 4, pp. 91 - 101, 2015.
[22] J. L. Massey, "SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm", in Proc of Conference Fast Software Encryption, pp. 1 - 17, 1993.
[23] M. Kwan, "The Design of the ICE Encryption Algorithm", in Proc of Conference Fast Software Encryption, pp. 69 C 82, 1997.
[24] A. Sorkin, "LUCIFER: a cryptographic algorithm", Cryptologia, vol. 8, no. 1, pp. 22 C 35, 1984.
[25] A. H. Al-Wattar, R. Mahmod, Z. A. Zukarnain3, and N. I. Udzir4, "A new DNA based appoach of generating key-depedent shiftrows transformation", arXiv:1502.03544 [cs.CR].
[26] X. Sun and X. Lai, "The key-dependent attack on block ciphers", In: Advances in Cryptology C ASIACRYPT 2009, vol. 5912 of LNCS, pp. 19 - 36, 2009.
[27] 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 - 66, Springer-Verlag, 2007.
[28] D. Engels, M. Saarinen, and E. Smith, "The Hummingbird-2 lightweight authenticated encryption algorithm", in Cryptology ePrint Archive, Report 2011/126, 2011.
[29] 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.
[30] M. Hell, T. Johansson, A. Maximov, and W. Meier, "The Grain Family of Stream Ciphers", in New Stream Cipher DesignsłThe eSTREAM Finalists, Lecture Notes in Compter Science, no. 4986, pp. 179 - 190, Springer-Verlag, 2008.
[31] C. D. Canniere and B. Preneel, "TRIVIUM Specifications", in ECRYPT Stream Cipher Project Report 2005/030, 2005.
[32] D. J. Bernstein, "The Salsa20 Family of Stream Ciphers", in New Stream Cipher DesignsłThe eSTREAM Finalists, Lecture Notes in Compter Science, no. 4986, pp. 84 - 97, Springer-Verlag, 2008.

APPENDIX A: PROOF OF BIT INDEPENDENCE CRITERION

The bit independence criterion (BIC) states that bits j and k of output should change independently when any single input bit i is inverted, for all i, j and k [12].

Theorem 4: Our scheme satisfies BIC.

1) Suppose A and B have the same length n and $(n-1)$ same binary digits.
2) $\star$ is an operator, its distribution of length of each chunk (subblock) is preset or can be calculated from all the possible partitions.
3) σ is a random permutation.
4) x_i is a n-bit binary number with digits random uniformly distributed in [0, 1].

Theorem 5: Assume $A^{(i)} = x_{i-1} \star \sigma(A^{(i-1)})$, $B^{(i)} = x_{i-1} \star \sigma(B^{(i-1)})$, where $A^0 = A$, and $B^0 = B$, then the 0 and 1 of $A^{(i)}$ and $B^{(i)}$ $(i \geq 1)$ are uniformly distributed.

Proof: 1 The sum of any number and a number with digits uniformly distributed in $\{0, 1\}$ is with digits uniformly distributed in $\{0, 1\}$.

2 The concatenation of several numbers with digits uniformly distributed in $\{0, 1\}$ is with digits uniformly distributed in $\{0, 1\}$.

Theorem 6: Assume the probability of $A^{(i)}$ and $B^{(i)}$ have the same bits is p_i, then the relationships between p_{i+1} and p_i are as follows:

1) $p_i < p_{i+1} < 1/2$, if $p_i < 1/2$
2) $p_i > p_{i+1} > 1/2$, if $p_i > 1/2$
3) $p_i \to 1/2, i \to \infty$

Theorem 6 guarantees that the probabilities of 0 and 1 in a number at last are 1/2 whatever the original probabilities are.

Theorem 7: For all different i, j, k, then

$$P(A_{[j]}^{(t)} = B_{[j]}^{(t)}, A_{[k]}^{(t)} = A_{[k]}^{(t)} \| A_{[i]}^{(t)} = A_{[i]}^{(t)}) \to 1/4 \tag{IX-5}$$

$$P(A_{[j]}^{(t)} \neq B_{[j]}^{(t)}, A_{[k]}^{(t)} \neq A_{[k]}^{(t)} \| A_{[i]}^{(t)} = A_{[i]}^{(t)}) \to 1/4 \tag{IX-6}$$

According to Theorem 6, we only need to prove that the probabilities of 0 and 1 at any two positions are irrelative.

That means, we only need to prove that

$$P(A_{[j]}^{(t)} = B_{[j]}^{(t)} | A_{[i]}^{(t)} = A_{[i]}^{(t)}) \to 1/2. \tag{IX-7}$$

Assume that $P(A_{[j]}^{(t)} = B_{[j]}^{(t)} | A_{[i]}^{(t)} = A_{[i]}^{(t)}) = q_{i,j,t}$, that means when t is large enough, $P(A_{[j]}^{(t)} = B_{[j]}^{(t)}, A_{[i]}^{(t)} = A_{[i]}^{(t)}) = 1/2 q_{i,j,t}$. Let the i', j' are the positions of i, j after permutation.

Then $P(A_{[i']}^{(t+1)} = B_{[i']}^{(t+1)}, A_{[j']}^{(t+1)} = B_{[j']}^{(t+1)})$ can be calculated as in the proof of Theorem 6.

We can also prove that: (1) If i', j' are in a chunk, then the maximum decreases and the minimum increases; (2) If i', j' are not in a chunk, the probabilities will approach $1/4$ due to the uniformly distribution.

Proof:

Suppose m bits of $A^{(i)}$ and $B^{(i)}$ are the same, and the other $n-m$ bits are different. Also the indices (positions) of the m bits are uniformly distributed. After operation $\star_i$, the number of the same bits in $A^{(i+1)}$ and $B^{(i+1)}$ can be calculated by the following rule:

We select a $\boxplus_l$ operator in a $\star_i$ to do the analysis: 1) The probability of the 0th bit is the same is p_i, after the $\boxplus_l$ operation, this probability is the same;

2) The probability of the 0th bit is different is $1-p_i$, after the $\boxplus_l$ operation, this probability is the same; however, it affects the previous bit due to a carrier with the probability of $(1-p_i)/2$.

3) The probability of the tth $(0 < t < l)$ bit does not change is p_i, assume the the probability of carrier is 1 at $(t-1)$th bit is α_{t-1}, then the probability of A^t and B^t stay the same is as follows:

$T_i = p_i(1-\alpha_{t-1}) + (1-p_i)\alpha_{t-1}$

The probability of there is a carrier is as follows:

$S_i = 1/2 p_i \alpha_{t-1} + 1/2(1-p_i)(1-\alpha_{t-1}) \leq 1/2$

4) In a summary,

$$T_i = p_i + (1-2p_i)\alpha_{t-1} = \begin{cases} > p_i, & p_i < 1/2 \ ; \\ < p_i, & p_i > 1/2 \ ; \\ = p_i, & p_i = 1/2 \ . \end{cases}$$

If there is one chunk (subblock) with the length longer than 1, then Theorem 2 is true. Since any d_i is greater than 1, then Theorem 2 is true. We can simulate the contraction speed on a computer as well. Table 1 shows the statistical results about the diffusion in our experiments.