32-bit RISC-V 아키텍처에서 ARX 기반 경량 블록암호 SPECK32와 CHAM64의 최적 병렬 구현*

신명수*, 신한범*, 홍득조**, 성재철***, 홍석희****

*, ****고려대학교 (대학원생, 교수), **전북대학교(교수), ***서울시립대학교(교수)

Optimized Parallel Implementation of ARX-based Lightweight Block Cipher SPECK32 and CHAM64 on 32-bit RISC-V Architectures

Myoungsu Shin*, Hanbeom Shin*, Deukjo Hong**, Jaechul Sung***, Seokhie Hong****

*, ****Korea University(Graduate student, Professor), **Jeonbuk National University(Professor), ***University of Seoul(Professor)

요 약

ARX 기반 경량 블록암호인 SPECK과 CHAM은 저사양 임베디드 기기 환경에서도 안전한 통신 채널을 확보하기 위한 요구사항으로 제안되었다. RISC-V는 오픈 소스 명령어 집합 아키텍처로 누구나 추가적인 비용 없이 사용할 수 있는 ISA로 높은 관심을 받고 있다. 본 논문에서는 32-bit RISC-V 아키텍처에서 ARX 기반 경량 블록암호인 SPECK32와 CHAM64에 대해 2블록 암호화 최적 병렬 구현을 제시한다. 본 논문에서 제안한 기법으로 구현한 결과, CHAM64의 경우 기존에 제시된 병렬 구현 기법보다 바이트당 클록 사이클 수 관점에서 66.8% 향상된 성능을 보여주고, SPECK32의 경우 레퍼런스 구현에 비해 62.7% 향상된 성능을 보여준다. 특히 본 논문에서 제안한 모듈러 덧셈 병렬 연산 기법은 다른 ARX 기반 암호를 병렬 구현하거나 벡터 연산을 지원하지 않는 임베디드용 아키텍처에서 병렬 구현할 때 좋은 성능을 보일 것으로 기대한다.

I. 서론

IoT 기술의 발전으로 인해 많은 종류의 임베디드 기기가 사용되고 있다. 임베디드 기기와 같은 제한된 컴퓨팅 환경에서 안전한 통신 채널을 확보하기 위해 경량 암호는 필수적이다. 이러한 요구사항으로 여러 경량 블록암호가 제안되었으며 본 논문에서 다루는 SPECK[1]과 CHAM[2]은 ARX(Addition, Rotation, XOR) 기반 경량 블록암호로, 범용 프로세서뿐만 아니라 저사양 컴퓨팅 환경에서도 효율적인 구현이 가능하여 다양한 환경에서의 구현에 관한 연구가 활발히 진행되고 있다.

RISC-V[4]는 오픈 소스 명령어 집합 아키텍처(Instruction Set Architecture, ISA)로 독점 ISA인 x86, ARM과 달리 라이센스 비용 없이 누구나 사용할 수 있다. 단순한 명령어 집합을 가지고 있어 경량화 프로세서 및 임베디드 시스템에 적합한 특징을 가지고 있다. RISC-V에 대한 높은 관심에 따라 암호 구현에 관한 연구 또한 진행되고 있다.

Sim et. al[5]은 32-bit RISC-V에서 CHAM64의 병렬 구현으로, 1블록 병렬 구현과 2블록 병렬 구현 방법을 제시하였고, 모듈러 덧셈 병렬 연산 시, 15번째 비트와 나머지 비트를 분리하여 연산하는 방법을 사용하였다.

