정 보 보 호 학 회 지
제 31 권 제1호, 2021. 2

안전한 부채널 공격 내성을 위한 Constant Timing 구현 동향

김 현 준*, 박 재 훈*, 심 민 주*, 서 화 정**

요 약

암호화 알고리즘은 수학적 안전성 확보가 중요하기 때문에 이론적인 측면에서의 정보 유출은 불가능에 가깝도록 설계된다. 하지만 암호화 알고리즘을 수행하는 컴퓨터 상에서 발생하는 부가적인 정보를 수집 및 분석하게 될 경우 안전한 암호화 알고리즘을 사용한 경우라 할지라도 비밀 정보가 쉽게 유출될 수 있는 가능성을 가지고 있다. 많은 부가적인 정보 중에서도 보다 직관적인 정보에 해당하는 시간 정보는 암호화 해킹 분야에서 많이 활용되고 있다. 본 고에서는 시간 정보를 활용한 부채널 공격기법에 대해 확인해 보며 이를 방어하기 위한 일정시간 암호화 구현 기법 동향에 대해 확인해 보도록 한다.

Ⅰ. 서 론

1970년도에 IBM에 의해 개발된 대표적인 대칭키 암호 DES (Data Encryption Standard)는 56-비트 키 길이와 64-비트 블록크기를 가지는 암호화 알고리즘으로써 고전 데이터 암호화 기술로써 활발히 활용되었다[1]. DES는 substitution과 permutation 과정을 거치며 데이터를 암호화해나가는 과정을 거친다. 소프트웨어 구현에서 substitution 과정의 경우 사전 테이블 접근을 통해 연산을 효율적으로 구현하는 것이 가능하다. 하지만 permutation과 같은 연산은 비트 단위 연산으로 구성되어 있어 하드웨어에는 적합하지만 바이트, 워드, 혹은 더블 워드를 기본 형식으로 가지는 소프트웨어 상에서 효과적으로 구현하는 것은 어렵다. 이를 효율적으로 구현하기 위해 소프트웨어 상에서는 하나의 워드 상에 비트들을 묶어서 연산 효율성을 도모하는 비트슬라이싱 기법이 제안되었다[2]. 비트 슬라이싱 기법은 단 한번의 암호화에는 적용이 어려우며 다수의 암호화가 필요한 경우에 보다 효율적인 적용이 가능하다. 하지만 한 번의 암호화가 16-바이트의 정보만을 암호화 할 수 있으며 5G 통신을 통해 기가바이트 통신이 가능한 지금 상황에서는 다수의 암호화가 많이 사용되고 있다. 특히 TLS (Transport Layer Security)와 VPN (Virtual Private Network)에 많이 활용되는 GCM (Galois/Counter Mode)의 경우 기본적으로 다수의 데이터에 대한 병렬 암호화가 가능한 CTR (Counter) 운영 모드를 사용하기 때문에 해당 구현 기법은 효용성이 높다.

비록 소프트웨어 상에서 하드웨어와 같은 비트 형식으로 데이터 형식을 변환함으로써 높은 연산 성능 달성을 목표로 하였던 비트슬라이싱 기법이지만 현재에 와서는 constant timing으로 연산이 가능한 구현 기법으로써 많이 활용되고 있다. 본 고에서는 variable timing 연산에 대한 공격 가능성과 더불어 constant timing을 달성하기 위한 비트슬라이싱 관련 최신 구현에 대해 확인해 보도록 한다.

본 고의 구성은 다음과 같다. 2장에서는 variable timing에 대한 공격에 대해 확인해 보도록 한다. 3장에서는 constant timing으로 암호 구현을 효율적으로 할 수 있는 하드웨어 암호 가속기와 비트슬라이싱을 구현 기법에 대해 확인해 보도록 한다. 4장에서는 본고의 결론내린다.

