DOI: 10.3745/KIPSTC.2010.17C.3.233

전력 분석 공격에 안전한 효율적인 SEED 마스킹 기법

조 영 인[†]·김 희 석[†]·최 두 호[††]·한 동 국[†††]·홍 석 희[††††]·이 옥 연[†††††]

요 약

전력분석 공격이 소개되면서 다양한 대응법들이 제안되었고 그러한 대응법들 중 블록 암호의 경우, 암/복호화의 연산 도중 중간 값이 전력 측정에 의해 드러나지 않도록 하는 마스킹 기법이 잘 알려져 있다. SEED는 비선형 연산으로 32 비트 덧셈 연산과 S-box 연산을 동시에 사용하고 각 연산에 대한 마스킹 방법이 조화를 이룰 수 있도록 마스킹 형태 변환 과정이 필요하다. 본 논문에서는 SEED의 구조적 특성을 고려하여, 연산 시간이 많이 필요한 마스킹 형태 변환 횟수를 최소화 하도록 새로운 마스킹 S-box 설계법을 제안한다. 또한 마스킹 S-box 테이블을 하나만 생성하고 이것으로 나머지 마스킹 S-box 연산을 대체할 수 있는 연산식을 만들어 기존 마스킹 기법에 비해 마스킹 S-box로 인한 RAM 사용량을 절반으로 줄여 메모리 크기면에서도 효율적이도록 구성하였다.

키워드 : SEED, 마스킹, 불 마스킹, 산술 마스킹

Efficient Masking Method to Protect SEED Against Power Analysis Attack

Young In Cho[†] · HeeSeok Kim[†] · Dooho Choi[††] · Dong-Guk Han[†††] · Seokhie Hong[††††] · Okyeon Yi[†††††]

ABSTRACT

In the recent years, power attacks were widely investigated, and so various countermeasures have been proposed. In the case of block ciphers, masking methods that blind the intermediate results in the algorithm computations(encryption, decryption) are well-known. In case of SEED block cipher, it uses 32 bit arithmetic addition and S-box operations as non-linear operations. Therefore the masking type conversion operations, which require some operating time and memory, are required to satisfy the masking method of all non-linear operations. In this paper, we propose a new masked S-boxes that can minimize the number of the masking type conversion operation. Moreover we construct just one masked S-box table and propose a new formula that can compute the other masked S-box's output by using this S-box table. Therefore the memory requirements for masked S-boxes are reduced to half of the existing masking method's one.

Keywords : SEED, Masking, Boolean Masking, Arithmetic Masking

1. 서 론

최근 들어 컴퓨터를 이용한 인터넷 사용의 급증과 정보통신 환경의 변화 등으로 스마트카드와 PDA같은 모바일 산업은 빠르게 성장하고 있다. 그리고 스마트카드 수요의 증가와 더불어 이에 대한 공격 방법으로 부채널 공격(Side Channel Attack)이 소개되었다[1]. 부채널 공격이란 수학적으로 안전한 것으로 알려진 알고리즘이 구현 단계에서 부가적인 정보를 누출함으로써 이로부터 비밀 키의 값을 알아낼 수 있는 방법이다. 이러한 부채널 공격이 소개되면서 많은 암호시스템 설계자들은 효율적인 대응법들을 연구하기 시작했고, 부채널 공격 중 가장 강력한 공격인 차분 전력 분석(Differential Power Analysis, DPA)[2-4]에 대한 대응법으로는 마스킹 기법(masking method)이 활발히 연구되고 있다[3, 5-7]. 마스킹 기법은 알고리즘의 변형을 통해 고차 및 일차 차분 전력 분석을 방어하는 방법으로 노이즈 삽입(Random Noise Insertion)[10], 임의의 지연(Random Delay)[11], 임의의 클록(Unstable Clock)[12]과 같은 하드웨어적인 대응법에 비해 그 비용이 저렴하다. 따라서 단가가 저렴한 스마트 카드와 같은 장비에서는 일반적으로 가장 선호하여 사용된다.

※ 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다.
† 준 회 원 : 고려대학교 정보경영공학과 박사과정
†† 정 회 원 : 한국전자통신연구원 정보보호연구본부 선임연구원/팀장
††† 정 회 원 : 국민대학교 수학과 조교수
†††† 정 회 원 : 고려대학교 정보경영공학전문대학원 부교수
††††† 정 회 원 : 국민대학교 수학과 부교수
논문접수 : 2010년 3월 16일
수 정 일 : 1차 2010년 5월 12일
심사완료 : 2010년 5월 12일

1999년 2월, 한국정보보호진흥원에서 개발한 SEED[8]는 비선형 연산으로 32 비트 덧셈 연산과 S-box 연산을 사용한다. 본 논문에서는 SEED 알고리즘을 위한 마스킹 기법이 현재 존재하지 않으므로 일반적으로 널리 사용되는 대칭키 마스킹 기법을 그대로 적용하여 설계한 마스킹 기법을 기존 SEED 대응법이라 한다.[1] 제안하는 SEED 마스킹 기법은 SEED 알고리즘의 특성을 충분히 고려하여, 기존 SEED 대응법의 메모리 효율성 및 연산 속도측면에서 생길 수 있는 문제를 개선하도록 하였다. 즉, 기존 SEED 대응법에서는 G 함수내의 마스킹 S-box를 Boolean 마스킹된 값을 입력으로 받아 다시 Boolean 마스킹된 S-box의 출력 값을 출력할 때 32비트 덧셈 연산 이후 G 함수의 S-box에 값이 입력되기 위해서는 32비트 덧셈 연산의 출력값이 Boolean 마스킹된 상태여야 하고, 이를 위해 연산 시간이 많이 소요되는 Arithmetic 마스크에서 Boolean 마스크로의 마스킹 형태 변환 과정이 필요하게 된다. 이러한 문제점을 해결하기위해 제안하는 SEED 마스킹 기법에서는 Arithmetic 마스크에서 Boolean 마스크로의 마스킹 형태 변환 과정을 최소화할 수 있도록 마스킹 S-Box를 새롭게 설계하였다. 또한 마스킹 S-box 테이블 두 개 중 하나만 생성하여 이것으로 나머지 마스킹 S-box 연산도 수행할 수 있는 연산식을 만들어 마스킹 S-box 테이블에 소요되는 메모리를 기존 SEED 대응법에서의 절반만 사용하였다. 따라서 본 논문에서는 기존 SEED 대응법보다 메모리 효율성과 연산 속도 측면에서 모두 뛰어난 새로운 SEED 마스킹 기법을 제안한다.

