Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.2, Apr. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.2.1

32-bit RISC-V상에서의 LEA 경량 블록 암호 GCM 운용 모드 구현*

엄 시 우,1† 권 혁 동,[1] 김 현 지,[1] 양 유 진,[1] 서 화 정[2‡]
[1,2]한성대학교 (대학원생, 교수)

Implementation of LEA Lightwegiht Block Cipher GCM operation mode on 32-bit RISC-V*

Si-Woo Eum,[1†] Hyeok-Dong Kwon,[1] Hyun-Ji Kim,[1] Yu-Jin Yang,[1] Hwa-Jeong Seo[2‡]
[1,2]Hansung University (Graduate student, Professor)

요 약

LEA는 2013년 국내에서 개발된 경량 블록암호이다. 본 논문에서는 블록 암호 운용 방식 중 CTR 운용 모드와 CTR 운용 모드를 활용하며 기밀성과 무결성을 제공하는 GCM 운용 모드의 구현을 진행한다. LEA-CTR의 최적화 구현은 CTR 운용 모드의 고정된 Nonce 값의 특성을 활용하여 사전 연산을 통한 연산 생략과 State 고정을 통해 State 간의 이동을 생략한 최적화 구현을 제안한다. 또한 제안 기법을 GCM 운용 모드에 적용 가능함을 보여주며, Galois Field(2^{128}) 곱셈 연산을 사용하는 GHASH 함수 구현을 통해 GCM 구현을 진행한다. 결과적으로 32-bit RISC-V상에서 제안하는 기법을 적용한 LEA-CTR의 경우 기존 연구 대비 2%의 성능 향상을 확인하였으며, 추후 다른 연구에서 성능 지표로 사용될 수 있도록 GCM 운용 모드의 성능을 제시한다.

ABSTRACT

LEA is a lightweight block cipher developed in Korea in 2013. In this paper, among block cipher operation methods, CTR operation mode and GCM operation mode that provides confidentiality and integrity are implemented. In the LEA-CTR operation mode, we propose an optimization implementation that omits the operation between states through the state fixation and omits the operation through the pre-operation by utilizing the characteristics of the fixed nonce value of the CTR operation mode. It also shows that the proposed method is applicable to the GCM operation mode, and implements the GCM through the implementation of the GHASH function using the Galois Field(2^{128}) multiplication operation. As a result, in the case of LEA-CTR to which the proposed technique is applied on 32-bit RISC-V, it was confirmed that the performance was improved by 2% compared to the previous study.In addition, the performance of the GCM operation mode is presented so that it can be used as a performance indicator in other studies in the future.

Keywords: LEA, CTR mode, GCM mode, Implementation, RISC-V

Received(01. 26. 2022), Modified(03. 02. 2022), Accepted(03. 03. 2022)

* 본 논문은 2021년도 한국정보보호학회 호남지부 학술대회에 발표한 우수논문을 개선 및 확장한 것임.

* 이 논문은 부분적으로 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No.2018-0-00264, IoT 융합형 블록체인 플랫폼 보안 원천 기술 연구, 25%) 그리고 이 논문은 부분적으로 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2021-0-00540, GPU/ASIC 기반 암호알고리즘 고속화 설계 및 구현 기술개발, 50%) 그리고 이 성과는 부분적으로 2022년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2020R1F1A1048478, 25%).

† 주저자, shuraatum@gmail.com

‡ 교신저자, hwajeong84@gmail.com (Corresponding author)

I. 서 론

사물인터넷의 발전으로 다양한 기기에서 암호가 사용되고 있다. 다양한 기기들은 각각의 컴퓨팅 환경을 가지며, 이 중 제한된 컴퓨팅 환경에서 사용하기 위한 암호가 개발되고 있다. 2015년부터 NIST(National Institute of Standards and Technology)에서도 이러한 제한된 컴퓨팅 환경에서 사용할 경량 암호 공모전을 개최하였으며 여러 경량 암호 알고리즘이 발표되고 있다[1]. 국내에서도 LEA, HIGHT, PIPO등과 같은 경량 블록암호가 개발되어 있으며, 최적화 연구 또한 활발하게 진행되고 있다.