Ⅱ. Variable Timing에 대한 공격

연산 시간에 있어서 variable timing이라는 것은 해

본 연구는 2020년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2020R1F1A1048478).
* 한성대학교 IT융합공학부 (대학원생, khj930704@gmail.com; 대학원생, p9595jh@gmail.com; 학부생, minjoos9797@gmail.com)
** 한성대학교 IT융합공학부 (조교수, hwajeong84@gmail.com)

당 연산이 수행될 때마다 상이한 연산 시간이 소모되는 것을 의미한다. 이러한 연산 시간의 상이함은 컴퓨터가 동일한 연산을 수행하는 경우 미리 이를 예측하여 연산 성능을 최적화하는 부분에서 발생하는 경우이다. 컴퓨터 상에서 가장 많은 연산 저하가 발생하는 것은 메모리 접근이다. 메모리는 연산이 필요한 대량의 정보가 저장된 공간으로써 연산 시 해당 정보가 필요할 때마다 접근하여 정보를 CPU 내부의 레지스터로 가지고 온다. 그리고 연산이 끝난 경우 일반적으로 메모리에 다시 값을 저장하는 수순으로 진행된다. 하지만 자주 접근하는 정보의 경우 메모리에 다시 저장하지 않고 CPU에 근접한 캐시에 정보를 저장함으로써 연산 효율성을 증가시킬 수 있다. 이러한 연산 최적화 기법은 일반적인 연산의 경우 큰 문제를 발생시키지 않는다. 하지만 캐시에 저장하는 정보가 암호화에서 사용하는 비밀정보와 연관된 경우라면 접근 속도의 상이함을 이용하여 비밀정보를 유출하는 것이 가능하다. 2010년 대표적인 암호 라이브러리인 OpenSSL 상에 구현된 AES 암호에 대한 캐시 공격이 모든 CPU 상 (Intel Pentium III, AMD Athlon, PowerPC RS64 IV, Sun UltraSPARC III)에서 가능함을 증명하였다[3]. 해당 공격은 32-비트 이상의 프로세서 상에서의 AES 최적화 구현에 많이 활용되고 있는 T Table 사용 시 발생하는 variable timing을 분석한 공격이다. 기본적으로 AES는 add-round-key, substitution, shiftrow, 그리고 mixcolumn 연산으로 구현된다. 여기에 더불어 AES는 기본적인 연산 단위가 8-비트라는 특징을 가지고 있다. 만약 32-비트 프로세서 상에서 AES를 구현하게 될 경우 32-비트에서 8-비트를 뺀 24-비트 만큼의 연산 범위를 활용할 수 없게 되는 문제점을 가지고 있다. 따라서 32-비트 프로세서 상에서 AES 연산을 효율적으로 구현하는 것이 가능하도록 AES 제안서에는 substitution, shiftrow, 그리고 mixcolumn 연산을 합쳐서 계산하는 T table 구현 기법을 제안하고 있다[4]. 해당 구현은 8-비트 입력이 3가지 연산을 수행하며 32-비트까지 확장되는 값을 사전에 계산한다. 해당 사전 계산된 값은 암호화 연산시 사전 테이블 형식으로 연산 최적화를 수행하게 된다. AES 암호화의 경우 가장 처음에 add-round-key를 통해 라운드 키와 평문을 XOR하며 결과값을 가지고 T table 접근을 수행하게 된다. 만약 특정 index값이 사전에 사용되었다면 해당 값에 대한 접근 속도는 점차 빨라지는 특징을 가지며 이를 분석하게 될 경우 킷값을 추출하는 것이 가능하다. 따라서 variable timing을 통한 정보 유출을 방지하기 위해서는 constant timing으로 암호화를 구현하는 것이 중요하며 이에 대해서는 3장에서 자세히 확인해 보도록 한다.

Ⅲ. Constant Timing 암호 구현

암호는 수학적으로 설계되지만 이를 실제로 컴퓨터 상에서 동작시키기 위해서는 프로그래밍 언어로 수학적 수식을 작성해야 한다. 기본적으로 프로그래밍은 논리적인 연산의 연속이며 variable timing이 발생하는 주된 원인은 분기문과 예측실행으로 볼 수 있다.

