JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 22, No. 9: 1257~1263, Sep. 2018

ARM Cortex-M3 상에서 곱셈 연산 최적화 구현

서화정*

Compact Implementation of Multiplication on ARM Cortex-M3 Processors

Hwa-jeong Seo*

Department of IT Engineering, Hansung University, Seoul 02876, Korea

요 약

경량 사물인터넷 디바이스 상에서의 암호화 구현은 정확하고 빠르게 연산을 수행하여 서비스의 가용성을 높이는 것이 중요하다. 특히 곱셈 연산은 RSA, ECC, 그리고 SIDH와 같은 공개키 암호화에 활용되는 핵심 연산으로 최적화된 구현이 요구된다. 하지만 최신 저전력 프로세서인 ARM Cortex-M3 프로세서의 경우에는 곱셈연산 입력 크기에 따라 수행속도가 달라지는 보안 취약점을 가지고 있다. 수행속도가 달라지게 될 경우 연산 시간의 차이점을 확인하여 비밀정보를 추출하는 것이 가능하다. 이를 보완하기 위해 최근 연구에서는 고정된 연산 시간 안에 곱셈 연산을 수행하는 기법이 제안되었다. 하지만 해당 구현에서는 여전히 속도가 완전히 최적화되어 있지 않다. 본 논문에서는 기존에 제안된 곱셈연산을 보다 효율적으로 연산하기 위한 기법을 제안한다. 제안된 기법은 기존 방식에 비해 연산 속도를 최대 25.7% 향상시킨다.

ABSTRACT

Secure authentication technology is a fundamental building block for secure services for Internet of Things devices. Particularly, the multiplication operation is a core operation of public key cryptography, such as RSA, ECC, and SIDH. However, modern low-power processor, namely ARM Cortex-M3 processor, is not secure enough for practical usages, since it executes the multiplication operation in variable-time depending on the input length. When the execution is performed in variable-time, the attacker can extract the password from the measured timing. In order to resolve this issue, recent work presented constant-time solution for multiplication operation. However, the implementation still missed various speed-optimization techniques. In this paper, we analyze previous multiplication methods over ARM Cortex-M3 and provide optimized implementations to accelerate the speed-performance further. The proposed method successfully accelerates the execution-time by up-to 25.7% than previous works.

키워드 : ARM Cortex-M3, 부채널 공격, 소프트웨어 구현, 곱셈

Keywords : ARM Cortex-M3, Side Channel Attack, Software Implementation, Multiplication

Received 8 May 2018, **Revised** 14 May 2018, **Accepted** 30 May 2018

*** Corresponding Author** Hwa-jeong Seo(E-mail:hwajeong@hansung.ac.kr, Tel:+82-2-760-8033)
Department of IT Engineering, Hansung University, Seoul 02876, Korea

Open Access http://doi.org/10.6109/jkiice.2018.22.9.1257 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

사물인터넷을 활용한 서비스는 저전력 경량 디바이스에서 수집된 센서 데이터가 인터넷을 통해 연결된 고성능의 서버 컴퓨터에 전달되고 서버 컴퓨터에서는 해당 데이터를 분석하여 새로운 정보를 도출해 내는 일련의 과정을 의미한다. 이처럼 사물인터넷은 이기종의 컴퓨터들이 네트워크를 통해 상호 연결되어야 하기 때문에 전 통신구간에 걸친 암호화가 매우 중요하다. 기존 암호화 연산은 고성능의 컴퓨터 상에서는 서비스 가용성을 만족하는 수준에서 연산 성능을 도출하는 것이 가능하였지만 연산능력과 저장공간 그리고 에너지가 부족한 경량 프로세서 상에서는 복잡한 암호화 연산을 가용성을 만족하는 시간 안에 구현하는 것이 매우 어렵다. 따라서 최근에는 이러한 문제점을 해결하기 위해 사물인터넷에서 사용되는 경량 디바이스 상에서도 효율적으로 암호화 연산을 수행하는 방법에 대한 연구가 지속적으로 연구되어 왔다. 하지만 사물인터넷 서비스에서 데이터를 수집하는 경량 디바이스의 경우 누구나 쉽게 물리적으로 접근 가능한 위치에 설치되는 특징을 가진다. 따라서 공격자가 직접 해당 경량 디바이스에 물리적 공격을 수행하게 될 경우 안전하게 암호화된 정보를 쉽게 복호화하는 것이 가능하며 이를 부채널 공격이라 한다. 따라서 경량 디바이스 상에서의 구현에서는 성능을 향상시키는 것도 중요하지만 이와 더불어 부채널 공격에 대한 방어기법도 함께 제시되어야 하는 특징을 가진다. 따라서 본 논문에서는 공개키 암호화 (RSA, ECC, 그리고 SIDH)의 필수 연산자인 곱셈연산을 보다 효율적이고 안전하게 구현하는 방안에 대해 확인해 보도록 한다. 이를 위해 본 논문에서는 기존의 곱셈기법이 가지는 문제점에 대해 먼저 확인해보도록 하며 최종적으로는 효율적이면서 안전한 방안에 대해 제시하도록 한다. 특히 제안된 기법은 부채널 공격 중 Timing 공격과 Simple Power Analysis (SPA)에 안전한 특성을 가진다.

