Journal of The Korea Society of Computer and Information
Vol. 23 No. 10, pp. 111-117, October 2018

www.ksci.re.kr
https://doi.org/10.9708/jksci.2018.23.10.111

Software Implementation of Lightweight Block Cipher CHAM for Fast Encryption

Taeung Kim*, Deukjo Hong**

Abstract

CHAM is a lightweight block cipher, proposed in ICISC 2017. CHAM-n/k has the n-bit block and the k-bit key, and designers recommend CHAM-64/128, CHAM-128/128, and CHAM-128/256. In this paper, we study how to make optimal software implementation of CHAM such that it has high encryption speed on CPUs with high computing power. The best performances of our CHAM implementations are 1.6 cycles/byte for CHAM-64/128, 2.3 cycles/byte for CHAM-128/128, and 3.8 cycles/byte for CHAM-128/256. The comparison with existing software implementation results for well-known block ciphers shows that our results are competitive.

▸Keyword: Block cipher, Lightweight cryptography, CHAM, Fast software encryption, SIMD

I. Introduction

블록 암호 알고리즘(Block Cipher)은 현대의 암호 시스템에서 데이터의 기밀성 제공을 위한 핵심 요소로 사용되고 있다. AES(Advanced Encryption Standard)[5]는 ISO/IEC 및 FIPS 표준으로서, 가장 잘 알려져 있고 전세계에서 가장 많이 사용되고 있는 암호 알고리즘이다. 그렇기 때문에, AES를 하드웨어 또는 소프트웨어 환경에서 효율적으로 구현할 수 있는 방법이 많이 연구되어 있다[7].

그러나, 가용자원이 제한적인 경량(Lightweight) 환경에서는 AES가 상대적으로 효율적인 성능을 갖지 못함이 밝혀졌고, 최근 십수년간 경량 환경에서의 효율적인 암호화를 위하여 경량 블록 암호 알고리즘들이 연구 및 개발되어 왔다[1, 2, 4, 8, 9, 10]. CHAM[1]은 2017년에 국가보안기술연구소에서 개발한 경량 블록 암호이며, CHAM의 개발자들은 경량 환경에서 우수한 암호화 성능을 갖는다는 것을 입증하였다. n비트 블록과 k비트 키를 갖는 CHAM은 CHAM-n/k로 표기 되는데, CHAM-64/128, CHAM-128/128, CHAM-128/256의 세 가지 버전이 개발되어 있다. 또한 경량 구현을 위하여 보편적이며 간단한 산술 연산들인 덧셈, 로테이션, XOR로 구성되는 ARX 구조가 설계에 응용되었다.

IoT 개념에서는 다양한 경량 기기들(Lightweight Devices)이 네트워크에 참여하므로, 기기간 암호 통신에 경량 블록 암호를 사용하는 것이 적합하다. 그런데, 기기와 서버간의 다대일 통신 또한 발생할 수 있으며, 서버는 다양한 정보를 수신 받아 안전하게 관리해야할 수도 있기 때문에, 경량 블록 암호가 서버와 같은 고성능 컴퓨팅 환경에서 사용될 수 있다.

이와 같은 이유로, 본 논문에서는 경량 블록 암호 알고리즘 CHAM을 경량 환경이 아닌 일반 PC 이상의 고성능 환경에서 최대 속도 암호화를 제공할 수 있도록 구현하는 방법을 연구한다. 이 연구는 두 가지 방향으로 진행되었는데, 첫 번째는 단일 데이터 블록을 최대 속도로 암호화하는 방식이고 두 번째는 복수의 연속된 데이터 블록을 동시에 암호화하는 방식이다. 두 번째 방법의 경우 ECB와 CTR 암호화 모드에만 적용가능하지만, CTR 모드가 효율성 면에서 다른 암호화 모드에 비해 많은 장점을 갖기 때문에 의미가 있다.

세 가지 버전의 CHAM 알고리즘에 대하여 단일 블록 고속 암호화(FSBE: Fast Single-Block Encryption)를 위한 최적 구현에는 임시 변수 등 필요 없는 연산의 최소화, 연산의 지연 시간 최소화 등이 적용되었다. 그리고, 복수 블록 고속 암호화(FMBE: Fast Multi-Block Encryption)를 위한 최적 구현에는 CPU에서 지원되는 SIMD(Single Instruction Multiple Data)를 이용한 벡터 프로세싱이 적용되었다.

