Journal of The Korea Institute of Information Security & Cryptology
VOL.31, NO.3, Jun. 2021
ISSN 1598-3986(Print)
ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2021.31.3.331

32-bit RISC-V 프로세서에서 국산 블록 암호 성능 벤치마킹*

곽 유 진,[1†] 김 영 범,[2] 서 석 충[3‡]
[1]국민대학교 정보보안암호수학과 (학생), [2,3]국민대학교 금융정보보안학과 (대학원생, 교수)

Benchmarking Korean Block Ciphers on 32-Bit RISC-V Processor*

YuJin Kwak,[1†] YoungBeom Kim,[2] Seog Chung Seo[3‡]
[1]Department of Information, Security and Cryptography, Kookmin University (Undergraduate),
[2,3]Department of Financial Information Security, Kookmin University (Graduate student, Assistant Professor)

요 약

5G를 포함한 통신 산업이 발전함에 따라, 모바일 임베디드 시스템을 위한 특수목적의 초소형 컴퓨터인 SoC (System on Chip)의 개발이 증대되고 있다. 이에 따라, 산업체와 기업들의 기술 설계의 패러다임이 변화하고 있다. 기존의 공정은 기업들이 마이크로 아키텍처를 구매하였다면, 지금은 ISA (Instruction Set Architecture)를 사들여, 기업이 직접 아키텍처를 설계한다. RISC-V는 축소 명령어 집합 컴퓨터 기반의 개방형 명령어 집합이다. RISC-V는 모듈화를 통하여 확장이 가능한 ISA를 탑재했으며, 현재 전 세계적 기업들의 지원을 통하여 ISA의 확장 버전 등이 개발되고 있다. 본 논문에서는 RISC-V에서 국산 블록 암호 ARIA, LEA, PIPO에 대하여 성능 벤치마킹과 분석 결과를 제공한다. 또한, RISC-V의 기본 명령어 집합과 특징을 활용한 구현 방법을 제안하고 성능을 논의한다.

ABSTRACT

As the communication industry develops, the development of SoC (System on Chip) is increasing. Accordingly, the paradigm of technology design of industries and companies is changing. In the existing process, companies purchased micro-architecture, but now they purchase ISA (Instruction Set Architecture), and companies design the architecture themselves. RISC-V is an open instruction set based on a reduced instruction set computer. RISC-V is equipped with ISA, which can be expanded through modularization, and an expanded version of ISA is currently being developed through the support of global companies. In this paper, we present benchmarking frameworks ARIA, LEA, and PIPO of Korean block ciphers in RISC-V. We propose implementation methods and discuss performance by utilizing the basic instruction set and features of RISC-V.

Keywords: Benchmarking, ARIA, PIPO, LEA, RISC-V

Received(04. 19. 2021), Modified(05. 28. 2021), Accepted(06. 01. 2021)
* 본 논문은 2021년도 정부의 재원으로 한국연구재단의 지원(No.2019R1F1A1068494)을 받아 수행된 연구임
† 주저자, youjk@kookmin.ac.kr
‡ 교신저자, scseo@kookmin.ac.kr(Corresponding author)

I. 서 론

5G 산업의 발달 및 각종 IoT 산업이 발전함에 따라, 임베디드 장치를 위한 특수목적의 초소형 컴퓨터인 SoC (System on Chip)의 개발이 증대하고 있다. 이에 따라, 산업체와 기업들에서 SoC 설계의 패러다임이 변화하고 있다. 기존의 SoC를 활용하는 공정은 기업들이 마이크로 아키텍처를 제조사로부터 구매하였다면, 지금은 기업이 제조사로부터 명령어 집합 구조 (ISA : Instruction Set Architecture)만을 구매하여, 기업이 직접 SoC를 설계한다. ISA는 마이크로프로세서가 인식해서 기능을 이해하고 실행할 수 있는 기계어 명령어를 말하며 실제, 최하위 레벨의 프로그래밍 인터페이스로, 프로세서가 실행할 수 있는 모든 명령어를 포함한다.

지금까지 SW (Software) 개발은 System SW의 개발과 Application SW의 개발의 큰 범주로 나뉘어 발전해왔다. Application SW 개발은 프로그래밍 언어로 개발되지만, 한계가 존재하기 때문에 특정 SW의 경우 효율성 및 재사용 측면에서 System SW를 기반으로 개발된다. 그러나, System SW의 개발은 복잡도가 매우 높고, 실제 ISA를 설계해야 해서 이미 존재하는 잘 정형화되어있는 ISA를 구매하여 SW에 적용한다. 예를 들어, SW 구현 최적화를 위해 Assembly Code를 사용하는 것처럼 Application SW가 System SW의 기능을 포함할 수 있도록 구현한다.

RISC-V는 축소 명령어 집합 컴퓨터 기반의 개방형 명령어 집합이다 [5]. 2010년부터 UC Berkeley에서 개발이 진행되었으며 2014년에 처음으로 공개되었다. RISC-V는 모듈화를 통하여 확장이 가능한 ISA를 탑재했으며, 현재 다양한 연구를 통하여 ISA의 확장 버전 등이 개발되고 있다. RISC-V의 가장 큰 장점으로는 ARM, x86 기반의 ISA와 다르게 누구나 사용할 수 있도록 공개된 ISA를 지원한다는 것이다. 따라서, 기존의 고착화된 ISA 시장에서 높은 기대를 받고 있으며, 오픈 코어 및 소프트웨어들로 인해 풍부한 개발환경을 지원한다. 또한, 전세계 기업들이 RISC-V 개발 참여 및 재정 지원을 통해 System SW 개발의 어려움을 해소해 줄 것으로 전망되고 있다.

