# 32-bit RISC-V 프로세서 상에서의 경량 블록 암호 SIMECK-CTR 최적 구현

https://youtu.be/HCdnCZa-Ydo

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

## 서론

- 사물인터넷이 사용되는 저사양 프로세서들의 보안 필수적
  - 사물인터넷에서 효율적으로 동작하는 경량 암호 알고리즘 제안

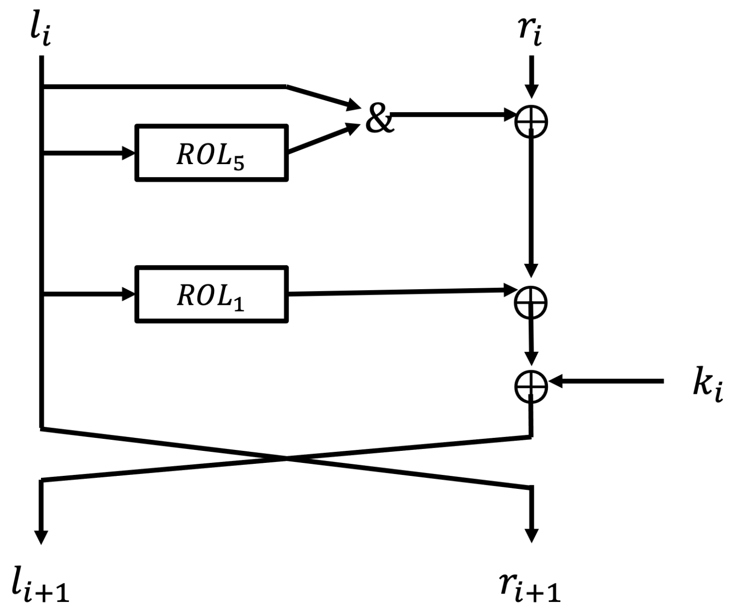
- 저사양 프로세서 중 하나인 RISC-V 상에서의 SIMECK-CTR 암호 알고리즘 구현
  - SIMECK 최적 구현 연구는 다양한 환경에서 활발히 진행
  - 저사양 프로세서 상에서의 SIMECK-CTR 에 대한 최적 구현 연구 X

# SIMECK

- CHES'15에서 발표된 Feistel 구조의 경량 블록암호

- NSA에서 개발한 경량 블록 암호인 SIMON과 SPECK의 장점을 결합하여 개발
  - SIMON의 라운드 함수를 일부 변형
  - SPECK의 키 스케줄링과 유사한 구조

| 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 |



Round function of SIMECK.