먼저 프로그래밍에서 분기문은 IF-ELSE문이 대표적이다. 특정한 조건인지 아닌지를 판단하며 이에 따라 상이한 연산을 수행하게 된다. 만약 특정한 조건을 결정짓는 요소에 비밀정보가 포함되어 있다면 정보가 유출되게 된다.

예측 실행의 예시로는 앞에서 살펴본바와 같이 미래에 사용될 값을 메모리가 아닌 캐시에 저장해두고 사용하는 것이다. 이는 데이터가 아닌 명령어에도 적용되는 사항이며 주기적으로 반복 실행되는 루틴이 있다면 이를 미리 예측하고 사전에 계산을 하게 된다.

이처럼 일반적인 논리에 대한 연산 최적화는 연산 지연을 대폭 감소시킴으로써 성능을 향상시킬 수 있다는 측면에서 큰 이점을 가진다. 하지만 암호 구현의 경우 비밀 정보의 유출로 이어질 수 있는 만큼 예외적인 상황에 대한 최적화가 적용되지 않도록 코드를 구현하는 constant timing이 중요하다.

3.1. 암호 확장 명령어 기반 Constant Timing 구현

Constant timing의 문제점을 해결하기 위한 노력으로는 하드웨어적 그리고 소프트웨어적 관점으로 진행되고 있다. 하드웨어적으로는 Intel과 ARM에서 제공하는 AES 하드웨어 가속기가 있다[5]. 해당 가속기는 프로그래밍 단계에서 [표 1]에 명시된 바와 같이 특정한 암호 확장 명령어를 통해 활용가능하다. 특히 AES 연산의 대표적인 운영모드인 GCM을 효과적으로 수행하기 위해서는 binary field 곱셈이 필요하게 되는데 이를 효과적으로 지원하기 위한 명령어 PCLMULQDQ와 VMULL을 제공하고 있다[6]. 만약 해당 명령어없이 binary field 곱셈을 수행하기 위해서는 비트 단위로

〔표 1〕 Intel과 ARM 상에서 AES와 GCM 가속화를 위한 암호 확장 명령어

Intel	ARM
AESENC, AESENCLAST, AESDEC, AESDELAST, AESIMC, AESKEYGENASSIST, PCLMULQDQ	AESE, AESMC, AESD, AESIMC, VMULL

연산을 수행하거나 사전테이블 형식을 취해야 한다. 여기서 비트 단위 구현은 연산 속도가 매우 느려지는 단점이 있고 사전테이블의 경우에는 캐시 타이밍 공격에 취약할 수 있는 문제점을 가지고 있다[7]. 따라서 하드웨어적 확장 연산자 지원을 통해 constant timing을 만족하면서 동시에 고속 암호화 구현이 가능하다.

3.2. Bitslicing 기반 Constant Timing 구현

하지만 모든 경우에 하드웨어적 지원을 받을 수 있는 것은 아니다. 하드웨어 칩에서는 원가를 절감하기 위해 암호화 확장 명령어를 탑재하지 않는 경우도 있다[8]. 대표적인 사물인터넷 플랫폼인 Raspberry Pi4의 경우 최신 64-비트 ARMv8 프로세서를 탑재하고 있지만 암호화 확장 연산자는 따로 제공하고 있지 않다. 이와는 별개로 하드웨어적 장치는 추가적으로 발견되는 보안 취약점에 유연하게 대응할 수 없다는 특징으로 인해 소프트웨어적 암호화 연산에 대한 고려는 필요하다.

소프트웨어 상에서 타이밍 정보 유출을 최소화하는 방법 중 가장 폭넓게 활용되고 있는 접근법은 앞에서 간략히 소개한 bitslicing 기법이다. DES에 적용되었던 bitslicing 기법은 AES에서도 사용 가능함을 CANS'06에서 증명하였다[9]. AES를 bitslicing으로 구현하는 방법은 크게 packing, 암호화, unpacking으로 나누어 생각해 볼 수 있다. packing과 unpacking은 다른 일반 구현에 없는 bitslicing 구현만의 독특한 연산이다.