본 논문의 구성은 다음과 같다. 2장에서는 곱셈 알고리즘이 구현되는 경량 디바이스인 32-비트 ARM Cortex-M3 프로세서와 곱셈 알고리즘 그리고 대표적인 부채널 공격 방법에 대해 확인해 보도록 한다. 3장에서는 이전에 타겟 프로세서 상에서 구현된 기법들의 특징에 대해 확인해보고 이를 개선한 새로운 기법을 제안하도록 한다. 4장에서는 제안된 기법의 성능을 타겟 보드 상에서 확인해보도록 한다. 마지막으로 5장에서는 본 논문의 결론을 내린다.

Ⅱ. 관련 연구

2.1. 타겟 프로세서: ARM Cortex-M3

아두이노 DUE의 메인 프로세서로 활용되고 있는 ARM Cortex-M3 프로세서는 사물인터넷 서비스 구현에 매우 효과적인 특징을 가지고 있다. ARM Cortex-M3 프로세서의 장점으로는 32-비트로 동작하는 연산자, 저전력 프로세싱 그리고 저렴한 가격으로 볼 수 있다. 프로세서는 ARMv7-M 구조를 따르며 3단계 파이프 라인을 지원한다. 프로그램 코드를 최적화하기 위해서 16-비트 단위로 연산자를 구성하는 Thumb-1 명령어 셋을 지원하며 이를 이용하면 동일한 32-비트 ARM 연산자로 동작하는 프로그램 코드를 절반에 가까운 코드로 구현하는 것이 가능하다. 이와 더불어 Thumb-2 명령어 셋도 지원한다. 기존 Thumb-1 명령어 셋의 경우 16-비트의 한계로 인해 작성이 불가능한 코드가 발생하게 된다. 이러한 경우에는 32-비트 연산자인 ARM 명령어를 활용하여 성능과 코드 크기를 조절한 것으로 살펴볼 수 있다. 본 논문에서는 ARM Cortex-M3 프로세서 상의 대표적인 명령어 셋인 덧셈, 이동, 곱셈, 치환 그리고 비트-시프트를 활용하게 되며 자세한 사항은 표 1과 같다.

Table. 1 Cortex-M3 instruction set summary

Assembler	Description	Cycles
ADD	Addition	1
MOV	Moving	1
MUL	16-bit wise multiplication	1
UMULL	32-bit wise multiplication	3 / 5
UXTH	Unsigned halfword	1
LSR	Logical shift right	1

2.2. 곱셈 구현

공개키 암호화 연산에서 가장 핵심적인 연산은 곱셈연산으로써 빠르고 안전하게 입력된 값들을 활용한 곱셈 결과를 도출하는 것이 중요하다. 이를 효과적으로 수행하기 위해 많은 연구가 진행되어 왔다. 가장 기본적인