• First Author: Taeung Kim, Corresponding Author: Deukjo Hong
*Taeung Kim (giantbearkim@naver.com), Division of Business Administration, Chonbuk National University
**Deukjo Hong (deukjo.hong@jbnu.ac.kr), Dept. of Information & Technology Engineering, Chonbuk National University
• Received: 2018. 07. 30, Revised: 2018. 09. 27, Accepted: 2018. 10. 05.
• This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.B0722-16-0006, Cross layer design of cryptography and physical layer security for IoT networks)

본 연구의 구현 환경은 Intel Core i5 Skylake 2.3GHz 이며, CHAM-64/128, CHAM-128/128, CHAM-128/256에 대한 우리의 FSBE 구현의 암호화 속도는 각각 43.5, 12.1, 14.2 cycles/byte이다. CHAM-64/128, CHAM-128/128, CHAM-128/256에 대한 우리의 FMBE 구현의 암호화 속도는 각각 1.6, 2.3, 3.8 cycles per byte로서, FSBE 구현에 비해 약 27.2배, 5.3배, 3.7배 향상되었다. 기존의 다른 경량 암호 알고리즘의 고속 암호화 구현 결과와 비교함으로써, 우리의 방법은 CHAM에 대해 적용할 수 있는 최고 수준의 고속 암호화 구현 기법임을 확인할 수 있었다(Table 2 참고).

본 논문의 나머지 부분은 다음과 같이 구성되어 있다. II절에서는 본문의 이해에 필요한 용어 및 기호가 설명되고, III절에서는 CHAM 알고리즘의 구조가 간단하게 기술된다. IV절에서는 CHAM의 FSBE 최적 구현에 대해 설명되고, V절에서는 CHAM의 FMBE 최적 구현에 대해 설명된다. VI절에서는 구현 결과를 비교하고, VII절에서 본 논문의 결론이 제시된다.

II. Definitions and Notations

본 논문에서 사용되는 표기법 및 연산 기호 등을 여기서 정의한다. CHAM 알고리즘의 근본적인 개발목적은 Lightweight에서의 구현이다. 때문에 복잡한 산술연산이 아닌 bit연산을 위주로 구현된다.

- $x \| y$: 비트열 x와 비트열 y의 연결
- $x \boxplus y$: n비트 값 x와 y의 법 2^n 덧셈
- $x \oplus y$: n비트 값 x와 y의 배타적 논리합
- $ROL_i(x)$: n비트 값 x에 대한 i 비트 좌순환시프트

III. Block Cipher CHAM

1. Brief Description of CHAM

n비트 평문 블록과 k비트 키를 입력 받아 n비트 암호문 블록을 출력하는 블록 암호 CHAM 알고리즘을 CHAM-n/k로 표기한다. n과 k는 각각 평문 블록과 키의 비트 수를 의미한다. r은 각 cipher의 라운드 수를 의미한다. w는 워드 크기로서, 평문 블록 비트 수의 1/4을 의미한다 (Table 1 참고).

Table 1. Parameters of CHAM

cipher	n	k	r	w	k/w
CHAM-64/128	64	128	80	16	8
CHAM-128/128	128	128	80	32	4
CHAM-128/256	128	256	96	32	8

평문 블록은 $P = X_0[0] \| X_0[1] \| X_0[2] \| X_0[3]$로 표기되며 이것은 첫 번째 라운드(Round)의 입력이 된다. 각 i 번째 라운드의 입력과 출력은 각각 $Xi = X_i[0] \| X_i[1] \| X_i[2] \| X_i[3]$과 $X_i+1 = X_{i+1}[0] \| X_{i+1}[1] \| X_{i+1}[2] \| X_{i+1}[3]$으로 표기된다($0 \leq i < r$).

RK[i]는 각 라운드에 사용되는 라운드 키(Round Key)를 의미하며, 이것은 키스케줄(Key Schedule) 함수가 키 입력으로부터 생성한다.

CHAM의 암호화 형식은 Fig. 1과 같다.