본 논문에서는 국내 경량 블록암호인 LEA의 CTR 운용 모드 최적 구현을 제안한다. 현재까지 LEA-CTR 최적 구현 연구는 8-bit AVR 마이크로컨트롤러와 ARMv8 마이크로컨트롤러 상에서의 최적화 구현 연구가 제안되었다[2][3]. 본 논문에서는 기존 연구의 최적화 기법을 참고하여 32-bit RISC-V 플랫폼 상에서의 최적화 구현을 진행한다. 또한 LEA-CTR 최적화 구현을 활용하여 CTR 운용 모드를 활용하는 GCM 운영 모드에 적용하는 확장을 포함한다.

본 논문의 구성은 다음과 같다. 2장에서는 경량 블록암호 LEA, 블록암호 운영 모드 그리고 RISC-V에 관하여 설명한다. 3장에서는 구현 기법을 설명한다. 4장에서는 구현 기법의 성능을 평가한다. 마지막으로 5장에서는 본 논문의 결론을 내린다.

II. 관련 연구

2.1 LEA 경량 블록암호

LEA는 빅데이터, 클라우드등 고속 환경뿐만 아니라 IoT기기, 모바일기기 등 경량 환경에서도 기밀성을 제공하기 위해 2013년 국내에서 개발된 경량 블록암호이다. 블록 길이는 128-bit, 키 길이는 128, 192, 256-bit 세 종류를 지원한다[4]. 자세한 매개 변수는 Table 1.과 같다. 가장 널리 사용되는 AES 대비 약 1.5배 빠른 암호화가 가능하며, 2019년 경량 블록 암호 분야 표준(ISO/IEC 29192-2:2019)으로 제정되었다[5].

Table 1. Parameters of LEA; n:block size, k:Key size, rk:round key size, r:number of rounds

Cipher	n	k	rk	r
128/128	128	128	192	24
128/192	128	192	192	28
128/256	128	256	192	32

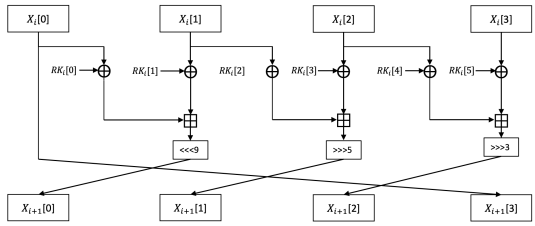

Fig. 1. Algorithm Structure of LEA

알고리즘은 ARX(Addition, Rotation, eXclusive-or)구조로 이루어져 있으며, 입력 받은 블록은 32-bit State로 나뉘어져 암호화가 진행된다. 전체적인 알고리즘 구조는 Fig.1.과 같다.

2.2 CTR(Counter) 운용 모드

블록 암호 운용 방식에는 ECB, CBC, CTR 등 여러 운용 방식이 존재한다. 그 중 CTR 운용 모드는 고정된 상수를 사용하는 Nonce 값과 변수인 Counter 값이 결합한 값을 입력 값으로 사용하여 암호화를 진행하고 마지막에 평문과 XOR 하는 방식으로 암호화가 진행된다[6]. 이때 Counter 값을 통해 현재 블록이 몇 번째 블록인지 알 수 있으며, 각 블록이 이전 블록에 의존하지 않기 때문에 병렬적으로 동작하는 것도 가능하다.

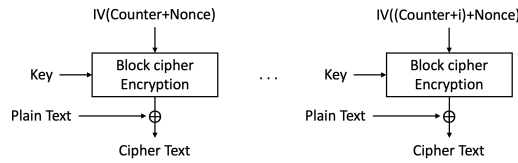

Fig. 2. Counter Operation mode of block cipher

2.3 GCM(Galois Counter Mode) 운용 모드

GCM 운용 모드는 데이터의 기밀성과 무결성을 제공하는 운영모드이다. 기밀성은 CTR 운용 모드를 통해 암호화되어 제공되며, 무결성은 암호화된 데이터와 AAD 데이터를 활용하여 GHASH 함수 연산

