32-bit RISC-V 프로세서 상에서의 경량 블록암호 SIMECK 최적 병렬 구현

심민주*, 엄시우*, 권혁동*, 서화정*†

*한성대학교 IT융합공학부(대학원생)

*†한성대학교 IT융합공학부 (교수)

Optimized Parallel Implementation of Lightweight Block Cipher SIMECK on 32-bit RISC-V Processor

Min-Joo SIM*, Si-Woo Eum*, Hyeok-Dong Kwon*, Hwa-Jeong Seo*†

*Dept. of IT Convergence Engineering, Hansung University (Graduate student)

*†Dept. of IT Convergence Engineering, Hansung University (Professor)

요 약

CHES'15에서 경량 블록 암호인 SIMON과 SPECK을 이점만을 결합한 SIMECK이 발표되었다. 본 논문에서는 RISC-V 프로세서 상에서의 SIMECK 암호 알고리즘에 대해 단일 평문 구현과 2개의 평문에 대한 병렬 구현을 제안한다. SIMECK 라운드 함수의 마지막 연산에 블록 이동이 존재하지만, 모든 구현에 대해서 블록 이동을 생략하는 최적 기법을 제안한다. 2개의 평문에 대한 병렬 구현은 모든 레지스터 내부 정렬과 동일 연산을 진행하는 서로 다른 값인 16-bit 2개의 값이 섞이지 않는 효율적인 로테이션 연산을 구현하였다. 32-bit 워드 사이즈를 갖는 암호에 대한 효율적인 로테이션 연산을 제안하였다. 결과적으로, RISC-V에서 단일 평문 구현물 중에서는 워드 크기가 32-bit인 SIMECK-64/128이 가장 효율적 구현이 가능하였으며, Reference-C 대비 446%의 성능 향상을 확인하였다. 2개의 평문 병렬 구현물 중에서는 워드 크기가 16-bit인 SIMECK-32/64가 가장 효율적 구현이 가능하였으며, 단일 평문 구현물 대비 206%의 성능 향상을 확인하였다.

I. 서론

SIMECK 암호 알고리즘은 효율적인 경량 암호 중 하나이다. SIMECK은 SIMON의 라운드 함수를 수정하고, SPECK과 유사한 키 스케줄링을 사용한다[1]. 본 논문에서는 저사양 프로세서 중 하나인 32-bit RISC-V 프로세서 상에서의 SIMECK-32/64와 SIMECK-64/128의 단일 평문 구현과 최적 병렬 구현 기법을 제안한다.

II. 관련연구

2.1 SIMECK

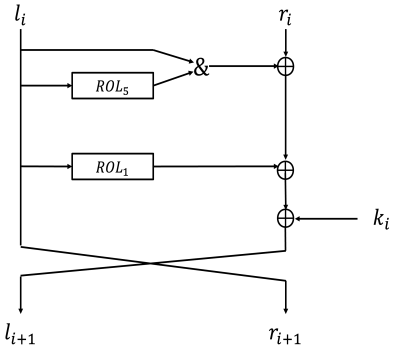
CHES'15에서 발표된 SIMECK은 Feistel 구조로 되어 있는 경량 블록 암호 알고리즘이다. SIMECK은 NSA(National Security Agency)에서 개발한 경량 블록 암호인 SIMON과 SPECK[2]의 장점을 결합하여 개발되었다. SIMECK의 각각의 암호화에 해당되는 파라미터는 [Table 1]과 같다.

Cipher	n	k	r	w
SIMECK-32/64	32	64	32	16
SIMECK-48/96	48	96	36	24
SIMECK-64/128	64	128	44	32

[Table 1] List of SIMECK ciphers and their parameters.

SIMECK 암호 알고리즘은 로테이션 연산, XOR 연산, AND 연산, 블록의 이동으로 단순한 구조로 되어 있다. [Fig 1]은 SIMECK의 라운드

함수를 나타낸 것이다.