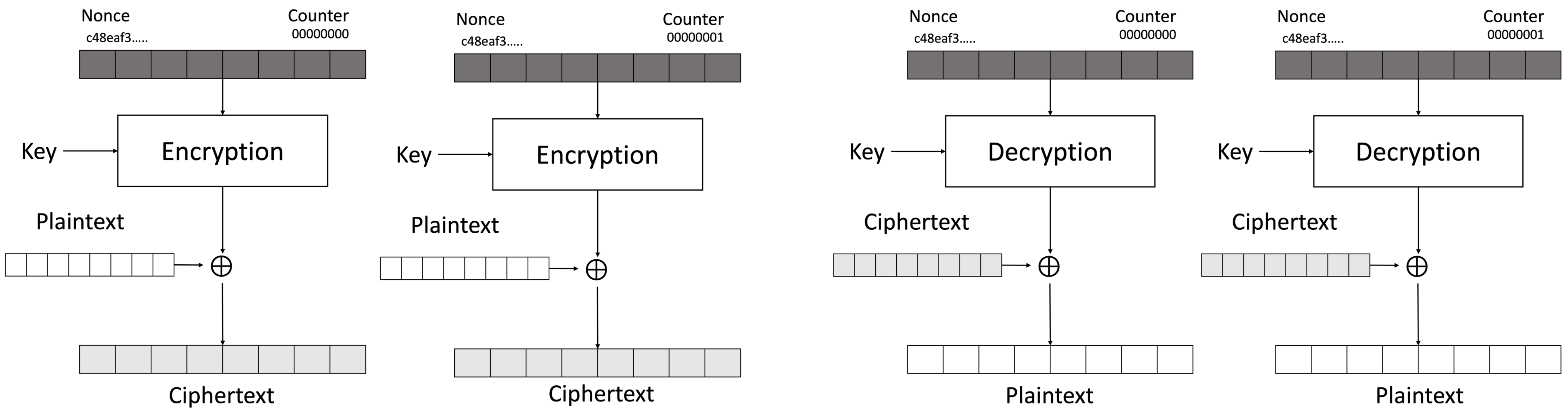
# 카운터 운용 모드

- 입력 : nonce 값 + Counter 값

  ( nonce : 고정된 임의의 상수 / Counter : 블록을 암호화 할 때마다 1씩 증가)

- 이렇게 결합된 입력 값을 암호화하여 키 스트림 생성

- 암호문 : 생성된 키 스트림 ⊕ 평문



# RISC-V Processor

- 오픈 소스로 제공되는 명령어 셋을 기반으로 한 프로세서

- 32-bit 프로세서인 RV32I에서는 32개의 32-bit 레지스터 제공

| X0 | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | X10 | X11 | X12 | X13 | X14 | X15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ZERO** | **RA** | **SP** | **GP** | **TP** | T0 | T1 | T2 | **S0** | **S1** | A0 | A1 | A2 | A3 | A4 | A5 |
| **X16** | **X17** | **X18** | **X19** | **X20** | **X21** | **X22** | **X23** | **X24** | **X25** | **X26** | **X27** | **X28** | **X29** | **X30** | **X31** |
| A6 | A7 | **S2** | **S3** | **S4** | **S5** | **S6** | **S7** | **S8** | **S9** | **S10** | **S11** | T3 | T4 | T5 | T6 |

RA : Return Address Register

SP : Stack Pointer Register

GP : Global Pointer Register

TP  : Tread Pointer Register

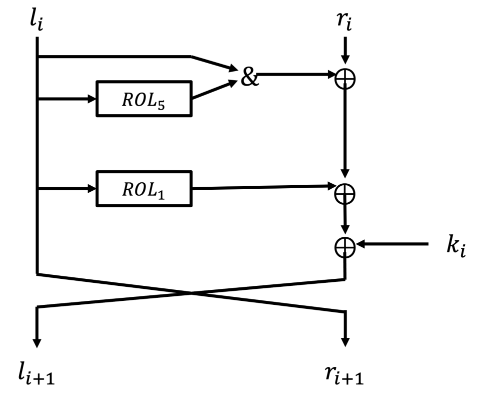
T0 ~ T6 : temporal registers

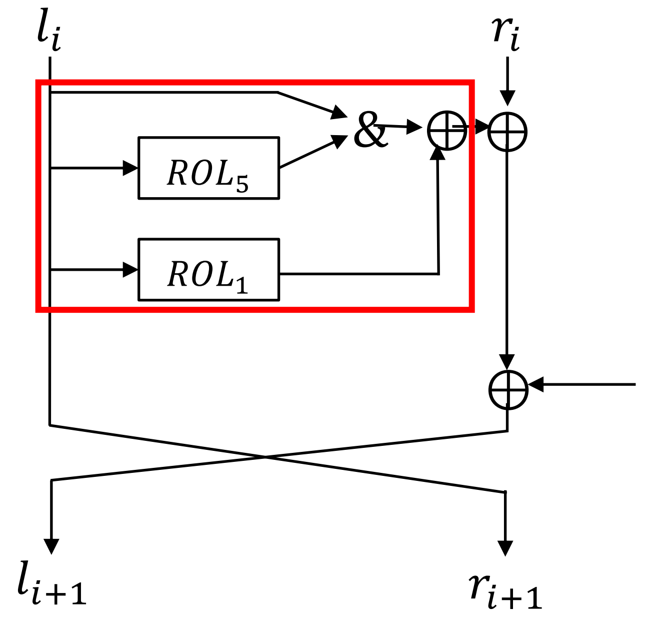
S0 ~ S11 : saved registers(callee saved)

