Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.4, Aug. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.4.943

ARM Cortex-M3 상에서 부채널 공격에 강인한 곱셈 연산 구현*

서 화 정†‡
한성대학교

Secure Multiplication Method against Side Channel Attack on ARM Cortex-M3*

Hwajeong Seo†‡
Hansung University

요 약

경량 사물인터넷 디바이스 상에서의 암호화 구현은 정확하고 빠르게 연산을 수행하여 서비스의 가용성을 높이는 것이 중요하다. 하지만 공격자가 해당 경량 디바이스 상에서 수행되는 연산 특징을 분석하여 비밀정보를 추출해 낼 경우 사용자의 비밀번호가 공격자에게 쉽게 노출될 수 있는 문제점을 가지고 있다. 특히 최신 ARM Cortex-M3의 경우 곱셈연산이 입력의 크기에 따라 수행 속도가 달라지는 취약점을 가지고 있다. 본 논문에서는 지금까지 제안된 안전한 곱셈 구현기법의 장단점을 분석하고 더 나아가 최신 곱셈기법을 최적화하는 방안에 대해 확인해 본다. 제안된 기법은 기존 방식의 속도를 최대 28.4% 향상시킨다.

ABSTRACT

Cryptography implementation over lightweight Internet of Things (IoT) device needs to provide an accurate and fast execution for high service availability. However, adversaries can extract the secret information from the lightweight device by analyzing the unique features of computation in the device. In particular, modern ARM Cortex-M3 processors perform the multiplication in different execution timings when the input values are varied. In this paper, we analyze previous multiplication methods over ARM Cortex-M3 and provide optimized techniques to accelerate the performance. The proposed method successfully accelerates the performance by up-to 28.4% than previous works.

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

I. 서 론

사물인터넷은 저전력의 경량 디바이스에서 수집된 센서 데이터가 인터넷을 통해 연결된 고성능의 서버 컴퓨터에 전달되고 서버 컴퓨터에서는 해당 데이터를 분석하여 새로운 정보를 도출해 내는 서비스를 의미한다. 이처럼 사물인터넷은 이기종의 컴퓨터들이 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 어렵다. 따라서 최근에는 이러한 문제점을 해결하기 위해 사물인터넷에서 사용되는 경량 디바이스 상에서도 효율적으로 암호화 연산

Received(06. 07. 2017), Modified(1st: 07. 12. 2017, 2nd: 07. 27. 2017), Accepted(07. 31. 2017)
* 본 연구는 한성대학교 교내학술연구비 지원과제임.
† 주저자, hwajeong84@gmail.com
‡ 교신저자, hwajeong84@gmail.com(Corresponding author)

을 수행하는 방법에 대한 연구가 지속적으로 되어왔다. 하지만 사물인터넷 서비스에 사용되는 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 대표적인 부채널 공격에 대해 확인해 보도록 한다. 3장에서는 이전에 타겟 프로세서 상에서 구현된 기법들의 문제점에 대해 확인해보고 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

ARM Cortex-M3 프로세서는 산업체에서 많이 사용되는 32 비트 프로세서이다. 저전력으로 동작할 뿐 아니라 가격이 저가이며 실시간 임베디드 응용 프로그램에 적합한 특성을 가진다. 프로세서는 ARMv7-M 구조를 따르며 3단계 파이프 라인을 지원한다. 또한 Thumb-1과 Thumb-2 명령어 셋을 모두 지원한다. Thumb-1의 경우 기존 32-bit 크기의 ARM 명령어와는 달리 16-bit로 명령어를 정의하는 방안으로써 코드크기를 획기적으로 줄임과 동시에 기존 ARM의 성능을 유지하는 장점을 가진다. Thumb-2의 경우에는 ARM과 Thumb-1 명령어를 혼용해서 사용함으로써 성능과 코드 크기의 적절한 절충점을 찾은 명령어 셋으로 볼 수 있다. 이를 이용하면 ARM의 성능을 Thumb-1의 코드 크기로 달성하는 것이 가능하다. 암호화 연산에 사용되는 대표적인 명령어 셋은 [표 1]과 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 부채널 공격

