# 32-bit RISC-V상에서의 경량 블록암호 PIPO 최적 병렬 구현

엄시우*, 장경배**, 송경주*, 이민우*, 서화정*
*한성대학교 IT융합공학과
**한성대학교 정보컴퓨터공학과
shuraatum@gmail.com, starj1023@gmail.com, thdrudwn98@gmail.com,
minunejip@gmail.com, hwajeong84@gmail.com

# Optimized parallel implementation of Lightweight blockcipher PIPO on 32-bit RISC-V

Si-Woo Eum*, Kyung-Bae Jang**, Gyeong-Ju Song*, Min-Woo Lee*, Hwa-Jeong Seo*
*Dept. of IT Convergence Engineering, Han-Sung University
**Dept. of Information and Computer Engineering, Han-sung University

요 약

PIPO 경량 블록암호는 ICISC'20에서 발표된 암호이다. 본 논문에서는 PIPO의 단일 평문 최적화 구현과 4평문 병렬 구현을 제안한다. 단일 평문 최적화 구현은 Rlayer의 최적화와 키스케쥴을 포함하지 않은 구현을 진행하였다. 결과적으로 키스케쥴을 포함하는 기존 연구 대비 70%의 성능 향상을 확인하였다. 4평문의 경우 32-bit 레지스터를 최대한 활용하여, 레지스터 내부 정렬과 Rlayer의 최적화 구현을 진행하였다. 또한 Addroundkey 구현에서 메모리 최적화 구현과 속도 최적화 구현을 나누어 구현하였다. 메모리 사용을 줄인 메모리 최적화 구현은 단일 평문 구현 대비 80%의 성능 향상을 확인하였고, 암호화 속도를 빠르게 구현한 속도 최적화 구현은 단일 평문 구현 대비 157%의 성능 향상을 확인하였다.

## 1. 서론

현재 암호화 알고리즘의 대부분은 데스크탑과 서버 환경을 위해 설계되었기 때문에 이러한 알고리즘 중 많은 부분이 제한된 환경으로 동작하는 장치에서 사용하기에는 알맞지 않다. NIST(National Institute of Standards and Technology)에서도 2015년에 경량 암호 공모전을 개최하였으며, 이를 통해 제한된 환경에서 사용하기 적합한 여러 경량 암호 알고리즘이 개발되고 있다.

PIPO는 ICISC'20에서 발표된 경량 블록암호이다. 11개의 비선형 연산과 23개의 선형 비트연산을 사용하는 효율적인 Bitslice로 구현되어 있어, 효율적인 마스킹 구현이 가능한 경량 블록암호이다[1].

본 논문에서는 32-bit RISC-V상에서 PIPO 경량 블록암호의 단일 평문 최적화 구현과 4평문 병렬 구현을 진행한다.

본 논문의 구성은 다음과 같다. 2장에서는 PIPO 경량 블록암호와 32-bit RISC-V에 대해서 설명한다. 3장에서는 단일 평문 최적화 구현과 4평문 병렬 구현의 기법을 설명한다. 4장에서는 제안 기법을 적용한 구현의 성능을 평가한다. 마지막으로 5장에서는 본 논문의 결론을 내린다.

## 2. 관련 연구

### 2.1 PIPO 블록 암호

PIPO의 암호화 알고리즘은 SPN(Substitution Permutation Network) 구조를 사용한다. 블록 길이는 64-bit, 키 길이는 128-bit, 256-bit를 지원한다. 각 키 길이에 따라 13, 17라운드를 진행하며 암호화가 진행되며 전체적인 알고리즘은 [그림 1]과 같다.

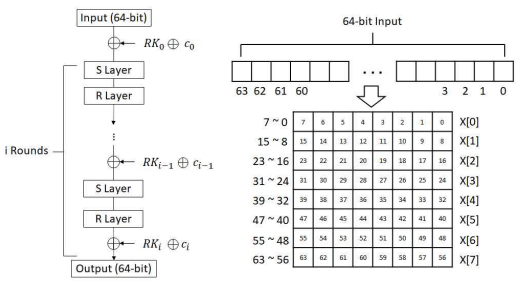


Fig 1. Structure of PIPO

PIPO는 [그림 1]의 오른쪽과 같은 구조로 암호화가 이루어지며, Slayer와 Rlayer의 동작은 [그림 2]와 같다.

