Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.31, NO.3, Jun. 2021 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2021.31.3.473

차세대 공개키 암호 고속 연산을 위한 RISC-V 프로세서 상에서의 확장 가능한 최적 곱셈 구현 기법*

서 화 정,[1†‡] 권 혁 동,[2] 장 경 배,[2] 김 현 준[2]
[1,2]한성대학교 (교수, 대학원생)

Optimized Implementation of Scalable Multi-Precision Multiplication Method on RISC-V Processor for High-Speed Computation of Post-Quantum Cryptography*

Hwa-jeong Seo,[1†‡] Hyeok-dong Kwon,[2] Kyoung-bae Jang,[2] Hyunjun Kim[2]
[1,2]Hansung University (Professor, Graduate student)

요 약

차세대 공개키 암호 고속 연산을 위해서는 목표로 하는 컴퓨터 프로세서의 구조를 활용하여 암호화 기본 연산을 최적화 구현하는 것이 중요하다. 본 논문에서는 RISC-V 프로세서 상에서 차세대 공개키 암호 고속 연산을 위해 핵심 곱셈기 연산을 최적화 구현하는 기법을 제안한다. 특히 RISC-V 프로세서의 기본 연산자를 열 기반 곱셈기 연산 알고리즘에 맞추어 최적 구현해봄으로서 이전 연구와 비교 시 256-비트 곱셈의 경우 약 19% 그리고 512-비트 곱셈의 경우 약 8%의 성능 향상을 RISC-V 프로세서 상에서 달성하였다. 마지막으로 RISC-V 프로세서에서 추가적으로 제공되면 곱셈 연산 성능 향상에 도움이 될 수 있는 확장 명령어 셋에 대해서도 확인해 보도록 한다.

ABSTRACT

To achieve the high-speed implementation of post-quantum cryptography, primitive operations should be tailored to the architecture of the target processor. In this paper, we present the optimized implementation of multiplier operation on RISC-V processor for post-quantum cryptography. Particularly, the column-wise multiplication algorithm is optimized with the primitive instruction of RISC-V processor, which improved the performance of 256-bit and 512-bit multiplication by 19% and 8% than previous works, respectively. Lastly, we suggest the instruction extension for the high-speed multiplication on the RISC-V processor.

Keywords: Post-quantum Cryptography, RISC-V, Software Implementation, Multiplication

I. 서 론

RISC-V 프로세서는 오픈 소스로 제공되는 명령어 셋을 기반으로한 프로세서로서 2010년부터 개발되어 활발히 활용되고 있다. 특히 산학연 모두에서 무료로 사용이 가능한 장점으로 인해 현재 구글, 퀄컴, 그리고 삼성을 포함한 200 여개의 단체가 RISC-V 컨소시엄에 포함되어 활동 중에 있다[1]. 산업체에서 RISC-V 프로세서를 채택하는 이유로는 임베디드 시장에서 현재 가장 높은 영향력을 보이고 있는 ARM 마이크로 컨트롤러에 대한 라이센스 비용을 절감하기 위한 목적도 상존하고 있다. 또한 내

Received(04. 21. 2021), Modified(05. 24. 2021), Accepted(05. 24. 2021)
* 이 성과는 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2020R1F1A1048478).
† 주저자, hwajeong84@gmail.com
‡ 교신저자, hwajeong84@gmail.com(Corresponding author)

1) https://riscv.org/

부 구조가 공개되지 않은 ARM 프로세서에 비해 내부 구조가 공개된 RISC-V 프로세서 상에 대한 접근이 학교에서의 연구에 용이한 면도 있기 때문이다.