본 논문의 구성은 다음과 같다 2절은 SEED 알고리즘에 대해 설명하고, 3절은 일반적인 대칭키 마스킹 기법을 SEED에 적용한 기존 SEED 대응법을 살펴본다. 본 논문에서 제안하는 마스킹 기법은 4절에서 소개한다. 마지막으로 5절에서는 기존 SEED 대응법과 제안하는 마스킹 기법의 효율성과 안전성을 비교, 분석한다.

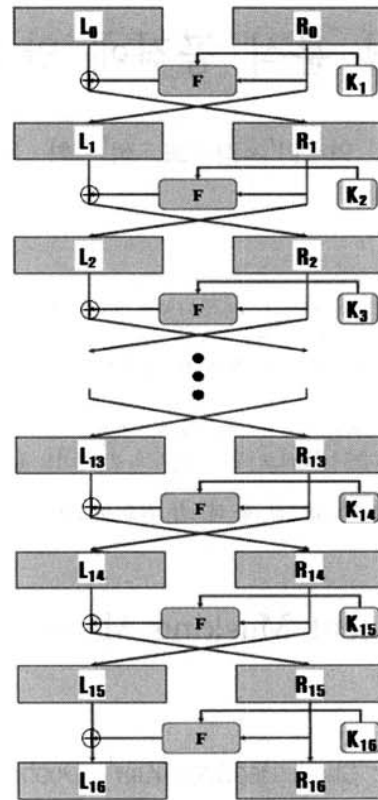
2. SEED

2.1 SEED Algorithm의 구조

SEED는 DES와 유사한 구조로, 128 비트의 비밀키를 사용하는 Feistel 구조를 가지고 있다. Feistel 구조란 각각 n/2비트인 L_0, R_0 블록으로 이루어진 n비트 평문 블록 (L_0, R_0)가 r라운드$(r \geq 1)$를 거쳐 암호문 (L_r, R_r)으로 변환되는 반복 구조이다[8].

■ 기호와 표기

- $a\,b$: 'a' 비트-wise AND 'b'
- L_i : i 라운드에서 출력된 왼쪽 메시지 블록(64 비트)
- R_i : i 라운드에서 출력된 오른쪽 메시지 블록(64 비트)



(그림 1) SEED Algorithm 구조

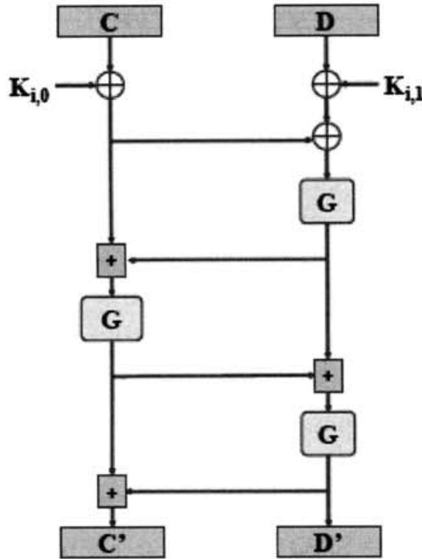
- $K_i = (K_{i,0}, K_{i,1})$: i 라운드의 라운드 키(64 비트)
- $K_{i,0}$: i 라운드 F 함수의 오른쪽 입력키(32 비트)
- $K_{i,1}$: i 라운드 F 함수의 왼쪽 입력키(32 비트)
- $X = (X_3 \| X_2 \| X_1 \| X_0)$: G 함수의 입력 값(32 비트)
- $Y = (Y_3 \| Y_2 \| Y_1 \| Y_0)$: G 함수에서 S-box(S_1, S_2)의 출력 값(32 비트)
- $Z = (Z_3 \| Z_2 \| Z_1 \| Z_0)$: G 함수의 출력 값(32 비트)
- m : 마스킹 난수

SEED는 128비트의 평문 블록과 128비트 키를 입력으로 사용하여 총 16라운드를 거쳐 128비트 암호문을 출력한다. SEED Algorithm은 크게 Feistel 구조로 되어 있는 F 함수와 라운드 키 생성함수로 나누어진다. 그리고 F 함수에는 S-box를 포함하는 G 함수가 포함되어 있고, 키 생성함수는 128비트의 키를 받아서 64비트의 16라운드의 키를 생성한다. (그림 1)은 SEED Algorithm의 구조를 도식화한 것이다.

2.2 F 함수

Feistel 구조를 갖는 블록 암호알고리즘은 F 함수의 특성에 따라 구분될 수 있다. SEED의 F 함수는 수정된 64비트 Feistel 형태로 구성된다. F 함수는 각 32비트 블록 2개 (C,D)를 입력으로 받아 32비트 블록 2개 (C',D')를 출력한다. 즉, 암호화 과정에서 64비트 블록(C,D)와 64비트 라운드 키 $K_i = (K_{i,0}; K_{i,1})$를 F 함수의 입력으로 처리하여

1) 기존 SEED 대응법으로 국내특허(출원번호 '10-2009-0065769')가 존재하나 현재 비공개 기간이므로 열람이 불가능하여 일반적으로 널리 사용되는 대칭키 마스킹 기법을 그대로 SEED에 적용함.



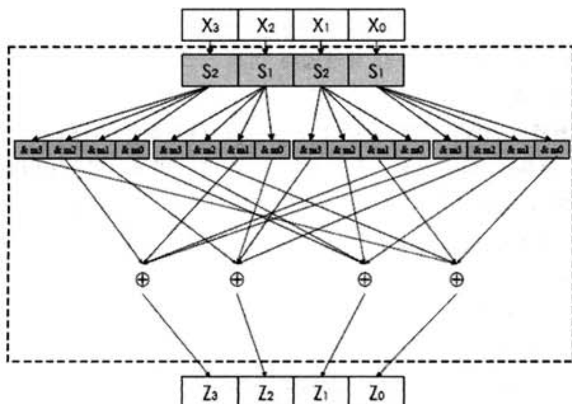
(그림 2) SEED F 함수

64비트 블록 (C',D')을 출력한다. (그림 2)는 F 함수의 구조를 도식화한 것이다.