본 연구는 32-bit RISC-V 아키텍처에서 기본적인 명령어 집합인 RV32I를 사용하여 SPECK32과 CHAM64에 대해 최적 병렬 구현 기법을 제시하고 성능을 평가한다. 기존 CHAM64 2블록 병렬 구현에 비해 바이트당 클록 사이클 수 관점에서 66.8% 성능 향상을 이루었으며, 기존 결과가 없는 SPECK32의 경우 레퍼런스 C 언어 구현에 비해 62.7% 성능 향상을 이루었다. 또한 본 논문에서 제안하는 개선된 모듈러 덧셈 병렬 구현 기법은 다른 ARX 기반 블록암호를 최적 병렬 구현하는데 적용할 수 있다. 32-bit RISC-V 아키텍처에서 SPECK32와 CHAM64에 대한 구현물은

`https://github.com/dampers/RISC-V-SPECK32-CHAM64`

에서 확인할 수 있다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 논문에 대한 배경지식으로 SPECK과 CHAM의 명세 및 32-bit RISC-V 아키텍처를 소개한다. Ⅲ장에서는 본 논문에서 사용한 32-bit RISC-V 아키텍처에서 SPECK32와 CHAM64의 최적 병렬 구현 기법을 소개한다. Ⅳ장에서는 32-bit RISC-V 실물 보드를 사

*

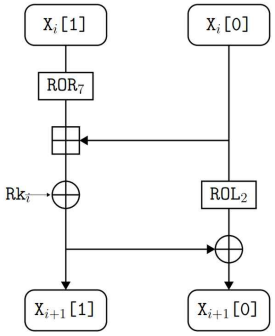

Figure 1. round function of SPECK32

용하여 구현물과 이전 연구 결과와 비교 및 평가하고 V장에서 본 논문에 대한 결론을 내린다.

II. 배경지식

2.1 표기법

$X_i[j]$: i-라운드 입력 블록의 j번째 워드
$x \| y$: x와 y의 연접(concatenation)
$x \boxplus y$: x와 y의 2^{16}상에서의 모듈러 덧셈
$x \underset{32}{\boxplus} y$: x와 y의 2^{32}상에서의 모듈러 덧셈
$x \underset{32}{\boxminus} y$: x와 y의 2^{32}상에서의 모듈러 뺄셈
$x \oplus y$: x와 y의 XOR 연산
$\mathrm{ROL}_n(x)$: x의 왼쪽으로 n만큼 비트 순환이동 연산 (bitwise ROtation Left)
$\mathrm{ROR}_n(x)$: x의 오른쪽으로 n만큼 비트 순환이동 연산 (bitwise ROtation Right)

2.2 경량 블록암호 SPECK

SPECK[1]은 NSA가 2013년에 공개한 ARX 기반 경량 블록암호이며 파라미터에 따라 10가지 종류를 가진다. Figure 1.은 SPECK의 10가지 종류 중 본 논문에서 다루는 SPECK32의 라운드 함수를 나타낸다. 22라운드, 32-bit 블록 크기, 64-bit 키 크기를 가지는 SPCEK32에서 32-bit 블록을 16-bit 워드 2개로 나누어 $(X[1], X[0])$로 정의한다. i번째 라운드 입력을 $(X_i[1], X_i[0])$, 16-bit 라운드키를 rk_i라고 하면, i번째 라운드 출력 $(X_{i+1}[1], X_{i+1}[0])$은 다음과 같이 계산된다.

$$X_{i+1}[1] = (ROR_7(X_i[1]) \boxplus X_i[0]) \oplus rk_i,$$
$$X_{i+1}[0] = ROL_2(X_i[0]) \oplus X_{i+1}[1].$$

2.3 경량 블록암호 CHAM

CHAM[2]은 국가보안기술연구소에서 제안한 ARX 기반 블록암호로, ICISC 2017에서 처음 제안되었다. 추가적인 안전성 분석을 통해 라운드 수를 추가한 reivsed CHAM[3]이 ICISC 2019에서 제안되었다. CHAM-n/k에서 n은 블록 크기를, k는 키 크기를 나타내며 CHAM-64/128, CHAM-128/128, CHAM-128/256 3가지 종류를 가지고 revised