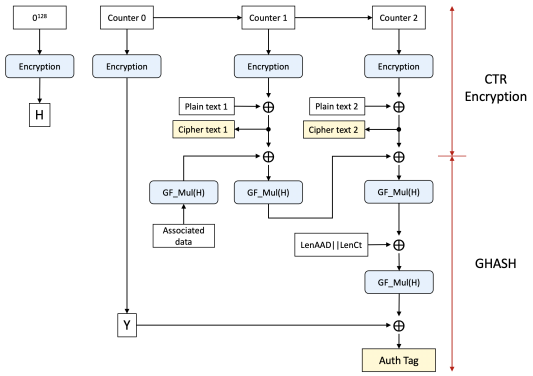

Fig. 3. GCM Operation mode of block cipher

을 통해서 제공된다. GHASH 함수에서는 Galois Field(GF) 곱셈 연산을 통해 인증 Tag 생성한다. 이때 GF 곱셈 연산은 $P(x) = x^{128} + x^7 + x^2 + x + 1$ 를 적용한 GF(2^{128})상에서 연산된다[7].

Fig.3.은 편의를 위해 2개의 블록을 암호화하고 인증 Tag값을 연산하는 과정을 도식화한 그림이다. H는 0으로 이루어진 값을 암호화한 값을 의미한다. Y는 CTR 운용 모드에서 Counter 값이 0일 때의 암호화 값을 의미한다. 이전에 연산된 H와 추가 인증 데이터(Additional Authentication Data:AAD)가 AAD의 블록의 수만큼 GF 곱셈 연산을 진행한다. 이후 CTR 운용 모드에서 연산된 암호화 값들과 GF 곱셈 연산 진행하며 인증 Tag 값이 출력된다.

2.4 RISC-V Processor

캘리포니아 대학교 버클리에서 2010년부터 개발중인 RISC(Reduced Instruction Set Computer) 기반의 컴퓨터 아키텍처이다[8]. RISC-V는 RV32I, RV64I의 두 가지 모델이 있으며, 각각 32-bit, 64-bit 레지스터를 사용한다. 본 논문에서 사용한 32-bit 구조의 RV32I는 32-bit 레지스터 32개를 제공하며, 각각의 레지스터 용도는 Table 2.와 같다[9].

Table 3.은 RV32I에서 사용하는 기본적인 명령어와 본 논문에서 활용한 명령어를 확인할 수 있다. 명령어 사용은 opcode destination, oprand1, oprand2 순으로 사용된다. 예를 들어 ADD a0, a1, a2 는 a1 register와 a2 register의 덧셈 연산이 a0 register에 저장된다.

Table 2. Purpose of RISC-V Register

Register	Description	Saver
zero(x0)	Zero register	
ra(x1)	return address	
sp(x2)	stack pointer	callee
gp(x3)	global pointer	
tp(x4)	thread pointer	
a0~a7	function arguments and return value	
s0~s11	saved registers	callee
t0~t6	temporal registers	

Table 3. Instruction of RISC-V

Instruction	Description
ANDI	And Immediate
XOR	Exclusive or
SLLI	Shift Left Logical Immediate
SRLI	Shfit Right Logical Immediate
JAL	Jump and Link
LI	Load Immediate
BNE	Branch Not Equal
BEQ	Branch Equal

III. 구현 기법

본 논문에서는 LEA-CTR 운용 모드 최적화 구현을 진행하며, 이를 활용한 LEA-GCM 운용 모드의 구현을 진행한다. 본 장의 구성은 LEA-CTR 운용 모드 구현을 설명하고, GCM 운용 모드 적용에 대해 설명한다.

3.1 LEA-CTR 구현

LEA-CTR 최적화 구현을 위해 CTR 운용 모드의 특성을 활용한 사전 연산 최적화 기법과 레지스터간의 이동을 생략하고 고정된 레지스터로 연산을 진행하는 고정 레지스터 사용 기법을 설명한다.

3.1.1 사전 연산 최적화

CTR 운용 모드는 고정된 Nonce 값과 변수인 Counter 값을 결합하여 입력 값으로 사용한다. 이때 고정된 Nonce 값으로 인하여 Counter 값이 바뀌어도 특정 라운드까지 암호화가 진행될 때, 각 라운드에서 고정된 값이 존재하게 된다. 따라서 사전 연산을 통해 고정된 값을 얻고 암호화를 진행하게 되면 고정된 값을 얻을 때를 제외하고 다음 블록의 암호화가 진행될 때는 일정 부분의 연산을 생략할 수 있다.