RISC-V 컨소시엄에서는 RISC-V 프로세서에 대한 명령어 셋을 32-비트 임베디드 마이크로 컨트롤러, 64-비트 프로세서, 그리고 128-비트 고성능 프로세서에 맞추어 제공하고 있다 [1]. RISC-V 프로세서의 가장 큰 특징은 기본적인 명령어셋은 컨소시엄에서 제공하지만 사용자가 추가할 수 있는 확장 명령어에 대한 제한이 없다는 점이다. 따라서 이를 활용하게 될 경우 목표로 하는 응용 서비스의 속도를 RISC-V 프로세서에 대한 커스터마이징을 통해 고속화하는 것이 가능하다. CHES'20에서는 Kyber와 NewHope와 같은 격자 기반 암호를 최적화하기 위해 RISC-V 상에서의 유한체 연산을 변경하여 85%의 기본 구현을 개선하는 연구가 진행되었다. CHES'21에서는 AES 암호화 연산에 필요한 명령어 셋을 RISC-V에 확장하여 4~10배 이상의 암호화 성능을 개선하는 연구가 제안되었다 [3]. 이러한 확장 연산자와 더불어 같이 연구가 활발히 진행되고 있는 부분은 기본적인 RISC-V 구조 상에서의 암호화 구현이다. 다양한 암호화 모듈을 추가적으로 RISC-V 프로세서 상에 구현하게 될 경우 해당 응용 서비스만 고속화가 되는 한계점이 있다. 하지만 해당 모듈을 하드웨어로 구현해야 하기에 임베디드 프로세서 환경에서는 이와 같은 추가적인 하드웨어 칩 면적은 비용과 환경 상의 한계로 그 활용이 제한적이다. LatinCrypt'19에서는 RISC-V 프로세서의 기본적인 연산자를 활용하여 AES, ChaCha, Keccak, 그리고 곱셈 연산자를 RISC-V 프로세서 상에 최적 구현한 결과가 발표되었다 [4]. [5]에서는 종합 암호화 라이브러리인 NaCl를 RISC-V 프로세서 상에서 고속구현하였다. 현재 RISC-V 프로세서 상에서는 올림 연산이 원활히 수행되지 않는다. 이를 보완하기 위해 redundant representation을 적용한 구현 기법이 제안되었다. CHES'21에서는 RISC-V 프로세서 상에서의 AES에 대한 최신 최적화 연구를 통해 기존 구현에 비해 26%의 성능 향상을 달성하였다 [6].

본 논문에서는 차세대 공개키 암호의 핵심 연산인 곱셈기 연산을 RISC-V 프로세서의 기본 연산자를 통해 최적화 구현한 결과를 제시한다. 본 논문의 구성은 다음과 같다. 2장에서는 RISC-V 프로세서의 구조와 차세대 공개키 암호를 위한 곱셈기 구현에 대해 확인한다. 3장에서는 제안하는 곱셈기 구현 기법을 제시한다. 4장에서는 성능을 비교 분석하며 확장 명령어 셋 후보군에 대해 확인해 본다. 마지막으로 5장에서는 본 논문의 결론을 내린다.

II. 관련 연구

2.1 RISC-V 프로세서의 상세 구조

RISC-V의 32-비트 구조인 RV32I에서는 32개의 32-비트 레지스터 (x0~x31)를 제공한다. 이 중에서 다수의 레지스터는 특수한 용도로 예약되어 있다. RISC-V 프로세서 상에서의 레지스터의 용도는 [표 1]에 나타나 있다. 이 중에서 sp와 s0-s11 레지스터는 callee-saved 레지스터이다.

RISC-V 프로세서는 기본적인 사칙연산과 함께 메모리 연산자 및 프로그램 플로우 연산자들을 제공한다. 본 논문에서 활용한 RISC-V 프로세서 상에서의 명령어 셋은 [표 2]와 같다.

Table 1. Purpose of registers in RISC-V processor

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

Table 2. Instruction set for RISC-V processor

Instruction	Description
add	add
addi	add immediate
sub	subtraction
sltu	set less than unsigned
li	load immediate
mv	move
sw	store word
lw	load word
bne	branch not equal
mul	multiply
mulhu	multiply upper half

2.2 차세대 공개키 암호를 위한 곱셈기 구현

현재 차세대 공개키 암호 알고리즘에 대한 표준화가 미국 NIST를 중심으로 진행되고 있다. 2020년도 7월 기준으로 공모전 Round 3에 해당하는 finalist와 alternate 후보군들이 발표되었다.

대표적인 finalist 후보군들 중에는 격자기반 암호가 있으며 alternate 후보군들 중에는 SIKE가 있다. 해당 암호군들의 핵심 연산자로는 곱셈 연산이 있다. CHES'19에서는 RSA의 곱셈 연산자 가속기를 활용하여 RLWE 암호화 알고리즘을 최적화하는 기법이 제안되었다 [7]. 해당 구현 기법은 특히 Kyber 암호화를 최적화하였다. CHES'18에서는 곱셈 연산자를 ARM 프로세서 상에서 최적화 구현한 결과가 제시되었다 [8]. 이와 더불어 현대 암호와 차세대 공개키 암호를 병합하여 TLS를 구현하는 연구결과도 진행 중에 있다 [9]. 현대 암호인 RSA와 ECC의 경우 곱셈기 연산이 전체 연산의 70% 이상의 부하를 차지하게 된다.

