Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.6, Dec. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.6.1217

라운드 키 선행 로드를 통한 CHAM-64/128 카운터 모드 고속 구현*

권 혁 동,[1†] 장 경 배,[1] 박 재 훈,[1] 서 화 정[2‡]
[1,2]한성대학교 (대학원생, 교수)

High-Speed Implementation to CHAM-64/128 Counter Mode with Round Key Pre-Load Technique*

Hyeok-dong Kwon,[1†] Kyoung-bae Jang,[1] Jae-hoon Park,[1] Hwa-jeong Seo[2‡]
[1,2]Hansung University (Graduate student, Professor)

요 약

CHAM은 저사양 프로세서를 지원하기 위한 경량 블록암호로, 한국의 국가보안기술연구소에서 개발되었다. 블록암호의 원활한 동작을 위해서는 블록암호 운용 모드를 적용하는데 그 중에서 카운터 모드는 낮은 구현 난이도와 병렬 연산 지원으로 뛰어난 효율을 자랑한다. 본 논문에서는 블록암호 CHAM의 카운터 운영 모드를 최적 구현한 결과물을 제시한다. 제안기법은 사전 연산을 통해 일부 라운드를 생략하는 것으로 기존 CHAM보다 빠른 연산 속도를 가진다. 또한, 라운드 함수 진입 전 라운드 키의 일부를 레지스터에 선행 로드하는 것으로 라운드 함수마다 라운드 키를 로드하는 시간을 160cycles만큼 감소시켰다. 제안하는 기법은 기존 기법에 비해 고정키 시나리오 상에서 6.8%, 가변키 시나리오 상에서 4.5%의 성능 향상이 있었다.

ABSTRACT

The Block cipher CHAM is lightweight block cipher for low-end processors, developed by National Security Research Institute from Korea. The mode of operation is necessity for efficient operation of block cipher, among them, the counter (CTR) mode has good efficiency because it is easy to implement and supporting parallel operation. In this paper, we propose the optimized implementation for block cipher CHAM-CTR. The proposed implementation can be skipped some rounds by pre-computation. Thus it has better calculating speed than existing CHAM. Also, this implementation pre-load some of round keys to registers, before entering round functions. It makes reduced 160cycles loading time for round key load. Finally, proposed implementation achieved higher performance about 6.8%, and 4.5% for fixed-key scenario, and variable-key scenario, respectively.

Keywords: 8-bit AVR Processors, CHAM block cipher, Optimized implementation

I. 서 론

사물인터넷 환경에서는 수많은 정보가 저사양 기기를 통해 전송된다. 이러한 정보에는 민감한 정보들도 있기에 안전하게 암호화 되어야 한다. 하지만 암호 알고리즘은 방대한 연산량과 계산의 난해함을 통해 안전성을 확보하며, 연산량이 떨어지는 저사양 기기는 이를 처리하기 어렵다. 이를 개선하기 위해서

Received(09. 25. 2020), Modified(11. 13. 2020), Accepted(11. 16. 2020)
* 이 성과는 2020년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF-2020R1F1A1048478).
† 주저자, korlethean@gmail.com
‡ 교신저자, hwajeong84@gmail.com(Corresponding author)

경량 암호가 제안되었다. 경량 암호는 저사양 프로세서 상에서도 효율적이며 안전한 암호화 환경을 제공해준다. PRESENT[1], SPECK, SIMON[2], CHAM 등의 암호가 경량 암호에 속한다. 이 중에서 블록암호 CHAM은 국내에서 개발된 암호로, 저사양 프로세서에 매우 효과적이다.

본 논문에서는 블록암호 CHAM의 카운터 모드 최적 구현을 개량한, 라운드 키 선행 로드 기법을 제안한다. 제안 기법은 8-bit AVR 프로세서 상에서 기존 CHAM 카운터 모드 최적 구현보다 더 빠른 성능을 지닌다.

