32-bit RISC-V 프로세서 상에서의 초경량 블록 암호 알고리즘 Revised CHAM 구현

심민주*, 엄시우*, 권혁동*, 송경주*, 서화정*†
*한성대학교 IT융합공학부
*† 한성대학교 IT융합공학부
minjoos9797@gmail.com, shuraatum@gmail.com, korlethean@gmail.com,
thdrudwn98@gmail.com, hwajeong84@gmail.com

Implementation of Ultra-Lightweight Block Cipher Algorithm Revised CHAM on 32-Bit RISC-V Processor

Min-Joo Sim*, Si-Woo Eum*, Hyeok-Dong Kwon*, Gyeong-Ju Song*,
Hwa-Jeong Seo*†
*Dept. of IT Convergence Engineering, Hansung University
*† Dept. of IT Convergence Engineering, Hansung University

요 약

ICISC'19에서 기존 CHAM과 동일한 구조와 규격을 갖지만, 라운드 수만 증가시킨 revised CHAM이 발표되었다. CHAM은 사물인터넷에서 사용되는 저사양 프로세서에서 효율적인 구현이 가능한 특징을 갖고 있다. AVR, ARM 프로세서 상에서의 CHAM 암호 알고리즘에 대한 최적 구현은 존재하지만, 아직 RISC-V 프로세서 상에서의 CHAM 구현은 존재하지 않는다. 따라서, 본 논문에서는 RISC-V 프로세서 상에서의 Revised CHAM 알고리즘을 최초로 구현을 제안한다. CHAM 라운드 함수의 내부 구조의 일부를 생략하여 최적 구현하였다. 그리고 홀수 라운드와 짝수 라운드를 모듈별로 구현하여 필요에 따라 모듈을 호출하여 손쉽게 사용할 수 있게 하였다. 결과적으로, RISC-V 상에서 제안 기법 적용하기 전보다 제안 기법 적용 후에 12%의 속도 향상을 달성하였다.

1. 서론

최근 사물인터넷의 발전 속도가 급격하게 빨라짐에 따라 사물인터넷이 탑재된 제품들이 상용화되고 있다. 이에 따라 사물인터넷에 사용되는 저사양 프로세서들의 보안이 필수적으로 요구되어, 사물인터넷에서 효율적으로 동작하는 암호 알고리즘들이 제안되고 있다. CHAM 암호 알고리즘은 사물인터넷에 사용하기 효율적인 알고리즘 중 하나이다[1]. 이러한 CHAM 암호 알고리즘에 대해 라운드 수만 증가된 Revised CHAM이 제안되었다[2].

본 논문에서는 저사양 프로세서 중 하나인 32-bit RISC-V 프로세서 상에서 Revised CHAM 암호 알고리즘을 최초로 구현하며, CHAM 라운드 함수의 내부 구조의 일부 생략하는 효율적인 구현을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서 Revised CHAM과 RISC-V 프로세서에 대해 설명한다. 3장에서는 구현 기법에 대해 설명하고, 4장에서는 제안하는 구현 기법에 대한 성능 평가를 진행한다. 마지막으로 5장에서는 본 논문에 대한 결론을 내린다.

2. 관련 연구

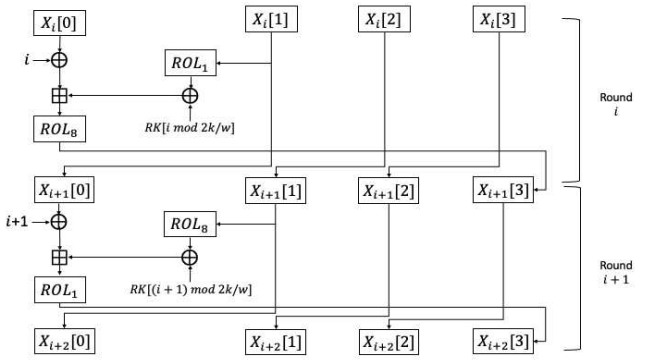
2.1 Revised CHAM cipher

ICISC'17에서 발표된 CHAM은 ARX(Addition, Rotation, XOR)연산을 사용하는 Feistel 구조로 되어 있는 초경량 국산 블록 암호 알고리즘이다.

cipher	n	k	r	w	k/w
CHAM-64/128	64	128	88	16	8
CHAM-128/128	128	128	112	32	4
CHAM-128/128	128	256	120	32	8

[Table 1] List of Revised CHAM ciphers and their parameters.

Revised CHAM은 기존 CHAM과 동일하게 3가지 암호화를 제공하고, 동일한 규격을 갖으며, 라운드 수의 차이만 있다. Revised CHAM의 각각의 암호화에 해당되는 파라미터는 [Table 1]과 같다.



[Fig 1] Round function of Revised CHAM block cipher.

CHAM 암호 알고리즘에서 사용되는 홀수, 짝수 라운드에 해당되는 연산은 각 라운드 내부에서 실행되는 두 개의 로테이션 연산을 제외하고 동일한 연산이 실행된다. CHAM의 라운드 함수는 [Fig 1]과 같다.