RISC-V가 최근에 주목받기 시작했기 때문에 국산 블록 암호에 대한 RISC-V에 관한 구현 결과 연구는 아직 미미한 편이다. 따라서, 본 논문에서는 RISC-V에서 국산 블록 암호 ARIA, LEA, PIPO의 성능을 벤치마킹한다 [1][2][3]. 참조 코드를 단순히 RISC-V 환경에서 포팅한 것이 아닌, RISC-V의 가용한 명령어 집합과 암호 알고리즘별 최적의 레지스터 스케줄링을 제안하여 성능을 개선한다. 암호 알고리즘별 핵심 연산은 Assembly Code를 이용하여 최적화를 수행하였다. 그 결과, ARIA는 295CPB(Clock Per Bytes)를 달성했고, LEA는 48.44CPB, PIPO는 259.75CPB를 달성했다. 본 논문에서 제시한 RISC-V에서의 블록 암호 벤치마킹 결과는 기존 결과와 비교하여 크게 개선된 성능을 보여준다. 아울러, SiFive사의 HiFiveRevB 보드에서 성능을 측정하고 이에 대해 논의한다.

II. 국산 블록 암호

2.1 ARIA

ARIA(Academy, Research Institute, Agency)는 2003년 ICISC에 발표된 학계(Academy), 연구소(Research Institute), 정부 기관(Agency)의 전문가들이 전자정부를 지원하기 위해 만들어진 국산 블록 암호 알고리즘이다 [1]. ARIA는 블록 크기 128-bit의 암·복호화를 지원하는 블록 암호 알고리즘으로 경량환경 및 하드웨어 구현을 위해 최적화되어 설계되었다. ARIA는 키 크기 128/192/256-bit에 따라 12/14/16라운드를 가진다. ARIA의 구조는 ISPN(Inverse SPN)구조로 암호화와 복호화의 구조가 같다. 또한, ARIA는 GF(2)에서 eXclusive OR(xor) 연산만을 이용하여 이루어진 알고리즘으로 간단하다는 특징을 갖는다. ARIA의 한 라운드는 치환계층(SubstLayer), 확산계층(DiffLayer), 라운드키(AddRoundKey) 과정으로 구성된다. 마지막 라운드에 확산계층의 과정이 빠져있다는 것은 암호화와 복호화 과정을 같게 만들어준다.

치환계층 과정은 8-bit Sbox인 S_1, S_2를 사용하여 각 바이트를 치환한다. S_1, S_2함수는 $S_1(x) = Bx^{-1}+b$, $S_2 = Cx^{247}+c$의 꼴을 가진다. ARIA가 다른 블록 암호 알고리즘과 구분되는 특징은 짝수 라운드와 홀수 라운드가 다르게 구성된다는 것이다. 짝수 라운드와 홀수 라운드의 치환계층 구성은 Fig. 1.과 같다. 이렇게 두 유형의 치환계층은 서로 역관계에 있게 되어 암호화 과정이 Involution 구조를 이루는 데 도움을 준다.

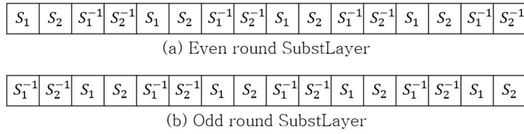

Fig. 1. Even Round/Odd Round SubstLayer Sbox Progress

확산계층 과정은 16x16 이진 행렬을 사용한 선형 변환 과정이다. 이 과정에서 사용되는 행렬 A는 $GF(2^8)^{16} \rightarrow GF(2^8)^{16}$으로 표현될 수 있다. 이때 사용되는 행렬 A는 Det(A)=1로 역행렬을 가지며 $A=A^{-1}$구조를 갖게 된다.

2.2 LEA

LEA(Lightweight Encryption Algorithm)는 2012년 국가보안기술연구소에서 개발된 국산 블록 암호 알고리즘이다 [2]. LEA는 128-bit의 블록을 암호화하며 키 크기 128/192/256-bit에 따라 라운드 수가 24/28/32로 결정된다. LEA는 소프트웨어와 하드웨어의 구현 시 효율적인 ARX(Addition, Rotation, Xor)기반의 GFN(Generalized Feistel Network)구조로 이루어져 있다. LEA의 암호화는 하나의 라운드가 변환 없이 라운드 수 만큼 반복되는 과정으로 구성된다.

LEA의 한 라운드는 32-bit의 워드 단위로 이루어진 4개의 state인 $X_i[j](0 \le i \le Nr,\ 0 \le j \le 3)$로 구성된다. 하나의 라운드에는 6개의 라운드키로 $RK_i[k](0 \le i \le Nr,\ 0 \le k \le 5)$를 사용한다. 6개의 라운드키는 마스터키로부터 스케줄링하여 라운드 수만큼의 192-bit의 라운드키를 만들어낸다. 이렇게 LEA는 4개의 state와 6개의 라운드키로 ARX구조의 특징에 따라 32-bit 단위의 xor, rotation, addition 연산을 사용하여 암호문을 만들어낸다. LEA는 Sbox와 같은 복잡한 연산 없이 단순한 ARX구조로 암호화가 진행되기 때문에 자원의 사용량을 줄여주면서 빠른 속도를 제공한다.