[Fig 1] Round function of SIMECK.

2.2 RISC-V processor

RISC-V 프로세서 중 32-bit 구조인 RV32I는 32개의 32-bit 레지스터를 제공한다. x0 레지스터는 항상 0의 값을 갖는 레지스터이고, 이외 x1~x4의 레지스터는 특수 용도에 사용되는 레지스터로 지정되어 있다[3]. RISC-V 프로세서의 레지스터의 용도는 [Table 2]와 같다.

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	

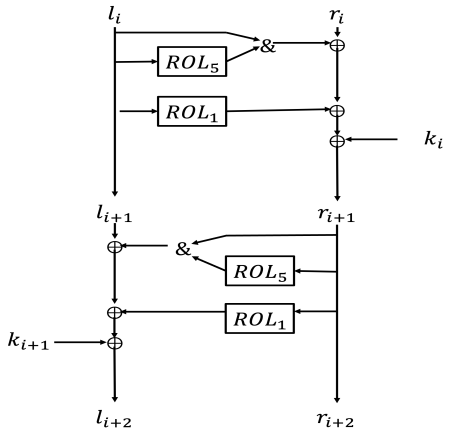
[Table 2] Purpose of registers in RISC-V processors.

III. 제안 기법

본 장에서는 32-bit RISC-V의 구조를 효율적으로 사용하기 위해 SIMECK-48/96은 제외하고, SIMECK-32/64와 SIMECK-64/128에 대한 단일 평문 구현과 2개의 평문 병렬 구현에 대해 설명한다.

3.1 블록 이동 제거

SIMECK 암호화 과정에서 수행되는 블록 이동은 1개의 라운드마다 진행된다. 하지만, 두 개의 블록에 대한 블록의 이동이 매 라운드마다 진행되는 것은 (라운드의 수*2) 만큼의 블록 이동을 위한 명령어를 사용함을 뜻한다. 따라서, 모든 라운드의 두 번째 블록에서 진행하는 연산에 대해 [Fig 2]와 같이 블록 이동을 생략하였다. 블록 이동 제거 기법은 기존과 달리 홀수, 짝수 라운드로 구분하여 홀수 라운드에서는 두 번째 블록에서 연산을 하고 짝수 라운드에서는 첫 번째 블록에서 연산을 진행하고 구현하였다.



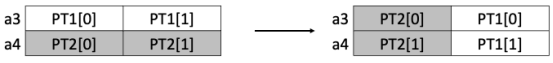
[Fig 2] Optimized structure in which the block movement step is omitted.

3.2 단일 평문 최적 구현

SIMECK-32/64는 워드 사이즈가 16-bit이기 때문에 32-bit 레지스터의 절반인 16-bit만 사용하여 구현을 하였고, SIMECK-64/128은 워드 사이즈가 32-bit이기 때문에 32-bit를 모두 활용하여 사용하여 구현하였다. RISC-V에서는 로테이션 명령어를 제공하지 않아 쉬프트 연산과 XOR 연산을 이용하여 구현하였다.

3.3 2개의 평문 최적 병렬 구현

3.3.1 SIMECK-32/64



[Fig 3] Register internal alignment in SIMECK-32/64.

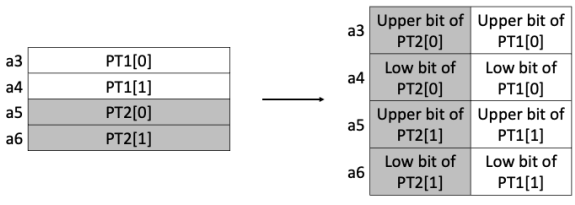
16-bit 블록 단위로 연산하는 SIMECK-32/64를 구현하기 위해 32-bit 레지스터를 16-bit만

사용하여 암호화하는 것은 효율적이지 않다. 따라서, 32-bit 레지스터를 모두 활용하기 위해 2개의 평문에 대한 암호화를 진행하는 병렬 구현을 진행한다. [Fig 3]과 같이 레지스터 내부 정렬을 하였다.