Packing은 데이터를 워드 단위 표기법에서 비트 단위 표기법으로 변환하는 과정을 의미하며 unpacking은 데이터를 비트 단위 표기법을 워드 단위 표기법으로 변환하는 과정을 의미한다. 해당 과정을 직접 수행하게 될 경우 비트 단위로 연산을 수행해야 하며 이는 여전히 워드 단위 소프트웨어 구현에서는 높은 연산 부하를 발생시키게 된다. 이를 효과적으로 구현하는 기법으로는 수식 (1)에 나타나 있는SWAPMOVE가 있다[10].

$$\begin{aligned} &SWAPMOVE(A,B,M,N):\\ &T=((A\gg N)\oplus B)\&\mathrm{M}\\ &B=B\ \oplus T\\ &A=A\oplus(T\ll N) \end{aligned} \tag{1}$$

SWAPMOVE 기법은 워드단위 데이터 표현을 비트 단위 데이터 표현으로 변경해 주기 위해 동일한 비트 위치에 따라 데이터를 묶어서 이동시켜주고 마스킹 값(M)을 통해 일정한 비트를 추출하는 방식으로 진행된다. SWAPMOVE를 통해 bitslicing에 적합한 데이터 형식이 완성되고 난 이후에는 AES 암호화 연산이 수행되게 된다. 그리고 마지막으로 초기에 수행한 SWAPMOVE의 역연산을 수행하여 다시 일반 워드 단위 데이터 표기 형식으로 변환해 주게 된다.

Bitslicing을 통해 암호화 과정을 수행하는 방법은 기존 AES의 add-round-key, substitution, shiftrow, 그리고 mixcolumn 연산을 선형 연산자들로 모두 변환하여 수행하는 것이다. add-round-key의 경우 라운드 키와 평문을 XOR 연산을 해주는 것이며 이는 기존의 구현 기법과 크게 다르지 않다.

Substitution 연산의 경우 비선형 연산자로써 $GF(2^8)$ 상에서의 역치와 아핀 변환으로 구성된다. 일반적으로 소프트웨어 상에서 substitution 연산의 경우 한 번의 8-비트 룩업 테이블 접근 방식을 통해 결과값을 도출하는 것이 가능하다. 하지만 룩업 테이블 접근 방식은 variable timing임과 동시에 bitslicing 구현에서는 적용 불가능한 문제점을 가지고 있다. 해당 역치 연산을 위해 확장 유클리디안 알고리즘을 사용할 수 있다. 하지만 해당 알고리즘 역시 variable timing이라는 문제점을 가지고 있다. 이를 해결하기 위해 페르마의 정리를 기반으로한 역치 계산 알고리즘을 활용할 수 있다. 수식 (2)와 같이 해당 값 (a)에 대한 역치 (m은 prime number)를 곱셈과 제곱 연산으로 대치할 수 있으며 이 횟수 ($a^{m-2} \bmod m$)는 항상 일정하다. 하지만 해당 구현 기법의 경우 복잡한 곱셈과 제곱 연산으로 인해 연산 속도가 저하되는 문제점을 가지고 있다.

$$a^{m-2} \equiv a^{-1} (\bmod\ m) \tag{2}$$

소프트웨어 구현에서 AES의 substitution은 완전 선

형 연산자로 대치하여 32개의 AND 게이트 그리고 83 XOR/XNOR 게이트로 효과적인 구현이 가능하다[11].