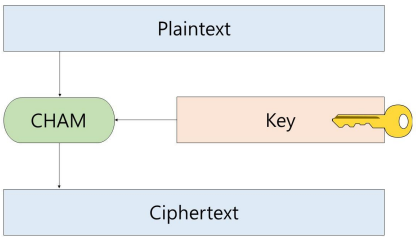

Fig. 1. Encryption Process of CHAM

CHAM은 Round 함수와 각 round에 활용되는 Round Key를 생성하는 Key_Schedule 함수로 구성 된다. RK(round key)는 입력받은 K(secret key)를 활용하여 만든다. CHAM에서 RK는 K 개수의 2배에 해당하는 개수로 생성된다. K[i]를 통해 RK[i]와 RK[(i + k/w)⊕1]를 생성한다. 각 RK를 만드는 연산은 다음과 같다.

For $0 \leq i < k/w$,
$RK[i] \leftarrow K[i] \oplus ROL_1(K[i]) \oplus ROL_8(K[i])$
$RK[(i+k/w) \oplus 1] \leftarrow K[i] \oplus ROL_1(K[i]) \oplus ROL_{11}(K[i])$

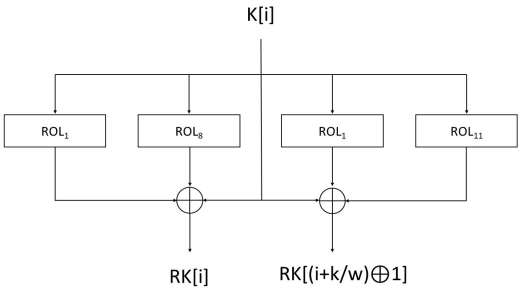

Fig. 2. Key schedule of CHAM

CHAM의 암호화 함수에서 각 라운드는 홀수와 짝수가 구분되어 실행된다. 홀수 라운드의 연산은 다음과 같다.

For $0 \leq j \leq 2$,
$X_{i+1}[3] \leftarrow ROL_8((X_i[0] \oplus i) \boxplus (ROL_1(X_i[1]) \oplus RK[i \bmod 2k/w]))$
$X_{i+1}[j] \leftarrow X_i[j+1]$

그리고, 짝수 라운드의 연산은 다음과 같다.

For $0 \le j \le 2$,

$X_{i+1}[3] \leftarrow ROL_8((X_i[0] \oplus i) \boxplus (ROL_1(X_i[1]) \oplus RK[i \bmod 2k/w]))$

$X_{i+1}[j] \leftarrow X_i[j+1]$

CHAM의 첫 번째 라운드의 인덱스 i 값은 0으로서, 짝수 라운드를 먼저 시작한다.

CHAM의 암호화 함수는 라운드 함수를 r번 반복 적용함으로써 평문 블록을 암호문 블록으로 변환한다.

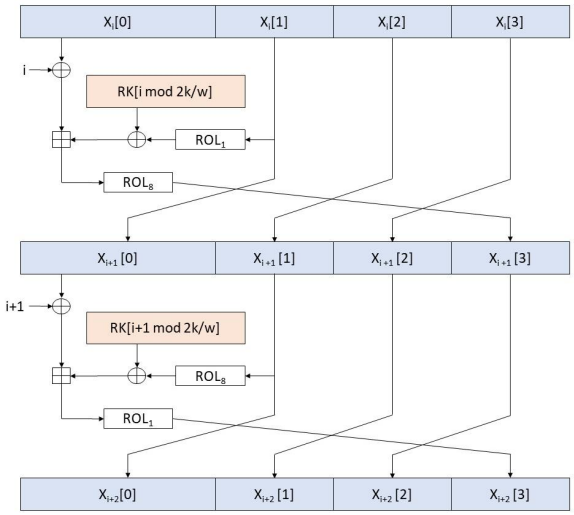

Fig. 3. i-th and (i+1)-th Round functions of CHAM

2. Encryption Process of CHAM

[RK 생성(K∈{0,1})] 비밀 키, 라운드 키 생성

- 비밀키의 개수는 각 모드의 k크기를 w크기로 나눈 것이다. 따라서 각 모드별로 키는 8개, 4개, 8개로 구성된다.
- 비밀키를 이용하여 RK를 생성한다. 이때에 각 비밀키 별로 같은 bit크기의 2개 RK가 생성되므로 각 모드별 RK는 16개, 8개, 16개가 생성된다.