라운드가 진행됨에 따라 확산이 이루어지는 과정과 고정되는 값은 Fig.4.와 같다.

Fig.4.에 색이 채워진 부분은 Counter 값이 변하였을 때, 영향을 받는 부분이다. Counter 값이 변하였을 때 각 라운드가 진행됨에 따라서 영향을 받는 부분이 고정되어 있는 것을 확인할 수 있으며, 3라운드에서 전체 확산이 이루어지는 것을 확인 할 수 있다. 1라운드에서는 X_1〔1〕, X_1〔2〕, 2라운드에서는 X_2〔1〕의 값이 Counter 값이 증가하여도 고정된 Nonce 값으로 인해 값이 고정된다. 따라서 해당 값을 매번 연산할 필요 없이 사전 연산을 통해 값을 얻고, 다음 블록을 연산할 때는 해당 값을 얻기 위한 연산을 생략할 수 있다.

또한 암호화 입력 값으로 사용되는 Nonce 값 96-bit 전부를 불러올 필요 없이 X_1〔0〕을 얻기 위해 사용되는 32-bit Nonce 값만 필요하다. X_1〔3〕 값의 경우 Counter 값이 그대로 옮겨지기 때문에, 1라운드에서의 연산은 X_1〔0〕의 값만 연산하게 된다.

결과적으로 평문 블록 한 개의 암호화 과정에서 6번의 라운드키 XOR 연산, 3번의 Addition 연산, 3번의 Rotation 연산이 생략 가능하다.

Input	Counter(32-bit)	Nonce(32-bit)	Nonce(32-bit)	Nonce(32-bit)
ROUND 1	$X_1[0]$	$X_1[1]$	$X_1[2]$	$X_1[3]$
ROUND 2	$X_2[0]$	$X_2[1]$	$X_2[2]$	$X_2[3]$
ROUND 3	$X_3[0]$	$X_3[1]$	$X_3[2]$	$X_3[3]$

Fig. 4. Counter value diffusion process

3.1.2 고정 레지스터 사용

LEA의 암호화 과정에서 각 State들은 다음 라운드로 넘어가면서 옆 State와 자리를 바꿔가며 암호화가 진행된다. 또한 X_i〔0〕 값은 연산 과정 없이 X_{i+1}〔3〕으로 값이 이동하게 된다. 이러한 이동하는 과정을 생략하고 고정된 자리에서 연산을 진행하게 되면 값이 이동하는 연산을 생략할 수 있다. 즉 X_i〔0〕의 값이 다음 라운드에서도 X_{i+1}〔0〕으로 그대로 고정되게 된다.

각 State의 이동이 생략됨에 따라서 연산의 각 라운드에 맞춰 연산의 수정이 필요하게 된다. 4개의 State가 라운드당 한 칸씩 이동하며 암호화가 진행되기 때문에 4라운드가 진행되게 되면 처음의 위치로 다시 돌아오게 된다. 따라서 4라운드 단위로 반복된 연산을 진행하게 된다.

이로 인해 전체 암호화 과정에서 128-bit 키 길이 기준으로 보았을 때, 24라운드가 진행되기 때문에 X_i〔0〕의 값이 X_{i+1}〔3〕 으로 이동하는 연산을 24번 생략 가능하다.

3.2 LEA-GCM 구현

본 장에서는 최적화된 LEA-CTR 구현을 활용하는 LEA-GCM 구현에 관하여 설명한다. GCM 운용 모드는 CTR 운용 모드로 암호화가 진행되고, 암호화된 암호문을 GHASH 함수 연산을 통해 인증 Tag 값을 생성한다.

구현은 Fig.3.에서 볼 수 있듯이, CTR을 통해 암호화하는 부분과 GHASH 함수 연산을 통해 인증 Tag를 생성하는 부분으로 나뉜다.