2.3 G 함수

SEED의 F 함수는 Feistel 구조를 이루고 있으며 F 함수 내에는 S-box을 포함하고 있는 G 함수가 위치하고 있다. G 함수는 다음과 같은 연산을 통해 이루어진다. (그림 3)은 SEED G 함수의 구조를 도식화한 것이다.

$$Y_3 = S_2(X_3),\ Y_2 = S_1(X_2),\ Y_1 = S_2(X_1),\ Y_0 = S_1(X_0),$$
$$Z_3 = (Y_0 \& m_3) \oplus (Y_1 \& m_0) \oplus (Y_2 \& m_1) \oplus (Y_3 \& m_2)$$
$$Z_2 = (Y_0 \& m_2) \oplus (Y_1 \& m_3) \oplus (Y_2 \& m_0) \oplus (Y_3 \& m_1)$$
$$Z_1 = (Y_0 \& m_1) \oplus (Y_1 \& m_2) \oplus (Y_2 \& m_3) \oplus (Y_3 \& m_0)$$
$$Z_0 = (Y_0 \& m_0) \oplus (Y_1 \& m_1) \oplus (Y_2 \& m_2) \oplus (Y_3 \& m_3)$$
$$(m_0 = 0xfc,\ m_1 = 0xf3,\ m_2 = 0xcf,\ m_3 = 0x3f)$$



(그림 3) SEED G 함수

2.4 S-box

G 함수의 내부에 사용되는 비선형 S-box S_1, S_2는 다음의 식을 이용하여 생성된다.

$$S_i : Z_{2^8} \rightarrow Z_{2^8},\ S_i(x) = A^{(i)} \cdot x^{n_i} \oplus b_i$$
$$(n_1 = 247,\ n_2 = 251,\ b_1 = 159, b_2 = 56)$$

$$A^{(1)} = \begin{pmatrix} 10001010 \\ 11111110 \\ 10000101 \\ 01000010 \\ 01000101 \\ 00100001 \\ 10001000 \\ 00010100 \end{pmatrix},\ A^{(2)} = \begin{pmatrix} 01000101 \\ 10000101 \\ 11111110 \\ 00100001 \\ 10001010 \\ 10001000 \\ 01000010 \\ 00010100 \end{pmatrix}.$$

3. 기존 SEED 대응법

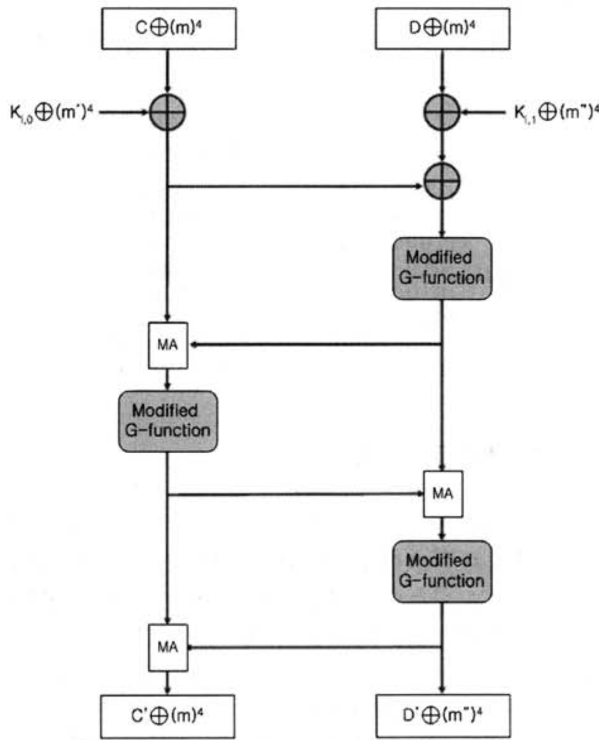
마스킹 기법 설계 시 대부분의 비용을 차지하는 것은 비선형 연산에 대한 것이다. 본 절에는 대칭키 암호 시스템에 널리 사용되는 마스킹 기법이 그대로 적용된 기존 SEED 대응법의 비선형 연산에 대한 구조를 살펴본다. Arithmetic 마스크에서 Boolean 마스크로 변환하는 연산을 AtoB, Boolean 마스크에서 Arithmetic 마스크로 변환하는 연산을 BtoA이라 한다. 이 때, BtoA은 비교적 연산 시간이 짧고 메모리 크기가 작으나 AtoB는 다수의 선형 연산을 필요로 하므로 연산 시간이나 메모리 크기면에서 모두 소요량이 많다[14].

본 논문에서는 마스킹이 적용된 32 비트 덧셈 연산과 S-box를 각각 마스킹 덧셈(Masked addition, MA), 마스킹 S-box(Masked S-box : MS_1(Masked S_1), MS_2(Masked S_2))라 한다.

기존 SEED 대응법은 먼저 전 라운드에 걸쳐 마스킹을 하는 Full 라운드 마스킹 기법을 사용한다. 또한 키와 평문에 모두 Boolean 마스킹을 하여 라운드 출력이 있을 때마다 마스킹 보정을 위한 추가 연산이 필요하게 된다.