[암호화(RK, m∈{0,1})] 평문 m에 대한 암호문 생성

- m를 w크기로 나누어 X[0] ~ X[3]을 구성한다.
- i = 라운드, i에 따라 짝수, 홀수 라운드 구분 및 RK[i mod 2k/w]사용한다.
- 각 라운드에 적합한 계산을 통해 새로운 X[3]을 구성한다. 이후 X[0] ~ X[2]는 기존의 X[1] ~ X[3]을 이동시켜 사용한다.

IV. Optimal FSBE Implementation of CHAM Encryption Algorithms

1. FSBE Implementation of CHAM-64/128

평범한 CHAM 구현에서는 반복문이 사용되며, 반복되는 라운드 함수 실행을 위해 라운드 순서의 홀수와 짝수여부 판단, $X_i[0]$와 $X_i[1]$의 연산을 통해 새롭게 계산되는 변수 $X_{i+1}[3]$값의 저장, 임시변수를 이용한 다음 라운드 입력 값 준비를 위한 변수 값 이동 등의 구현 기법이 적용되어야한다.

이와 같은 이유로, CHAM-64/128에 대한 우리의 FSBE(Fast Encryption of Single Block) 구현에서는 반복문을 사용하지 않았다. 이를 통해 논리연산, 변수이동, 임시변수 등에 소요되는 연산들을 최소화할 수 있었다. Fig. 4와 같은 연속된 4 라운드 구조는 최소화된 연산으로 반복될 수 있다. CHAM-64/128의 라운드 수는 80이기 때문에 이것이 20번 반복되면 암호문 블록이 생성된다. 이 4 라운드 연산에 대한 구현 코드는 다음과 같다.

```
void CHAM_64_128_Encryption(unsigned short
RK[16],unsigned short X[4],unsigned short C[4]) {

X[0]=ROL(((X[0]^0)+(ROL(X[1],1)^(RK[0]))),8);
X[1]=ROL(((X[1]^1)+(ROL(X[2],8)^(RK[1]))),1);
X[2]=ROL(((X[2]^2)+(ROL(X[3],1)^(RK[2]))),8);
X[3]=ROL(((X[3]^3)+(ROL(X[0],8)^(RK[3]))),1); }
```

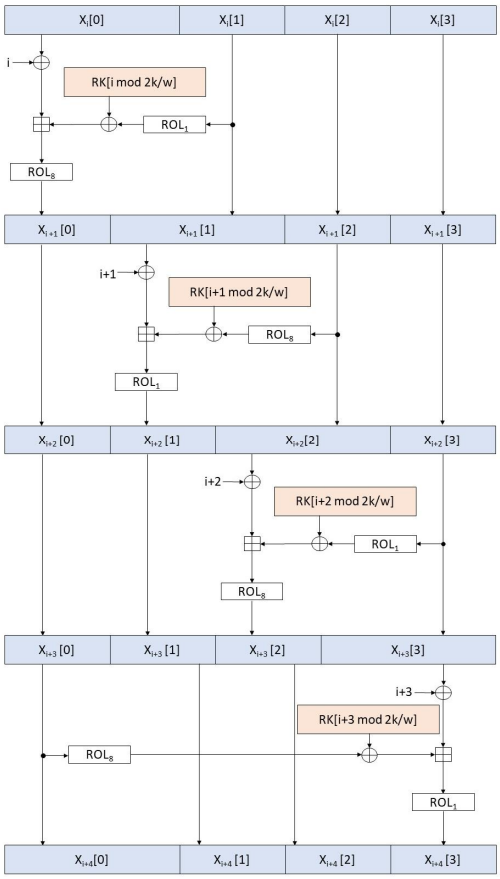

Fig. 4. Four rounds of CHAM

2. FSBE Implementation of CHAM-128/128 and CHAM-128/256

블록 길이가 128비트인 경우, 워드 크기가 16비트에서 32비트로 증가했다는 점을 제외하면 FSEB 구현에 사용되는 개념은 CHAM-64/128과 같다.

3. FSBE Process of CHAM

[RK 생성(K∈{0,1})] 비밀 키, 라운드 키 생성

- CHAM의 기본 구현 방식과 동일하다.