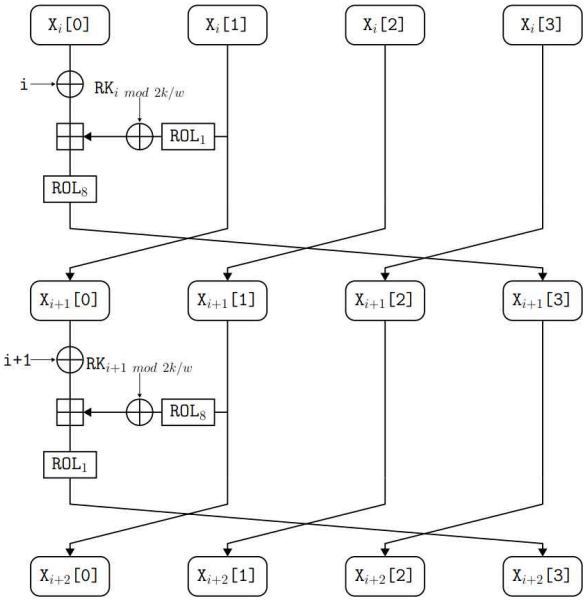

Figure 2. 2-round encryption process of CHAM

CHAM을 기준으로 각각 88, 112, 120 라운드가 사용된다. CHAM은 4-branch GFN(General Feistel Network) 구조를 가지며 CHAM64의 워드 크기 w는 16-bit, CHAM-128/128과 CHAM-128/256의 워드 크기는 32-bit이다. Figure 2.는 CHAM의 2-라운드 암호화 과정을 나타낸 것이다. CHAM의 2-라운드 암호화 과정에서 i번째(짝수) 라운드와 $i+1$번째(홀수) 라운드의 차이는 비트 순환이동의 크기에 있다. 짝수 라운드는 $X[0]$에 8만큼, $X[1]$에 1만큼 수행하고 홀수 라운드는 $X[0]$에 1만큼, $X[1]$에 8만큼 수행한다. 본 논문에서는 CHAM의 3가지 종류 중 CHAM-64/128만을 다루므로 이를 CHAM64로 축약하여 표기한다.

2.4 32-bit RISC-V 아키텍처

32-bit RISC-V 아키텍처는 누구나 무료로 사용할 수 있는 오픈 소스 명령어 집합 아키텍처이다[4]. 기본 ISA는 모든 RISC-V 프로세서가 구현해야 하는 최소한의 기능 집합을 의미한다. 32-bit 용 RISC-V ISA는 RV32I로 32개의 32-bit 레지스터를 가지고 있으며, 산술 및 논리연산과 메모리 접근 연산을 포함하여 최소한의 명령어로 이루어져 있다. 비트 이동 연산 명령어는 있지만 비트 순환이동 명령어는 정의되어있지 않다. 따라서 비트 이동 연산 및 AND, OR, XOR와 같은 논리연산을 사용하여 구현하여야 한다.

III. 제안 기법

본 장에서는 32-bit RISC-V 아키텍처에서 SPECK32와 CHAM64의 2블록 병렬 구현에 사용한 기법을 설명한다. 먼저 SPECK32와 CHAM64 모두에 사용한 기법을 설명하고 각 암호에 적용한 기법들을 설명한다.

a3	$X[0]$	$X'[0]$
a4	$X[1]$	$X'[1]$

(a) SPECK32

a3	$Y[0]$	$Y'[0]$
a4	$Y[1]$	$Y'[1]$
a5	$Y[2]$	$Y'[2]$
a6	$Y[3]$	$Y'[3]$

(b) CHAM64

Figure 3. RV32I register internal alignment in SPECK32 and CHAM64

3.1 레지스터 내부 정렬

16-bit 워드를 사용하는 SPECK32와 CHAM64를 32-bit 아키텍처에서 2블록 병렬 구현하기 위해 32-bit 레지스터에 2개의 16-bit 워드를 넣어 연산을 수행한다. 두 블록의 워드들이 같은 연산이 수행될 수 있도록 교차하여 배치한다.