2.3 PIPO

PIPO는 2020년 ICISC에서 발표된 국산 블록 암호 알고리즘이다 [3]. PIPO는 64-bit 블록의 암호화를 제공하는 경량 블록 암호이다. 또한, 8-bit AVR 소프트웨어 구현에서 뛰어난 성능을 제공한다. PIPO는 성능의 오버헤드를 최소화하기 위해 비선형 과정의 연산 수를 줄이는 데에 목적을 두고 설계되었다. PIPO는 리소스가 제한된 다양한 환경에서 적용될 수 있다. PIPO의 한 라운드는 S-Layer, R-Layer, AddRoundKey로 구성된다.

S-Layer 과정은 8-bit Sbox를 사용하는 과정이다. S-Layer의 8-bit Sbox는 1개의 3-bit Sbox와 2개의 5-bit Sbox를 사용하여 8-bit Sbox를 구성하는 Unbalanced-Bridge 구조를 사용한다. Fig. 2.는 PIPO의 S-Layer에서 사용되는 Unbalanced-Bridge 구조를 보여준다. 이 구조는 3/5-bit Sbox를 사용하여 생성된 8-bit Sbox가 2보다 큰 선형 분기 수를 갖도록 설계되었다. 이러한 조건은 Sbox의 생성에 필요한 검색공간을 크게 줄여준다. 또한, 이는 8-bit Sbox가 11개의 비선형 비트 연산만 포함하는 효율적인 Bit-slicing 구현을 제공한다. R-Layer 과정은 효율적인 하드웨어 및 소프트웨어 구현을 보장하기 위해 바이트 단위의 비트 회전만을 사용하여 설계되었다. R-Layer 과정은 Fig. 3.과 같이 MSB에 위치한 바이트부터 차례로 0, 7, 4, 3, 6, 5, 1, 2-bit를 왼쪽으로 순환시켜준다. 이는 모든 입력 비트가 전체 출력 비트에 영향을 미칠 수 있는 완전한 확산을 만들어주기 위한 라운드 수를 최소화하는 데 도움을 준다. PIPO는 비선형 연산의

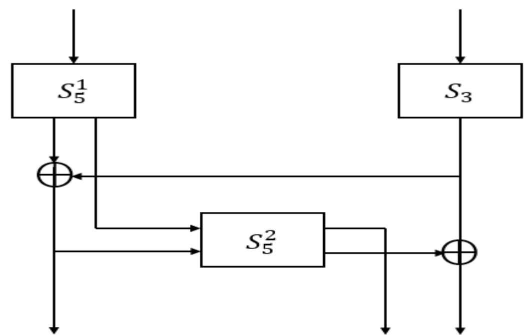

Fig. 2. Unbalanced-Bridge structure

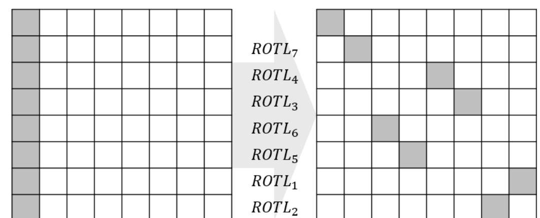

Fig. 3. PIPO's R-Layer process

수를 최소화하는 데 중점을 두었다. 이것은 효율적인 고차 마스킹 구현을 위한 가장 중요한 요소가 된다.

III. RISC_V 프로세서 개요 및 개발환경

3.1 RISC-V 프로세서 특징

RISC-V는 2010년부터 UC Berkeley 대학에서 개발 중인 RISC(Reduced Instruction Set Computer) 기반의 새로운 컴퓨터 아키텍처이다 〔4〕. RISC-V는 공개적 표준 공동 작업을 기반으로 무료 개방형 명령어 집합 아키텍처를 지원하고 있다.

또한, RISC-V의 하드웨어 구조와 명령어 집합(Instruction Set Architecture, ISA)이 무료로 공개되어 ISA는 모듈화 방식으로 자유롭게 확장이 가능할 뿐만 아니라, 하드웨어 및 소프트웨어 개발의 자유를 보장하기에 접근성과 확장성 모두 높이 평가되고 있다 〔5〕. RISC-V는 이러한 장점들 때문에 AMD, Google, Microsoft, IBM, NVDIA 등의 많은 기업이 개발에 힘쓰고 있어 앞으로 IoT 환경에서 RISC-V의 관심이 더욱 높아질 것으로 보인다.