곱셈 연산에 대한 구현은 크게 행 중심으로 곱셈을 수행하는 것과 열 중심으로 곱셈을 수행하는 방법이 있다 [10]. 행 중심으로 연산을 수행하는 것을 school-book 기법이라고 하며 가장 일반적인 곱셈 기법이다. 열 중심 기법은 comba 기법이라고 하며 행 중심과는 반대 방식으로 연산이 수행된다. 최근에는 행과 열 중심을 각각의 타겟으로하는 프로세서 환경에 맞추어 구현한 연구 결과들이 발표되고 있다 [11]. 여기서 곱셈 구현에서 변하지 않는 원칙은 행과 열 중심 기본 연산자에 대한 최적화는 모든 구현에서 필요하다는 점이다. [4]에서는 행 중심 곱셈 기법이 RISC-V 프로세서 상에서 제안되었으며 본 논문에서는 열 중심 곱셈 기법이 RISC-V 프로세서 상에서 보다 효율적임을 증명한다.

이 외에도 32-비트 워드 중 일부 비트를 올림과 내림 연산에 할당하는 redundant 구현 기법이 있다 [12, 13]. 해당 구현 기법은 올림 연산이 비효율적인 컴퓨터 구조의 경우 활용이 가능하다. 하지만 모든 암호군마다 상이한 redundant 표기법을 사용해야 하기 때문에 암호들 간에 호환이 안되는 문제점을 가지고 있다.

본 논문에서는 non-redundant 구현 기법을 활용하여 RISC-V 프로세서 상에서 모든 암호군에 하나의 코드로 동작이 가능한 구현 기법을 제안한다. 이를 통해 코드 크기를 최소화함과 동시에 높은 성능을 달성할 수 있다.

III. RISC-V 프로세서 상에서 제안하는 곱셈기 구현 기법

곱셈기를 구현하기 위해서는 RISC-V 프로세서 상에서 제공하는 사칙 연산자들을 확인하고 이를 효율적으로 활용해야 한다. 현재 RISC-V 프로세서 상에서는 기본적인 사칙 연산자 (덧셈, 뺄셈, 곱셈, 그리고 나눗셈)을 제공한다. RISC-V 프로세서 상에서 고려해야 할 부분은 상태 플래그가 제공되지 않기 때문에 올림과 내림에 대한 값을 확인하는 것이 불가능하다. 따라서 입력값 a와 b를 더하여 출력값 r에 저장하는 경우 올림 값을 저장하기 위해 추가적인 변수 c를 할당하고 활용해야 한다. 해당 올림 값을 확인하기 위한 용도로 sltu 명령어를 활용가능하다. 따라서 이를 적용하면 add r, a, b; sltu c, r, a를 통해 연산 결과를 얻는 것이 가능하다. 곱셈의 경우 mul과 mulh를 통해 32-비트 곱셈에 대한 상위 32-비트와 하위 32-비트 값을 추출하는 것이 가능하다. [표 3]에서는 32-비트 입력값 a0와 b0를 곱하여 c0와 c1에 저장하는 기법이 제시되어 있다. 본 논문에서는 확장 가능한 곱셈기를 구현하기 위해서 26개의 레지스터 (a0-a5, s0-s11, t0-t6, ra)를 활용하였으며 해당 레지스터의 역할은 [표 4]와 같다.

Table 3. 32-bit multiplication on RISC-V processor (macro_mul_32)

```
mul c0, a0, b0
mulhu c1, a0, b0
```

Table 4. Purpose of registers in RISC-V processor for implementation of multiplication

Register	Purpose
a0-a5	memory pointer and loop counter
s0-s11	argument and result
t0-t6	temporal registers
ra	carry