2.2 RISC-V processor

RISC-V 프로세서 중 32-bit 구조인 RV32I는 32개의 32-bit 레지스터를 제공한다[3]. RISC-V 프로세서의 레지스터의 용도는 [Table 2]와 같다.

$x0$ 레지스터는 항상 0의 값을 갖는 레지스터이고, 이외 $x0$~$x4$의 레지스터는 특수 용도에 사용되는 레지스터로 지정되어 있다. Callee saved 레지스터인 $s0$~$s11$를 제외한 $a0$~$a7$, $t0$~$t6$은 사용하기 전에 이전 값을 보존할 필요 없이 사용이 가능하다. 이때, $a0$~$a7$ 레지스터는 function argument를 고려한 후, 사용해야한다.

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

[Table 2] Purpose of registers in RISC-V processor.

3. 제안 기법

3.1 블록 이동 단계 생략[4, 5]

CHAM 암호화 과정은 평문을 4개의 블록으로 나누고, 한 라운드의 마지막 부분에서 블록들은 왼쪽으로 하나씩 블록의 이동이 진행된다. 이 블록의 이동 과정은 4개의 블록으로 구성되어 있기 때문에 4 라운드마다 블록을 이동하지 않았을 때와 동일한 블록의 자리로 돌아오게 된다.

[Fig 1]을 보면 매 라운드마다 동일하게 첫 번째, 두 번째 블록에서만 암호화 과정에 필요한 연산이 진행되는 것을 확인 수 있다. 하지만, 88 라운드로 동작하는 Revised CHAM-64/128의 특징을 활용하면, 블록 이동 단계를 생략할 수 있다. 블록 이동을 생략한 암호화 과정은 [Fig 2]와 같다.

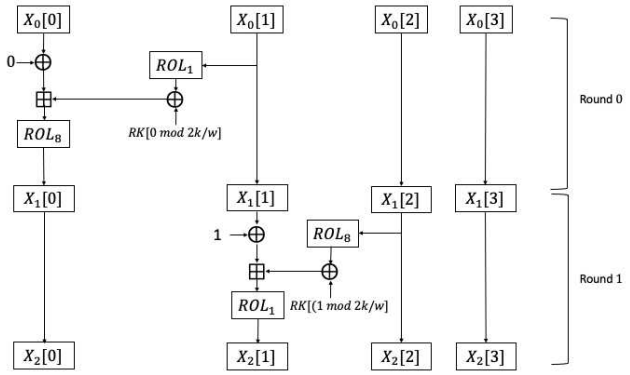


Fig 2. Structure in which block-movement step is omitted.

0 ~ 3 라운드를 기준으로 생각해보면, 0 라운드에서는 기존과 동일하게 첫 번째, 두 번째 블록에서 연산을 진행하고, 1 라운드에서는 두 번째, 세 번째 블록, 2 라운드에서는 세 번째, 네 번째 블록, 마지막으로 3 라운드에서 네 번째, 첫 번째 블록에서 연산을 진행한다. 그리고 이후, 4 라운드에서는 연산을 진행하는 블록의 위치가 첫 번째, 두 번째 블록인 0 라운드와 동일한 위치에서 연산을 진행한다.

이와 같이 4 라운드마다 블록의 위치가 되돌아오는 것을 알 수 있다. 4 라운드를 확장하여 생각해보면, 88 라운드에도 동일하게 적용가능하다. 따라서, 해당 기법을 전체 라운드에 적용하여 전체 라운드에 블록 이동 과정을 생략한다.

3.2 구현 코드

라운드 별로 암호화 연산이 진행되는 블록에 해당

되는 블록($a2$~$a5$)을 입력으로 넣어 사용이 가능하다. 스택을 사용하게 되면, 메모리 접근을 하여 속도가 저하된다. 그렇기 때문에 제안 기법에 대해 구현할 때, 스택을 사용하지 않는 레지스터를 활용하여 구현하였다. Function argument 값을 갖고 있는 $a0$,$a1$을 각각 평문과 라운드 키 값을 갖는다.

```
Input: \o0, \o1
Output : \o0, \o1
1. mv       t4, \o0

2. slli     t0, t4, 1
3. srli     t1, t4, 15
4. xor      t4, t0, t1
5. and      t4, t4, t5    // ROL_1

6. lh       t2, 0(a1)
7. xor      t4, t4, t2
8. addi     a1, a1, 2

9. xor      \o1, \o1, t3
10. and     \o1, \o1, t5

//add on RISC-V without carry flag
11. add     \o1, \o1, t4
12. sltu    \t0, \o1, t4
13. add     \o1, \o1, t0
14. and     \o1, \o1, t5

15. slli    t0, t4, 8
16. srli    t1, t4, 8
17. xor     t4, t0, t1
18. and     t4, t4, t5    //ROL_8

19. addi    t3, t3, 1
```

[Table 3] Odd round code implemented with macro; One of the values of \o0 and \o1 is (a3, a4), (a4, a5), (a5, a6) and (a6, a3).