RISC-V는 32개의 범용 레지스터로 구성된다는 것을 알 수 있다 〔4〕. 각 레지스터의 사용 용도는 Table 1.에 명시된다. 레지스터의 크기는 RV32I와 RV64I의 모델에 따라 동일한 명령어 셋으로 32-bit와 64-bit로 지원된다 〔5〕. RISC-V의 정수형 ISA는 다른 RISC 프로세서와 같지만, 분기에 대한 지연 슬롯이 없고, 가변 길이의 명령어 암호화를 지원한다는 점이 다르다 〔6〕. RISC-V의 명령어 포맷은 R, I, S, B, U, J로 구성된다. Table 2.는 본 논문에서 사용한 명령어 일부에 대한 용도와 CPI(Clock Per Instruction)를 제공한다. 명령어는 고정된 32-bit의 넓이를 가지며 메모리의 4바이트 경계에 정렬되어 명령어를 수행한다 〔6〕. 또한, RISC-V는 다른 임베디드 장치들과는 다르게 연산 결과에 대한 상태 코드가 존재하지 않는다. 따라서 이를 위한 조건부 점프 명령어가 존재한다는 특징이 있다 〔4〕.

Table 1. RISC-V register purpose〔4〕

Register	Usage
zero	Zero
ra	Return address
sp	Stack pointer
gp	Global pointer
tp	Thread pointer
t0~t6	Temporaries
s0	Frame pointer
s1~s11	Saved register
a0~a7	Function arguments
pc	Program counter

3.2 RISC-V 개발 환경

RISC-V은 리눅스 등과 같은 다양한 개발 환경을 가진다. 본 논문에서는 RISC-V에서의 최적화된 개발 환경인 FreedomStudio를 사용한다. FreedomStudio은 SiFive 기반 프로세서를 대상으로 하

Table 2. Assembly Instruction Set of RISC-V〔5〕

Asm	Operands	Description	Operation	CPI
lb	rd, imm12(rs1)	Load byte	R〔rd〕 = SignExt(Mem$_1$〔R〔rs1〕 +SignExt(imm12)〕)	2
sb	rs2, imm12(rs1)	Store byte	Mem$_1$〔R〔ra1〕 + SignExt(imm12)〕 = R〔rs2〕(7:0)	2
and	rd, rs1, rs2	And	R〔rd〕=R〔rs1〕&R〔rs2〕	1
or	rd, rs1, rs2	OR	R〔rd〕=R〔rs1〕|R〔rs2〕	1
xor	rd, rs1, rs2	Exclusive OR	R〔rd〕=R〔rs1〕^R〔rs2〕	1
sll	rd, rs1, rs2	Shift left logical	R〔rd〕=R〔rs1〕〈〈R〔rs2〕	1
add	rd, rs1, rs2	Add	R〔rd〕=R〔rs1〕+R〔rs2〕	1
srl	rd, rs1, rs2	Shift right logical	R〔rd〕=R〔rs1〕〉〉R〔rs2〕	1
bne	rs1, rs2, imm12	Branch not equal	if(R〔rs1〕 != R〔rs2〕) pc = pc + SignExt(imm12 〈〈 1)	1/2

는 소프트웨어를 작성하고 디버그하는데 사용할 수 있는 통합 개발환경이다 [7]. FreedomStudio는 산업 표준 Eclipse를 기반으로 하며 사전 빌드된 RISC-V GCC Toolchain, OpenOCD, freedome-sdk와 함께 번들로 제공된다. FreedomStudio는 SiFive의 웹사이트에서 제공 받아서 사용할 수 있다.

FreedomStudio(ver. 2020)에서는 GCC Toolchain이 gcc 버전 10.1.0과 gcc 버전 8.3.0으로 두 개의 Toolchain이 번들로 제공된다. 또한, 디버깅 시 쓰이는 OpenOCD은 SRAM을 읽는 속도가 이전 버전보다 최대 20배 빨라졌다.

IV. RISC-V에서 블록 암호 구현 동향

국외에서 RISC-V 환경에서의 블록 암호 구현 연구는 활발하게 진행되고 있다. 2019년 Ko Stoffelen에 의해 RISC-V 환경에서 AES, ChaCha, Keccak 알고리즘에 관한 구현 연구가 진행되었다[8]. 또한, 공개키 암호화와 관련하여 캐리 플래그가 없는 임의의 정밀도 정수 연산이 제안되었다. AES의 구현을 위해 Ko Stoffelen는 AES 32-bit T-table을 이용하는 방법과 Bit-slicing 기법을 사용하는 구현 두 가지를 제안했다.

T-table을 이용한 AES 암호화 방식은 가장 빠른 구현 방법의 하나고 RISC-V의 레지스터의 크기도 32-bit이므로 T-table 기법을 기존 ARM 기반 프로세서와 마찬가지로 동일하게 적용할 수 있다. Ko Stoffelen의 T-table AES 구현은 타 플랫폼의 T-table을 이용한 AES 구현 방식과 같이 4KB의 조회 테이블을 사용하고, 키 확장 과정은 340 clock cycles를 달성한다. 최종적으로 RISC-V에서 T-table을 사용한 AES는 40.3 CPB를 달성했다.