3.1 64-비트 곱셈기

본 연구에서는 기본적인 곱셈의 단위를 64-비트로

정하여서 구현하였다. 그 이유는 최신에 발표되고 있는 Intel과 ARM 프로세서 상에서도 기본적인 곱셈 연산 단위로 64-비트를 채택하고 있으며 차세대 공개키 암호와 현대 공개키 암호에서도 64-비트의 배수 형식으로 파라미터에 대한 곱셈 연산이 수행되고 있기 때문이다. [표 5]에는 64-비트 곱셈을 수행하는 코드가 나타나 있다. 연산의 처음에는 [표 4]에 제시된 기법을 활용하여 32-비트 기반 곱셈 연산을 수행한다. 해당 연산은 곱셈 결과값을 현재 결과값에 반영을 하는 것은 아니며 실제 값만을 계산한 결과이다. 따라서 그 다음으로 수행해야 하는 부분은 기존 결과값에 현재 계산된 결과값을 반영해 주는 연산이 필요하다. [표 5]에 제시된 64-비트 곱셈은 연산 가장 초기에 수행되는 곱셈 연산으로써 기존 결과값이 존재하지 않는다. 따라서 곱셈된 결과값들만을 축적하는 과정만이 요구되어 진다. 64-비트 곱셈 결과값은 총 4개의 32-비트 레지스터 (c0, c1, c2, 그리고 c3)에 저장되며 128-비트를 초과하는 오버플로우는 발생하지 않는다. 구현 시 열 중심 구현 기법이 행 중심 구현 기법보다 RISC-V 프로세서 상에서 이점을 가지는 부분은 올림 연산을 축적하여 연산하는 것이 가능하다는 점이다. [표 5]에서 second accumulation 연산의 경우 올림값을 t4, t5, 그리고 t6 레지스터들에 나누어 저장하게 된다. 해당 값들을 행 중심 구현을 통해 구현하게 될 경우 올림값들이 상위 워드로 전파되는 문제가 발생한다. 하지만 열 중심 구현에서는 올림을 저장해 두는 레지스터들을 두고 해당 값들을 한 번에 모아서 연산하는 것이 가능하다. 특히 올림값들간의 연산은 2^32 워드에 대한 곱셈이 아닌 이상 오버플로우가 발생하지 않기 때문에 add 연산자만으로 효과적인 연산이 가능하다. 현재 연산자의 크기가 큰 RSA의 경우에도 2048-비트 (2^6) 곱셈이 사용되고 있기에 2^32 워드 단위 곱셈은 암호화 연산에서 발생하지 않는다.

Table 5. Initial stage of 64-bit multiplication on RISC-V processor (macro_mul_64_init)

```
//32-bit wise multiplication
macro_mul_32 a0, b0, c0, c1
macro_mul_32 a1, b1, c2, c3
macro_mul_32 a0, b1, t0, t1
macro_mul_32 a1, b0, t2, t3

//first accumulation
add c1, c1, t0
sltu t4, c1, t0
add c1, c1, t2
sltu t5, c1, t2
add t5, t5, t4

//second accumulation
add c2, c2, t5
sltu t4, c2, t5
add c2, c2, t1
sltu t5, c2, t1
add c2, c2, t3
sltu t6, c2, t3
add t5, t5, t4
add t6, t6, t5

//third accumulation
add c3, c3, t6
```

3.2 열 중심 곱셈기 스캐쥴링

열 중심 곱셈기는 [표 5]에서 제시한 바와 같이 64-비트 단위로 곱셈 연산을 구현하고자 하는 전체 비트 수만큼 분배하여 계산하는 과정이 필요하다. [그림 1]에서는 64-비트 워드 단위 곱셈을 활용하여 192-비트 곱셈 연산을 수행하는 방법론에 대해 명시되어 있다. 각각의 직사각형은 64-비트 워드 곱셈을 의미하며 특히 녹색 직사각형은 중간 결과값이 없는 초기 곱셈을 의미하고 빨간 직사각형은 그 이후에 중간 결과값이 있는 경우 64-비트 워드 곱셈을 의미한다. 중간 결과값이 있는 경우 값을 축적시키면서 올

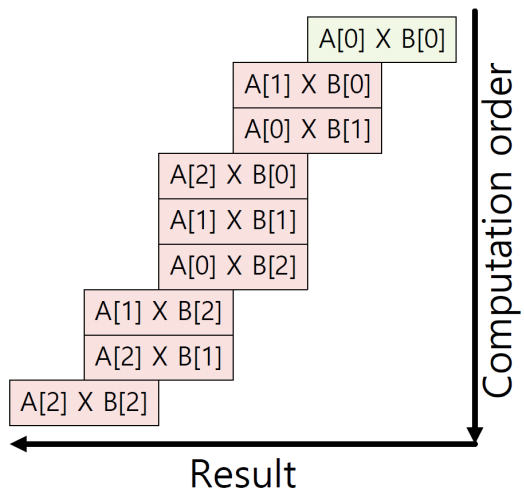

Fig. 1. Column-wise multiplication for RISC-V processor (3-limb cases; each limb represents 64-bit wise word)

림 연산을 효율적으로 수행해 주는 단계가 필요하다.