GCM에서는 CTR 암호화 과정에서 Counter=0 일 때 암호화한 값을 Y값으로써 사용한다. 따라서 기존 구현에서 약간의 수정을 통해 Counter=1부터 암호화한 값을 평문과 XOR 하여 암호문을 생성한다. 결과적으로 GHASH 함수에서 마지막 연산에 필요한 Y값을 CTR 암호화 과정에서 같이 연산한다. 예를 들어 3개의 블록(즉, 입력 길이가 384-bit)을 CTR 암호화를 진행할 때, 기존 LEA-CTR에서는 Counter가 0, 1, 2로 암호화 되지만, GCM 운용 모드에서는 Counter가 하나 더 추가되어 0, 1, 2, 3으로 암호화가 된다.

GHASH 함수에서는 Galois Field(GF)곱셈 연산을 진행한다. 따라서 GF(2^{128}) 곱셈 연산의 구현이 필요하다. GF 곱셈 연산의 수도 코드는 Table 4.와 같다.

수도 코드를 어셈블리로 변환한 코드는 Table 5.와 같다. GF 곱셈 연산의 구현은 64-bit 단위로 진행되며, 32-bit 레지스터를 사용하기 때문에 64-bit

Table 4. Pseudocode of GF(2^{128}) multiplication

Input : X(128-bit), Y(128-bit)
Output : X • Y

1 R = 0xe100000000000000ULL;
2 X -〉 $x_0x_1 \cdots x_{127}$
3 $Z_0 = 0^{128}$, $V_0 = Y$
4 **For** i=0 **to** 127:
5 $Z_{i+1} = Z_i$ **if** $x_i = 0$
6 $Z_{i+1} = Z_i \oplus V_i$ **if** $x_i = 1$
7 $V_{i+1} = V_i \gg 1$ **if** $LSB_1(V_i) = 0$
8 $V_{i+1} = (V_i \gg 1) \oplus R$ **if** $LSB_1(V_i) = 1$
9 **return Z**

에 맞춰 두 개의 레지스터에 64-bit의 상위 32-bit와 하위 32-bit를 나누어 저장하여 연산을 진행한다.

Table 5.는 GF 곱셈 연산 128-bit중에서 상위 64-bit의 연산을 의미한다. 하위 64-bit 연산도 위와 동일한 코드이며 레지스터의 차이만 존재한다.

Table 5. Assembly code of GF(2^{128}) multiplication(register pupose; t0, t1, t2, t3 : input X; s3, s2 : High 64-bit of input Y; a5 : 0xe1; a6 : count; a7 : 0x80000000; t4, t5 : temp register)

```
1.  loop1:
2.      and     t5, s3, a7
3.      beq     zero, t5, branch1
4.      xor     s4, s4, t0
5.      xor     s5, s5, t1
6.      xor     s6, s6, t2
7.      xor     s7, s7, t3
8.  branch1:
9.      andi    t5, t0, 1
10.     srli    t0, t0, 1
11.     slli    t4, t1, 31
12.     or      t0, t0, t4
13.     srli    t1, t1, 1
14.     slli    t4, t2, 31
15.     or      t1, t1, t4
16.     srli    t2, t2, 1
17.     slli    t4, t3, 31
18.     or      t2, t2, t4
19.     srli    t3, t3, 1
20.     beq     zero, t5, branch2
21.     xor     t3, t3, a5  //0xe1
22. branch2:
23.     addi    a6, a6, -1
24.     slli    s3, s3, 1
25.     srli    t4, s2, 31
26.     or      s3, s3, t4
27.     slli    s2, s2, 1
28.     bne     zero, a6, loop1
```

loop1:은 Table 4.의 5번 6번 줄에 해당하는 코드이다. beq 명령어로 x_i의 값이 0이면 Branch1:로 넘어가며 1일 경우 6번 줄의 연산을 진행하고 Branch1:로 넘어간다.

Branch1:은 $V_i \gg 1$ 의 연산을 진행한다. 128-bit 단위의 Shift Right를 해야 하기 때문에 4개의 레지스터간에 비트 이동이 필요하다. 20번째 줄에서 V_i의 최하위 비트가 0일 경우 Branch2:로 넘어가며 1일 경우 R(0xe1)의 값과 XOR 연산된다. 이때 0xe1은 하위 bit는 0이기 때문에 상위 32-bit 레지스터에만 연산한다.