[암호화(RK, m∈{0,1})] 평문 m에 대한 암호문 생성

- m를 w크기로 나누어 X[0]~X[3]을 구성한다.
- i = 라운드, i에 따라 짝수, 홀수 라운드 구분 및 RK[i mod 2k/w]사용한다.
- 반복문이 없기 때문에 X[0] ~ X[3]을 이동시키는 과정이 생략된다. 기존의 과정은 새로운 값은 항상 X[3]에 입력되었지만, FSBE에서는 이동과정이 생략되고, 그대로 라운드별 계산이 수행된다.

V. Optimal FMBE Implementation of CHAM Encryption Algorithms

1. FMBE Implementation of CHAM-64/128

1.1 SIMD Operations

FMBE를 위해 사용한 SIMD(Single Instruction Multiple Data) 연산을 먼저 설명한다. CPU에서 지원하는 일종의 명령어 set으로 CPU에 따라 다른 결과 값과 지원 여부가 다르다. SIMD는 다수의 데이터를 한 번의 벡터 연산으로 처리할 때 주로 사용된다. SIMD 연산에는 어셈블리 언어를 이용하는 방법, intrinsic 함수를 이용하는 방법, Vector class를 이용하는 방법 등이 있다. 그중 SIMD연산을 위해 이번 구현에서 사용한 방법은 Vector class이다.

Vector class에서 변수는 선언된 Vector 변수를 설명한다. 여기서 사용되는 변수는 2가지 형태이다. 먼저 'Iu16vec8 X'은 각각 알파벳이 정수(integer), unsigned, 16bit, vector, 8개를 의미한다. 즉 unsigned short X[8]와 같은 의미이다. 같은 맥락으로 'Iu32vec4 X'는 unsigned int X[4]와 같은 의미이다. xmm 레지스터의 크기가 128bit이기 때문에 각각 벡터 요인은 16bit 8개, 32bit 4개로 구성된다. Vector class를 사용하면 Vector를 하나의 변수로 계산이 가능하다는 이점이 있다. 이를 위해서는 선언된 Vector 변수에 각각 적합한 변수를 넣어야한다.

CHAM-64/128에서 vX을 만드는 과정은 다음과 같다. 먼저 원하는 개수의 Vector를 생성한다. CHAM-64/128의 경우 4개의 w가 필요하기 때문에 vX도 4개를 생성한다. w의 크기가 16bit이기 때문에 8개 블록의 w를 각각 Vector요소로 넣어준다.

```
Iu16vec8 vX[4];

vX[0][0] = X0[0];          vX[2][0] = X0[2];
vX[0][1] = X1[0];          vX[2][1] = X1[2];
vX[0][2] = X2[0];          vX[2][2] = X2[2];
vX[0][3] = X3[0];          vX[2][3] = X3[2];
vX[0][4] = X4[0];          vX[2][4] = X4[2];
vX[0][5] = X5[0];          vX[2][5] = X5[2];
vX[0][6] = X6[0];          vX[2][6] = X6[2];
vX[0][7] = X7[0];          vX[2][7] = X7[2];

vX[1][0] = X0[1];          vX[3][0] = X0[3];
vX[1][1] = X1[1];          vX[3][1] = X1[3];
vX[1][2] = X2[1];          vX[3][2] = X2[3];
vX[1][3] = X3[1];          vX[3][3] = X3[3];
vX[1][4] = X4[1];          vX[3][4] = X4[3];
vX[1][5] = X5[1];          vX[3][5] = X5[3];
vX[1][6] = X6[1];          vX[3][6] = X6[3];
vX[1][7] = X7[1];          vX[3][7] = X7[3];
```

CHAM-128/128, CHAM-128/256에서 vX을 만드는 과정은 다음과 같다. CHAM-64/128의 경우와 마찬가지로 4개의 w가 필요하기 때문에 vX도 4개를 생성한다. 이후 w의 크기가 32bit이기 때문에 4개 블록의 w를 각각 Vector 원소로 넣어준다.

```
Iu32vec4 vX[4];

vX[0][0] = X0[0];          vX[2][0] = X0[2];
vX[0][1] = X1[0];          vX[2][1] = X1[2];
vX[0][2] = X2[0];          vX[2][2] = X2[2];
vX[0][3] = X3[0];          vX[2][3] = X3[2];

vX[1][0] = X0[1];          vX[3][0] = X0[3];
vX[1][1] = X1[1];          vX[3][1] = X1[3];
vX[1][2] = X2[1];          vX[3][2] = X2[3];
vX[1][3] = X3[1];          vX[3][3] = X3[3];
```

1.2 SIMD-based FMBE Implementation

CHAM-64/128은 워드 크기가 16 비트이다. SIMD 구현에서는 128 비트 xmm 레지스터를 이용하여 8개의 워드 값을 동시에 처리할 수 있다.