그 다음으로 shiftrow 연산은 기존 word 단위 연산자에서는 가장 많은 시간이 소모되는 부분이다. 하지만 bitslicing 형식으로 변환된 데이터 형식에서는 최상의 경우 모든 연산을 생략하는 최적화가 가능하다. 마지막으로 mixcolumn의 경우 column 기준으로 곱셈 연산을 수행하게 된다. AES의 경우 상수 (1, 2, 혹은 3)에 대한 곱셈이기 때문에 단순히 XOR 연산자만을 통해 mixcolumn을 수행하는 것이 가능하다.

위에서 제시한 바와 같이 bitslicing을 통해 기존 암호화 연산을 최적화하고 timing 정보 유출을 방지하는 것이 가능하다. 하지만 모든 암호화 기법에 bitslicing이 적용가능한 것은 아니다. 최근에 경량 암호에 많이 사용되고 있는 ARX 구조 상에서는 기본 연산자로만 구현이 가능한 특징으로 인해 항상 constant timing을 만족하는 구현이 가능하다[12].

3.3. 변형된 Bitslicing 기반 Constant Timing 구현

Bitslicing 기반 구현의 장점은 constant timing과 빠른 연산 속도이다. 하지만 연산 초기에 packing과 unpacking과 같은 부가적인 표현 형식 변환 단계가 필요하다. CHES'17에서는 경량 암호 PRESENT에 대한 bitslicing 구현 시 packing과 unpacking이 필요하지 않은 방법론을 제시하였다[13]. PRESENT 암호화는 add-round-key, substitution, 그리고 permutation 연산을 하나의 라운드로 구성하여 수행하게 된다. 여기서 substitution은 packing을 하지 않은 데이터 형식의 경우 효율적인 bitslicing 구현 기법이 존재하지 않는다. 하지만 본 구현에서는 substituion과 permutation의 순서를 변경함으로써 바로 bitslicing 구현이 가능한 구조를 확인하였다. PRESENT는 permutation 연산 시 substitution이 바로 되는 형식으로 데이터가 정렬되게 된다. 따라서 단순히 순서를 바꾸는 접근법을 통해서 packing과 unpacking 연산을 생략할 수 있는 장점을 가진다.

ICISC'20에서는 암호 설계 초반부터 bitslicing이 가능한 블록 암호 PIPO가 발표되었다[14]. 암호화의 순서의 변경 그리고 packing/unpacking이 모두 필요하지 않은 장점을 가진다. 이와 더불어 부채널 방지 기법 또한 쉽게 적용이 가능하도록 설계되었다.

CHES'20에서는 PRESENT와 유사한 암호화 구조를 가진 GIFT 블록 암호에 대한 변형된 bitslicing 구현 기법이 제시되었다[15]. GIFT의 경우 PRESENT 보다 경량화된 구조 (add-round-key 그리고 substitution)를 가진다. 대신 경량성과 보안성을 동시에 만족시키기 위해 GIFT 블록 암호에서는 permutation을 PRESENT 보다 불규칙적으로 나타나도록 설계하였다. 이러한 이유로 CHES'17에서 제시된 PRESENT 구현 기법을 GIFT에 바로 적용하는 것은 불가능하다. CHES'20에서 제안된 방식은 bitslicing의 경우 동일한 위치의 bit들을 묶어준 것과는 달리 GIFT의 permutation 연산이 효율적으로 될 수 있는 방식으로 데이터 형식을 변경하는 fixslicing을 제안하였다. Fixslicing은 특히 최신 ARM 프로세서에 탑재되어 있는 barrel-shifter 연산자를 이용할 경우 상당수의 rotation 연산을 최적화함으로써 높은 성능을 달성하였다. 해당 구현 기법은 AES 구현에도 적용가능함을 CHES'21에서 제시하였다[16]. 특히 shiftrow과 mixcolumn 연산을 fixslicing 표현법과 barrel-shifter를 통해 효과적으로 계산할 수 있음을 증명하였다.