캐시 타이밍 공격관점에서 T-table 기반의 구현은 안전하지 않기 때문에, Bit-slicing 기법을 사용한 AES도 구현되었다. RISC-V는 32-bit 레지스터를 갖기 때문에 여러 블록을 병렬로 처리할 수 있고 32개의 범용레지스터를 가지기 때문에 Bit-slicing 기법이 더 효율적일 수 있다. Bit-slicing AES 구현은 내부병렬 처리를 고려하여 구현되었다. SubBytes 연산의 내부 병렬처리는 일반적으로 AES 상태가 모든 바이트의 특정 비트를 포함하게 레지스터를 유지할 수 있도록 구현되었다. RISC-V의 명령어 집합을 사용하고 한 평문의 암호화를 위해 레지스터의 16-bit만 사용하여 AES 내부상태가 표현되었다. 즉, 32-bit 레지스터를 전부 사용하여 2개의 블록을 병렬로 처리하였다. 이를 이용하여 Ko Stoffelen는 RISC-V 환경에서 SubBytes 구현을 113개의 단일 사이클로 구현하였다. RISC-V에서 Bit-slicing 기반 AES 구현은 4096바이트를 암호화할 때 101.2 CPB를 달성했다.

국내에서 RISC-V 프로세서상에서 블록 암호 구현 연구는 아직 미미한 편이나 CISC-W'20에서 RISC-V 프로세서상에서 ARIA의 고속연산을 위한 확장 명령어 셋 구현에 관한 연구가 진행되었다[9]. ARIA의 치환계층 명령어 arias1, arias2가 제안되었으며 확산계층 명령어 ariad1, ariad2, ariad3, airad4가 제안되었다. 치환계층 명령어는 테이블을 참조하는 구현이 아닌 Galois field 연산 기반의 치환 함수를 지원하며 RISC-V의 특성을 이용하여 한 번에 32-bit의 치환계층 연산을 수행한다. ARIA의 확산계층 명령어는 32-bit 데이터에 대하여 각 홀/짝 라운드의 확산계층 연산을 4개로 나누어 구성하고 라운드별로 4개의 명령어만 거칠 수 있도록 설계하였다. 명령어의 구현은 SPIKE 시뮬레이터와 LLVM의 컴파일러를 사용하였으며 최종적으로 치환계층은 라운드 당 4 clock cycles를 달성하고, 확산계층은 16 clock cycles를 달성하였다.

V. RISC-V에서 국산 블록 암호 구현

본 절에서는 RISC-V 환경에서 RC32I를 사용하는 국산 블록 암호 구현 방법론에 대해 논의한다. 구현 시 추가 확장 명령어 세트는 사용하지 않는다. 본 절에서 제시한 구현은 메모리 접근을 최소화하기 위한 최적의 레지스터 스케줄링 방법 사용과 가용한 명령어 집합에서 최적의 명령어를 조합하여 암호 연산의 성능을 최적화한다.

5.1 ARIA 구현 방법

ARIA는 128-bit의 평문이 암호화되며 DiffLayer 과정에서 이전 16바이트에 대한 정보가 다시 필요하기 때문에 레지스터의 수가 다른 프로세서보다 제한적인 RISC-V에서 최적화된 평문의 레지스터 스케줄링이 ARIA의 벤치마킹에서 중요한 부분이 된다. 먼저 ARIA의 RISC-V 상에서 레지스터의 사용은 Table 3.에 명시되어 있다.

Table 3. ARIA register scheduling on RISC-V

Register	Our use
a0	Function argument address (Plain text)
a1	Function argument address (Roundkey)
s2~s11, t0~t5	Plaintext (P[0]~P[15])
t6, a5~a7	Temp

a0와 a1은 각각 함수 인자로 할당된 평문과 라운드키의 주소를 레지스터로 받아온다. 평문의 주소가 할당된 a0 레지스터와 lb 명령어를 사용하여 s1~s11, t0~t5의 총 16개의 레지스터에 평문을 각각 한 바이트씩 로드한다. 따라서, 16개의 레지스터에 1바이트가 할당되어 암호화 과정을 진행한다. 본 논문에서는 암호화가 진행될 때 한 라운드의 구성을 라운드 키 과정, 치환계층 과정, 확산계층 과정의 순서로 하였으며 치환계층 과정에서는 룩 업 테이블을 이용하여 구현하였다.

16개의 레지스터에 스케줄링 된 평문의 바이트들은 라운드 키 과정과 치환계층 과정에서는 처음 스케줄링 된 맨 오른쪽 한 바이트에서 평문의 이동 없이 연산이 진행된다. 한 바이트에 대한 연산이 진행된 후, 확산계층의 연산이 시작되기 전에 Fig. 4.와 같이 16개의 레지스터 위에 있는 평문을 왼쪽으로 1바이트씩 shift 한다. 이는 확산계층 과정에서 새로운 평문에 지난 평문에 대한 정보가 쓰이는 데 이를 제한적인 레지스터를 가진 RISC-V에서 구현하기 위해 사용하였다. 이동된 평문이 확산계층에서 연산된 뒤 새로운 평문에 대한 정보는 원래 평문이 있던 자리인 맨 오른쪽 1바이트에 저장이 된다. 이렇게 진행된 한 라운드 과정이 라운드 수 만큼 반복되면 결국 16개의 레지스터의 맨 오른쪽 1바이트가 암호문이 된다.