부채널 공격은 암호화 연산이 수행되는 컴퓨팅 환경의 특징을 이용하여 암호화에 사용된 비밀정보를 추출해 내는 기법을 의미한다. 부채널 공격 기법은 연산 속도의 차이점을 분석하여 현재 암호화 키를 찾아내는 실행 속도 분석 기법과 연산 시 소모되는 전력이 다르다는 점을 이용하여 비밀키를 확인하는 전력 소모 분석 기법으로 나누어 생각해 볼 수 있다. 해당 부채널 공격 기법은 기본적으로 암호화 구현 시에 방어되어야 하는 공격 기법으로써 많은 고속 구현 논문에서도 기본적으로 해당 부채널 공격에 대한 대응방안이 제시되어야 한다.

2.2.1 실행 속도 분석

실행 속도 분석은 프로세서 상에서 연산을 수행하는 속도가 암호화 키 정보를 포함하고 있을 때 공격자가 이를 분석해 내는 기술을 의미한다. 크게 프로세서 특성에 따라 연산 속도가 다르게 나오는 경우와 암호화 알고리즘의 특성에 따라 연산 속도가 다르게 나오는 경우로 나누어 생각해 볼 수 있다. 먼저 프로세서에 따라 연산속도가 다르게 나오는 경우는 각각의 프로세서가 가지는 컴퓨터 구조 (파이프 라인, 캐시, 명령어 셋)의 차이점에 의해 발생하게 된다. 따라서 암호학적으로 안전한 암호 구현의 경우에도 특

정한 경량 디바이스 상에서는 공격자에 의해 비밀키가 노출되는 경우가 발생하게 된다. 두 번째로 암호학적인 취약점으로 인해 발생하는 실행속도 취약점이 있다. 가장 간단한 예시로 RSA의 지수승 연산의 경우 가장 기본적인 지수승 연산을 곱셈과 제곱 연산으로 수행하게 될 경우 곱셈의 횟수가 비밀키 값에 따라 상이해 지는 문제점이 있으며 이는 연산 속도를 명확히 나타내는 지표로 동작하게 된다. 이러한 문제점을 해결하기 위한 방어 기법으로는 Montgomery Ladder 알고리즘으로써 비밀키값과 상관없이 언제나 동일한 횟수의 곱셈과 제곱연산을 수행하도록 한다.

2.2.2 전력 소모 분석

암호화 연산이 수행되는 경량 디바이스는 해당 암호화 연산을 수행하기 위해 전력을 소모하게 된다. 여기서 전력은 프로세서에 인가되는 에너지로써 특정한 연산을 수행 시 연산자마다 소모되는 에너지는 서로 상이하다. 따라서 특정 프로세서 상에서 특정 연산자가 소모하는 에너지의 특성을 분석하여 암호화 연산에 사용되는 비밀키값을 추출해내는 것이 가능하다. 이를 방지하기 위해 비밀키값에 의존적이지 않은 전력 소모 패턴을 만들어 내는 구현이 필수적이다.