Figure 3.은 SPECK32 2블록(X,X')과 CHAM64 2블록(Y,Y')에 대한 레지스터 내부 정렬을 그림으로 나타낸 것이다. RV32I 명령어 중 16-bit 로드 명령어 `lhu`(load half unsigned)를 사용하여 각 32-bit 레지스터에 16-bit 워드씩 로드한 후, $X(Y)$블록의 16-bit 워드 값들에 16만큼 왼쪽으로 비트 이동 연산을 한다. 그리고 $X(Y)$블록과 $X'(Y')$블록의 같은 인덱스 워드를 저장한 레지스터끼리 XOR 연산하여 Figure 3.과 같이 두 블록을 배치한다.

3.2 모듈러 덧셈 병렬 연산

하나의 32-bit 레지스터에 서로 다른 두 블록의 16-bit 워드가 연접하여 배치되어 있어, 모듈러 덧셈 연산 시 캐리 발생에 유의해야 한다. 32-bit 모듈러 덧셈 연산을 수행했을 때 15번째 비트에서 발생한 캐리가 16번째 비트에 더해져 다른 블록의 연산에 영향을 주게 된다. 이를 방지하기 위해 2블록 모듈러 덧셈 병렬 연산 기법을 제안한다. 제안 기법의 기본 아이디어는 32-bit 모듈러 덧셈에서 15번째 비트에서 캐리가 발생하여 16번째 비트에 캐리 비트가 더해졌을 때, 이 값을 빼는 것이다. branch를 활용하여 캐리 발생 여부를 확인하면 상수시간 구현이 아니므로 타이밍 공격이 가능하다. 따라서 산술 및 비트 논리 연산으로만 이를 구현한다.

먼저 15번째 비트에서 캐리 발생 여부를 확인해야 한다. 이를 위해 XOR 연산한 값과 32-bit 모듈러 덧셈 연산한 값에 대해 XOR 연산을 수행한다. 그 결과 32-bit 모듈러 덧셈 연산에서 모든 비트에 더해

Table 1. 2 block modular addition in RV32I

input : $A(x\|x'), B(y\|y'), M(=0\text{x}00010000), T(temp)$
output : $D(x\boxplus y\|x'\boxplus y')$

```
.macro ADDITION A, B, T, M, D
    xor   \T, \A, \B
    add   \D, \A, \B
    xor   \T, \T, \D
    and   \T, \T, \M
    sub   \D, \D, \T
.endm
```