3.1 기존 SEED 대응법의 마스킹 덧셈 연산

마스킹 덧셈 연산(MA)는 다음과 같이 두 8 비트 난수 $m,\ m'' = (m \oplus m')$에 대해 두 32비트 입력 $x \oplus (m'')^4$ $(= x \oplus (m'' \parallel m'' \parallel m'' \parallel m''))$과 $y \oplus (m'')^4$으로부터 $(x + y) \oplus (m)^4$을 출력하도록 한다. MA의 출력 값이 G 함수에 입력될 때, 마스킹 S-box의 Boolean 마스크에 적합하도록 설계된 것으로 이 때, Boolean 마스크과 Arithmetic 마스크 사이에 형태 변환이 필요하며 BtoA는 [9]에서, AtoB는 [14]에서 제안한 방법을 이용한다. (그림 4)는 기존 SEED 대응법이 적용된 F 함수이며 이로부터 MA의 출력값이 G 함수에 입력될 때마다 AtoB 연산을 수행해야 함을 알 수 있고 이는 많은 연산량이 소요된다. 한 번의 마스킹 덧셈을 수행할 때마다 MA 입력값의 마스크를 Boolean 마스크에서 Arithmetic 마스크로 바꾸어 주어야 하므로 BtoA 2 번, MA의 출력 값이 Boolean 마스킹되어 G 함수에 입력되어



(그림 4) 기존 SEED 대응법의 F 함수

야 하므로 AtoB 1 번을 수행해야 한다. SEED 알고리즘 한 라운드에서 MA가 세 번 수행되므로 모두 AtoB 3 번을 수행해야 한다. AtoB에 필요한 메모리 크기를 살펴보면, 마스킹 변환 테이블과 캐리 테이블에 각각 16 바이트 RAM이 필요하므로 테이블 생성 시간 및 모두 32 바이트 RAM이 소요된다.

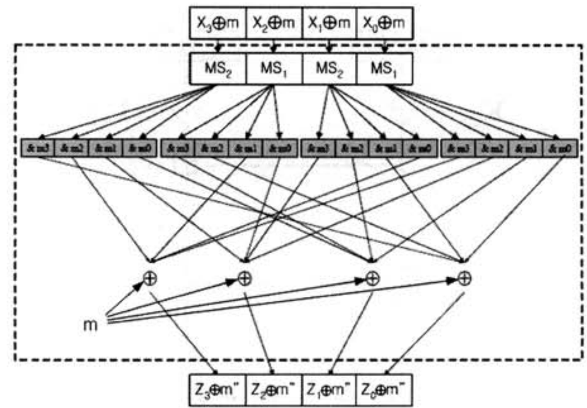
3.2 기존 SEED 대응법의 마스킹 S-box 설계

G 함수는 $(m)^4$으로 Boolean 마스킹 된 입력 값에 대해 $(m'')^4$으로 Boolean 마스킹 된 G 함수 출력값 Z를 출력하도록 한다. 기존 SEED 대응법의 G 함수에서는 마스킹 S-box MS_1, MS_2가 S_1, S_2 대신에 사용된다. MS_1, MS_2의 계산은 S_1, S_2와 같이 테이블을 이용하여 생성하는데 생성 알고리즘은 Algorithm 1과 같다. 마스킹 S-box 생성 알고리즘은 두 S-box S_1, S_2에 대하여 m으로 Boolean 마스킹 된 x 값이 입력되었을 때, m'으로 Boolean 마스킹 된

[Algorithm 1] 기존 마스킹 S-box 생성 알고리즘

Input : $S_1(x)$ S_2, m, m'
Output : MS_1, MS_2

1. For x from $0x00$ to $0xff$
   1.1 : $MS_1(x \oplus m) = S_1(x) \oplus m'$
   1.2 : $MS_2(x \oplus m) = S_2(x) \oplus m'$
2 : *Return* MS_1, MS_2



(그림 5) 기존 SEED 대응법의 G 함수

$S_1(x)$, $S_2(x)$의 값을 출력하는 테이블을 생성하도록 한다. (그림 5)는 기존 SEED 대응법의 마스킹 S-box를 나타낸다. 기존 SEED 대응법에서는 마스킹 S-box MS_1, MS_2에 대한 테이블 2 개를 모두 생성하므로 각 마스킹 S-box에 대하여 256 바이트 RAM이 필요하여 모두 512 바이트 RAM이 소요된다.

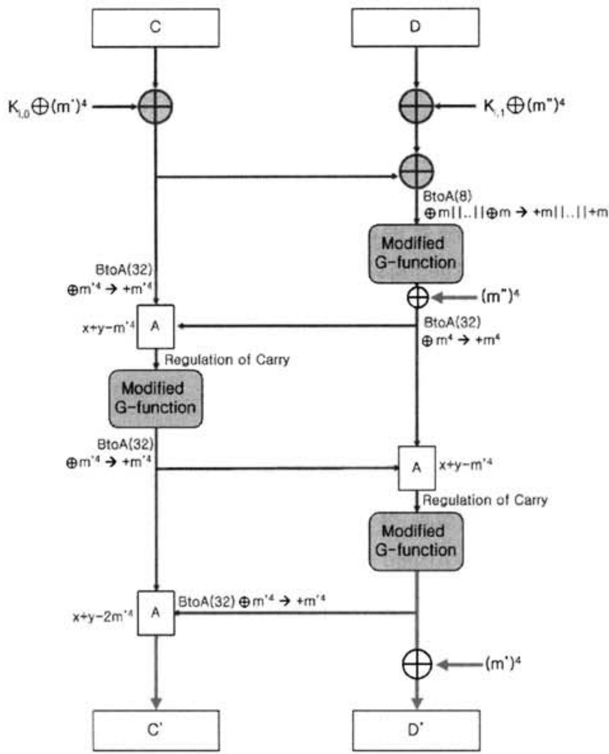
따라서 기존 SEED 대응법에서는 32비트 덧셈 연산과 S-box 연산에 마스킹을 적용하기 위해 테이블 생성시간 및 544 바이트 RAM이 필요하다.

4. 제안하는 마스킹 기법

기존 SEED 대응법에서 소요 비용의 대분을 차지하는 연산은 Arithmetic 마스킹에서 Boolean 마스킹으로 변환하는 함수의 호출 부분이었으며, 두 개의 마스킹 S-box 생성으로 인한 512 바이트의 RAM 또한 무시할 수 없는 부분이었다. 본 논문에서는 이러한 기존 SEED 대응법의 문제점을 보완하기 위해 Arithmetic 마스킹에서 Boolean 마스킹으로 변환하는 함수의 호출 횟수를 줄일 수 있는 새로운 마스킹 S-box를 설계하며 소요 RAM 사이즈를 줄일 수 있는 마스킹 S-box 사이의 연산 식을 세운다.