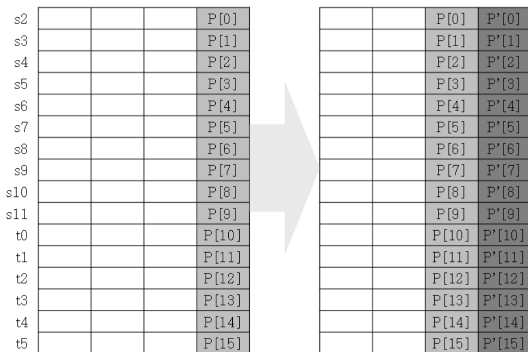

Fig. 4. Moving plaintext on register in ARIA's RISC-V implementation

5.2 LEA 구현 방법

LEA는 평문의 워드 단위와 RISC-V의 레지스터 크기가 32-bit로 같다는 것을 이용하여 RISC-V 상에서 최적화하여 벤치마킹하였다. 따라서 RISC-V의 32-bit 레지스터 한 개에 한 개의 워드 단위 평문이 들어가고 그 안에서 연산이 진행되게 된다. RISC-V 위에서 레지스터의 사용은 Table 4.와 같다.

a0와 a1은 각각 함수 인자로 할당된 평문과 라운드키의 주소를 레지스터로 받아온다. 평문의 주소가 할당된 a0 레지스터를 사용하여 s2~s5의 총 4개의 레지스터에 평문을 로드한다. LEA는 32-bit 단위로 암호화 과정을 수행하기 때문에 실제 암호화 과정에서 4개의 레지스터만을 이용하여 암호화가 가능하다.

LEA의 레지스터 연산 과정은 간단하다. 6개의 라운드키를 라운드키 레지스터에 로드한다. 한 라운드는 로드된 6개의 라운드 키와 4개의 워드 단위 평문이 xor, add, rotation, or 연산만을 이용해서 구성된다. 하지만 RISC-V에서는 carry를 처리하는 rotation 명령어가 존재하지 않는다. 따라서 RISC-V에서 rotation을 위한 연산을 설계해야 한다. 본 논문에서는 RISC-V에서 LEA의 바이트 단위의 비트 rotation의 구현을 Fig. 5.와 같이 구현하였다. rotation 구현을 위해 3개의 레지스터가 추가로 필요로 하게 되며, sll, srl, or 연산자를 이용한다. rotation 연산은 다음과 같이 구현된다. Fig. 6.에서 (a)를 레지스터 s2라고 가정하자. 우리가 구현하고 싶은 $ROTR_9$는 s2 레지스터를 오른쪽으로 9-bi

Table 4. LEA register scheduling on RISC-V

Register	Our use
a0	Function argument address (Plaintext)
a1	Function argument address (Roundkey)
s2~s5	Plaintext (P[0]~P[16])
s6~s11	Roundkey
t3~t6	Temp

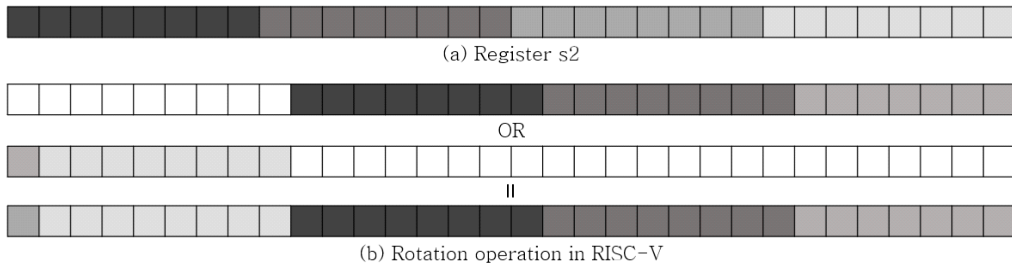

Fig. 5. Implementation of LEA rotation in RISC-V

t만큼 shift 시키고 s2 레지스터를 왼쪽으로 23-bit 만큼 shift 시킨 뒤 두 결괏값을 or 연산을 하게 되면 오른쪽으로 9-bit rotation 시킨 s2를 만들어 낼 수 있게 된다.

5.3 PIPO 구현 방법

PIPO는 8-bit AVR 소프트웨어 구현에서 뛰어난 성능을 제공하기 때문에 32-bit 프로세서인 RISC-V에서 최적화하여 암호화를 진행하기 위해 평문을 위한 8개의 레지스터를 사용하여 PIPO를 벤치마킹하였다. RISC-V 위에서 레지스터의 사용은 Table 5.와 같다.

a0와 a1은 각각 함수 인자로 할당된 평문과 마스터키의 주소를 레지스터로 받아온다. 평문의 주소가 할당된 a0 레지스터를 사용하여 s2~s9에 평문을 한 바이트씩 로드하여 암호화 과정을 진행한다. 따라서, 8개의 레지스터를 이용하여 암호화를 진행한다. 8개의 레지스터 위에 평문 8바이트를 한 바이트씩 잘라서 암호화를 진행할 때, 평문의 바이트들이 서로 영향을 주지 않도록 레지스터를 사용하였다. 또한, PIPO는 다른 알고리즘과는 다르게 라운드 키를 생성하는 키 스케줄링 과정을 함께 구현하였기 때문에

Table 5. PIPO register scheduling on RISC-V

Register	Our use
a0	Function argument address (Plaintext)
a1	Function argument address (Masterkey)
s2~s9	Plaintext (P[0]~P[7])
s10	Rcon (Number of Round)
a4-a7	MasterKey
t3~t5	Temp

a1 레지스터에 마스터키의 주소가 할당된다. PIPO의 키 스케줄링 과정은 마스터키에 단순히 라운드 수를 의미하는 Rcon을 xor해주는 과정이기 때문에 레지스터 s10을 라운드 수를 위한 레지스터로 할당한 뒤 키 스케줄링 과정을 암호화 과정과 함께 구현함으로써 제한된 환경에서 PIPO 알고리즘의 특성을 활용하여 적합하게 구현하였다.