본 논문의 구성은 다음과 같다. 2장에서 블록암호 CHAM과 기존 카운터 모드 최적 구현 기법에 관한 내용을 확인한다. 3장에서는 제안하는 라운드 키 선행 로드 기법과 구현 방법에 대해 제안한다. 4장에서는 제안하는 구현 기법과 기존 기법의 성능 비교를 한다. 5장에서는 본 논문의 결론을 내린다.

II. 관련 연구 동향

2.1 경량 블록암호 CHAM

CHAM은 ICISC'17에서 발표된 국산 경량 블록암호로 저사양 프로세서를 대상으로 하여 최적 구현에 용이하게 설계되어 있다[3]. 이후 수정된 버전인 revised CHAM이 ICISC'19에서 발표되었다[4]. 기존 CHAM과 revised CHAM은 라운드 수만 다르고 모든 구조가 동일하다. CHAM은 Fig.1.과 같은 구조로, 입력된 평문을 4개의 블록으로 나누어 연산한다. 연산에는 Addition, Rotation, XOR의

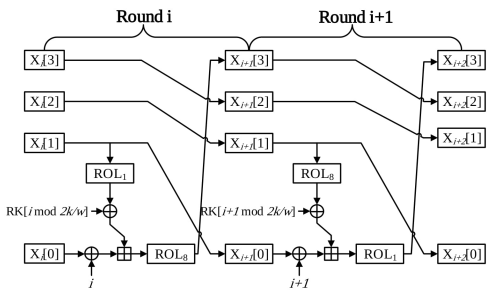

Fig. 1. Round function structure of CHAM. Xi[N]: Nth plaintext block of round i, RK[N]: Nth round key, i: round counter, ROLN: rotate left N times

Table 1. Parameters of CHAM. n: length of plaintext, k: length of key, r: number of round, w: bit-length of word.

Type	n	k	r	w	k/w
64/128	64	128	88	16	8
128/128	128	128	112	32	4
128/256	128	256	120	32	8

세 가지 연산만을 활용하는 ARX 구조가 적용되어 있다. CHAM은 총 세 종류의 입출력 규격을 가지며 이는 Table 1.과 같다. Table 1.의 파라미터는 revised CHAM 기준으로 작성되어 있다. 본 논문에서는 ICISC'19에서 발표된 revised CHAM을 대상으로 구현하였다.

본 절 이후로 편의상 revised CHAM을 CHAM으로 표기한다.

2.2 대상 프로세서 ATmega128

ATmega128은 Atmel사 에서 제조 및 판매하는 초경량 사물인터넷 프로세서 중 하나이다. ATmega128은 32개의 8-bit 범용 레지스터를 가지고 있기에 제안 기법 구현에 적합한 형태의 프로세서이다. 하드웨어 아키텍처는 Harvard 구조가 적용되어 16MHz의 주파수로 동작한다. 제공되는 명령어는 총 81개이며 128KB의 플래시 메모리와 4KB의 EEPROM, 4KB의 SRAM을 가지고 있다[5].

2.3 CHAM 카운터 모드 최적 구현

WISA'20에서 CHAM의 카운터 모드 최적 구현이 발표되었다[6]. 카운터 모드는 블록암호 운용 모드 중 하나로, 평문 대신 상수인 논스(Nonce) 값과 블록의 번호를 뜻하는 카운터(Counter)를 입력 값으로 사용하는 운용 모드이다. [6]에서 제안하는 기법은 평문을 구성하는 값 중, 카운터는 변수이며 논스는 상수인 것을 활용한 구현이다. 제안하는 기법은 8-bit AVR 마이크로컨트롤러 중 하나인 ATmega128 프로세서를 대상으로 구현했다.

Fig.1.과 같이, CHAM의 각 라운드의 입력 값은 평문 블록, 라운드 키, 라운드 카운터로, 총 세 종류의 입력 값이 존재한다. 이때, 평문 블록이 가진 정보가 논스라면 해당 라운드의 입력 값 세 종류는 모두 고정 값임을 알 수 있다. 모든 입력 값이 고정

값이면 연산 결과는 항상 동일하다. 따라서 [6]에서는 논스가 사용되는 지점의 결과를 사전 연산하여 일부 라운드의 연산을 생략하는 기법을 제시하고 있다. [6]의 기법은 CHAM-64/128 16-bit 카운터 기준으로 기존보다 약 15.5% 향상되었다.