4.1 제안하는 마스킹 대응법의 전체 구조

제안하는 마스킹 대응법은 차분 전력 분석에 안전하도록 구성하기 위해 앞뒤 적당한 라운드에 마스킹을 수행하는 축소 라운드 마스킹을 적용한다. 물론, 16 라운드인 SEED의 앞뒤 8 라운드를 마스킹할 경우, Full 라운드 마스킹이 되므로 제안하는 마스킹 기법을 확장할 경우 기존 SEED 대응법과 같이 알고리즘의 모든 중간 연산 값을 마스킹할 수 있다. 본 논문에서는 앞뒤 한 라운드(1, 16 라운드) 마스킹(대응법 1) 또는 앞뒤 두 라운드(1, 2, 15, 16 라운드) 마스킹(대응법 2)에 초점을 두고 설명을 하도록 한다. 다음 장에서는 이러한 축소 라운드 마스킹에 대한 안전성 논의와 기존 SEED 대응법과 동등한 조건에서의 연산 속도를 비교하도



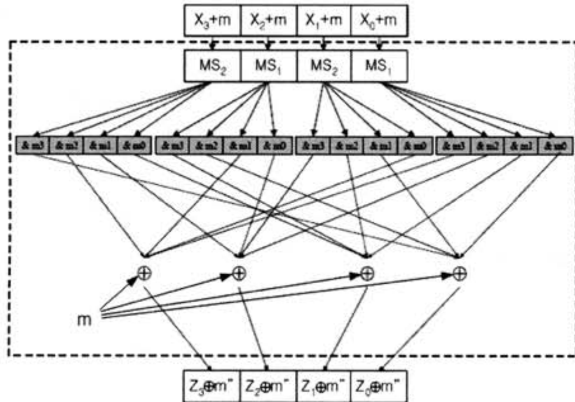
(그림 6) 제안하는 마스킹 기법 중 대응법 1의 1, 16 라운드 F 함수

록 한다. 다음은 대응법 1의 1, 16 라운드 구조이다.

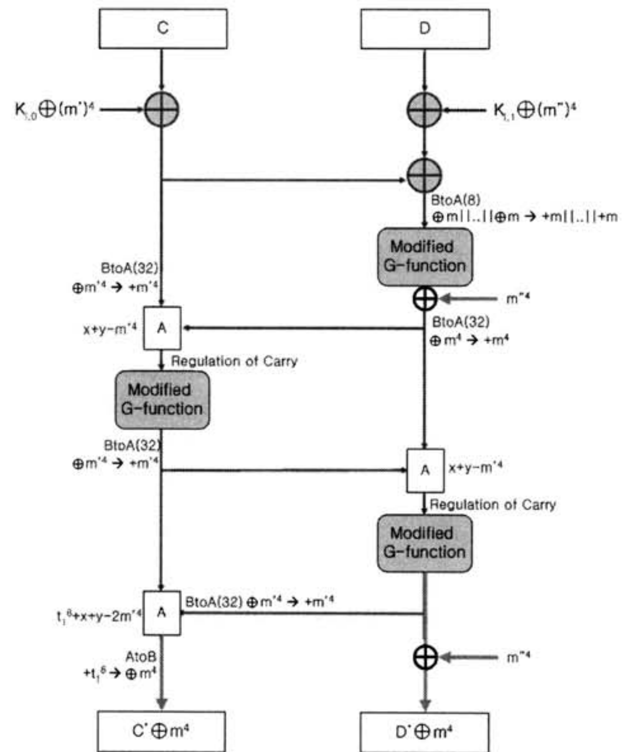
대응법 1은 알고리즘 수행 전 생성된 두 난수(m, m')와 두 난수로부터 생성된 새로운 난수$(m''=m \oplus m')$로 중간 값을 Arithmetic 또는 Boolean 마스킹 시킨다. (그림 4)의 구조에서 BtoA(8) 함수와 BtoA(32) 함수는 Boolean 마스킹 되어있는 데이터를 Arithmetic 마스킹으로 안전하게 변환하는 알고리즘으로써 [9]에서 제안된 방법에 따라 수행된다. 차이점이 있다면, BtoA(8) 함수는 32 비트의 변수를 8 비트씩 네 바이트로 나눠 각 바이트를 법 2^8에 대하여 Artihmetic 마스킹으로 변환하는 함수이고 BtoA(32) 함수는 32 비트를 법 2^{32}에 대하여 Arithmetic 마스킹으로 변환하는 알고리즘이다. (그림 4)에서 보는바와 같이 대응법 1은 기존 SEED 대응법과 달리 덧셈 연산 수행 후 추가적인 마스킹 형태 변환(Arithmetic에서 Boolean 마스킹으로의 변환 3회, 한 라운드에서 대부분의 연산량을 차지)이 수행되어질 필요가 없으며 상당한 연산 시간을 단축시킬 수 있다. 앞뒤 두 라운드(1, 2, 15, 16 라운드)를 마스킹 시키는 대응법 2에 대한 구조는 (그림 8), (그림 9)과 같다.

앞뒤 두 라운드를 마스킹 시키는 대응법 2에서는 마스킹 된 1 라운드 출력 값이 2 라운드의 입력 값이 되어야 한다. 따라서 2 라운드 시작 전 키 에디션 연산을 위해 1 라운드의 Arithmetic 마스킹 된 출력 값을 반드시 (그림 8)에서 보는 바와 같이 Boolean 마스킹 변환해야만 한다. 본 논문에서는 이 변환 기법으로 [15]의 방법을 참고하였다. 하지만 이 변환은 1, 16 라운드 축소 라운드 마스킹 적용했을 때, AtoB 연산이 한 라운드당 3번에서 1번으로 줄었으므로 3*2회 필요하던 연산을 단지 1*2회의 연산으로 수행하도록 한 것으로 상당한 연산량이 감소된다.