III. 기존 기법의 문제점 제시 및 새로운 기법 제안

ARM 프로세서 상에서는 두 개의 32 비트 입력값을 곱하여 64 비트의 출력값을 도출하는 연산자 UMULL를 지원한다. 해당 연산자는 하나의 어셈블리 명령어를 통해 64 비트 곱셈을 수행하기 때문에 복잡한 곱셈 연산에 효율적이다. 하지만 해당 곱셈 연산은 특정 ARM 프로세서 상에서는 입력인자의 크기에 따라 연산 수행속도가 달라진다. 특히 ARM7TDMI 구조를 가지는 프로세서 상에서 연산 속도가 일정하지 않은 취약점을 이용하여 암호 연산의 비밀키를 공격하는 방안이 제시되었다 [1]. 해당 논문에서는 AES 라운드 함수 중 MixColumns의 경우 ARM7TDMI에서 제공하는 UMULL 명령어 셋을 이용하여 Polynomial 연산을 수행할 경우 입력 인자의 크기에 따라 연산이 수행되는 속도가 달라짐을 확인하고 이를 이용하여 비밀값을 추출하였다. 이는 하드웨어 구현 수준에서 최적화 기법을 적용한 점을 이용한 부채널 공격 기법이라고 볼 수 있다. 해당 명령어 셋의 취약점은 가장 최신의 임베디드 프로세서인 ARM Cortex-M3 상에서도 [표 1]에서와 같이 여전히 발견되고 있다. 2015년 Wouter de Groot에 의해 발표된 논문에서는 ARM Cortex-M3 상에서 UMULL의 연산 속도의 차이를 [표 2]와 같이 정리하여 나타내고 있다 [2]. 표에서와 같이 UMULL 연산자의 경우 연산에 사용되는 입력값에 따라 연산 수행속도가 상이하게 나타나게 된다. 예를 들어 두 개의 입력값이 모두 16비트 이하인 경우 3 클록에 연산이 수행되지만 두 개의 입력값이 모두 16비트를 초과하는 경우에는 5 클록이 연산에 사용되게 된다.

Table 2. Clock cycles for UMULL instruction in Cortex-M3 processors [2]

Condition	Cycles
Either operand zero	3
Both operands ≤16-bit	3
One operand ≤16-bit & other operand 〉16-bit	4
Both operand 〉16-bit	5

ARM7TDMI 상에서의 불규칙한 연산 속도를 보완하기 위해서 입력값을 변경하여 곱셈 연산을 수행하는 방안이 제시되었다 [3]. [표 3]에서와 같이 현재 입력 레지스터의 전체 길이를 확인한 이후에 R1 레지스터가 16 비트 이하인 경우에는 해당 레지스터의 25 번째 비트를 세팅해 줌으로써 인위적인 16 비트를 만들어 주고 해당 값을 추후에 변경해 주는 것이다.

[표 3]에서 접미 연산인 EQ의 경우 특정 경우를 만족하는 경우에만 해당 연산이 수행되도록 하고 그렇지 않은 경우에는 no operation (NOP)을 수행함으로써 원하는 연산 결과를 동일한 수행 시간에 얻을 수 있는 특징이 있다. 하지만 NOP 연산의 경우

Table 3. Constant time implementation of UMULL [3]

```
Input: operands R0, R1
Output: results R2, R3
1: TST R1, #0xFF000000
2: EOREQ R1, R1, #0x01000000
3: UMULL R2, R3, R0, R1
4: EOREQ R1, R1, #0x01000000
5: SUBEQ R3, R3, R0, LSR #8
6: SUBEQS R2, R2, R0, LSL #24
7: SBC R3, R3, #0x00000000
```

마이크로프로세서의 연산자 유닛 (ALU)가 동작하지 않기 때문에 연산에 소모되는 전력량이 상이할 수 있다. 이러한 문제점을 이용한 공격 기법은 [4]에 나타나 있다. 해당 논문에서는 NOP 연산과 XOR 연산을 수행시키고 두 개의 연산자가 각각 소모하는 전력량을 측정하였다. 그 결과 NOP 연산의 경우 육안으로 확인이 가능할 정도로 전력소모가 다름을 확인할 수 있었다. 이는 [표 3]에 나타난 접미 연산인 EQ의 경우에도 공격자가 해당 연산 수행 시의 전력을 분석하여 그 결과를 도출할 수 있음을 의미한다. 만약 인자가 24-비트 미만인 경우 Step 2, 4, 5, 6에서는 EOR/SUB 연산이 수행되게 된다. 반대로 인자가 24-비트 이상인 경우에는 Step 2, 4, 5, 6, 에서 NOP 연산이 수행되게 된다. 따라서 두 경우에 소비되는 전력 패턴이 상이하게 나타나게 된다.