대표적인 대칭키 운영모드인 GCM에서 요구되는 binary field 곱셈의 경우 각 비트의 상태에 따라 XOR 연산을 수행한다. 하지만 레지스터 상에 모든 데이터를 올려두고 연산을 하기에는 레지스터의 수가 부족한 한계를 가진다. 이를 극복하기 위해 동시에 연산이 필요한 비트끼리 묶어서 연산하는 기법이 제시되었다[17]. 해당 기법의 경우에 연산 초기 packing이 필요하지만 이를 감안하더라도 레지스터를 효율적으로 활용함으로써 높은 연산 속도를 달성할 수 있음을 보였다.

3.4. 벡터 연산자 기반 Constant Timing 구현

최신 Intel 그리고 ARM 프로세서 상에서는 대량의 정보를 고속으로 처리하기 위해 벡터 연산자인 Intel AVX 그리고 ARM NEON 명령어 셋을 제공하고 있다[18]. AVX는 256-비트 그리고 512-비트 단위 벡터 연산이 가능하며 NEON은 128-비트 단위 벡터 연산이 가능하다. 메모리에 대한 접근없이 한 번에 처리할 수 있는 데이터의 크기가 크기 때문에 연산 속도가 빨라짐과 동시에 캐시 공격으로부터 높은 안전성을 확보할 수 있다. 이와 더불어 경량 대칭키 암호에서 많이 활용되고 있는 4-비트 사전 테이블의 경우 벡터 연산자

(ARM의 경우 VTBL)를 통해 병렬로 한번에 연산하는 것이 가능하다.

대표적인 경량 암호 HIGHT에 대한 NEON 상의 최적 구현시에는 수식 (3)에 제시된 내부 연산자 F0와 F1를 사전 테이블로 계산함으로써 연산 성능을 최적화하는 것이 가능하다.

$$F_0(x) = x^{\lll 1} \oplus x^{\lll 2} \oplus x^{\lll 7}$$
$$F_1(x) = x^{\lll 3} \oplus x^{\lll 4} \oplus x^{\lll 6} \quad (3)$$

F0와 F1 함수의 경우 8-비트 입력값을 받아서 회전 연산과 XOR 연산을 수행하여 8-비트 출력값을 생성한다. 여기서 8-비트 입력값의 상위와 하위 4-비트씩을 나누어서 이에 해당하는 8-비트 출력값을 생성하는 것이 가능하다. 생성한 2개의 사전 테이블을 VTBL 연산자를 통해 병렬로 계산하는 것이 가능하다[19]. 이처럼 최신 프로세서에서 제공하는 벡터 연산자를 활용하게 되면 대량의 정보를 병렬로 효율적 그리고 안전하게 암호화하는 것이 가능하다.

Ⅳ. 결 론

본 고에서는 암호 구현 시 연산 최적화로 인해 발생할 수 있는 variable timing의 보안 취약점과 이를 해결하기 위한 constant timing 최신 구현 기법에 대해 확인해 보았다. Variable timing의 주된 원인은 암호화 연산에 대한 최적화 부분에서 발생하게 되며 이를 해결하는 방법은 언제나 동일한 연산을 수행하는 constant timing 구현이다. 지금까지 다양한 방법론을 적용하여 constant timing이 연구되었으며 본 고에서는 최신 구현 동향에 대해 확인해 보았다. 앞으로도 최신 프로세서 구조의 등장과 새로운 경량 암호화의 등장으로 인해 constant timing을 만족하는 암호 고속 구현에 대한 수요는 계속 증가할 것으로 보인다. 다만 constant timing이 기존 구현 기법에 비해 추가적인 연산을 포함함으로 인해 성능이 저하되는 부분을 지속적으로 개선해 나가기 위한 방법론을 모색하는 것이 필요해 보인다.

참 고 문 헌

[1] NIST, “The official document describing the DES standard,” *Technical Report*, 1999.

[2] Biham, E. “A fast new DES implementation in software,” *In International Workshop on Fast Software Encryption*, pp. 260-272, 1997.

[3] Bernstein, D. J., “Cache-timing attacks on AES,” 2005.

[4] Daemen, J., Rijmen, V., “AES proposal: Rijndael,” 1999.