A0 ~ A7 : function arguments and return value

# 구현 기법

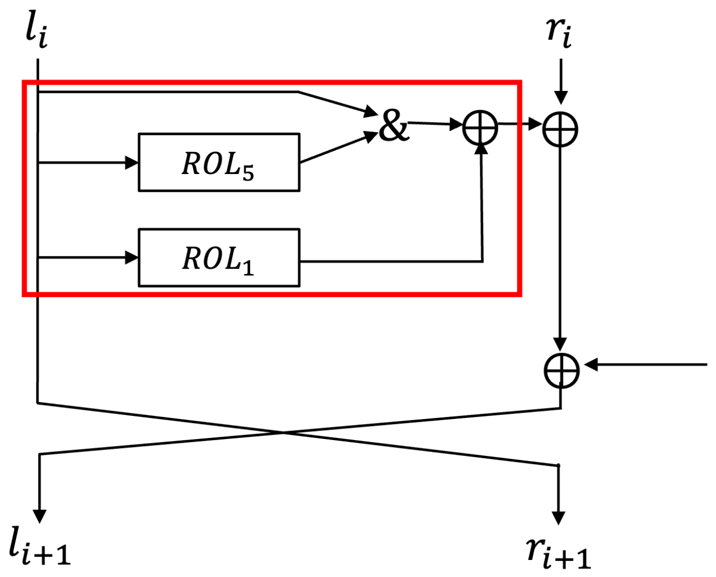
- 고정된 nonce 블록이 카운터 블록에 영향을 받기 전까지 해당되는 연산에 대해 사전 연산 가능
- 카운터 블록에 영향을 받기 전 **1라운드 일부만 사전 연산 가능**
- 왼쪽 블록: nonce, 오른쪽  블록 : 카운터
  - 일반적인 카운터 값 32-bit
  - SIMECK-32/64의 경우, 카운터 값으로 16-bit 사용
  - SIMECK-64/128의 경우, 카운터 값으로 32-bit 사용