〔표 6〕에는 중간 64-비트 곱셈 연산에 대한 구현이 구체화되어 나타나 있다. 해당 곱셈 구현이 〔표 5〕에 제시된 방법론과 상이한 점은 기존 결과값이 존재하는 것과 올림 연산이 발생가능하다는 점이다. 따라서 이러한 점을 고려하여 64-비트 곱셈 연산이 설계되었다. 먼저 32-비트 단위로 곱셈을 4번 수행

Table 6. Middle stage of 64-bit multiplication on RISC-V processor (macro_mul_64_mid)

```
//32-bit wise multiplication
macro_mul_32 a0, b0, q0, q1
macro_mul_32 a1, b1, q2, q3
macro_mul_32 a0, b1, t0, t1
macro_mul_32 a1, b0, t2, t3

//first accumulation
add c0, c0, q0
sltu t4, c0, q0
add c1, c1, q1
sltu t5, c1, q1
add c1, c1, t4
sltu t6, c1, t4
add  t6, t6, t5

//second accumulation
add c1, c1, t0
sltu t4, c1, t0
add c1, c1, t2
sltu t5, c1, t2
add t5, t5, t4
add  t5, t6, t5

//third accumulation
add c2, c2, t5
sltu t4, c2, t5
add c2, c2, t1
sltu t5, c2, t1
add c2, c2, t3
sltu t6, c2, t3
add t5, t5, t4
add t6, t6, t5
add c2, c2, q2
sltu t5, c2, q2
add t6, t6, t5

//fourth accumulation
add c3, c3, t6
sltu t5, c3, t6
add c3, c3, q3
sltu t6, c3, q3
add t6, t6, t5
add carry, carry, t6
```

해 준다. 그리고 최하위 워드인 c0 레지스터에 64-비트 곱셈 결과값인 q0를 더해주는 연산을 수행해 준다. 이때 발생하는 올림을 c1 워드까지 연쇄적으로 올림 연산을 수행해주며 최종적으로 t5와 t6 레지스터에 최종 올림값들을 저장한다. 이와 동일한 방식으로 c2와 c3에서 발생하는 올림값까지 처리를 해주도록 한다. 해당 구현 결과는 기존 결과값인 128-비트와 계산된 128-비트를 더해서 128-비트를 초과하는 올림값은 carry 레지스터에 저장하고 다음 곱셈 연산에 반영하도록 한다.

3.3 확장 가능한 곱셈 및 코드 크기 최소화 구현

본 구현 결과는 하나의 구현을 통해 모든 파라미터를 만족하는 확장 가능한 구현으로서 코드 크기는 최소화하면서 성능은 극대화하는 결과를 도출한다. 이를 위해 전체 곱셈을 각 열을 반복 수행하는 내부 루프와 전체 곱셈 열을 관리하는 외부 루프로 나누어서 구현하였다. 특히 내부 루프 카운터는 입력 인자 포인터에 카운터를 더하는 형식으로 구현하였으며 외부 루프 카운터는 결과값 포인터에 카운터를 더하는 형식으로 구현하였다. 또한 반복문을 구현하기 위해 bne 명령어를 통해 조건을 분별하여 분기하도록 하였다.

IV. 평 가

평가에 활용한 프로세서는 HiFive사의 HiFive1 Rev B 플랫폼이다. 해당 플랫폼은 E31 코어가 탑재되어 있다. RV32IMAC 명령어 셋이 사용되며 기본적인 곱셈기 연산자가 제공된다. E31 코어는 5단계 single-issue 순차 파이프라이닝을 제공하며

Table 7. Code for timing measurement in RISC-V processor

```
.text

.globl getcycles
.align 2
getcycles:
    csrr a1, mcycleh
    csrr a0, mcycle
    csrr a2, mcycleh
    bne a1, a2, getcycles
    ret
.size getcycles,.-getcycles
```

320MHz로 연산이 수행된다. 프로그래밍 및 디버깅 환경은 HiFive사에서 제공하는 FreedomStudio 4.7.2. IDE를 활용하였다. 연산 수행시간을 측정하기 위해 RISC-V 내부의 clock cycle을 〔표 7〕과 같이 추출하였으며 이를 64-비트 데이터 형식으로 반환하여 측정에 활용하였다 (uint64_t count = getcycles();).