각 라운드에서 사용되는 상수도 Vector class로 묶어 연산한다. Fig. 5에서 볼 수 있듯이 8 블록의 라운드 함수 연산(RF)이 병렬적으로 처리된다. CHAM-64/128에 대한 FMBE 개념의 4 라운드 구현 코드는 다음과 같다. 다만 FSBE와 다르게 상수값 또한 저장된 워드 값의 수에 맞는 벡터로 만들어 계산해야 한다.

```
void CHAM_64_128_Encryption(Iu16vec8 vRK[16],Iu16vec8 vX[4],Iu16vec8 vC[4]){

   vX[0]=ROL(((vX[0]^vi[0])+(ROL(vX[1],1)^(vRK[0]))),8);
   vX[1]=ROL(((vX[1]^vi[1])+(ROL(vX[2],8)^(vRK[1]))),1);
   vX[2]=ROL(((vX[2]^vi[2])+(ROL(vX[3],1)^(vRK[2]))),8);
   vX[3]=ROL(((vX[3]^vi[3])+(ROL(vX[0],8)^(vRK[3]))),1);
}
```

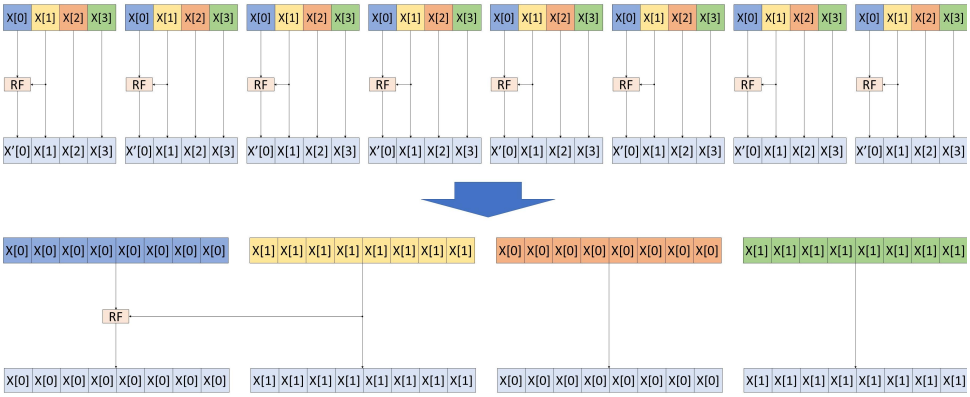

Fig. 5. FMBE implementation of CHAM-64/128

2. FMBE Implementation of CHAM-128/128 and CHAM-128/256

블록 크기가 128비트인 경우, 워드 크기가 32비트이므로 SIMD 구현에서는 128비트 xmm 레지스터로 4개의 블록을 동시에 처리할 수 있다 (Fig. 6 참고). CHAM-128/128 및 CHAM-128/256에 대한 FMBE 개념의 4 라운드 구현 코드는 다음과 같다.

```
void CHAM_128_128_Encryption(lu32vec4 vRK[8],lu32vec4
vX[4],lu32vec4 vC[4]){

  vX[0]=ROL(((vX[0]^vi[0])+(ROL(vX[1],1)^(vRK[0]))),8);
  vX[1]=ROL(((vX[1]^vi[1])+(ROL(vX[2],8)^(vRK[1]))),1);
  vX[2]=ROL(((vX[2]^vi[2])+(ROL(vX[3],1)^(vRK[2]))),8);
  vX[3]=ROL(((vX[3]^vi[3])+(ROL(vX[0],8)^(vRK[3]))),1);
}
```

```
void CHAM_128_256_Encryption(lu32vec4 vRK[8],lu32vec4
vX[4],lu32vec4 vC[4]){

  vX[0]=ROL(((vX[0]^vi[0])+(ROL(vX[1],1)^(vRK[0]))),8);
  vX[1]=ROL(((vX[1]^vi[1])+(ROL(vX[2],8)^(vRK[1]))),1);
  vX[2]=ROL(((vX[2]^vi[2])+(ROL(vX[3],1)^(vRK[2]))),8);
  vX[3]=ROL(((vX[3]^vi[3])+(ROL(vX[0],8)^(vRK[3]))),1);
}
```