또한 [표 2]에서와 같이 Cortex-M3 보드 상에서는 두 개의 인자 상태 모두에서 영향을 받게 된다. 따라서 [표 3]에서와 같이 하나의 인자를 32-비트 인자로 만들 경우 4 클록 혹은 5 클록으로 변동적인 연산 속도를 가지게 됨으로써 Timing 공격에 취약함을 확인할 수 있다. 예를 들어 두 인자가 모두 16-비트 이하인 경우와 두 인자가 모두 16-비트 초과인 경우를 비교해 보면 전자는 4 클록 그리고 후자는 5 클록이 수행됨을 확인할 수 있다. 이는 [3]의 기법을 사용하더라도 ARMv6-M 프로세서 상에서는 Constant timing 구현이 불가능함을 의미한다.

따라서 안전하게 곱셈연산을 ARM Cortex-M3 상에서 수행하기 위해서는 UMULL 연산자를 사용하지 않는 기법 적용이 필요하다. 적용 가능한 방안으로는 [5]에서와 같이 16 비트 입력값에 대한 32 비트 곱셈 연산을 수행하는 MUL 연산자를 활용한 방안을 생각해 볼 수 있다. 해당 연산자를 이용하면 일정한 연산 속도를 보장할 수 있는 장점이 있으며 자세한 사항은 [표 4]에 나타나 있다. 해당 곱셈 연산은 먼저 32 비트 안에 할당된 입력값을 16 비트 단위로 나누어 저장하게 된다. 그리고 총 4번의 곱셈 연산을 통해 32 비트 연산을 완성하게 된다. 해당 연산 결과값은 결과값 레지스터에 차례로 저장되며 연산을 마무리하게 된다.

하지만 해당 구현의 경우 Cortex-M0+ 상에서의 구현으로써 Cortex-M3와 비교 시 크게 두 가지 차이점을 가지고 있다. 첫 번째로 Cortex-M0+의 경우 Thumb 연산자만을 지원하게 된다. 해당 연산자의 경우 R0-R7까지의 레지스터만을 연산에 활용할 수 있고 나머지 레지스터의 경우에는 저장 용도로 밖에 사용할 수 없다. 반면에 Cortex-M3의 경우 Thumb2를 제공하여 레지스터 활용의 제한이 없다. 두 번째로 Barrel Shifter가 Cortex-M3 상에서만 가능하다. 해당 기능을 이용하게 될 경우 레지스터에 대한 shift 혹은 rotation 연산을 클록 소비없이 구현 가능하다. 자세한 구현 사항은 [표 5]에 나타나 있다. Step 1, 2에서는 레지스터 R8 그리고 R9에서 바로 연산자를 불러오도록 한다. Step 7에서는 입력과 출력 레지스터를 다르게 선택가능하다. 이를 통해 연산 인자를 재사용하는 것이 가능하도록 하여 연산자를 복사하는 불필요한 연산을 회피하도록 하였다. 또한 Step 13, 16, 19에서와 같이 세 번째

Table 4. Previous Multiply Accumulate operation for multiplication [5]

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: ADDS 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

Table 5. Proposed Multiply Accumulate operation for multiplication

```
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
```

인자에 바로 #0을 넣어서 계산하는 것이 가능하다. 이를 통해 하나의 레지스터 소비를 줄임과 동시에 레지스터 복사 연산을 줄일 수 있다. 마지막으로 Step 14, 15, 17, 18에서와 같이 세 번째 인자를 연산에 사용할 때 Barrel Shifter를 활용하여 계산하는 것이 가능하다. 이를 통해 결과값을 미리 shift 해 놓을 필요가 없어지는 장점을 가진다.

또한 전체적인 구현 상에서도 다양한 최적화 기법을 적용하였다. 먼저 함수를 호출하는 시기에 PUSH/POP 연산자 안에 들어가는 인자를 R4부터 R11까지 한 번에 선택하여 연산 가능하다. 또한 이전에 연산을 하기 위해 상위 레지스터 (R8~R12)에 저장된 값을 하위 레지스터 (R0-R7)으로 불러오는 과정을 생략하고 바로 상위 레지스터에 대한 연산이 가능하도록 하였다. 이를 통해 레지스터의 활용도를 높임과 동시에 필요한 연산도 줄이는 효과를 얻을 수 있다.