4.1 RISC-V 상에서의 곱셈 연산 성능 평가

〔표 8〕에서는 RISC-V 프로세서 상에서 non-redudant 형식으로 구현한 곱셈에 대한 성능을 나타내고 있다. 곱셈 연산은 다양한 비트 길이에 맞추어서 수행되었으며 이를 〔4〕에서 구현한 school-book 곱셈기와 성능을 비교분석하였다. 256-비트의 경우 현대 암호인 ECC에서 활용이 가능하며 512~1024-비트의 경우에는 차세대 공개키 암호인 SIKE 그리고 Kyber 혹은 현대 암호인 RSA에서 활용이 가능하다. 256-비트의 경우 약 19%의 성능 향상이 있었으며 512-비트의 경우 약 8%의 성능 향상을 달성하였다. 본 구현 기법은 입력인자에 의존하여 분기문을 수행하는 형식이 아닌 입력인자에 값과 무관하게 항상 일정한 연산을 수행하는 constant timing으로 구현되었다. 따라서 timing attack을 통한 구현 공격에는 높은 보안성을 달성함을 확인할 수 있다.

본 연구 결과의 차별점은 하나의 코드를 통해 모든 파라미터를 만족하는 확장 가능한 곱셈 구현이라는 점이다 〔14〕. 핵심 64-비트 구현을 내부와 외부 루프를 통해 반복적으로 수행하는 방법을 통해 모든 파라미터를 만족하는 구현을 수행하였다. 전체 곱셈 코드의 크기는 640 바이트로서 임베디드 프로세서에 보다 최적화된 코드 크기를 제시한다. 이를 이용할 경우 현대 공개키 암호와 차세대 공개키 암호의 핵심 곱셈 연산자를 하나의 최적화된 코드를 통해 구현하는 것이 가능하다.

Table 8. Performance of arbitrary-precision multiplication on RISC-V processor, timing is measured in clock cycles

Bit length	〔4〕	This work
256-bit	≈ 1,800	1,460
512-bit	≈ 6,100	5,625
1024-bit	-	21,857

4.2 곱셈 연산 최적화를 위한 RISC-V 프로세서 상에서의 추가 명령어 셋 제안

본 논문에서는 기본적인 RISC-V 프로세서 상에서의 명령어셋을 활용하여 곱셈 연산을 최적화 구현하였다. 구현을 진행하면서 추가적인 하드웨어 지원이 있다면 성능이 향상될 것 같은 부분은 다음과 같다. 첫 번째로 32-비트 곱셈을 상위와 하위 워드로 나누어서 결과값을 도출해 주는 부분을 합쳐서 64-비트 결과값을 한 번에 도출해 주게 된다면 보다 효율적인 연산이 가능할 것으로 보인다. 이와 더불어 〔표 5〕와 〔표 6〕에서는 64-비트 단위의 곱셈을 위한 RISC-V 상의 연산자들의 조합이 나타나 있다. 만약에 해당 알고리즘을 지원하는 64-비트 기반 곱셈기가 추가된다면 연산 성능을 획기적으로 줄이는 것이 가능할 것으로 보인다. 이와 더불어 RISC-V 상에서 지원하지 않는 덧셈 연산에 대한 올림 플래그 지원 혹은 add 연산자와 sltu를 조합한 명령어 셋이 지원된다면 덧셈과 올림을 확인하는 부분에 대한 부하가 줄어 들 것이다.

V. 결 론

본 논문에서는 RISC-V 상에서의 차세대 공개키 암호의 고속 구현을 위해 필요한 곱셈기 연산자를 최적 구현하였다. 이를 통해 기본적인 RISC-V 프로세서 상에서의 차세대 공개키 암호 및 현대 공개키 암호 구현 연산 성능을 개선할 수 있었다. 이와 더불어 곱셈 연산 중 높은 부하가 발생하는 영역에 대해서 추후 확장 명령어 도입을 통해 가속화가 가능한 부분도 함께 제안하였다. 본 연구 결과를 통해 차세대 공개키 암호와 현대 공개키 암호의 고속화와 추가적인 확장 명령어를 통해 전용 고속 암호화 칩으로의 양산이 가능할 것으로 보인다.