# 구현 기법-SIMECK-32/64

- 단일 평문 & 2개의 평문 병렬 구현
  - 3번의 XOR 연산 , 4번의 쉬프트 연산, 1번의 AND 연산 생략



- 암호화 시작 전, 레지스터 내부 정렬 진행
  - 서로 다른 두 개의 평문을 통해 연속 정렬

SIMECK-32/64

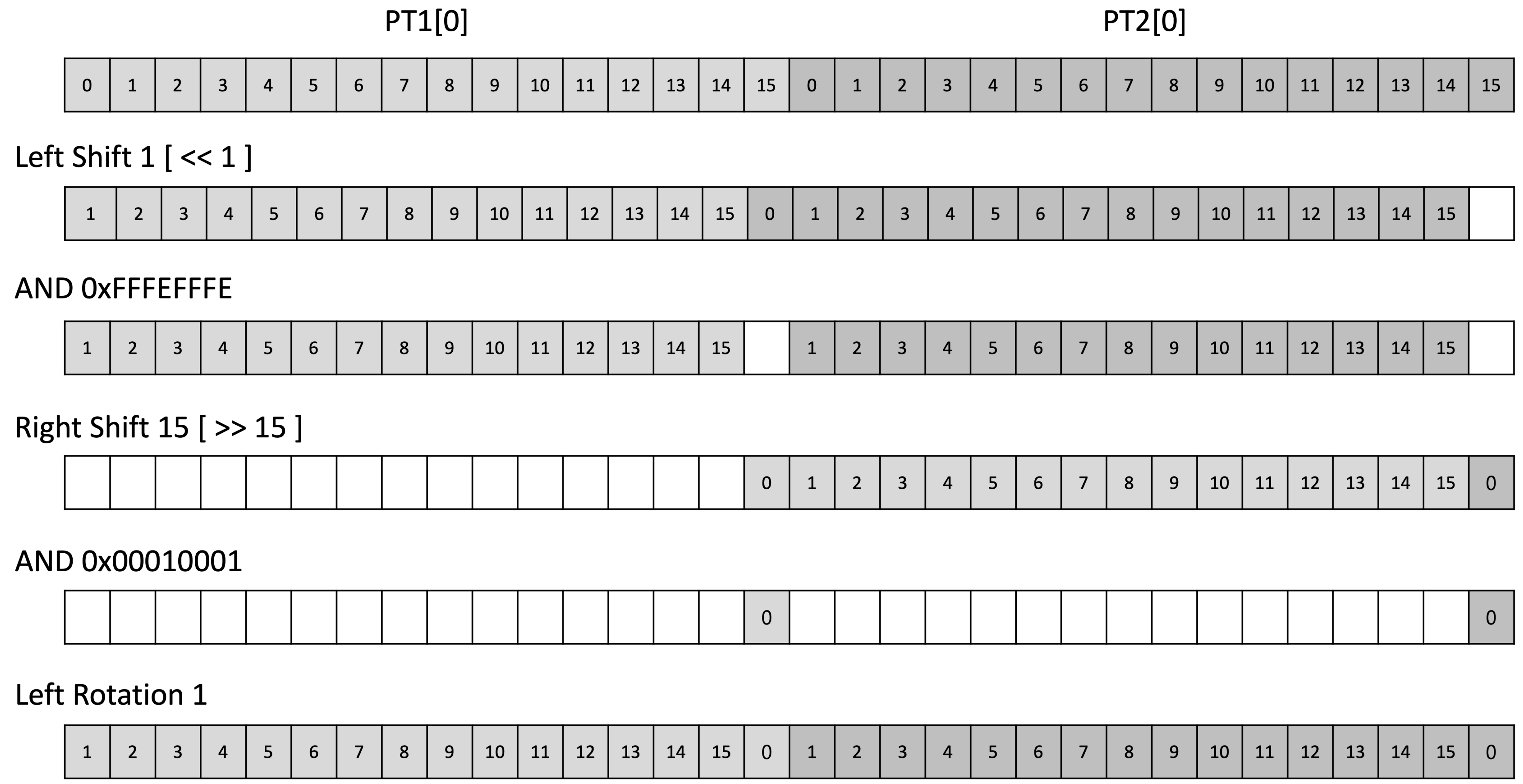
| | | |
|---|---|---|
| a3 | PT1[0] | PT1[1] |
| a4 | PT2[0] | PT2[1] |