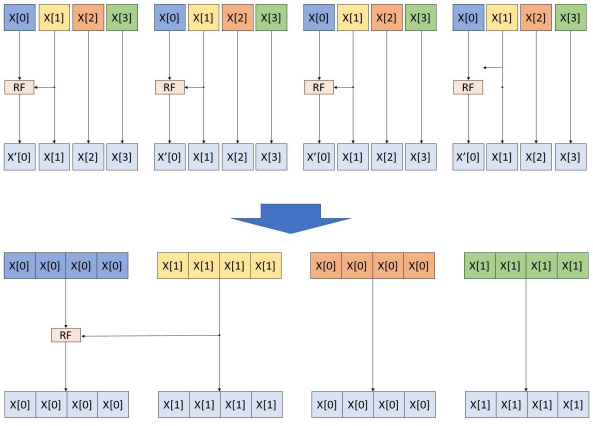

Fig. 6. FMBE implementation of CHAM-128/128 and CHAM-128/256

3. FMBE Process of CHAM

[RK 생성(K∈{0,1})] 비밀 키, 라운드 키 생성

- CHAM의 기본 구현 방식과 동일하다.

[암호화(RK, m∈{0,1})] 평문 m에 대한 암호문 생성

- m를 w크기로 나누어 X[0]~X[3]을 구성한다.
- xmm 레지스터를 사용하기 위한 Vector 변수를 선언한다. (vRK[2k/w],vX[4],vC[4])
- 이와 동시에 각 벡터 클래스에서 상수 계산을 하기 위해 상수를 xmm 레지스터 크기에 맞게 벡터로 묶어 주어야 한다. (예: vi[0][0] = vi[0][1] = … = vi[0][n] = 0. n은 벡터 원소 수이다.)
- RK도 vRK라는 벡터로 묶어서 활용해야 한다. (예: vRK[0][0] = vRK[0][1] = … = vRK[0][n] = RK[0]. n은 벡터 원소 수이다.)
- 평문 m으로 만든 X 또한 vX로 묶어 주어야한다. (예: vX[0][0] = vX[0][1] = … = vX[0][n] = RK[0]. n은 벡터 원소 수이다.)
- 이후 암호화는 CHAM의 FSBE 방식과 동일하다.

VI. Performance

본 연구에서 CHAM 알고리즘 구현은 Intel Core i5 Skylake@2.3GHz에서 수행되었으며, Dev-C++ 5.11이 구현 도구로 사용되었다. Table 2는 CHAM에 대한 우리의 구현 결과 및 기존의 다른 경량 블록 암호에 대한 고속 암호화 구현 결과를 나열하고 있다.

CHAM-64/128의 경우 단일 블록 구현에서는 암호화 속도가 CHAM의 세 가지 알고리즘 중 가장 느린 반면, 복수 블록 구현에서는 암호화 속도가 가장 빠른 것으로 관찰되었다. 그 이유는 단일 블록 암호화 구현에서는 같은 양의 데이터를 암호화하기 위해서 CHAM-64/128는 다른 두 알고리즘에 비해 더 많이 호출되어야하는 반면, 복수 블록 암호화 구현에서는 xmm 레지스터로 동시에 처리할 수 있는 블록이 상대적으로 더 많고

라운드 수가 적기 때문인 것으로 추정된다.

Table 2에서의 비교를 통해, 우리의 방법이 CHAM 알고리즘에 적용될 수 있는 최고 수준의 최적 구현 기법임을 확인할 수 있다. Table 2의 암호 알고리즘들 간의 속도 차이는 주로 알고리즘의 구조 및 특성에 의존한다.