References

〔1〕 K. Asanovic, and A. Waterman, "The RISC-V Instruction Set Manual. In Privileged Architecture," *RISC-V Foundation*, 2(1), pp. 1-91, May. 2017.

〔2〕 E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, R. Petri, "ISA Extensions for Finite Field Arithmetic," *IACR*

Transactions on Cryptographic Hardware and Embedded Systems, pp. 219-242, Aug. 2020.

[3] B. Marshall, G. R. Newell, D. Page, M. J. O. Saarinen, and C. Wolf, "The design of scalar AES Instruction Set Extensions for RISC-V," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 109-136, Aug. 2021.

[4] K. Stoffelen, "Efficient cryptography on the RISC-V architecture. *In International Conference on Cryptology and Information Security in Latin America*, pp. 323-340, Oct. 2019.

[5] S. van den Berg, "RISC-V implementation of the NaCl-library," *Master Thesis*, 1(1), pp. 1-52, 2020.

[6] A. Adomnicai, and T. Peyrin, "Fixslicing AES-like Ciphers," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 402-425, Aug. 2021.

[7] M. R. Albrecht, C. Hanser, A. Hoeller, T. Pöppelmann, F. Virdia, A. Wallner, "Implementing RLWE-based schemes using an RSA co-processor," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 169-208, Aug. 2019.

[8] H. Seo, Z. Liu, P. Longa, and Z. Hu, "SIDH on ARM: faster modular multiplications for faster post-quantum supersingular isogeny key exchange," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1-20, Aug. 2018.

[9] H. Seo, R. Azarderakhsh, "Curve448 on 32-Bit ARM Cortex-M4," *In International Conference on Information Security and Cryptology*, pp. 125-139, Dec. 2020.

[10] P. G. Comba, "Exponentiation cryptosystems on the IBM PC," *IBM systems journal*, 29(4), pp. 526-538, 1990.

[11] H. Seo, H. Kim, "Multi-precision multiplication for public-key cryptography on embedded microprocessors," *In International Workshop on Information Security Applications*, pp. 55-67, Aug. 2012.

[12] H. Seo, P. Sanal, R. Azarderakhsh, "SIKE in 32-bit ARM Processors Based on Redundant Number System for NIST Level-II," *ACM Transactions on Embedded Computing Systems (TECS)*, 20(3), pp. 1-23, 2021.

[13] H. Seo, Z. Liu, Y. Nogami, T. Park, J. Choi, L. Zhou, H. Kim, "Faster ECC over F_{2^{521}-1} (feat. NEON)," *In ICISC 2015*, pp. 169-181, Dec. 2015.

[14] H. Seo, "Memory efficient implementation of modular multiplication for 32-bit ARM Cortex-M4," *Applied Sciences*, 10(4), pp. 1539, 2020.

〈저자소개〉

서 화 정 (Hwa-jeong Seo) 종신회원
2010년 3월: 부산대학교 컴퓨터공학과 졸업
2012년 3월: 부산대학교 컴퓨터공학과 석사
2016년 3월: 부산대학교 컴퓨터공학과 박사
2016년~2017년: 싱가포르 과학기술청 연구원
2017년~현재: 한성대학교 IT융합공학부 조교수
〈관심분야〉 정보보호, 암호화 구현, IoT

권 혁 동 (Hyeok-dong Kwon) 학생회원
2018년 2월: 한성대학교 IT응용시스템공학부 졸업
2020년 3월: 한성대학교 IT융합공학부 석사
2020년 3월~현재: 한성대학교 정보컴퓨터공학과 박사과정
〈관심분야〉 정보보안, 암호구현

장 경 배 (Kyung-bae Jang) 학생회원
2019년 3월: 한성대학교 IT응용시스템공학부 졸업
2021년 3월: 한성대학교 IT융합공학부 석사
2021년 3월~현재: 한성대학교 정보컴퓨터공학과 박사과정
〈관심분야〉 정보보호, 암호, 양자컴퓨터

김 현 준 (Hyun-jun Kim) 학생회원
2019년 3월: 한성대학교 IT응용시스템공학부 졸업
2021년 3월: 한성대학교 IT융합공학부 석사
2021년 3월~현재: 한성대학교 정보컴퓨터공학과 박사과정
〈관심분야〉 부채널 분석, 블록체인