$\longrightarrow$

| | | |
|---|---|---|
| a3 | PT2[0] | PT1[0] |
| a4 | PT2[1] | PT1[1] |

# 구현 기법-SIMECK-32/64

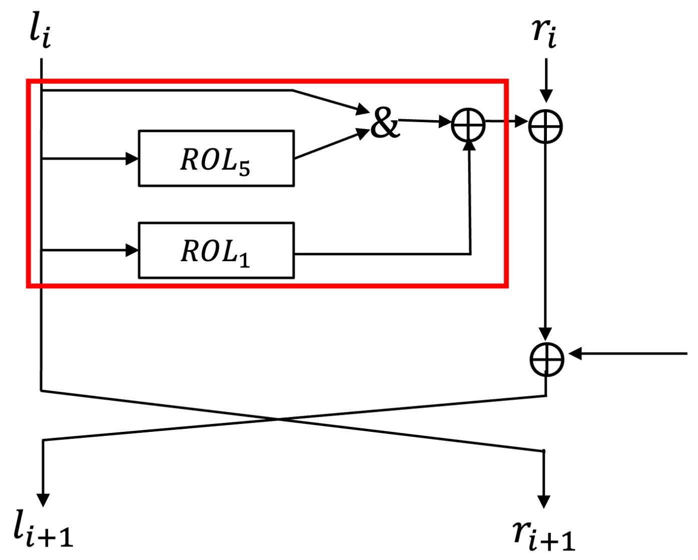
- 로테이션 연산



# 구현 기법-SIMECK-64/128

- 단일 평문 구현의 사전 연산을 통한 생략
  - 3번의 XOR 연산 , 4번의 쉬프트 연산, 1번의 AND 연산 생략

- 2개의 평문 병렬 구현의 사전 연산을 통한 생략
  - 6번의 XOR 연산, 8번의 쉬프트 연산, 10번의 AND 연산 생략



SIMECK-64/128

| | |
|---|---|
| a3 | PT1[0] |
| a4 | PT1[1] |
| a5 | PT2[0] |
| a6 | PT2[1] |

$\longrightarrow$

| | | |
|---|---|---|
| a3 | Upper bit of PT2[0] | Upper bit of PT1[0] |
| a4 | Low bit of PT2[0] | Low bit of PT1[0] |
| a5 | Upper bit of PT2[1] | Upper bit of PT1[1] |
| a6 | Low bit of PT2[1] | Low bit of PT1[1] |

# 구현 기법-SIMECK-64/128

- 로테이션 연산

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

AND 0xF800F800

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 2 | 3 | 4 |  |  |  |  |  |  |  |  |  |  |  |
| 16 | 17 | 18 | 19 | 20 |  |  |  |  |  |  |  |  |  |  |  | 16 | 17 | 18 | 19 | 20 |  |  |  |  |  |  |  |  |  |  |  |

Right Shift 11[>>11]

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 2 | 3 | 4 |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 2 | 3 | 4 |
|  |  |  |  |  |  |  |  |  |  |  | 16 | 17 | 18 | 19 | 20 |  |  |  |  |  |  |  |  |  |  |  | 16 | 17 | 18 | 19 | 20 |

Left Shift 5 [<<5]

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |  |  |  |  |  |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |  |  |  |  |  |

AND 0xFFE0FFE0

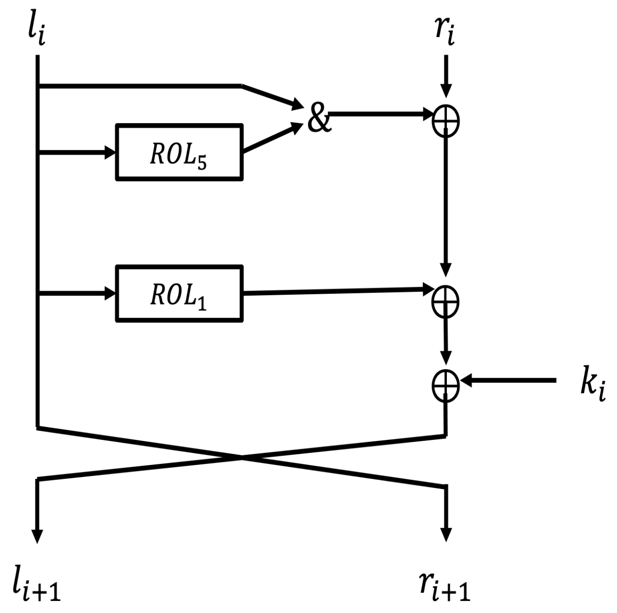
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |  |  |  |  |  | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |  |  |  |  |  |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |  |  |  |  |  | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |  |  |  |  |  |