[5] Gueron, S., “Intel advanced encryption standard (AES) new instructions set,” 2010.

[6] Gueron, S., Kounavis, M. E., “Intel® carry-less multiplication instruction and its usage for computing the GCM mode,” *White Paper*, 2010.

[7] Liu, Z., Seo, H., Chen, C. N., Nogami, Y., Park, T., Choi, J., Kim, H., “Secure GCM implementation on AVR,” *Discrete Applied Mathematics*, 241, pp. 58-66, 2018.

[8] Fujii, H., Rodrigues, F. C., López, J., “Fast AES Implementation Using ARMv8 ASIMD Without Cryptography Extension,” *In International Conference on Information Security and Cryptology,* pp. 84-101, 2019.

[9] Rebeiro, C., Selvakumar, D. Devi, A. S. L., “Bitslice implementation of AES,” *In International Conference on Cryptology and Network Security*, pp. 203-212, 2006.

[10] May, L., Penna, L., Clark, A., “An implementation of bitsliced DES on the pentium MMX TM processor,“ *In Australasian Conference on Information Security and Privacy*, pp. 112-122, 2000.

[11] Boyar, J., Peralta, R., “A new combinational logic minimization technique with applications to cryptology,” *In International Symposium on Experimental Algorithms,* pp. 178-189, 2010.

[12] Kwon, H., Kim, H., Choi, S. J., Jang, K., Park, J., Kim, H., Seo, H., “Compact Implementation of CHAM Block Cipher on Low-End Microcontrollers,” *In International Conference on Information Security Applications*, pp. 127-141, 2020.

[13] Reis, T. B., Aranha, D. F., López, J., “PRESENT

runs fast," In *International Conference on Cryptographic Hardware and Embedded Systems,* pp. 644-664, 2017.

[14] Kim, H., Jeon, Y., Kim, G., Kim, J., Sim, B. Y., Han, D. G., Seo, H., Kim, S., Hong, S., Sung, J., Hong, D. A, "New Method for Designing Lightweight S-Boxes with High Differential and Linear Branch Numbers, and Its Application," In *International Conference on Information Security and Cryptology,* 2020.

[15] Adomnicai, A., Najm, Z., Peyrin, T., "Fixslicing: A New GIFT Representation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020.

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

[17] Seo, S. C., Seo, H., "Highly efficient implementation of NIST-compliant Koblitz curve for 8-bit AVR-based sensor nodes," *IEEE Access*, vol. 6, pp. 67637-67652, 2018.

[18] Lomont, C., "Introduction to intel advanced vector extensions," *Intel white paper*, 2011.

[19] Seo, H., Jeong, I., Lee, J., Kim, W. H., "Compact implementations of ARX-based block ciphers on IoT processors," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 3, pp. 1-16, 2018.

〈저 자 소 개〉

김 현 준 (Hyunjun Kim)

학생회원

2019년 2월 : 한성대학교 IT응용시스템공학과 공학 학사

2019년 3월~현재 : 한성대학교 IT융합공학과 석사과정

<관심분야> 부채널 분석, 블록체인

박 재 훈 (Jaehoon Park)

학생회원

2020년 2월 : 한성대학교 IT응용시스템 공학 학사

202년 3월~현재 : 한성대학교 IT융합공학과 석사과정

<관심분야> 웹 보안

심 민 주 (Minjoo Sim)

학생회원

2017년 3월~현재 : 한성대학교 IT응용시스템공학과 재학 중

<관심분야> 부채널 분석, 정보보안

서 화 정 (Hwajeong Seo)

종신회원

2010년 2월 : 부산대학교 컴퓨터공학과 학사

2012년 2월 : 부산대학교 컴퓨터공학과 석사

2016년 1월 : 부산대학교 컴퓨터공학과 박사

2016년 1월~2017년 3월 : 싱가포르 과학기술청

2017년 4월~현재 : 한성대학교 IT 융합공학부 조교수

<관심분야> 암호구현