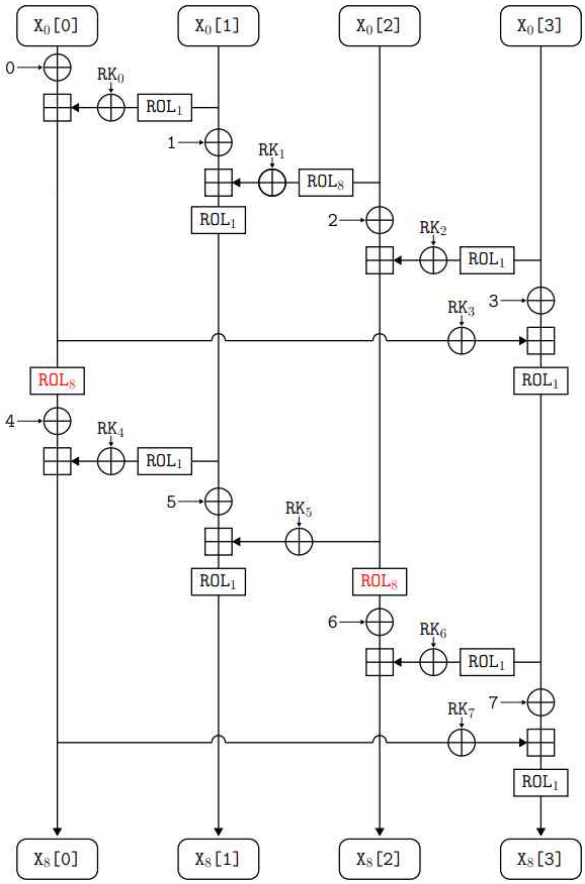

Figure 4. Initial 8-round modified encryption process of CHAM64

진 캐리 비트를 얻을 수 있다. 32-bit 레지스터 모든 비트에 더해진 캐리 비트와 0x00010000 값을 AND 연산하여 16번째 비트 값만 남기면 16번째 비트에 더해진 캐리 값을 얻을 수 있다. 32-bit 모듈러 덧셈한 값에서 16번째 비트에 더해진 캐리 값을 빼면 상위 및 하위 16-bit 워드끼리 모듈러 덧셈한 값을 정확하게 얻을 수 있고 상수시간 구현이 가능하다. 이를 수식으로 나타내면 다음과 같다.

$$D = (A \underset{32}{\boxplus} B) \underset{32}{\boxminus} (((A \oplus B) \oplus (A \underset{32}{\boxplus} B)) \wedge 0\text{x}00010000).$$

Table 1.은 2블록 모듈러 덧셈 연산을 RV32I 명령어로 구현한 것이다. 마스크값인 M은 0x00010000으로 고정이고 T는 임시 변수를 나타낸다.

3.3 SPECK32 2블록 병렬 구현

SPECK32는 워드 단위 순환이동이 없고, 라운드마다 비트 순환이동 연산 크기가 같으므로 한 라운드를 구현하여 22회 반복하는 방법으로 구현한다. 비트 순환이동을 위한 마스크값 4개와 3.2절에서 제시한 덧셈 병렬 연산에서 사용할 마스크값 1개로 총 5개의 레지스터를 마스크값 저장에 사용해야 한다.

3.4 CHAM64 2블록 병렬 구현

CHAM64 워드 단위 순환이동이 있으며 주기가 4로 4라운드마다 초기 위치로 돌아오게 된다. 따라서

4라운드를 구현하여 워드 단위 순환이동 생략 기법[2, 6]을 사용할 수 있다. 또한 16라운드마다 같은 라운드 키를 사용하므로 모든 라운드 키를 미리 로드하고 16라운드를 구현하여 라운드 키 로드 비용을 최소화할 수 있다. revised CHAM[3]을 기준으로 CHAM64은 88라운드를 가지므로 16라운드 구현을 5회 반복하고 따로 8라운드 구현을 수행한다.

Figure 4.는 CHAM64의 초기 8-라운드 암호화 함수를 재구조화하여 나타낸 그림이다. CHAM64의 경우 16-bit 워드를 사용하기 때문에 ROL_8 연산을 2회 연속으로 수행하면 비트 순환이동 연산을 수행하지 않은 것과 같다. 따라서 0번째 라운드와 2번째 라운드의 ROL_8을 제거하고 1번째 라운드를 제외한 나머지 홀수 라운드의 ROL_8을 제거할 수 있다. 그리고 4번째 라운드부터 모든 짝수 라운드의 ROL_8을 라운드 상수 XOR 연산 앞에 배치한다. 마지막으로 CHAM64의 88번째 라운드 수행 후 0번째와 2번째 워드에만 ROL_8을 수행하여 위치를 맞춰준다. 최종적으로 총 88회 ROL_8 연산을 45회로 축소할 수 있다.