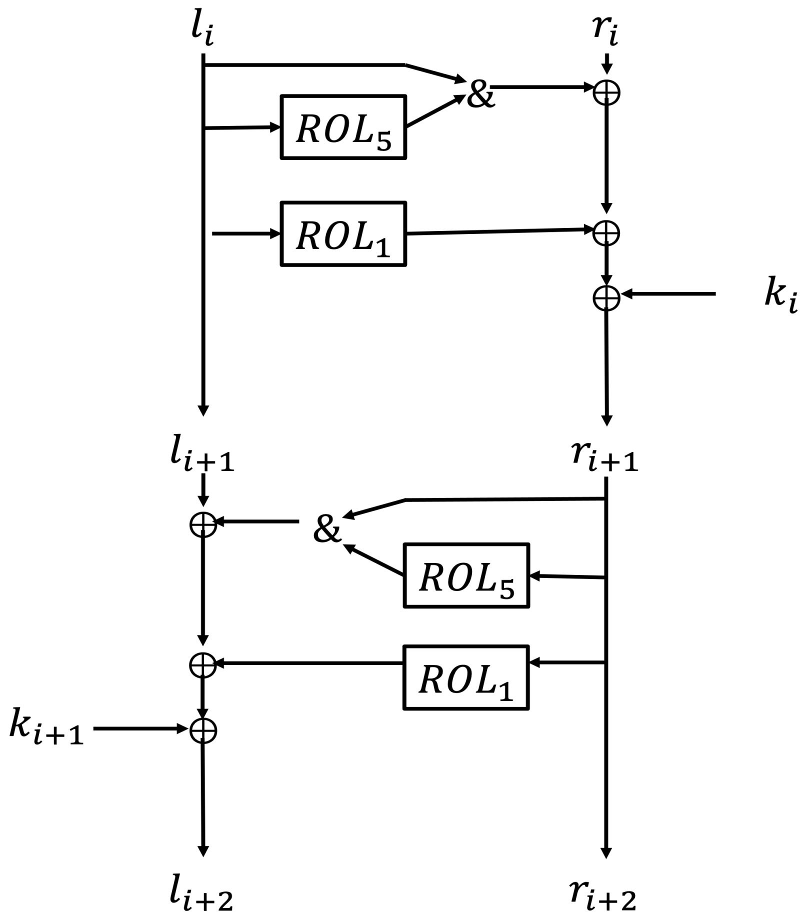
Left Rotation 5

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 0 | 1 | 2 | 3 | 4 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 0 | 1 | 2 | 3 | 4 |

# 구현 기법

- 블록 이동 단계 생략
  - 매 라운드마다 반복되는 블록 이동
  - 2라운드마다 블록의 위치가 되돌아오는 특징 활용





## 성능 평가

- 제안한 기법이 적용된 최적 구현은 기존 연구[1] 대비 **1%** 성능 향상

| SIMECK-32/64 | clock cycles |
|---|---|
| Ref-c | 3,298 |

| SIMECK-64/128 | clock cycles |
|---|---|
| Ref-c | 2,734 |

| SIMECK-32/64 | [1] | Our work |
|---|---|---|
| 1-PT | 655 | **646** |
| 2-PT | 635 | **625** |

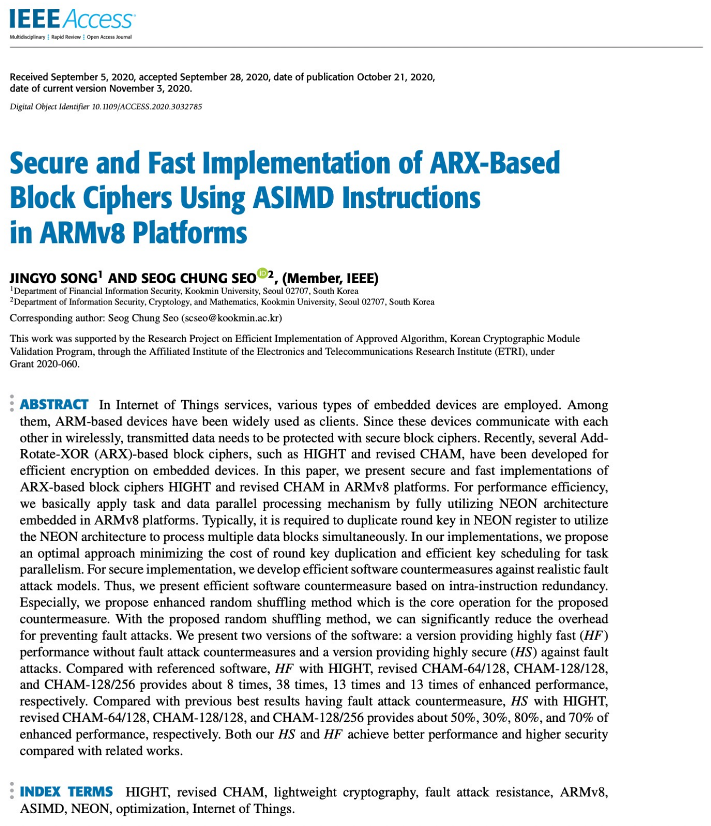
| SIMECK-64/128 | [1] | Our work |
|---|---|---|
| 1-PT | 612 | **594** |
| 2-PT | 1,560 | **1,540** |

Evaluation result : clock cycles

[1]M. Sim, S. Eum, H. Kwon, H. Seo, "Optimized Parallel Implementation of Lightweight Block Cipher SIMECK on 32-bit RISC-V Processor", Conference on Information Security and Cryptography Summer 2022.