본 논문에서 제안한 MAC 연산 기법은 32-비트 연산을 수행하게 된다. 이를 위해서는 총 4번의 16-비트 곱셈 연산 (Step 7-10)이 수행되어야 하며 이를 위해 각각의 입력 인자는 16-비트 단위로 나뉘는 과정 (Step 3-6)이 필요하다. 또한 32-비트 단위의 곱셈의 결과값을 기존 결과값에 업데이트해주기 위해서는 3번의 덧셈이 최소 필요하다. 따라서 제안하는 방안은 이론적으로도 가장 최적화된 연산 루틴을 가진다고 할 수 있다.

IV. 성능 평가 및 분석

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.

[표 6]에서는 Micro-ECC와 본 논문에서 제안한 기법 Multiply Accumulate 연산을 비교하여 나타내고 있다. 이전 구현과 비교하여 본 논문의 결과는 MOV와 LSL/LSR 연산 횟수를 각각 4회씩 줄여서 연산 속도를 향상 시켰다. 해당 연산자에 대한 최적화가 가능한 이유는 ARM Cortex-M3에서 제공하는 Thumb2 명령어 셋이 보다 자유롭게 레지스터를 활용할 수 있도록 해 줄뿐 아니라, barrel shifter 기능을 통해 다른 연산과 LSL/LSR을 통합하여 연산하는 것이 가능하기 때문이다.

본 논문에서 제시한 MAC 기법의 성능을 확인하기 위해 다양한 길이에 대한 곱셈 연산을 수행해 보았다. MAC 연산을 통해 다양한 길이에 대한 곱셈을 구성하는 방식은 Product-Scanning 기법을 적용하여 가능하다 [6]. Product-Scanning 기법은

Table 6. Comparison of execution timings (in clock cycle) for Multiply Accumulate operation

	Micro-ECC [5]	This work
ADD/ADC	9	9
MOV/UXTH	6	2
LDR	2	2
MUL	4	4
LSL/LSR	6	2

동일한 열에 위치하는 곱셈에 대해 먼저 연산을 수행함으로써 중간 결과값을 저장하기 위해 필요한 레지스터의 개수를 최적화할 수 있는 장점을 가진다. [표 7]에는 인자의 길이를 달리하며 연산 수행속도를 확인한 결과물이며 해당 실험에 사용한 코드는 다음 주소에서 확인가능하다[1].

Micro-ECC의 MAC 기법과 제안한 MAC 기법을 이용하여 ECC 혹은 RSA에서 사용하는 인자 길이에 대한 연산 결과를 확인해 보면 Micro-ECC의 경우 256 비트 연산을 위해서는 약 3,528 클락이 소모되며 2048 비트의 경우에는 약 203,700 클락이 소모됨을 확인할 수 있다. 반면에 제안하는 기법의 경우에는 256 비트 연산의 경우 약 2,688 클락이 소모되며 2048 비트의 경우에는 약 145,908 클락이 소모됨을 확인할 수 있다. 따라서 해당 결과를 통해 확인해 볼 때 기존 기법에 비해 제안하는 기법은 256 비트의 경우 23.8% 그리고 2048 비트의 경우에는 28.4%의 성능 향상이 나타남을 확인할 수 있다. 해당 구현 결과는 항상 일정한 클락 안에 연산이 수행되는 장점을 가지며 이는 기존의 UMULL을 통한 구현이 가지는 부채널 보안 취약점을 해결할 수 있다.