[Fig 4] Process of Left Rotation 1 in SIMECK-32/64.

단일 평문 구현과 달리 병렬 구현에서는 서로 다른 두 개의 16-bit 값이 연속으로 정렬된 하나의 레지스터에 대한 로테이션 연산을 하기 위해서는 값이 섞이지 않아야 한다. 하지만, 로테이션 연산을 하기 위해 쉬프트 연산을 사용하는데 쉬프트 연산을 하면, 서로 다른 값들이 섞이게 된다. 이를 막기 위해 AND 연산을 사용하여 섞이는 위치의 값을 0으로 바꿔주고, 섞이는 위치에 대한 값들을 temp로 사용하는 레지스터에 저장해 준다. 쉬프트 연산을 완료 후, temp에 저장된 값을 XOR 연산하여 로테이션을 구현하는 기법을 [Fig 4]와 같이 구현하였다.

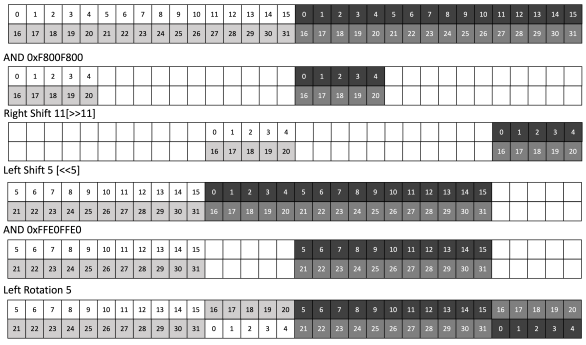
3.3.2 SIMECK-64/128



[Fig 5] Register internal alignment in SIMECK-64/128.

32-bit 블록 단위로 연산하는 SIMECK-64/128을 구현하기 위해서 하나의 레지스터에 서로 다른 평문의 상위 16-bit 혹은 하위 16-bit를 묶어주기 위해 [Fig 5]와 같이 레지스터 내부 정렬을 하였다.

단일 평문 구현과 SIMECK-32/64 병렬 구현은 왼쪽 쉬프트 연산과 오른쪽 쉬프트 연산한다. 하지만, SIMECK-64/128의 경우, 왼쪽 쉬프트 연산만 진행한다. 하위 비트가 위치한 레지스터를 왼쪽 쉬프트 연산하여 없어지는 값이 상위 비트에 위치되어야 하고, 상위 비트에 위치한 값도 동일하게 하위 비트에 위치하여야 하기 때문에 쉬프트 연산을 하기 전에 해당 값들을 temp에 저장한다. temp에 저장된 값들은 로테이션 연산 이후에 로테이션 연산으로 값이 섞이는 위치에 해당되는 값이기 때문에 오른쪽 쉬프트 연산을 해준다. 그리고 SIMECK-32/64 병렬 구현과 동일하게 값이 섞이지 않게 구현하였다. temp에 위치한 값은 하위 비트에 대한 값은 상위 비트에 상위 비트에 대한 값은 하위 비트에 값을 XOR 연산하여 [Fig 6]과 같이 구현하였다. 이 기법을 사용하면, 워드 사이즈가 32-bit인 암호의 병렬 구현 시 효율적인 로테이션 연산이 가능하다.



[Fig 6] Process of Left Rotation 5 in SIMECK-64/128.

IV. 성능평가

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

32-bit RISC-V 프로세서에 대한 SIMECK 구현이 존재하지 않아 SIMECK의 Reference-C와 블록 이동 제거 기법이 적용된 구현물에 대한 Clock Cycles과 CPB(Cycle Per Block)를 측정하여 성능을 비교한다. 성능 비교 결과는 [Table

3]과 같다.