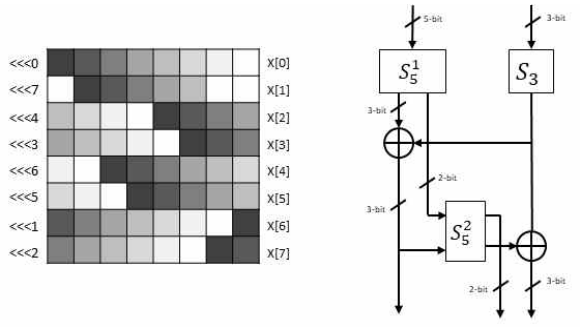


Fig 2. Rlayer(left) and Slayer(right) of Round Functions

## 2.2 RISC-V Processor

UC 버클리 대학에서 2010년부터 개발 중인 RISC(Reduced Instructuon Set Computer) 기반의 컴퓨터 아키텍쳐이다[2]. 무료로 공개된 하드웨어 구조와 명령어 집합으로 접근성과 확장성이 좋아 많은 기업과 연구 기관에서 주목하고 있다.

RISC-V는 RV32I, RV64I의 모델이 있으며, 각각 32-bit, 64-bit 레지스터를 사용한다. 본 논문에서는 32-bit를 지원하는 RV32I 사용한다.

RISC-V의 32-bit 구조인 RC32I는 32-bit 레지스터 32개를 제공한다. 전체적인 레지스터 용도는 [표 1]과 같다[3].

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

Table 1. Purpose of RISC-V Register

## 3. 제안 기법

본 장에서는 32-bit RISC-V 상에서의 PIPO의 단일 평문 최적화 구현과 4평문 병렬 구현에 대하여 설명한다. 단일 평문 최적화 구현에서는 64-bit 하나의 블록만 암호화 하는 구현을 제안한다. 4평문 병렬 구현에서는 4개의 64-bit 블록 병렬 구현을 제안한다.

### 3.1 단일 평문 최적화 구현

PIPO의 암호화 과정은 8-bit 단위로 연산이 진행되기 때문에 한 개의 레지스터에 한 개의 8-bit 값만 저장하여, 총 8개의 레지스터에 평문이 저장되게 된다. 이때 연산과정에서 레지스터가 8-bit 이상의 값을 가지고 있을 수 있지만 8-bit 이상의 값은 무시하고 연산을 진행한다. 이때 상위 24-bit의 값이 존재해도 라운드 함수중 Slayer와 Addroundkey에는 영향이 없다. 하지만 Rlayer에 경우 상위 24-bit가 연산에 영향을 주게 된다.

Rlayer에서는 로테이션 연산을 진행하고, RISC-V 상에서는 로테이션 연산을 Shift 명령어를 통해 구현한다. 레지스터에서 8-bit 이상의 값을 가지고 있을 때 logical shift right를 하게 되면, 무시하고 있던 상위 bit가 하위 8-bit로 이동하게 되고, 이로 인해 다른 결과 값이 나오게 된다. 따라서 Rlayer를 진행하기 전에 상위 24-bit를 0으로 비워주는 작업이 추가로 진행된다.

```
Input : X[1]
Output : X[1]<<<7
```

```
//X[1] = ((X[1] << 7)) ^ ((X[1] >> 1));
1. andi    a3, a3, 0xff
2. slli    t6, a3, 7
3. srli    a3, a3, 1
4. xor     a3, a3, t6
```

Table 2. A part of Rlayer; (a3:X[1], t6:temp)

[표 2]는 Rlayer의 일부 코드이다. 1줄에서 0xff의 값을 AND 연산하여 상위 24-bit의 값을 0으로 비워준 후 SLLI, SRLI, XOR 명령어를 사용하여 로테이션을 구현한다.

### 3.2 4평문 병렬 구현

PIPO는 8-bit 단위로 연산이 진행된다. 이때 32-bit 레지스터에서 8-bit의 값만 사용하는 것은 비효율적이다. 32-bit 레지스터는 4개의 8-bit 값을 저장할 수 있다. 따라서 4개의 블록을 입력받는 병렬 구현을 제안한다. 병렬 구현 시 Slayer의 경우 단일 평문 구현의 Slayer 코드가 변경 없이 바로 적용 가능하다. 하지만 Rlayer와 Addroundkey의 경우 병렬 구현에 맞춰 수정이 필요하다.