Table 7. Comparison of execution timings (in 1000 clock cycle) for multi-precision multiplication

Method	Operand length (bit)			
	256	512	1024	2048
Micro-ECC [5]	3.5	13.3	51.7	203.7
This work	2.7	9.6	37.1	145.9

V. 결 론

본 논문에서는 사물인터넷을 위한 경량 디바이스인 32-비트 ARM Cortex-M3 상에서 부채널 공격에 안전한 곱셈기법에 대해 확인해 보았다. 먼저 기존의 곱셈기법에 대해 살펴보고 해당 기법이 경량 디바이스 상에서 가지는 문제점을 확인해 보았다. 해당 문제점을 해결하고 연산 성능을 향상시키기 위해 본 논문에서는 새로운 곱셈 구현 기법을 제안하였다. 해당 기법을 사용할 경우 기존 기법에 비해 2048 비트 곱셈 연산의 경우 약 28.4%의 성능이 개선되며 이를 이용하여 공개키 암호화의 성능 개선도 가능하다. 추후 연구로는 해당 기법을 적용하여 공개키 암호화를 구현해 보는 것과 이에 따른 성능을 평가하는 것이다. 또한 [4, 7]에서는 동일한 명령어를 통해 연산을 수행하는 경우에도 사용되는 인자의 Hamming Weight 값에 따라 전력소모가 다르게 나타나는 특이점을 이용한 CPA (Correlation Power Analysis) 가 수행되었다. 이와 더불어 선택 평문을 이용하여 전력 소모량의 차이를 구분하는 SPA 공격도 수행되었다 [8, 9]. 즉 본 논문에서 제안한 기법의 경우에는 기본적인 Timing 공격과 SPA (Simple Power Analysis) 에는 안전하지만 CPA와 선택 평문에 대한 SPA 공격 모델에는 취약한 면모를 가진다. 따라서 이를 해결하기 위해 입력 인자값을 난수화하거나 마스킹을 하는 방안을 확인해 보도록 하겠다.

1) https://goo.gl/LTbLHB

References

[1] Großschädl, J., Oswald, E., Page, D., & Tunstall, M, "Side-channel analysis of cryptographic software via early-terminating multiplications," *In International Conference on Information Security and Cryptology*, pp. 176-192, 2009.

[2] de Groot, W, "A Performance Study of X25519 on Cortex-M3 and M4," Master thesis in Eindhoven University of Technology, 2015.

[3] Hamouda, F. B, "Exploration of efficiency and side-channel security of different implementations of RSA," 2011.

[4] Seo, H., Chen, C. N., Liu, Z., Nogami, Y., Park, T., Choi, J., & Kim, H., "Secure Binary Field Multiplication," *In International Workshop on Information Security Applications*, pp. 161-173, 2015.

[5] K. MacKay, "ECDH and ECDSA for 8-bit, 32-bit, and 64-bit processors," available for download at https://github.com/kmackay/micro-ecc, 2017.

[6] Comba, P. G., "Exponentiation cryptosystems on the IBM PC," *IBM systems journal*, vol. 29, no. 4, pp. 526-538, 1990.

〔7〕 Chen, C. N., "Memory address side-channel analysis on exponentiation," *International Conference on Information Security and Cryptology*, pp. 421-432, 2014.
〔8〕 Yen, S. M., Lien, W. C., Moon, S. J., Ha, J., "Power analysis by exploiting chosen message and internal collisions-vulnerability of checking mechanism for RSA-decryption," *In Mycrypt'05*, vol. 3715, pp. 183-1956, 2005.
〔9〕 Miyamoto, A., Homma, N., Aoki, T., Satoh, A., "Chosen-message SPA attacks against FPGA-based RSA hardware implementations," *In Field Programmable Logic and Applications, 2008*, pp. 35-40, 2008.

〈저자소개〉

서 화 정 (Hwa-jeong Seo) 종신회원
2010년 2월: 부산대학교 컴퓨터공학과 학사 졸업
2012년 2월: 부산대학교 컴퓨터공학과 석사 졸업
2016년 2월: 부산대학교 컴퓨터공학과 박사 졸업
2015년 4월~5월: 난양공대 인턴쉽
2016년 1월~2017년 3월: 싱가포르 과학기술청 연구원
2017년 4월~현재: 한성대학교 조교수
〈관심분야〉 정보보호, 암호화 구현, IoT