SIMECK-32/64	clock cycles	cpb
Ref-c	3,298	824.50
1-pt*	655	163.75
2-pt*	635	158.75
SIMECK-64/128	clock cycles	cpb
Ref-c	2,734	341.75
1-pt*	612	76.50
2-pt*	1,560	195

[Table 3] Evaluation result on RISC-V with; Notation(*) indicates with optimization techniques(block movement step omitted).

SIMECK-32/64의 단일 평문 구현물은 Reference-C 대비 503%의 성능 향상을 확인하였다. 2개의 평문에 대한 병렬 구현물은 Reference-C 대비 1,038%의 성능 향상을 확인하였고, 단일 평문 구현물 대비 206%의 성능 향상을 확인하였다. SIMECK-64/128의 단일 평문 구현물은 Reference-C 대비 446%의 성능 향상을 확인하였다. 2개의 평문에 대한 병렬 구현물은 Reference-C 대비 350%의 성능 향상을 확인하였지만, 단일 평문 구현물 대비 127%의 성능 저하를 확인하였다.

SIMECK-32/64 병렬 구현은 2개의 레지스터에 위치한 값에 대해 5개의 명령어를 사용하여 로테이션 연산을 구현하였다. 반면, SIMECK-64/128의 워드 사이즈는 32-bit이기 때문에 병렬 구현을 하기 위한 로테이션 연산은 4개의 레지스터에 위치한 값에 대해 12개의 명령어를 사용하여 로테이션 연산을 구현하였다. 로테이션 연산을 효율적으로 구현하였지만, 명령어를 더 많이 사용하여 이와 같은 성능이 저하된 것으로 생각된다.

V. 결론

본 논문에서는 RISC-V 상에서의 SIMECK 단일 평문 구현과 2개의 평문에 대한 최적 병렬 구현을 제안한다. SIMECK 암호화 과정에서 수행되는 블록 이동은 매 라운드마다 진행되는데 홀수, 짝수 라운드로 구분하여 홀수 라운드에서는 두 번째 블록에서 연산을 진행하고 짝수 라운드에서는 첫 번째 라운드에서 연산을 진행하게 최적 구현하여 블록 이동 제거하였다. 블록 이동 생략 기법은 모든 구현물에 대해 적용하였다. 단일 평문 구현의 경우, SIMECK-32/64는 32-bit 레지스터의 절반인 16-bit만 사용하여 구현을 하였고, SIMECK-64/128은 32-bit 모두 사용하여 구현하였다. 2개의 평문에 대한 병렬 구현의 경우, SIMECK-32/64는 32-bit 레지스터를 모두 사용하기 위한 효율적인 레지스터 정렬과 32-bit 레지스터에 저장된 두 개의 값들이 서로 섞이지 않는 로테이션 연산을 효율적으로 구현하였다. 그 결과, 단일 평문 구현보다 206%의 성능 향상을 확인하였다. SIMECK-64/128은 단일 평문 구현물 대비 127% 성능 저하를 확인하였지만, 워드 사이즈가 32-bit인 다른 경량 블록 암호의 병렬 구현 시 효율적인 로테이션 기법을 제안하였다. 향후 연구로 해당 기법을 활용한 효율적인 병렬 구현을 제안한다.

VI. Acknowledgement

이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No.2018-0-00264, IoT 융합형 블록체인 플랫폼 보안 원천 기술 연구, 50%) 그리고 이 성과는 2022년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2020R1F1A1048478, 50%)

[참고문헌]

[1] G. Yang, B. Zhu, V. Suder, M.D. Aagaard, , G. Gong, "The SIMECK family of lightweight block ciphers," In Cryptographic Hardware and Embedded Systems, CHES 2015, pp. 307-329. 2015.

[2] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith and L. Wingers, "The SIMON and SPECK light- weight block ciphers," Proceedings of the 2015 IEEE Design Automation Conference, pp. 1-6,

Jun. 2015.
[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/risc v-spec-v2.2.pdf.