[6]은 키가 고정된 환경에서만 구현되었기 때문에, 이를 개선한 [7]이 제안되었다. [7]은 키가 바뀌는 가변키 상황을 고려하여 구현하였다. [7]의 구현물은 두 가지 형태로 사전 연산만을 계산하는 형태와 사전 연산을 진행하며 1개의 블록을 암호화하는 형태가 제시되었다. [7]의 기법은 CHAM-64/128 16-bit 카운터 기준으로 기존보다 약 14.4%의 성능 향상을 보인다.

III. 제안 기법

3.1 기존 레지스터 할당

[6]과 [7]에서는 최적 구현을 위해 레지스터 정렬을 최적화 하였다. Fig.2.는 기존 연구에서 제시한 레지스터 정렬 기법으로, CHAM-64/128에 해당하는 레지스터를 표현한 것이다.

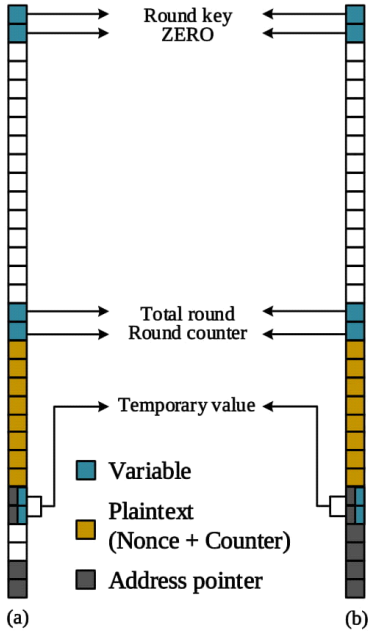

Fig. 2. Registers allocation plan from previous study, (a): Fixed-key model, (b): Variable-key model.

제안된 기법에서는 R2~R17 그리고 R27, R28 레지스터를 사용하고 있지 않다. 이는 callee saved 레지스터로서, 호출된 함수가 레지스터를 사용하기 전에 반드시 기존 값을 보존해야 할 레지스터이다. 기존 연구에서 callee saved 레지스터를 사용하지 않은 이유는, callee saved 레지스터의 값을 보존하는데 드는 시간을 절약하기 위해서다. 따라서 callee saved 레지스터 값 보존을 위한 PUSH, POP 연산에 소요되는 시간을 줄일 수 있었다.

3.2 라운드 키 선행 로드 기법

Table 1.에서 CHAM-64/128은 32바이트의 라운드 키를 가짐을 알 수 있고, Fig.2.에 따르면, 기존 구현물은 14개에서 16개의 레지스터를 사용하지 않고 있다. 즉, 미사용 레지스터에 라운드 키의 일부를 미리 로드해둘 수 있다. 이는 라운드 함수 동작 중에 라운드 키 로드 수를 경감시킬 수 있으며, 생략되는 라운드 키 로드 횟수만큼 연산 시간을 줄일 수 있다.

구체적으로는 전체 라운드 키는 32바이트이며, 그 중에서 선행 로드하는 라운드 키는 16바이트 이다. 라운드 키를 선행 로드하기 위해서는 16개의 LD 명령어가 필요하다. 따라서 첫 8라운드 중에 16개의 LD 명령어를 사용한다. 이후 80라운드 중에서는 절반의 라운드 키를 레지스터에 확보해 둔 상태이므로, 선행 로드한 라운드 키를 사용하는 40라운드 동안에는 LD 명령어가 생략된다. 하나의 라운드에서 사용되는 라운드 키는 2바이트 이므로, 두 개의 LD 명령어가 필요하다. 최종적으로 40라운드 동안 80개의 LD 명령어가 제거되므로 동작에 소요되는 160cycles가 감소한다.