곱셈 연산으로는 곱셈 시 동일한 곱셈 열에 위치하는 값들을 한 번에 묶어서 계산하는 product-scanning 기법이 있다[1]. 해당 기법은 지속적으로 결과값을 저장하기 위해 메모리에 접근하지 않아도 되기 때문에 연산 속도가 빠르다. 하지만 해당 기법은 입력 인자를 지속적으로 메모리에서 불러와야 하는 문제점을 가지고 있다. 이를 해결하기 위해 제안된 consecutive-operand-caching 기법은 연산 시작부터 마지막까지 입력인자를 지속적으로 caching하는 방법을 통해 메모리 접근을 효과적으로 최적화하고 있다[2]. 이러한 구현 기법은 n개의 워드에 대한 연산 복잡도가 n^2으로 나타나게 된다. 최근에는 ARM Cortex-M4 프로세서 상에서 consecutive-operand- caching을 효과적으로 수행하는 방법이 제안되었다[3]. 해당 기법에서는 ARMv7 구조 상에서 제안된 UMAAL 연산자를 활용하여 한 번의 곱셈과 두 번의 덧셈을 한 번에 수행하는 것이 가능하도록 하였다. 이를 활용하여 기존의 기법보다 약 50%의 성능 향상이 가능함을 제시하였다. 이와 더불어 최근에는 기존의 곱셈 구현 결과를 효율적으로 향상시키기 위해 Karatsuba 알고리즘을 적용한 예시가 제시되었다[4]. 기존에는 additive 방식을 통해 Karatsuba 알고리즘이 구현되었다면 해당 구현에서는 subtractive 방식을 통해 Karatsuba 알고리즘을 구현함으로써 성능을 효과적으로 향상시키는 방안을 제시하였다. Subtractive 방식의 경우 additive 방식과 달리 carry가 발생하지 않아 입력인자의 길이가 항상 일정한 크기로 고정되도록 함으로써 성능 향상이 가능함을 확인할 수 있었다. 이와 더불어 hybrid-scanning 과 같은 방식을 이용하여 연산 성능을 효과적으로 개선하는 방안에 대한 연구가 진행되었다[5]. Cortex-M0 프로세서 상에서의 구현 결과는 [6]에서 제시되었다. 해당 구현에서는 32-비트 곱셈 연산자(MUL)를 사용하여 정수 곱셈을 효율적으로 수행하는 방안이 제시되었다. 최근에는 Cortex-M3 프로세서 상에서 안전하고 빠르게 곱셈을 구현하는 방안이 연구되었다[7]. 하지만 해당 구현에서는 최적화가 가능한 부분에 대한 검토가 많이 이루어지지 않았다. 따라서 본 논문에서는 기존의 구현 기법을 보다 효과적으로 최적화하는 방안에 대해 확인해 보도록 한다.

2.3. 부채널 공격

암호화 모듈 구현 시 빠른 속도를 제공하는 것과 더불어 해당 암호화 모듈이 안전하게 연산되는 것을 보장해주는 것은 중요한 요소이다. 그 이유는 해커가 암호화 자체에 대한 공격은 어렵기 때문에 이와는 다른 채널 즉 부가적인 채널을 통해 암호화 키를 추출하는 공격을 수행하기 때문이다. 부채널 공격 기법은 암호화 연산이 수행되는 동안의 실행 속도 분석 혹은 연산 시 소모되는 전력 패턴을 분석하여 암호화 키를 추출하는 방식이다. 따라서 암호화 연산 시 해당 연산이 키에 의존하지 않도록 구현하는 것이 매우 중요하다. 부채널 공격 혹은 분석 기법으로는 크게 실행 속도 분석과 전력 소모 분석으로 나누어 볼 수 있다. 실행 속도 분석은 프로세서 상에서 연산을 수행하는 속도가 암호화에 사용되는 키가 다름에 따라 달라짐을 이용하는 기술로써 키에 의존적인 연산속도를 분석함으로써 실제 암호화 키를 확인하는 방식이다. 따라서 암호화 모듈은 암호화 키와는 무관하게 언제나 동일한 연산 속도를 도출하도록 구현하는 것이 중요하다. 이와 함께 많은 분석에 활용되는 기법으로는 전력 소모 패턴 분석으로 볼 수 있다. 암호화 연산을 수행하는 프로세서는 특정한 연산에 보다 많은 전력을 소모하게 된다. 따라서 실행 속도와 더불어 암호화 모듈은 항상 일정한 전력소모 패턴을 나타내도록 구현하는 것이 중요하다.

Ⅲ. 곱셈 연산 최적화 기법

곱셈 연산은 곱셈과 곱셈 결과를 지속적으로 가산 해주는 부분(MAC: Multiply Accumulate Operation)으로 구성된다. 따라서 곱셈과 가산해주는 부분은 곱셈 길이만큼 반복되기 때문에 MAC을 최적화하여 구현하는 것이 가장 중요하다.