PIPO의 S-Layer에서 사용되는 Sbox는 Bit-slicing 방법을 사용하여 임베디드 장비에서보다 효율적으로 구현하였다. RISC-V에서 PIPO의 R-Layer의 구현 역시 rotation 연산을 사용하기 때문에 LEA의 32-bit 단위의 rotation 구현을 8-bit 단위로 줄여서 구현했다. 하지만 PIPO의 경우 우리가 평문을 8개의 레지스터에 각각 한 바이트씩 스케줄링해주었기 때문에 Bit-masking의 개념이 추가로 필요하게 된다. 예를 들어 $(10011111)_2$가 s2 레지스터에 할당되어 있을 때, $ROTL_3$ 해주기 위해서 먼저 s2와 0xE0의 xor 연산을 진행한다. 그 결과 $(10000000)_2$와 같이 왼쪽의 3-bit가 계산된다. 이와 마찬가지로 s2와 0x1F를 xor 하여 오른쪽 5-bit를 계산한다. 마지막으로 살아남은 왼쪽 3-bit를 5-bit만큼 Right shift하고, 오른쪽 5-bit를 3-bit 만큼 Left shift 해준 뒤, 두 결괏값을 or 해주면 Bit-masking을 사용하여 $(10011111)_2$를 $ROTL_3$한 값이 도출된다. 다음과 같은 방법을 8개의 레지스터에 적용하여 RISC-V 환경에서 PIPO의 R-Layer를 설계하였다.

VI. 성능 평가

6.1 Reference code와의 비교

본 논문에서는 32-bit 프로세서 RISC-V 환경에서 국산 블록 암호의 코드 벤치마킹의 성능을 제시한

Table 6. Table of benchmarking performance of ARIA, LEA, PIPO using 128-bit key

Language	C [Reference code]		Code size	Asm [Our Works]		Code size
Algorithm	Clock cycles	Cycles/byte		Clock cycles	Cycles/byte	
ARIA-128[1]	28452	1778.25	23998	4720	295	8192
LEA-128[2]	9070	566.88	16574	775	48.44	2432
PIPO-64/128[3]	10733	1341.63	23906	2078	259.75	6124

다. 성능 측정을 위해 SiFive 사의 HiFiveRevB 보드를 사용하였으며, 측정 환경은 SiFive 사의 FreedomStudio(ver. 2020)를 사용하였다. Table 6.은 국산 블록 암호인 ARIA, LEA, PIPO의 RISC-V에서의 벤치마킹 성능 결과를 제시한다. 성능 비교를 위해 사용한 Reference code는 기존 공개 코드를 RISC-V 상에서 C언어로 포팅한 것을 의미한다. 포팅의 무결성 검증과 벤치마킹 코드의 구현 정확성을 위해, ARIA 및 LEA의 경우 KISA의 암호 알고리즘 검증기준 V3.0에 제시된 테스트 벡터를 사용하여 검증하였다. PIPO의 경우는 PIPO 제안 논문에서 소개된 테스트 벡터를 사용하여 검증하였다.

본 논문에서는 국산 블록 암호를 벤치마킹하는 데 있어서 한 블록 크기의 평문에 대해서 벤치마킹을 진행했다. 즉, ARIA-128과 LEA-128은 128-bit 단위의 평문 블록에 대해서 벤치마킹하였고, PIPO-64/128은 64-bit에 대해서 벤치마킹하였다. 그 결과, 본 논문에서 제안하는 ARIA의 최적화 코드는 295 CPB를 달성했고, LEA는 48.44CPB를 달성했다 [1][2]. PIPO의 경우 259.75CPB를 달성했다 [3]. 아울러 Reference code보다 Code size 측면에서도 성능이 향상했다는 것을 볼 수 있다.

결과적으로, 본 논문에서는 32-bit 프로세서 RISC-V 환경에서의 어셈블리 언어를 통한 국산 블록 암호(ARIA, LEA, PIPO)의 단순 벤치마킹 구현으로도 큰 성능향상 폭이 존재한다는 것을 확인했다.

6.2 기존 구현들과의 비교

지금까지 RISC-V에서 벤치마킹된 국외 블록 암호들과 비교한다. Table 7.은 [8]에 제시되어있는 Table AES-128, Bit-sliced AES-128, ChaCha20, Keccak과 본 논문에서 제시하는 국산 블록 암호 알고리즘에 대한 비교를 CPB를 기준으로 보여준다. 본 논문에서 구현한 3종의 알고리즘 중 LEA가 RISC-V에서 높은 성능을 달성했다. ARIA의 경우 Reference 코드를 바탕으로 레지스터 스케줄링 및 구현을 시도했지만, 실제 Table-AES와 비교하여 낮은 성능을 보인다. 이 부분에 대해서는 [9]와 같이 특정 ISA를 설계하거나 최적화 기법을 도입해야 할 것으로 예상된다. PIPO의 경우 RISC-V에서 259.75 CPB를 달성했다. PIPO는 경량암호이지만 내부상태가 64-bit이기 때문에, 실제로 32-bit 레지스터 크기를 갖는 RISC-V에서는 높은 성능을 끌어내기 힘들 것으로 보인다. 따라서, RISC-V 환경에서 ARIA 및 PIPO 암호에 대한 병렬화 기법 등의 최적화 기법이 필요할 것으로 예상된다.