CHAM-128/128의 경우에는 평문 길이가 CHAM-64/128보다 2배 길다. 이에 따라 미사용 레지스터가 절반으로 줄어든다. CHAM-128/256은 평문 뿐만 아니라 라운드 키 사이즈도 2배 길다. 때문에 CHAM-64/128 구현물 만큼의 속도 향상을 보여주지 못하기 때문에 구현에서 제외한다.

3.2.1 고정키 시나리오의 레지스터 할당

Fig.2.의 (a)에서 총 16개의 레지스터가 사용되고 있지 않음을 알 수 있다. 따라서 32바이트의 라운드 키 중에서 절반을 미리 로드해 둘 수 있다. 하

지만 기존 레지스터 할당을 그대로 사용하지 않는다. 본 구현물은 다음과 같이 레지스터 할당을 수정한 다음, 미사용 레지스터에 라운드 키를 선행 로드한다. 수정 내용은 다음과 같다.

첫 번째로 임시 변수 레지스터를 R26, R27에서 R28, R29로 할당하였다. 본디 R26부터 R31까지의 레지스터는 포인터 레지스터로 사용된다. 하지만 고정키 시나리오 구현물에서는 평문과 라운드 키 포인터 두 개만 필요하기 때문에 R28, R29 레지스터를 다른 용도로 사용할 수 있다. R28, R29 레지스터에 선행 로드한 라운드 키를 저장할 수 있으나, 임시 변수를 저장하는 용도로 사용하였다. 기존과 마찬가지로 임시 변수 레지스터를 R26, R27과 공유한다면, 포인터 값을 보존하기 위한 명령어인 PUSH, POP이 2개의 레지스터에 필요하다. 하지만 R28, R29를 사용한다면 상기의 과정이 필요 없기 때문에 R26, R27 보존을 위한 PUSH, POP 연산이 제외되어 8cycles의 이득이 발생한다.

두 번째로 전체 라운드 변수를 레지스터 상에서 제외하였다. 기존 구현물 상에서 해당 변수는 R16 레지스터에 저장되고 있다. 전체 라운드 변수는 라운드 함수의 반복을 중지할지 분기를 형성할 때 사용된다. 이때 AVR에서 제공하는 CPI 명령어를 사용한다면, 레지스터에 저장된 값과 비교하지 않고 지정 상수와 비교할 수 있게 된다. 따라서 R16에서 전체 라운드 변수를 제외하는 것으로 선행 로드 라운드 키의 레지스터로 사용할 수 있다.

마지막으로 라운드 카운터 레지스터를 R17에서 R27로 이동하였다. 이는 선행 로드 라운드 키 레지스터를 붙여두기 위함이다. 라운드 키는 1바이트 단위로 사용되기 때문에 레지스터를 붙이더라도 성능 면에서는 개선이 없다. 하지만 코드 관리 면에서 직관적으로 알아보기 쉽기 때문에 이와 같이 할당한다.

최종적으로 고정키 시나리오의 레지스터 할당은 Fig.3.의 (a)와 같이 할당된다. 전체적인 할당은 기존 구현과 유사하나, 일부 변수의 레지스터 위치가 조정되었으며 선행 로드 되는 라운드 키의 레지스터를 확인할 수 있다.

3.2.2 가변키 시나리오의 레지스터 할당

Fig.3.의 (b)는 가변키 시나리오의 레지스터 할당을 보여준다. 고정키 시나리오의 할당과 비교하면 하나의 차이점이 있는데, 평문 레지스터와 선행 로드

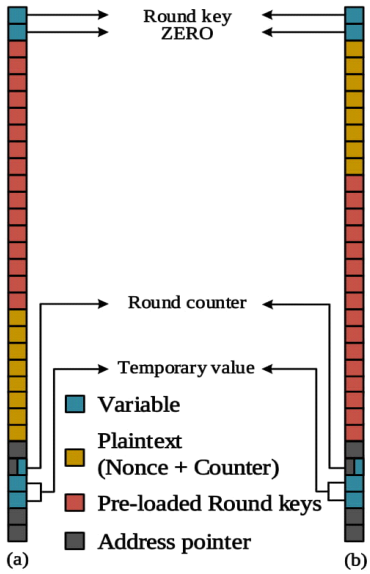

Fig. 3. Re-allocated registers for proposed implementation, (a): Fixed-key model, (b): Variable-key model.