### 3.2.1 레지스터 내부 정렬

병렬 구현을 위해 입력 받은 4개의 블록 중 같은 연산을 진행하는 state를 하나의 레지스터로 합쳐주는 내부 정렬이 필요하다. 4개의 블록이 정렬된 상태는 [그림 3]과 같다.

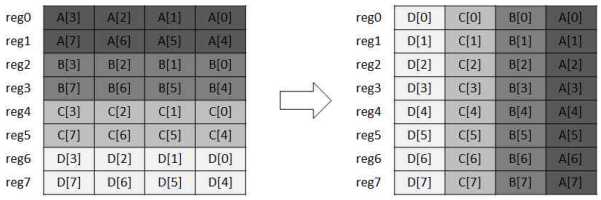


Fig 3. Internal alignment of registers

레지스터 내부 정렬을 위한 방법으로는 스택을 사용하는 방법과 LBU 명령어를 사용하는 방법이 있다. 스택을 사용할 경우 103Cycle, LBU 명령어를 사용할 경우 153Cycle이 발생한다. 따라서 본 논문에서는 스택을 사용한 레지스터 내부 정렬을 구현하였다.

### 3.2.2 로테이션 구현

[그림 4]은 Rlayer 연산 중 X[1]에서의 로테이션 과정을 나타낸다. 로테이션의 구현은 Left Shift의 값과 Right Shift의 XOR 연산을 통해 구현된다. 하지만 하나의 레지스터에 4개의 평문이 합쳐져 있기 때문에 Shift 연산을 하면 다른 평문의 값과 섞이게 된다. 따라서 다른 평문의 값을 0으로 바꿔주기 위해 AND 연산을 활용하여 섞인 부분의 값을 0으로 바꿔준다. [그림 4]에서는 0x80808080, 0x7F7F7F7F를 Left Shift 한 값과 Right Shift 한 값에 AND 연산을 해주어 다른 평문이 섞인 부분을 0으로 바꾸어 준다. 이때 AND 연산을 해주는 값은 로테이션 범위에 따라 다르다. 이를 구현한 코드는 [표 3]과 같다.

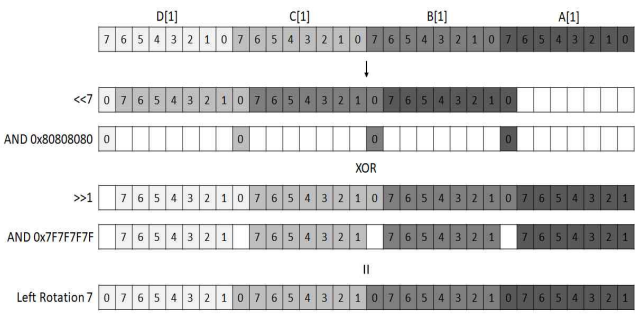


Fig 4. The process of Left Rotation 7

```
Input : A[1], B[1], C[1], D[1]
Output : (A[1], B[1], C[1], D[1])<<<7
```

```
//X[1] = ((X[1] << 7)) ^ ((X[1] >> 1));
1. li    t3, 0x80808080
2. li    t4, 0x7F7F7F7F
3. slli    t2, a3, 7
4. and     t2, t2, t3
5. srli    a3, a3, 1
6. and     a3, a3, t4
7. xor     a3, a3, t2
```

Table 3. A part of Rlayer; (a3:X[1], t2,t3,t4:temp)

### 3.2.3 Addroundkey 구현

Addroundkey 구현은 메모리 최적화 구현과 속도 최적화 구현 두가지 구현 방법을 제안한다.

메모리 최적화 구현은 단일 평문 구현과 동일한 라운드키를 사용한다. 동일한 라운드키를 사용하기 때문에 한번의 Addroundkey 연산을 진행할 때 64-bit 라운드키를 불러와 바이트 단위로 나누어 각 State와 연산을 진행하게 되는데, 바이트로 나뉜 라운드키는 8-bit의 크기이기 때문에 1개의 블록만 연산이 가능하다. 따라서 8-bit로 나뉘어진 라운드키를 32-bit로 늘려주는 추가적인 작업이 필요하다.