Table 7. Table of comparision of our work and existing implementation

Algorithm	Author	Cycles/byte
ARIA-128	our work	295
LEA-128	our work	48.44
PIPO-64/128	our work	259.75
Table AES-128[8]	Ko Stoffelen	57
Bit-sliced AES-128-CTR[8]	Ko Stoffelen	124.4
ChaCha20 encrypt[8]	Ko Stoffelen	27.9
Keccak-f[1600] permute[8]	Ko Stoffelen	68.9

VII. 결 론

본 논문에서는 국산 블록 암호(ARIA, LEA, PIPO)를 32-bit 프로세서인 RISC-V 상에서 벤치마킹한 방안과 그 성능을 제시하고 있다. AIRA의 경우 본 논문에서 제안한 RISC-V에서 Asm으로 벤치마킹한 코드는 295CPB를 달성했다. LEA의 경우는 48.44CPB, PIPO의 경우에는 259.75CPB를 달성했다.

5G 산업이 발달함에 따라 임베디드 장치들이 증가하고 임베디드 장치의 보안에 대한 중요성이 증가하고 있다. 이에 따라 앞으로 32-bit 임베디드 장치인 ARM, RISC-V가 기존의 8-bit AVR MCUs, 16-bit MSP430에 비해 수요가 증가할 것으로 보인다. 본 논문에서는 국산 블록 암호 ARIA, LEA, PIPO에 대해 RISC-V 환경에서 벤치마킹하고 그 결과를 다른 국외 블록 암호들과 비교하였다. 그 결과 어떤 알고리즘의 경우 경량암호임에도 불구하고 RISC-V 환경에서 높은 성능을 보이지 못했다. 따라서 향후 다양한 32-bit 임베디드 장치에서 블록 암호 알고리즘의 단순 벤치마킹을 통한 성능향상에 대한 연구뿐만 아니라 32-bit 임베디드 장치에서 블록 암호 알고리즘의 최적화 연구를 진행할 예정이다.

References

〔1〕 Kwon, Daesung, et al. "New block cipher: ARIA." International Conference on Information Security and Cryptology. Springer, Berlin, Heidelberg, pp. 432-445, Nov. 2003.

〔2〕 Hong, Deukjo, et al. "LEA: A 128-bit block cipher for fast encryption on common processors." International Workshop on Information Security Applications. Springer, Cham, pp. 3-27 Aug. 2013.

〔3〕 Kim, Hangi, et al. "A New Method for Designing Lightweight S-Boxes with High Differential and Linear Branch Numbers, and Its Application*." Proceedings of the 23rd Annual International Conference on Information Security and Cryptology (ICISC 2020), Seoul, Korea. pp. 105-132, Dec. 2020.

〔4〕 Waterman, Andrew, et al. "The risc-v instruction set manual, volume i: Base user-level isa." EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62 116, 2011.

〔5〕 Waterman, Andrew Shell. Design of the RISC-V instruction set architecture. Diss. UC Berkeley, 2016.

〔6〕 Lee, Jongbok. "Simulation and Synthesis of RISC-V Processor." The Journal of The Institute of Internet, Broadcasting and Communication 19.1, pp.239-245, Feb. 2019.

〔7〕 SiFive, "Freedom Studio User Manual“, pp.176, 2020.

〔8〕 STOFFELEN, Ko. “Efficient cryptography on the RISC-V architecture." In: Internatio nal Conference on Cryptology and Informa tion Security in Latin America. Springer, Cham, pp.323-340, Sep. 2019.

〔9〕 Jin-Jae Le, Min-Jae Kim, Jong-Uk Par, Ho-won Kim. “High-Speed ARIA cryptographic extension for a low performance RISC-V processor." , CISC-W20, onlineProceeding, pp.542-545, Nov. 2020

〈저자소개〉

곽 유 진(Yujin Kwak) 학생회원
2017년~현재: 국민대학교 정보보안암호수학과 학사과정
〈관심분야〉 암호최적화, 임베디드 보안

김 영 범 (YoungBeom Kim) 학생회원
2021년: 국민대학교 정보보안암호수학과 졸업
2021년~현재: 국민대학교 금융정보보안학과 석사과정
〈관심분야〉 암호모듈검증, 양자내성암호

서 석 충 (Seog Chung Seo) 정회원
2011년 8월: 고려대학교 정보보호대학원 박사
2013년 11월: 삼성전자 종합기술원 전문연구원
2014년 4월: 삼성전자 DMC 연구소 책임연구원
2019년 2월: 국가보안기술연구소 선임연구원
2019년 3월~현재: 국민대학교 금융정보보안학과 조교수
〈관심분야〉 암호최적화, 공개키 암호, 암호모듈검증,네트워크보안