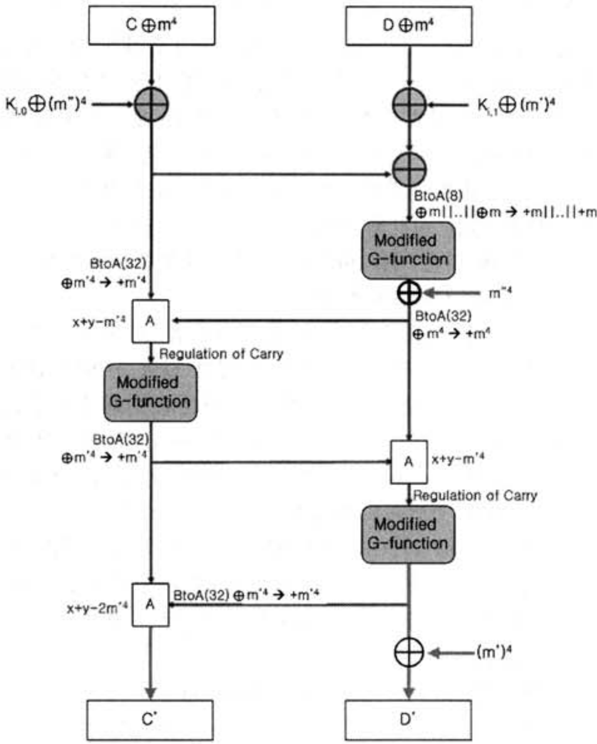
이러한 Arithmetic 마스킹에서 Boolean 마스킹으로 변환하는 함수의 호출 횟수를 줄일 수 있는 근본적인 이유는 새로운 마스킹 S-box의 설계에 의해 가능하다. 다음 소절에서는 이 새로운 마스킹 S-box를 설계하는 방법과 마스킹 S-box의 RAM 사이즈를 줄이기 위한 연산 방법에 대해 논



(그림 7) 제안하는 마스킹 기법의 G 함수



(그림 8) 대응법 2의 1, 16 라운드 F 함수



(그림 9) 대응법 2의 2, 15 라운드 F 함수

한다. 또한 32 비트 단위의 Arithmetic 마스킹을 8 비트 단위의 Arithmetic 마스킹으로 변환하기 위한 캐리 조절 (Regulation of Carry) 알고리즘을 소개한다.

4.2 Arithmetic에서 Boolean 마스킹으로의 변환 횟수를 줄이기 위한 마스킹 S-box의 설계

기존 SEED 대응법은 일반적인 방법인 $MS(x \oplus m) = S(x) \oplus m'$을 만족하는 마스킹 S-box를 설계하였다. 따라서 Arithmetic 마스킹된 중간 연산 값이 마스킹 S-box로 입력되기 위해선 반드시 Arithmetic에서 Boolean 마스킹으로 마스킹 변환을 수행해야만 했다. 본 논문에서는 이러한 연산을 줄이기 위해 마스킹 변환과 S-box의 연산을 동시에 수행하는 마스킹 S-box를 설계한다. 즉, Arithmetic 마스킹된 입력 값에 대해 Boolean 마스킹된 값을 출력하는 마스킹 S-box를 생성한다. S_1, S_2에 대한 마스킹 S-box MS_1, MS_2가 만족해야 하는 식은 다음과 같다.

$$MS_1((x+m) \bmod 2^8) = S_1(x) \oplus m'$$
$$MS_2((x+m) \bmod 2^8) = S_2(x) \oplus m'$$

본 논문에서는 RAM 사이즈를 최소화하기 위해 MS_2의 하나의 마스킹 S-box만을 다음 알고리즘에 의해 암호 연산 전 생성하며 MS_2로부터 MS_1의 출력 값을 연산한다.

[Algorithm 2] 마스킹 S-box 생성 알고리즘

Input : $S_2 m, m'$
Output : MS_2

1 : For x from 0x00 to 0xff
  $MS_2((x+m) \bmod 2^8) = S_2(x) \oplus m'$
2 : Return MS_2

4.3 RAM 사이즈를 줄이기 위한 마스킹 S-box의 연산식

앞에서 설명한 바와 같이 제안하는 마스킹 기법은 $MS_1((x+m) \bmod 2^8) = S_1(x) \oplus m'$에 해당하는 $MS_1(x')$을 얻기 위해 MS_2 테이블을 사용한다. 이는 기존에 RAM에 저장되었던 512 바이트의 MS_1, MS_2 테이블 사이즈를 256 바이트로 줄이기 위함이다. 본 소절에서는 MS_2 테이블로부터 $MS_1(x')$의 출력 값을 얻기 위한 연산식을 소개한다.

각 S-box 연산은 행렬 A_1, A_2와 벡터 b_1, b_2에 대해 다음과 같은 아핀(Affine) 연산을 수행한다.

$$S_1(x) = A_1 x^{247} \oplus b_1 = A_1 x^{-8} \oplus b_1$$
$$S_2(x) = A_2 x^{251} \oplus b_2 = A_2 x^{-4} \oplus b_2$$

이 식으로부터,

$$\begin{aligned} S_1(x) &= A_1((A_2^{-1}(S_2(x) \oplus b_2))^2) \oplus b_1 \\ &= A_1(B(A_2^{-1}(S_2(x) \oplus b_2))) \oplus b_1 \\ &= (A_1 B A_2^{-1}) S_2(x) \oplus (A_1 B A_2^{-1} b_2 \oplus b_1) \\ &= A_3 S_2(x) \oplus b_3 \end{aligned}$$

$$A_3 = \begin{pmatrix} 1&0&0&1&0&0&0&1 \\ 0&0&0&0&1&0&0&1 \\ 0&1&1&0&0&1&0&0 \\ 0&0&0&0&0&1&0&0 \\ 1&1&0&1&1&0&0&0 \\ 1&1&1&1&0&1&1&1 \\ 0&0&0&0&1&1&1&0 \\ 0&1&0&1&0&0&1&0 \end{pmatrix}, \quad b_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

가 만족하므로 $S_2(x)$로부터 $S_1(x)$가 연산 가능하다. 이 때, $GF(2^8)$에서의 제곱 연산은 선형 연산으로, 모든 $x \in GF(2^8)$에 대해 $B(x) = x^2$을 만족하는 행렬 B가 존재하므로 대체가 가능하다. 또한 $m_a = A_3 m' \oplus m'$라 할 때,