ARM Cortex-M3 프로세서 상에서 UMULL 연산자를 이용한 32-비트 단위 곱셈의 경우 연산 길이에 따라 연산 속도가 변경되는 문제점을 가지고 있다. 따라서 이러한 문제점을 해결하기 위해 선택할 수 있는 방안은 MUL 연산자를 이용하여 32-비트 곱셈을 16-비트 단위 곱셈으로 구현하는 것이다. MUL 연산자를 활용한 구현 기법은 표 2에 나타나 있다. 32-비트 곱셈의 인자를 하위 16-비트 그리고 상위 16-비트 씩 나누어서 각각 곱셈을 수행함으로써 연산 수행에 소모되는 시간을 효율적으로 줄이고 안전성을 제고하는 것이 가능하다.

Table. 2 Multiply accumulate operation for multiplication using MUL instruction [6]

```
Input: operand pointers R8, R9
Output: results R3, R4, R5

 1: MOV R1, R8
 2: LDR R1, [R1, #offset1]
 3: MOV R2, R9
 4: LDR R2, [R2, #offset2]

 5: UXTH R6, R1
 6: UXTH R7, R2
 7: LSR R1, R1, #16
 8: LSR R2, R2, #16

 9: MOV R0, R6
10: MUL R0, R0, R7
11: MUL R6, R6, R2
12: MUL R2, R2, R1
13: MUL R1, R1, R7

14: MOV R7, #0
15: ADDS R3, R3, R0
16: ADCS R4, R4, R2
17: ADCS R5, R5, R7

18: LSL R0, R6, #16
19: LSR R2, R6, #16
20: ADS R3, R3, R0
21: ADCS R4, R4, R2
22: ADCS R5, R5, R7

23: LSL R0, R1, #16
24: LSR R2, R1, #16
25: ADDS R3, R3, R0
26: ADCS R4, R4, R2
27: ADCS R5, R5, R7
```

최근에는 표 2의 기법을 ARM 프로세서 상의 Barrel Shifter 연산자를 통해 효율적으로 구현한 결과가 제시되었다. Barrel Shifter는 연산에 필요한 인자를 입력으로 넣어 줄 때 해당 인자를 원하는 비트만큼 쉬프트 혹은 회전 연산한 이후에 연산자의 인자로 값을 넣어 준다. 해당 기법은 쉬프트와 회전 연산에 필요한 연산 시간 소모를 최소화하는 데 도움을 준다.

Table. 3 Multiply accumulate operation for multiplication using MUL instruction and barrel shifter [7]

```
Input: operand pointers R8, R9
Output: results R3, R4, R5

 1: LDR R1, [R8, #offset1]
 2: LDR R2, [R9, #offset2]

 3: UXTH R6, R1
 4: UXTH R7, R2
 5: LSR R1, R1, #16
 6: LSR R2, R2, #16

 7: MUL R0, R6, R7
 8: MUL R6, R6, R2
 9: MUL R2, R2, R1
10: MUL R1, R1, R7

11: ADDS R3, R3, R0
12: ADCS R4, R4, R2
13: ADCS R5, R5, #0

14: ADDS R3, R3, R6, LSL #16
15: ADCS R4, R4, R6, LSR #16
16: ADCS R5, R5, #0

17: ADDS R3, R3, R1, LSL #16
18: ADCS R4, R4, R1, LSR #16
19: ADCS R5, R5, #0
```

표 3을 살펴보면 프로그램 코드의 14, 15, 17, 그리고 18 줄에는 연산된 결과값을 저장할 때 16-비트 결과값이 회전이 된 경우에는 Barrel Shifter 기법을 적용하여 연산 결과를 Shift 시킨 상태로 저장하게 된다. 이와 더불어 ARM Cortex-M3 프로세서의 경우에는 상수값을 바로 연산자의 인자로 활용하는 것이 가능하다. 이를 활용하여 결과값 가산 과정이 필요한 코드의 14~16 줄과 17~19줄에서는 상수값을 0으로 설정하여 값을 축적하게 된다.

Table. 4 Proposed multiply accumulate operation for multiplication using MUL instruction, barrel shifter, and re-order

```
Input: operand pointers R8, R9
Output: results R3, R4, R5
 1: LDMDB R8!, R1
 2: LDM R9!, R2

 3: UXTH R6, R1
 4: UXTH R7, R2
 5: LSR R1, R1, #16
 6: LSR R2, R2, #16

 7: MUL R0, R6, R7
 8: MUL R6, R6, R2
 9: MUL R2, R2, R1
10: MUL R1, R1, R7

11: ADDS R0, R0, R6, LSL #16
12: ADCS R2, R2, R6, LSR #16

13: ADDS R0, R0, R1, LSL #16
14: ADCS R2, R2, R1, LSR #16

15: ADDS R3, R3, R0
16: ADCS R4, R4, R2
17: ADCS R5, R5, #0
```