홀수, 짝수 라운드에서 진행되는 로테이션 연산이 서로 다르기 때문에 홀수 라운드와 짝수 라운드를 모듈별로 구현하여 필요에 따라 모듈을 호출함으로써, 효율적으로 코드를 관리를 하고 손쉽게 사용할 수 있게 하였다. [Table 3]은 홀수 라운드를 모듈로 구현한 코드이다.

CHAM의 구조를 활용하기 위해 한 블록에 32-bit 레지스터에서 하위 16-bit를 사용하여 구현하였다. 로테이션 연산은 따로 명령어가 존재하지 않기 때문에 쉬프트 연산인 slli, srli 명령어와 xor 명령어를 사용하여 구현하였다. 로테이션 연산을 위해 사용한 오른쪽 쉬프트 연산 시에 원하지 않는 값인 상위 비트의 값이 추가될 수 있다. 이를 사전에 방지하기 위해 로테이션 연산 후, 0xffff 값이 저장되어 있는 $t5$와 로테이션 연산이 저장되어 있는 $t4$를 and 연산을 하여 상위 16-bit를 모두 0으로 변경한다. 그리고 다른 연산을 진행할 때에도 발생할 수 있기 때문에 위와 동일한 연산을 통해 상위 16-bit를 0으로 유지한다.

RISC-V는 캐리 플래그를 제공하지 않기 때문에, 캐리가 발생한 것을 확인하기 위해 sltu 명령어를 사용하였다. [Table 3]의 12번째 줄에서 \o1은 $t4$와 \o1의 값이 더해진 결과가 저장되어 있는 상태이다. sltu 명령어는 \o1과 $t4$의 값을 비교하여 $t4$의 값이 \o1보다 클 경우, 1을 반환한다. 캐리가 발생하지 않았다면, \o1의 결과 값이 $t4$보다 항상 크지만, 덧셈 연산과정에서 캐리가 발생하였다면 $t4$가 더 큰 것으로 판단되기 때문에 반환된 1을 \o1에 다시 더해줌으로써 덧셈 연산을 진행한다.

4. 성능 평가

본 장에서는 SiFive사의 HiFive1 Rev B 플랫폼인 RISC-V를 활용한 제안 기법에 대한 성능을 비교한다.

32-bit RISC-V 프로세서에 대한 CHAM 구현은 존재하지 않아 Revised CHAM에 대한 Reference-C 코드, RISC-V 상에서 구현한 revised CHAM, 제안 기법이 적용된 구현물에 대해 cpb(Cycle Per Block)를 측정하여 성능 비교를 한다. 성능 비교 결과는 [Table 4]와 같다.

Reference C code	This work	This work*
9215	2378	1931

[Table 4] Evaluation result on RISC-V with optimization level *-O2* in terms of execution timing(i.e. clock cycles); Notation(*) indicates with optimization techniques.

제안 기법을 적용한 것은 Reference-C 코드보다 47%의 속도 향상을 확인하였으며, RISC-V 상에서 구현한 Revised CHAM보다 12%의 속도 향상을 확인하였다.

5. 결론

본 논문에서는 RISC-V 상에서 CHAM 구현을 최초로 구현하였다. 또한, 블록 이동 단계를 생략한 Revised CHAM을 구현하였다. CHAM 암호화 과정에서 수행되는 블록 이동은 4 라운드 마다 다시 동일한 블록으로 돌아온다. 이 특성을 활용하여 블록 이동 연산을 생략하였다.

결론적으로 본 논문은 해당 기법을 통해 RISC-V 상에서 구현한 revised CHAM 대비 12%의 속도 향상을 확인하였다. 향후 연구로 해당 기법을 활용하여 카운터 모드 등 다양한 운용모드의 효율적인 최적화 구현을 제안한다.

6. Acknowledgment

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

참고문헌

[1] Bonwook Koo, Dongyoung Roh, Hyeonjin Kim, Younghoon Jung, Dong-Geon Lee, and Daesung Kwon, “CHAM: A Family of Lightweight Block Ciphers for Resource-Constrained Devices”, International Conference on Information Security and Cryptology(*ICICS'17*), pp 3-25, 2017.

[2] Dongyoung Roh, Bonwook Koo, Younghoon Jung, IlWoong Jeong, Dong-Geon Lee, Daesung Kwon, and Woo-Hwan Kim, “Revised Version of Block Cipher CHAM”, International Conference on Information Security and Cryptology(*ICISC'19*), pp 1-19, 2019.

[3] SiFive, Inc. “The RISC-V Instruction Set Manual Volume I: User-Level ISA Document Version 2.2”, 2017.
https://risicv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf.

[4] Taeung Kim and Deukjo Hong. “Software Implementation of Lightweight Block Cipher CHAM for Fast Encryption”, Journal of the Korea Society of Computer and Information, Vol. 23, No. 10, pp 111-117, 2018.

[5] Hyeokdong Kwon, SangWoo An, YoungBeom Kim, Hyunji Kim, SeungJu Choi, Kyoungbae Jang, Jaehoon Park, Hyunjun Kim, Seogchung Seo and Hwajeong Seo, “Designing a CHAM Block Cipher on Low-End Microcontrollers for Internet of Things”, Electronics 9, No.9:1548, 2020.
https://doi.org/10.3390/electronics9091548, 2020.