[Algorithm 3] MS_2 테이블을 이용한 MS_1 출력값 연산 알고리즘

Input : x', $m_a (= A_3 m' \oplus m')$
Output : $MS_1(x')$

1 : $MS_1(x') = A_3 MS_2(x') \oplus m_a \oplus b_3$
2 : Return $MS_1(x')$

$$\begin{aligned} S_1(x-m) &= A_3S_2(x-m)\oplus b_3 \\ &= A_3MS_2(x)\oplus b_3\oplus A_3m' \\ MS_1(x) &= A_3MS_2(x)\oplus A_3m'\oplus m'\oplus b_3 \\ &= A_3MS_2(x)\oplus m_a\oplus b_3 \end{aligned}$$

가 성립하므로 $MS_2(x)$로부터 $MS_1(x)$의 연산이 가능하다. 이 때, $A_3MS_2(x)\oplus m_a\oplus b_3$의 연산에 소요되는 연산시간을 줄이기 위해 입력값 x에 대해 $A_3x\oplus b_3$의 값을 출력하는 Affine 테이블을 생성해 ROM에 저장하는 것이 가능하다. 즉, 이 Affine 테이블로부터 $MS_1(x')$의 값은 $MS_1(x')=Affine[MS_2(x')]\oplus m_a$의 연산에 의해 얻어질 수 있다. 따라서 전체 마스킹 S-box의 테이블을 저장하는데 필요한 RAM은 절반이 되고 추가로 Affine 테이블 저장에 256 바이트 ROM이 소요된다. 하지만 ROM은 RAM에 비해 일반적으로 충분한 공간이 있으며 그 메모리 크기가 크지 않으므로 Affine 테이블을 사전 연산해 저장하는 것이 MS_1 테이블을 생성하는 것보다 메모리 크기면에서 부담이 적게 된다.

4.4 캐리 조절(Regulation of Carry) 알고리즘

제안하는 마스킹 기법이 적용된 F 함수 구조에서는 모두 세 번의 32 비트 덧셈 연산을 수행한다. 32 비트 덧셈 연산이 끝난 후에는 결과 값이 바이트 단위로 나뉘어 수정된 G 함수에 입력되는데 이 때, 입력 값은 법 2^{32}에 대한 32 비트 Arithmetic 마스킹된 값이고 이 값이 마스킹 S-box에 해당하는 테이블을 호출 또는 연산을 수행하기 위해서는 32 비트의 각 바이트가 법 2^8에 대해 8 비트 Arithmetic 마스킹된 값으로 변환되어져야만 한다. 따라서 32 비트 덧셈 연산에서 8 비트마다 생겼던 캐리를 마스킹 S-box 테이블 호출 또는 연산 전 반드시 제거해야만 한다.

이 캐리를 제거하기 위해 32 비트 Arithmetic 마스킹 된 중간 값 $x_3'|x_2'|x_1'|x_0'$에서, $x_i'=(x_i+m+carry(x_{i-1}'))$ $(carry(x_{-1}')=0, i=0,1,2,3)$의 캐리 생성 유무를 최하위 바이트부터 판단해야한다. $x_i'>m,\ (i=0,1,2,3)$이면 캐리가 생성된 것이므로 이 값을 안전하게 계산해야 한다. 하지만, 이 연산을 직접 수행하는 것은 $x_i'-m$, 즉 x_i가 그대로 노출되고 이는 차분 전력 분석에 취약함을 의미한다. 따라서 본 논문에서는 이 캐리에 대한 부분을 안전하게 구성하기 위해 캐리에 대한 테이블을 암호 연산 전 생성해 RAM에 저장하는 방식을 사용한다. 이러한 방식을 사용할 경우 일반적으로 256 바이트의 RAM이 요구되지만 본 논문에서는 이 소요 RAM을 최소화하기 위해 4 비트 테이블 두 개(32 바이트 RAM 소요)를 사용해 캐리를 연산한다. 즉, 8 비트 캐리 조절을 위해 상·하위 4 비트로 나누어 캐리 테이블을 생성한다. 이러한 캐리 테이블을 생성했을 경우 4 비트 캐리 테이블 두 개를 사용하므로 256 바이트 RAM에서 32 바이트 RAM으로 소요량을 상당히 줄일 수 있다. $x_i+m\ (i=0,1,2,3)$의 하위 4 비트에 해당하는 캐리 테이블을 C_1, 상위 4 비트에 해당하는 캐리 테이블을 C_2라 하면 아래의 알고리즘과 같이 캐리 테이블 C_1, C_2를 생성한다. 캐리 테이블 생성 알고리즘에서 캐리 생성 여부를 감추기 위해 랜덤 상수 λ를 사용하였다.

[Algorithm 4] 캐리 테이블 생성 알고리즘

Output: *Carry Tables* C_1, C_2

1 : 8 비트 랜덤 상수 λ를 생성한다.

2 : For A from 0 to 15

2.1 : If $A<(m\&0x0f)$, $C_1[A]=\lambda+1$ // 캐리가 생성된 경우

2.2 : else $C_1[A]=\lambda$ // 캐리가 생성되지 않은 경우

2.3 : If $A<(m\&0xf0)^{\gg 4}$, $C_2[A]=\lambda+1$ // 캐리가 생성된 경우

2.4 : else if $A=(m\&0xf0)^{\gg 4}$, $C_2[A]=\lambda+2$

2.5 : else $C_2[A]=\lambda$ // 캐리가 생성되지 않은 경우

3 : Return C_1, C_2

[Algorithm 5] 캐리 조절(Regulation of Carry) 알고리즘

Input : 32 비트 Arithmetic masked value

$x_3'|x_2'|x_1'|x_0'=(x_3|x_2|x_1|x_0)+(m|m|m|m)$

Output: 8 비트 Arithmetic masked value

$x_3'|x_2'|x_1'|x_0'=((x_3+m)|(x_2+m)|(x_1+m)|(x_0+m))$

1 : For i from 0 to 2

1.1 : $carry=C_2[(x_i\&0xf0)^{\gg 4}]$

1.2 : If $carry=\lambda+2$, $carry=C_1[x\&0x0f]$

1.3 : $(x_3'|x_2'|\cdots|x_{i+1}')=(x_3'|x_2'|\cdots|x_{i+1}')-carry$

1.4 : $(x_3'|x_2'|\cdots|x_{i+1}')+=\lambda$

2 : Return $x_3'|x_2'|x_1'|x_0'$

암호 연산 수행 전 생성된 캐리 테이블로부터 8 비트마다 캐리를 조절해 주는 알고리즘은 아래와 같다. 이 때, Algorithm 5의 1.2에서 $carry=\lambda+2$라면 아래의 두 가지 경우를 생각해 볼 수 있다.