표 4에는 제안하는 기법을 통한 구현이 제시되어 있다. 기존 기법과 비교해서 가장 큰 차이점은 16-비트 단위의 곱셈 결과(R1 그리고 R6)를 곱셈 결과 저장에 사용되는 다른 변수(R0 그리고 R2)에 먼저 더해주는 기법이다. 이는 32-비트 곱셈의 경우 절대로 64-비트 이상의 결과값이 도출되지 않는 점에 착안하여 제안한 기법이다. 새롭게 조합된 연산 순서를 이용하면 Carry가 발생하지 않게 됨으로써 기존 연산순서에서 Carry를 위해 수행하였던 2번의 덧셈을 수행하지 않아도 되는 장점이 있다. 이를 구현하기 위해 코드의 11~14줄에서는 연산 결과를 barrel shifter 연산을 통해 사전 이동을 한 후에 중간 결과값에 더해주게 된다.

이와 더불어 코드의 1~2줄에서와 같이 메모리에 저장된 인자들에 대한 효율적인 접근을 위해 이후 증가 명령어인 LDM과 이전 감소 명령어인 LDMDB를 활용하여 메모리 접근을 위한 주소 조정에 소모되는 컴퓨터의 2 클록 사이클을 최적화하였다. 자동으로 조정된 메모리 주소는 메모리 접근을 효과적으로 한다.

마지막으로 공개키 암호 구현에 사용되는 곱셈 연산자는 짧은 정수부터 긴 정수까지 다양한 길이로 수행되게 된다. 특히 긴 정수 연산에서는 Karatsuba 곱셈기를 활용하게 될 경우 곱셈에 대한 복잡도를 약 25% 감소시키는 장점을 가진다[8]. 따라서 제안하는 기법의 성능을 보다 향상시키기 위해 Karatsuba 곱셈 기법을 긴 정수 연산일 경우 제안하는 기법과 동시에 활용하여 연산 효율성을 높였다.

Ⅳ. 평가

본 장에서는 Micro-ECC에서 제안된 기법과 본 논문에서 제안하는 기법의 성능을 비교 분석한다. 성능 분석을 위해 ARM Cortex-M3 프로세서를 탑재하고 있는 Arduino-DUE 개발 보드 상에서 소프트웨어를 Arduino IDE로 개발하였다. 프로그램은 어셈블리언어 처리를 위해 inline assembly 기법을 적용하여 Arduino IDE가 인식가능한 방향으로 제작하였다. 연산속도를 도출하기 위해 프로세서의 SysTick 사용하였다. 특히 Arduino-DUE 개발 보드는 84MHz로 동작을 시켰으며 정확한 결과 도출을 위해 각각의 연산은 1000번씩 수행하여 결과를 도출하였다.

4.1. 안전성

본 논문에서 제안하는 구현 기법은 UMULL과 같이 연산 결과에 의존적인 구현이 아닌 항상 일정한 속도로 곱셈 결과를 도출하는 MUL 연산자를 활용하여 Timing 공격에 안전하도록 구현되었다.

4.2. 효율성

본 논문의 가장 큰 특징으로는 연산 성능을 효과적으로 개선하였다는 점을 들 수 있다. 크게 핵심 연산인 곱셈과 가산 부분에서 덧셈의 순서를 변경함으로써 필요한 연산자의 수를 감소시켰으며 메모리에 대한 접근을 자동화하기 위해 이후 증가 명령어와 이전 감소 명령어를 활용하였다. 이와 더불어 기존 연구에서는 적용하지 않았던 Karatsuba 알고리즘을 활용한 결과를 제시함으

로써 긴 연산자에 대한 곱셈 결과를 보다 효율적으로 최적화하였다. 표 5와 그림 1을 살펴보면 기존 기법보다 제안하는 기법의 성능이 Karatsuba 곱셈을 쓰지 않은 경우 약 11%정도 높게 나타나고 있다. Karatsuba를 적용할 경우 짧은 정수의 경우에는 감소하는 곱셈 연산보다 부가적인 연산의 증가로 인해 속도가 감소하지만 512-비트 연산 이후부터는 성능이 크게 개선됨을 확인할 수 있다. 해당 증가폭은 2048-비트에서 가장 크게 관찰되며 이는 Karatsuba 곱셈을 적용하지 않은 결과보다 25.7% 향상된 결과이다.