라운드 키 레지스터가 상반된다는 점이다. 서로가 다른 형태로 할당 된 이유는 두 가지가 있다.

첫 번째 이유는 사전 연산 값 호출에 사용되는 명령어가 다르다. 고정키 시나리오 구현물은 사전 연산 값이 항상 같기 때문에 메모리를 통해서 호출하지 않고, LDI 명령어를 사용하여 정해진 값을 로드한다. 이때 LDI 명령어는 R16부터 R31 레지스터에 한해서만 사용이 가능하다. 따라서 고정키 시나리오의 평문 레지스터는 R16 이후의 레지스터를 할당해야 한다. 하지만 가변키 시나리오에서는 사전 연산 값이 바뀔 가능성이 존재하기 때문에, 사전 연산 테이블을 사용한다. 값을 가져오기 위해서는 LD 명령어를 통해 테이블에 접근한 다음 값을 가져온다. LD 명령어는 LDI 명령어와 다르게 레지스터 번호의 제약이 없으므로 평문을 R2부터 R9까지 할당할 수 있다. 하지만 LD 명령어를 사용하더라도 레지스터 상반과는 관계없이 기존과 동일한 구현이 가능하다.

두 번째 이유는 매개변수로 입력 받은 포인터 값의 상실을 방지하기 위함이다. 가변키 시나리오는 테이블 주소 값이 필요하므로, 고정키 시나리오보다 두 개 더 많은 포인터 매개변수가 필요하다. AVR 상에서 각각의 포인터 매개변수는 (R24, R25), (R22, R23), (R20, R21) 레지스터를 통해 16-bit 주소 값 형태로 전달된다. 이때 기존과 같은 형태로 평문

을 R18~R25에 할당하면, 평문을 로드하는 과정에서 최소 한 개의 주소 값을 상실할 가능성이 발생한다.

따라서 평문을 R2~R9에 할당하면 주소 값 상실을 방지할 수 있다. 주소 값이 상실되는 현상은 Fig.4.에서 시각적으로 확인할 수 있다. Fig.4.의 3단계는 주소 값이 소실되는 상황이고, Fig.4.의 4단계는 소실을 방지할 수 있는 상황이다.

레지스터 할당을 바꾸지 않고, 포인터를 통해 값을 호출하는 순서를 바꾸어도 이 문제를 해결할 수 있다. 가령 평문보다 라운드 키를 먼저 호출하는 방법이 있다. 하지만 이 방법을 사용한다면 값 상실을 방지하기 위해 추가적인 PUSH, POP 연산이 필요하기 때문에 더 많은 시간이 소요된다. 따라서 레지스터 할당을 바꿔주는 것으로 효율적으로 구성한다.

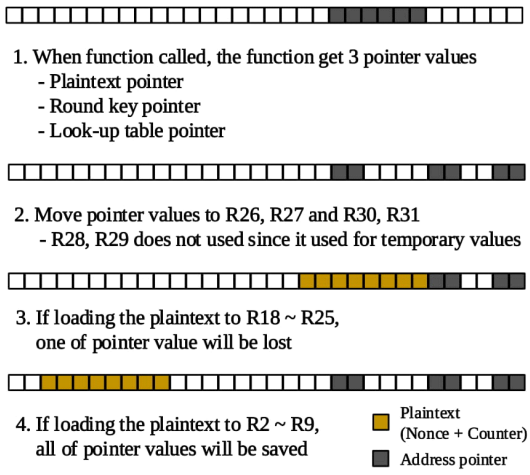

Fig. 4. Visualization of address pointer loss

IV. 성능 평가

구현은 [6], [7]과 동일한 환경인 8-bit AVR 마이크로컨트롤러인 ATmega128 프로세서를 대상으로 구현한다. 구현 결과물의 성능 비교는 Fig.5.에서 확인할 수 있다. 구현물의 성능 비교에는 clockcycle per byte, 속칭 cpb 단위로 비교를 행한다. cpb는 동작에 소요된 cycle을 입력 byte 단위로 나눈 값으로, 일정 단위의 데이터를 처리하는데 소요된 cycle을 확인하는 지표로 사용된다.