Table 2. Performance of lightweight block ciphers (cpb : Cycles Per Byte)

Cipher	size(bits)		speed(cpb)		Ref.
	block	key	FSBE	FMBE	
CHAM-64/128	**64**	**128**	**43.5**	**1.6**	**Ours**
CHAM-128/128	**128**	**128**	**12.1**	**2.3**	**Ours**
CHAM-128/256	**128**	**256**	**14.2**	**3.8**	**Ours**
SPECK-64/128	64	128	10.1	2.4	[4]
SPECK-128/128	128	128	5.7	2.6	[4]
SPECK-128/256	128	256	6.1	2.8	[4]
SIMON-64/128	64	128	28.7	5.2	[4]
SIMON-128/128	128	128	21.6	7.5	[4]
SIMON-128/256	128	256	23.0	8.0	[4]
AES-128	128	128	11.4	6.9	[6]
LEA-128	128	128	9.3	4.2	[2]
PRESENT	64	80/128	-	5.8	[3]
Piccolo-80	64	80	-	5.7	[3]
Piccolo-128	64	128	-	6.8	[3]

VII. Conclusion

CHAM은 국내에서 개발되어 ICISC 2017에서 발표된 경량 블록 암호 알고리즘이다. 본 논문에서는 고성능 컴퓨팅 환경에서의 CHAM 고속 암호화 구현 기법에 대해 연구한 결과를 소개하였다. 기존의 다른 경량 블록 암호 알고리즘의 비슷한 환경에서의 유사한 구현 방법의 결과들의 비교에 따르면, 본 논문의 결과가 CHAM에 대해 적용할 수 있는 최고 수준의 암호화 속도를 제공한다는 것을 확인할 수 있었다.

본 논문의 연구 결과를 바탕으로 다양한 연구들이 향후에 진행될 수 있다. 본 논문에서 제시한 CHAM 구현 방법은 더욱 최적화될 여지가 있다. 각 CPU 별로 지원되는 연산들의 Latency를 고려함으로써, 가장 최적화된 구현 방법을 연구할 수 있다. 또한, 최근에 메모리 덤프, 실행 파일 역공학 등으로 소스 코드를 분석하는 공격을 차단하기 위한 화이트박스 암호구현 분야가 많이 연구되고 있는데, CHAM의 화이트박스 구현 기술 적용에 의한 효율성 저하를 최소화하기 위한 방법이 개발된다면 응용 가치가 높을 것으로 예상된다.

REFERENCES

[1] B. Koo, D. Roh, H. Kim, Y. Jung, D. Lee, D. Kwon, "CHAM: A Family of Lightweight Block Ciphers for Resource-Constrained Devices." ICISC 2017, pp. 3-25, Springer, 2017

[2] D. Hong, J.-K. Lee, D.-C. Kim, D. Kwon, K. H. Ryu, D. Lee, "LEA: A 12 8-Bit Block Cipher for Fast Encryption on Common Processors." WISA 2013, pp. 3-27, 2013

[3] S. Matsuda, S. Moriai, "Lightweight Cryptography for the Cloud: Exploit the Power of Bitslice Imple mentation." CHES 2012, pp. 408-425, 2012

[4] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, L. Wingers, "SIMON and SPECK: Block Ciphers for the Internet of Things." IACR ePrint Archive 2015/585

[5] NIST, "Advanced Encryption Standard", FIPS 197, N ovember 26th, 2011

[6] E. Käsper, P. Schwabe, "Faster and Timeing-Attack Resistant AES-GCM." CHES 2009, LNCS 5747, pp. 1-27, Springer, 2009

[7] G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti, S. Marchesin, "Efficient Software Implementation of AES on 32-bit Platforms." CHES 2002, LNCS 2523, pp. 159-171, Springer, 2002

[8] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, T. Shirai, "Piccolo: An Ultra-Lightweight Block cipher." CHES 2011, LNCS 6917, Springer, 342-357, 2011

[9] C. D. Canniere, O. Dunkelman, M. Knezevic, "KATA N and KTANTAN - a family of small and efficient hardware-oriented block ciphers." CHES 2009, LNCS 5747, pp. 272-288, Springer, 2009.

[10] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, S. Chee, "HIGHT: A new block cipher suitable for low-resource device." CHES 2006, LNCS 4249, pp. 46-59, Springer-Verlag, 2006.

Authors

Taeung Kim is currently an undergraduate student in division of business administration at Chonbuk National University. He is interested in business administration, computer engineering, and cryptography.

Deukjo Hong received B.S. and M.S. degrees in mathematics from Korea University in 1999 and 2002, respectively, and Ph.D. degree in information security from Korea University in 2006. He is currently an assistant professor of department of information technology & engineering at Chonbuk National University. He is interested in cryptography and network security.