속도 최적화 구현은 라운드키를 늘려주는 작업을 키스케쥴에서 진행하게 된다. 이로 인해 라운드키의 저장 공간이 증가하게 된다. 하지만 암호화 과정에서 256-bit의 라운드키를 32-bit 단위로 불러와 각 State에 늘리는 작업 없이 바로 연산 가능하기 때문에 빠른 암호화가 가능하다.

## 4. 성능 평가

본 논문에서는 32-bit RISC-V 프로세서 환경에서 PIPO의 단일 평문 최적화 구현과 4평문 병렬 구현의 성능을 제시한다. SiFive사의 HiFive RevB 보드를 사용하여 측정을 진행하였으며, SiFive사에서 제공하는 FreedomStudio 프레임워크를 사용한다.

기존 연구인 Kwak et al.[4]은 같은 플랫폼인 32-bit RISC-V 프로세서 상에서 키스케쥴을 암호화에 포함시켜 구현하였다. 라운드 키를 저장하지 않아 제한된 저사양 디바이스에서 메모리 효율을 높힌 구현이다. 본 논문에서 제안하는 기법은 키스케쥴을 암호화에 포함시키지 않고 속도 효율을 높힌 구현이다. 임베디드 환경에서 키 갱신은 자주 이루어지지 않기 때문에 키스케쥴을 포함하지 않는 것이 속도 면에서 더 유리하다.

[표 3]은 제안하는 기법을 적용한 PIPO의 Cycle과 cpb(Cycle Per Byte)이다. 결과적으로 단일 평문 최적화 기법은 키스케쥴을 포함하지 않음으로 70%의 성능 향상을 확인하였다. 4평문 병렬 구현의 경우 단일 평문 구현과 비교하였을 때, 메모리 최적화 구현은 80%, 속도 최적화 구현은 157% 성능 향상을 확인하였다.

| PIPO-64/128 | Cycle | Cpb |
|---|---|---|
| Kwak et al. [4] | 2078 | 259.75 |
| **1-PT Our Work** | **1217** | **152.21** |
| **4-PT Our Work[1]** | **2715** | **84.85** |
| **4-PT Our Work[2]** | **1918** | **59.93** |

Table 4. Performance results in RISC-V ([1]:Memory optimized, [2]:Speed optimized)

## 5. 결론

본 논문에서는 PIPO 경량 블록암호의 단일 평문 최적화 구현과 4평문 병렬 구현을 제안한다. 단일 평문 최적화 구현은 PIPO의 라운드 함수중 Rlayer의 효율적인 구현과 키스케쥴을 포함하지 않은 암호화 구현을 통해 기존의 암호화 과정에 키스케쥴을 포함하는 연구 대비 70% 성능 향상을 확인하였다. 4평문 병렬 구현의 경우 32-bit 레지스터를 최대한 활용하여, 레지스터 내부 정렬과 Rlayer의 최적화를 진행하였다. 또한 Addroundkey의 메모리 최적화 구현과 속도 최적화 구현을 나누어 메모리에 효율적인 구현과 암호화 속도에 효율적인 구현을 제안하였다. 결과적으로 단일 평문 구현 대비 각각 80%, 157%의 성능 향상을 확인하였다. 향후 과제로 다양한 경량 블록암호의 병렬 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] H.G. Kim, Y.J. Jeon, G.Y. Kim, J.S. Kim, B.Y. Sim, D.G. Han, H.J. Seo, S.G. Kim, S.H. Hong, J.C. Sung, and D.J. Hong. “A New Method for Designing Lightweight S-Boxes with High Differential and Linear Branch Numbers,” and Its Application. International Conference on Information Security and Cryptology (ICISC 2020), Seoul, Korea, 2020, 62 pages.

[2] Waterman, Andrew, et al. "The risc-vinstruction set manual, volume i: Baseuser-level isa." EECS Department, UCBerkeley, Tech. Rep.UCB/EECS-2011-62 116, 2011

[3] SiFive, Inc. “The RISC-V Instruction Set Manual Volume I: User-Level ISA Document Version 2.2”. May 7, 2017. https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

[4] Y.J. Kwak, Y.B. Kim, S.C. Seo. “Benchmarking Korean Block Ciphers on 32-Bit RISC-V Processor”, Journal of the Korea Institute of Information Security & Cryptology, vol. 31, Issue. 3, pp. 331-340, 2021.