고정키 구현물에 관한 비교를 한다. 기존 구현물은 158.75cpb의 성능을 지니나, 제안하는 구현물은

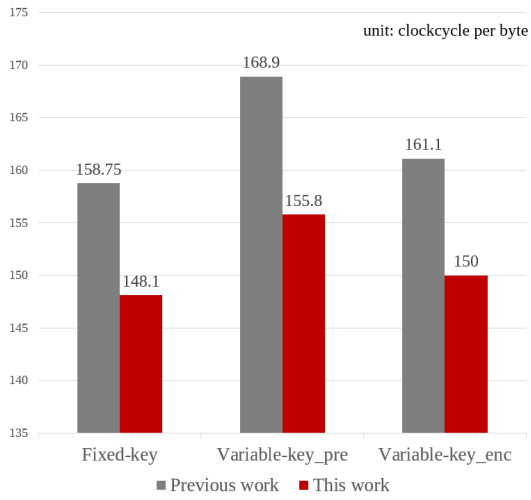

Fig. 5. Evaluation result between previous work and this work.

148.1cpb의 성능을 보여준다. 이는 약 6.8% 정도의 성능 향상을 가진다.

가변키 구현물은 사전 연산을 진행하는 부분과, 사전 연산을 활용하는 부분 두 가지 모델이 존재하므로 각각 비교를 진행한다. 첫 번째로 사전 연산 진행 모델을 비교한다. 기존 구현물은 168.9cpb의 성능을 보이지만, 제안하는 기법은 155.8cpb의 성능을 가진다. 이는 약 7.8% 성능 개선에 해당한다. 두 번째는 사전 연산을 사용한 암호화 모델을 비교한다. 비교 결과 기존 구현 기법은 161.1cpb의 성능을 보여주는데 반해, 제안하는 구현물은 150.0cpb의 성능으로 동작한다. 따라서 약 7.0%의 성능 향상을 가진다.

결과적으로 제안하는 기법이 기존 기법에 비해 모든 시나리오에서 평균적으로 약 7%의 속도 향상을 보이며, 이는 기존 기법에 비해 훨씬 고속으로 동작하는 이점을 지니고 있다.

V. 결 론

본 논문에서는 기존에 제안되었던 CHAM-CTR 최적 구현 중, 64/128 규격에 관한 최적 구현을 진행하였다. 제안하는 기법은 사용하지 않는 레지스터에 라운드 키의 일부를 미리 로드해두는 기법을 제시하였다. 이는 라운드 함수 동작 중에 절반의 라운드 키를 호출하는 시간을 절약할 수 있었다.

최적화 관점에는 저장 공간을 우선시하거나 속도

를 빠르게 하는데 집중, 또는 두 가지를 절충하는 형태의 최적화로 나눌 수 있다. 본 논문에서 제안한 구현물은 속도 위주로 최적화 하는 것에 주목하였고, 모든 레지스터를 사용하여 저장 공간을 최대한 활용하였다. 결과적으로 기존 구현물보다 세 가지 비교 상황을 고려한 결과, 평균 7% 정도의 빠른 연산 성능을 보여주었다.

ARX 기반 블록암호는 단순한 구조와 적은 양의 연산량을 가진다. 때문에 최적 구현이 활발하게 진행되는 분야이며, 같은 ARX 기반 블록암호라면 다른 블록암호에도 최적 구현 기법의 적용이 용이한 장점이 있다. [8]은 제안 기법과 유사한 방법으로 다른 블록암호인 LEA, HIGHT를 최적 구현 한 것이다. 이처럼 본 논문의 제안 기법은 ARX 기반 블록암호라면 적용 가능한 것을 확인할 수 있다. 다만 적용에 있어서 암호 알고리즘의 내부 구조를 면밀히 분석하여 최적 구현이 가능한 지점을 정확히 확보해야 하며, 그렇지 않다면 정확한 구현이 어려울 수 있다.