# 향후 진행 방향…

- 아래 논문 참조하여 Fault Attack에 안전한 SIMECK 구현 예정
  - SIMON, SPECK …



IEEE Access

Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 5, 2020, accepted September 28, 2020, date of publication October 21, 2020, date of current version November 3, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3032785*

**Secure and Fast Implementation of ARX-Based Block Ciphers Using ASIMD Instructions in ARMv8 Platforms**

**JINGYO SONG[1] AND SEOG CHUNG SEO[2], (Member, IEEE)**

[1]Department of Financial Information Security, Kookmin University, Seoul 02707, South Korea

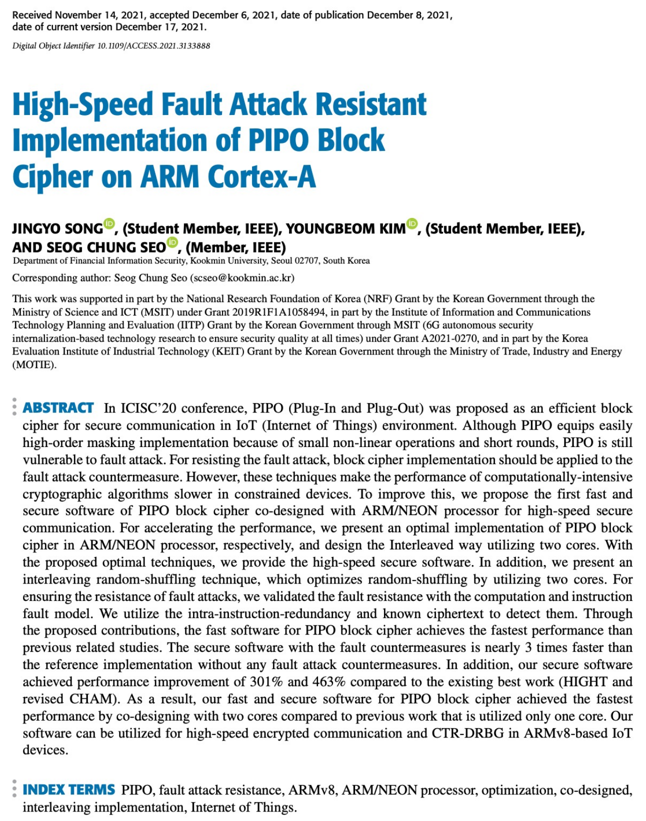
[2]Department of Information Security, Cryptology, and Mathematics, Kookmin University, Seoul 02707, South Korea

Corresponding author: Seog Chung Seo (scseo@kookmin.ac.kr)

This work was supported by the Research Project on Efficient Implementation of Approved Algorithm, Korean Cryptographic Module Validation Program, through the Affiliated Institute of the Electronics and Telecommunications Research Institute (ETRI), under Grant 2020-060.