Table. 5 Comparison of execution timings (in 1000 clock cycle) for multi-precision multiplication

Method	Operand length (bit)			
	256	512	1024	2048
Micro-ECC [6]	3.5	13.3	51.7	203.7
Seo [7]	2.7	9.6	37.1	145.9
This work (w/o Karatsuba)	2.4	9.0	34.5	135.6
This work (w/ Karatsuba)	3.2	8.8	29.5	108.4

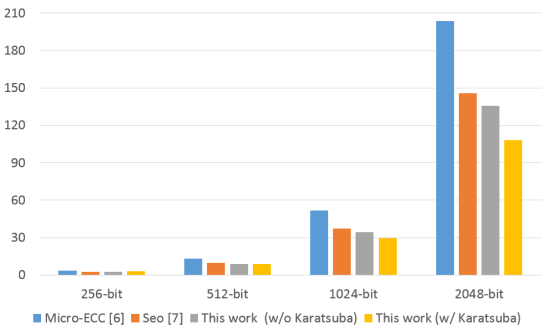

Fig. 1 Comparison of execution timings (in 1000 clock cycle) for multi-precision multiplication

V. 결 론

본 논문에서는 ARM Cortex-M3 프로세서 상에서 안전하고 효율적으로 동작하는 곱셈 구현 기법에 대해 확인해 보았다. 연산 성능 향상을 위해 MAC의 핵심 연산자의 수를 최적화하였으며 메모리 접근을 효율화하였다. 이와 더불어 Karatsuba 곱셈 기법을 통해 긴 정수 연산을 최대 25.7% 향상시켰다. 본 논문에선 제시하는 기법은 RSA, ECC, SIDH와 같은 다양한 공개키 암호화 구현에 활용가능한 기술로써 그 효용성이 매우 높다. 이를 통해 클라우드 환경 상에서의 보안제공에 이바지 할 수 있을 것으로 예상된다[9].

ACKNOWLEDGEMENT

This work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2017R1C1B5075742) and the MSIT(Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program(2014-1-00743) supervised by the IITP (Institute for Information & communications Technology Promotion)

References

[1] H. Seo, Y. Lee, H. Kim, T. Park, and H. Kim, "Binary and prime field multiplication for public key cryptography on embedded microprocessors," *Security and Communication Networks*, vol. 7, no. 4, pp. 774-787, Apr. 2014.

[2] H. Seo and H. Kim, "Consecutive operand-caching method for multiprecision multiplication, revisited," *Journal of information and communication convergence engineering*, vol. 13, no. 1, pp. 27-35, Mar. 2015.

[3] H. Fujii and D. F. Aranha, "Curve25519 for the Cortex-M4 and beyond," *Progress in Cryptology Latincrypt*, La Habana, pp. 1-18, 2017.

[4] M. Hutter and P. Schwabe, "Multiprecision multiplication on AVR revisited," *Journal of Cryptographic Engineering*, vol. 5, no. 3, pp. 201-214, Apr. 2015.

[5] Z. Liu and J. Groschdl, "New speed records for Montgomery modular multiplication on 8-bit AVR microcontrollers," *International Conference on Cryptology in Africa*, Marrakesh, pp. 215-234, 2014.

[6] K. MacKay. ECDH and ECDSA for 8-bit, 32-bit, and 64-bit processors [Internet]. Available: https://github.com/kmackay/micro-ecc.

[7] H. Seo, "Secure Multiplication Method against Side Channel Attack on ARM Cortex-M3," *Journal of The Korea*

Institute of Information Security & Cryptology, vol. 27, no. 4, pp. 943-949, Aug. 2017.

[8] A. Karatsuba and Y. Ofman, “Multiplication of Multidigit Numbers on Automata," *Cybernetics and Control Theory*, vol. 7, no. 7, pp. 595-596, Jan. 1963.

[9] V. Bhavana, "Data Security in Cloud environments," *Asia-pacific Journal of Convergent Research Interchange*, vol.1, no.4, pp. 25-31, Dec. 2015.

서화정(Hwa-jeong Seo)

2010년 2월: 부산대학교 컴퓨터공학과 학사 졸업
2012년 2월: 부산대학교 컴퓨터공학과 석사 졸업
2012년 3월~2016년 1월: 부산대학교 컴퓨터공학과 박사 졸업
2016년 1월~2017년 3월: 싱가포르 과학기술청
2017년 4월~현재: 한성대학교 IT 융합공학부 조교수
※관심분야 : 정보보호, 암호화 구현, IoT