이후로 본 기법을 사용하여 또 다른 ARX 기반 블록암호를 최적화 하기 위해 지속적인 알고리즘 분석 및 실제 구현을 통한 성능 향상 측정을 후속 연구로 제시한다.

References

[1] 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," International Workshop on Cryptographic Hardware and Embedded Systems (CHES'07), pp. 450-466, Sep. 2007.

[2] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK lightweight block ciphers," Proceedings of the 52nd Annual Design Automation Conference (DAC'15), pp. 1-6, June 2015.

[3] B.W. Koo, D.Y. Roh, H.J. Kim, Y.H. Jung, D.G. Lee, and D.S. Kwon, "CHAM: A family of lightweight block ciphers for resource-constrained devices," International Conference on Information Security and Cryptology (ICISC'17), pp. 3-25, Nov. 2017.

[4] D.Y. Roh, B.W. Koo, Y.H. Jung, I.W. Jeong, D.G. Lee, D.S. Kwon, and W.H. Kim, "Revised version of block cipher CHAM," International Conference on Information Security and Cryptology (ICISC'19), pp. 1-19, Dec. 2019.

[5] H.J. Seo, "Memory-efficient implementation of ultra-lightweight block cipher algorithm CHAM on low-end 8-bit AVR processors," Journal of the Korea Institute of Information Security & Cryptology, 28(3), pp. 545-550, June 2018.

[6] H.D. Kwon, H.J. Kim, S.J. Choi, K.B. Jang, J.H. Park, H.J. Kim, and H.J. Seo, "Compact implementation of CHAM block cipher on low-end microcontrollers," World Conference on Information Security Applications (WISA'20), pp. 120-134, Aug. 2020.

[7] H.D. Kwon, S.W. An, Y.B. Kim, H.J. Kim, S.J. Choi, K.B. Jang, J.H. Park, H.J. Kim, S.C. Seo, and H.J. Seo, "Designing a CHAM block cipher on low-end microcontrollers for internet of things," Multidisciplinary Digital Publishing Institute Electronics, 9(9), 1548, pp 1-16, Sep. 2020.

[8] Y.B. Kim, H.D. Kwon, S.W. An, H.J. Seo, and S.C. Seo, "Efficient implementation of ARX-based block ciphers on 8-bit AVR microcontrollers," Multidisciplinary Digital Publishing Institute Mathematics, 8(10), 1837, pp 1-22, Oct. 2020.

〈저자소개〉

권 혁 동 (Hyeok-dong Kwon) 학생회원
2018년 2월: 한성대학교 정보시스템공학과 공학 학사 졸업
2020년 2월: 한성대학교 IT융합공학부 석사 졸업
2020년 3월~현재: 한성대학교 정보컴퓨터공학과 박사과정
〈관심분야〉 정보보안, 암호구현

장 경 배 (Kyoung-bae Jang) 학생회원
2019년 2월: 한성대학교 IT응용시스템공학과 공학 학사 졸업
2019년 3월~현재: 한성대학교 IT융합공학부 석사과정
〈관심분야〉 IoT, 정보보안

박 재 훈 (Jae-hoon Park) 학생회원
2020년 2월: 한성대학교 IT응용시스템공학과 공학 학사 졸업
2020년 3월~현재: 한성대학교 IT융합공학부 석사과정
〈관심분야〉 웹보안

서 화 정 (Hwa-jeong Seo) 종신회원
2010년 2월: 부산대학교 컴퓨터공학과 학사 졸업
2012년 2월: 부산대학교 컴퓨터공학과 석사 졸업
2016년 2월: 부산대학교 컴퓨터공학과 박사 졸업
2015년 4월~5월: 싱가포르 난양공대 인턴쉽
2016년 1월~2017년 3월: 싱가포르 과학기술청 연구원
2017년 4월~현재: 한성대학교 조교수
〈관심분야〉 암호구현