**ABSTRACT** In Internet of Things services, various types of embedded devices are employed. Among them, ARM-based devices have been widely used as clients. Since these devices communicate with each other in wirelessly, transmitted data needs to be protected with secure block ciphers. Recently, several Add-Rotate-XOR (ARX)-based block ciphers, such as HIGHT and revised CHAM, have been developed for efficient encryption on embedded devices. In this paper, we present secure and fast implementations of ARX-based block ciphers HIGHT and revised CHAM in ARMv8 platforms. For performance efficiency, we basically apply task and data parallel processing mechanism by fully utilizing NEON architecture embedded in ARMv8 platforms. Typically, it is required to duplicate round key in NEON register to utilize the NEON architecture to process multiple data blocks simultaneously. In our implementations, we propose an optimal approach minimizing the cost of round key duplication and efficient key scheduling for task parallelism. For secure implementation, we develop efficient software countermeasures against realistic fault attack models. Thus, we present efficient software countermeasure based on intra-instruction redundancy. Especially, we propose enhanced random shuffling method which is the core operation for the proposed countermeasure. With the proposed random shuffling method, we can significantly reduce the overhead for preventing fault attacks. We present two versions of the software: a version providing highly fast (*HF*) performance without fault attack countermeasures and a version providing highly secure (*HS*) against fault attacks. Compared with referenced software, *HF* with HIGHT, revised CHAM-64/128, CHAM-128/128, and CHAM-128/256 provides about 8 times, 38 times, 13 times and 13 times of enhanced performance, respectively. Compared with previous best results having fault attack countermeasure, *HS* with HIGHT, revised CHAM-64/128, CHAM-128/128, and CHAM-128/256 provides about 50%, 30%, 80%, and 70% of enhanced performance, respectively. Both our *HS* and *HF* achieve better performance and higher security compared with related works.

**INDEX TERMS** HIGHT, revised CHAM, lightweight cryptography, fault attack resistance, ARMv8, ASIMD, NEON, optimization, Internet of Things.



Received November 14, 2021, accepted December 6, 2021, date of publication December 8, 2021, date of current version December 17, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3133888*

**High-Speed Fault Attack Resistant Implementation of PIPO Block Cipher on ARM Cortex-A**

**JINGYO SONG, (Student Member, IEEE), YOUNGBEOM KIM, (Student Member, IEEE), AND SEOG CHUNG SEO, (Member, IEEE)**

Department of Financial Information Security, Kookmin University, Seoul 02707, South Korea

Corresponding author: Seog Chung Seo (scseo@kookmin.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) Grant by the Korean Government through the Ministry of Science and ICT (MSIT) under Grant 2019R1F1A1058494, in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant by the Korean Government through MSIT (6G autonomous security internalization-based technology research to ensure security quality at all times) under Grant A2021-0270, and in part by the Korea Evaluation Institute of Industrial Technology (KEIT) Grant by the Korean Government through the Ministry of Trade, Industry and Energy (MOTIE).

**ABSTRACT** In ICISC'20 conference, PIPO (Plug-In and Plug-Out) was proposed as an efficient block cipher for secure communication in IoT (Internet of Things) environment. Although PIPO equips easily high-order masking implementation because of small non-linear operations and short rounds, PIPO is still vulnerable to fault attack. For resisting the fault attack, block cipher implementation should be applied to the fault attack countermeasure. However, these techniques make the performance of computationally-intensive cryptographic algorithms slower in constrained devices. To improve this, we propose the first fast and secure software of PIPO block cipher co-designed with ARM/NEON processor for high-speed secure communication. For accelerating the performance, we present an optimal implementation of PIPO block cipher in ARM/NEON processor, respectively, and design the Interleaved way utilizing two cores. With the proposed optimal techniques, we provide the high-speed secure software. In addition, we present an interleaving random-shuffling technique, which optimizes random-shuffling by utilizing two cores. For ensuring the resistance of fault attacks, we validated the fault resistance with the computation and instruction fault model. We utilize the intra-instruction-redundancy and known ciphertext to detect them. Through the proposed contributions, the fast software for PIPO block cipher achieves the fastest performance than previous related studies. The secure software with the fault countermeasures is nearly 3 times faster than the reference implementation without any fault attack countermeasures. In addition, our secure software achieved performance improvement of 301% and 463% compared to the existing best work (HIGHT and revised CHAM). As a result, our fast and secure software for PIPO block cipher achieved the fastest performance by co-designing with two cores compared to previous work that is utilized only one core. Our software can be utilized for high-speed encrypted communication and CTR-DRBG in ARMv8-based IoT devices.

**INDEX TERMS** PIPO, fault attack resistance, ARMv8, ARM/NEON processor, optimization, co-designed, interleaving implementation, Internet of Things.

# Q & A