Branch2:는 반복 횟수를 줄이고 x_i의 값을 다음 x_{i+1}값으로 바꾸는 연산을 진행한다. x_{i+1}값은 x_i 값을 Shift Right를 통하여 최하위 bit를 다음 비트의 값으로 바꾸는 연산을 진행한다.

IV. 성능 평가

본 논문에서는 SiFive 사의 HiFive RevB 보드를 사용하여 측정을 진행하였으며, SiFive 사의 Freedom Studio를 사용한다.

32-bit RISC-V 상에서의 LEA-CTR 최적화 구현에 관한 기존 연구가 없다. 따라서 같은 32-bit RISC-V 플랫폼 상에서 LEA 최적화 구현 연구인 Kwak et al.[10]과 성능 비교를 진행하였다. 해당 결과는 Table 6.과 같다.

본 논문에서 제안한 기법을 활용하여 128-bit 하나의 블록을 암호화 할 때의 LEA-CTR의 Cycle을 나타낸다. 기존 LEA 최적화 구현을 진행한 연구 대비 2%의 성능 향상을 확인하였다.

마찬가지로 32-bit RISC-V 상에서의 LEA-GCM 구현 또한 기존 연구가 없기 때문에 성능 비교 없이 성능 측정 결과만 나타낸다. 성능 측정 결과는 Table 7.과 같으며 한 번의 GF 곱셈 연산(GF_Mult), GCM 운용 모드에서의 CTR 암호화(CTR Encrytion), 인증 Tag를 생성하는 GHASH 함수(GHASH)를 나누어 성능 측정을 진행하였다. 입력 데이터 길이는 16-byte,

Table 6. Performance Result of LEA-CTR on 32-bit RISC-V(unit:cycle)

	Cycle
Kwak et al.[10]	775
Our Work	757

Table 7. Performance Result of LEA-GCM on 32-bit RISC-V

Function	Data Lenghth	Cycle	Cycle/ Byte
CTR Encrytion	16-byte	1489	93.06
	64-byte	3654	57.09
	128-byte	6548	51.16
GF_Mult	-	3499	-
GHASH	16-byte	8256	516
	64-byte	16447	256.98
	128-byte	27347	213.65

64-Byte, 128-Byte로 나누어 측정하였으며, AAD 길이는 128-bit를 사용하였다.

LEA-GCM의 CTR 암호화에서 16-byte를 암호화하는데 1489cycle이 발생한다. 데이터 길이는 16-byte이지만 실제로 암호화는 2개의 블록(32-byte)을 암호화하기 때문에 이전 LEA-CTR 성능과 2배 정도의 차이를 보여준다. 이는 GHASH에서 사용될 Y(count=0 일 때의 암호화 값)를 연산하기 위해 한 개의 블록을 더 암호화하기 때문이다. 결과적으로 CTR 암호화와 GHASH 함수 모두 데이터 길이가 길어질수록 좀 더 나은 성능을 보여준다.

V. 결 론

본 논문에서는 LEA 블록 암호의 CTR 운용 모드와 CTR 운용 모드를 활용하는 GCM 운용 모드의 구현을 진행하였다. 최적화 구현을 위해 CTR 운용 모드에서의 고정된 Nonce 값의 특성을 활용하여 사전 연산을 통한 연산 생략과 State 고정을 통해 State간 값의 이동을 생략한 최적화 구현을 제안한다. 제안 기법을 통해 암호화 과정에서 6번의 라운드키 XOR 연산, 3번의 Addition 연산, 3번의 Rotation 연산이 생략 가능하다. 또한 값의 이동을 생략하여 $X_i[0]$의 값이 $X_{i+1}[3]$ 으로 이동하는 연산을 24번 생략 가능하다. 이를 활용하여 GCM 운용 모드에서의 CTR 암호화에 제안 기법을 적용하여 암호화를 진행하며, Galois Field(GF)곱셈 연산을 사용하는 GHASH 함수 구현을 통해, 기밀성과 무결성을 동시에 제공하는 GCM 운용 모드 구현을 진행하였다. 결과적으로 기존의 연구 대비 LEA-CTR에서는 2%의 성능 향상을 보여준다. 또한 GCM 운용 모드의 경우 LEA-CTR 최적화 구현을 적용 가능함을 보여주며, 추후 다른 연구에서 성능 지표로 사용될 수 있도록 성능을 제시하였다.