IV. 평가

본 장에서는 32-bit RISC-V 아키텍처에서 SPECK32와 CHAM64를 구현한 결과에 대해 성능을 분석하고 평가한다. 타켓 보드는 SiFive 사의 HiFive1 Rev B 모델이고 SiFive 사의 FreedomStudio (ver. 2020-6)를 활용하여 성능 측정 환경을 구축한다. 컴파일러는 riscv64-unknown-elf-gcc 8.3.0을 사용하고, 컴파일 최적화 옵션은 -O3로 설정한다. 키스케줄을 미리 계산하여 라운드키를 저장하여 사용하고, 2블록 암호화 과정의 성능을 평가한다. Table 2.는 본 연구의 구현 결과와 기존 연구의 결과를 비교한 표이다. SPECK32의 경우 32-bit RISC-V 아키텍처에서 구현에 관한 기존 연구가 없기 때문에 SPECK32의 제안 논문의 C 언어 구현, C 언어 구현을 어셈블리로 구현한 코드와 제안 기법을 적용한 구현물에 대해 2블록 암호화한 바이트당 클록 사이클 수를 비교한다. CHAM64은 2블록 병렬 구현에 대한 기존 연구 결과와 바이트당 클록 사이클 수를 비교한다. 그 결과, SPECK32는 제안 논문의 C 코드보다 62.7%, 어셈블리 구현 코드보다 55% 성능 향상을 이루었다. CHAM64는 기존 최적 결과인 Sim et. al[5] 구현 결과에 비해 66.8% 성능 향상을 이루었다.

Table 2. Comparison of SPECK32 and CHAM64 implementations on 32-bit RISC-V
(cpb : cycles per byte)

Cipher	latency (cpb)	Method
SPECK32	96.00	reference C [1]
SPECK32	91.50	asm code of ref C
SPECK32	**59.00**	[this work]
CHAM64	152.25	Sim et. al [5]
CHAM64	**91.25**	[this work]

V. 결론

본 논문에서는 32-bit RISC-V 아키텍처에서 SPECK32의 2블록 병렬 최적 구현을 최초로 제시하고 제안 논문의 C 코드 및 이의 어셈블리 구현물과 비교하였다. 그리고 CHAM64의 2블록 병렬 최적 구현을 제시하여 그 성능을 기존 구현물과 비교하였다. 제안 기법은 이전 연구보다 향상된 모듈러 덧셈 병렬 연산 기법을 적용하여 덧셈 연산에 필요한 명령어 수를 줄일 수 있다. 또한 CHAM64의 라운드 함수를 재구조화하여 ROL_8 연산을 제거하는 방법을 적용하였다. 그 결과, SPECK32는 제안 논문의 C 코드보다 62.7%, 어셈블리 구현 코드보다 55% 성능 향상을 이루었고, CHAM64는 기존 연구 결과에 비해 66.8% 성능 향상을 이루었다.

특히 본 논문에서 제안한 모듈러 덧셈 병렬 연산 기법은 RV32I 명령어로 16-bit 워드를 사용하는 다양한 ARX 기반 암호를 병렬 구현 시 활용할 수 있다. 이 기법을 응용하여 벡터 연산을 지원하지 않는 아키텍처에서 병렬 구현할 때 적용할 수 있다.

[참고문헌]

[1] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The Simon and Speck Families of Lightweight Block Ciphers" Cryptology ePrint Archive, Jun. 2013.

[2] 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.

[3] 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.

[4] A. Waterman and K. Asanovic, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA" EECS Department, UCBerkeley, Technical Report, UCB/EECS-2011-62 116, May 7, 2017.

[5] M. Sim, J. Jang, S. Eum, H. Kwon, G. Song, and H. Seo, "Optimized Parallel Implementation of Lightweight Block Cipher Revised CHAM on 32-bit RISC-V Processor" CISC-W'21, 2021.

[6] Kwon, Hyeokdong, SangWoo An, YoungBeom Kim, Hyunji Kim, Seung Ju Choi, Kyoungbae Jang, Jaehoon Park, Hyunjun Kim, Seog Chung Seo, and Hwajeong Seo. "Designing a CHAM Block Cipher on Low-End Microcontrollers for Internet of Things" Electronics 9, no. 9: 1548., 2020.