References

[1] NIST, Lightweight Cryptography: Project Overview, https://csrc.nist.gov/projects/lightweight-cryptography, 2021.

[2] Young-Beom Kim, Hyeok-Dong Kwon, Sang-Woo An, Hwa-Jeonng Seo and Seong-Chung Seo, "Efficient Implementation of ARX-Based Block Ciphers on 8-bit AVR Microcontrollers," *Mathematics 2020*. vol. 8, no. 10, 1837., Oct 2020

[3] Jin-Gyo Song and Seog-Chung Seo, "Efficient Parallel Implementation of CTR Mode of ARX-Based Block Ciphers on ARMv8 Microcontrollers," *Applied Sciences 2021*. vol. 11, no. 6, 2548., Mar. 2021.

[4] D.J Hong, et al. "LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors," *WISA 2013: Revised Selected Papers of the 14th International Workshop on Information Security Applications*, Volume 8267, pp 3 - 27, Aug. 2013.

[5] ISO. "ISO/IEC 29192-2:2019 Information Security-Lightweight Cryptography-Part 2: Block Ciphers," *International Organization for Standardization*, pp 56, Nov. 2019.

[6] H. Lipmaa, P. Rogaway and D. Wagner, "CTR-mode encryption," *First NIST Workshop on Modes of Operation*, Vol. 39, Sep. 2000.

[7] D. McGrew and J. Viega, "The Galois/counter mode of operation (GCM)," *submission to NIST Modes of Operation Process*, 20, 0278-0070,

May. 2005.
〔8〕 A.Waterman and K.Asanovi´c, "The RISC-V Instruction Set Manual Volume I: User-Level ISA Document Version 2.2," UCBerkeley Tech., May. 2017.
〔9〕 A.Waterman, Y.Lee, D.Patterson, and K. Asanovi´c, "The risc-v instruction set manual, volume i: Baseuser-level isa," UCB/EECS-2011-62 116, UCBerkeley Tech., May. 2011.
〔10〕 Y.J. Kwak, Y.B. Kim, S.C. Seo. "Benchmarking Korean Block Ciphers on 32-Bit RISC-V Processor," *Journal of the Korea Institute of Information Security & Cryptology*, 31(3), pp. 331-340, Jun. 2021.

〈저자소개〉

엄 시 우 (Si-woo Eum) 학생회원
2021년 3월: 한성대학교 IT융합공학부 학사 졸업
2021년 3월~현재: 한성대학교 IT융합공학부 석사과정
〈관심분야〉 정보보호, 암호구현

권 혁 동 (Hyeok-dong Kwon) 학생회원
2018년 3월: 한성대학교 IT응용시스템공학부 학사 졸업
2020년 3월: 한성대학교 IT융합공학부 석사 졸업
2020년 3월~현재: 한성대학교 정보컴퓨터공학과 박사과정
〈관심분야〉 정보보호, 암호구현

김 현 지 (Hyun-ji Kim) 학생회원
2020년 3월: 한성대학교 IT융합공학부 학사 졸업
2022년 3월: 한성대학교 IT융합공학부 석사 졸업
2022년 3월~현재: 한성대학교 정보컴퓨터공학과 박사과정
〈관심분야〉 정보보호, 인공지능

양 유 진 (Yu-jin Yang) 학생회원
2022년 3월: 한성대학교 IT융합공학부 학사 졸업
2022년 3월~현재: 한성대학교 IT융합공학부 석사과정
〈관심분야〉 인공지능 보안, 양자 암호

서 화 정 (Hwa-jeong Seo) 종신회원
2010년 3월: 부산대학교 컴퓨터공학과 졸업
2012년 3월: 부산대학교 컴퓨터공학과 석사
2016년 3월: 부산대학교 컴퓨터공학과 박사
2016년~2017년: 싱가포르 과학기술청 연구원
2017년~현재: 한성대학교 IT융합공학부 조교수
〈관심분야〉 정보보호, 암호화 구현, IoT