1. $C_1[A]=\lambda+1$이고 x_i'의 상위 4 비트가 $0x0f$ → $C_2[A]=\lambda+1$
2. $C_1[A]=\lambda$이고 x_i'의 상위 4 비트가 $0x00$ → $C_2[A]=\lambda$

두 경우 모두 $C_1[A]=C_2[A]$이므로 1.3과 같이 연산하도록 한다.

5. 축소 라운드 마스킹 기법에 대한 안전성

제안하는 마스킹 기법은 편중되지 않은 난수로 암호 연산의 1, 16 라운드(대응법 1) 혹은 1, 2, 15, 16 라운드(대응법 2)의 모든 중간 값을 가린다. 일반적인 차분 전력 분석을 수행하기 위해서는 대응법 1에 대해서는 1 라운드 키 혹은 16 라운드 키 64 비트를 추측해야 차분 전력 분석을 수행할 수

있으며 대응법 2에 대해서는 1, 2 라운드 키 혹은 15, 16 라운드 키 128 비트를 추측해야 분석을 수행할 수 있다. 하지만 64 비트 혹은 128 비트 키를 추측해 차분 전력 분석을 수행하는 것은 실질적으로 불가능하며 따라서 제안하는 마스킹 기법은 기존의 일차 차분 전력 분석 방법에 충분한 안전성을 보장한다.

하지만 제안하는 대응 방법은 Full 라운드 마스킹이 아닌 축소 라운드 마스킹을 적용하였기 때문에 마스킹 되지 않은 중간 라운드 연산의 안전성을 고려해야 한다. 먼저 덧셈 연산이 없는 대칭키 암호(AES, DES)에 대해서는 축소 라운드 마스킹 기법에 대한 이론적인 분석 방법[13]이 존재하지만 SEED와 같이 덧셈 연산이 존재하는 대칭키 암호의 축소 라운드 마스킹에 대한 분석법은 존재하지 않는다. 그리고 축소 라운드 마스킹에 대한 기존 분석법은 하나의 전력 파형으로부터 연산되는 중간 값의 해밍 웨이트를 정확하게 알 수 있다는 가정 하에서 수행되어질 수 있다. 하지만 이러한 가정은 전력 파형에서 해당 중간값이 연산되는 위치를 정확히 알아야 한다는 것을 의미한다. 또한, 이 연산 위치를 정확히 안다고 가정하더라도 현실적으로 중간값의 해밍 웨이트를 오류 없이 정확하게 아는 것은 잡음에 의해 불가능하므로 실질적으로 이러한 분석으로 키를 찾는 것은 불가능하다. 만약 이러한 분석법으로부터도 안전성을 보장하고 싶다면 제안하는 대응 기법을 변형해 Full 라운드 마스킹으로 확장하는 것은 어려운 일이 아니다.

6. 비교 및 결론

본 논문에서는 SEED 알고리즘의 구조에 최적화된 새로운 마스킹 대응법을 제안하였다. 제안하는 마스킹 기법에서는 SEED의 마스킹 기법 설계에 있어 연산량이 가장 큰 Arithmetic에서 Boolean 마스킹으로의 변환 함수 호출 횟수를 완전히 제거하거나 최소화할 수 있는 마스킹 S-box를 설계하였다. 또한 RAM 사이즈를 줄이기 위해 두 S-box에 대한 마스킹 S-box 테이블을 하나만 생성해 사용할 수 있는 연산식을 제안하였다. 다음 <표 1>은 마스킹 기법이 적용되지 않은 SEED, 기존 SEED 대응법 그리고 제안하는 마스킹 기법이 적용된 SEED를 8 비트용으로 구현했을 때 메모리 크기와 연산 속도를 나타낸 표이다. <표 1>에서와 같이 제안하는 마스킹 기법은 기존의 마스킹 기법과 동일한 안전성을 제공하면서도 메모리 크기뿐만 아니라 속도 면에서도 모두 효율적이다.

참 고 문 헌

[1] P. Kocher, J. Jaffe, and B. Jun, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Others Systems" CRYPTO'6, LNCS 1109, pp.104-113, Springer-Verlag, 1996.

[2] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," CRYPTO'9, pp.388-397, Springer-Verlag, 1999.

[3] P. Kocher, J. Jaffe, and B. Jun, "Introduction to differential power analysis and related attacks," http://www.cryptography.com/dpa/technical, 1998.

[4] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Power analysis attacks on modular exponentiation in Smart cards," Proc. of Workshop on Cryptographic Hardware and Embedded Systems, pp.144-157, Springer-Verlag, 1999.

[5] E. Oswald nd K. Schramm. "An Efficient Masking Scheme for AES Software Implementations," TM WISA 2005, LNCS 3786, pp.292-305, Springer, 2006.

[6] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen., "A Side-Channel Analysis Resistant Description of the AES S-box," FSE 2005, LNCS 3557, pp. 413-423, Springer, 2005.

[7] J. Bl¨omer, J. Guajardo, and V. Krummel. "Provably Secure Masking of AES," SAC 2004, LNCS 3357, pp.69-83, Springer, 2005.

〈표 1〉 제안하는 SEED 마스킹 기법의 메모리 크기와 속도 비교 (8 비트)

1000만 번 수행	속도(초)					RAM (table, 바이트)	ROM (바이트)
	테이블 생성	키 스케줄	암호 연산	합계	비율		
SEED	0	32	39	71	1	-	512
기존 1, 16 라운드 마스킹	19	33	76	128	1.80	544	512
기존 1, 2, 15, 16 라운드 마스킹	19	34	123	176	2.48	544	512
기존 Full 라운드 마스킹	19	35	397	451	6.35	544	512
제안 1, 16 라운드 마스킹	12	33	55	100	1.41	288	768
제안 1, 2, 15, 16 라운드 마스킹	12	34	80	126	1.77	304	768
제안 Full 라운드 마스킹	12	35	230	277	3.9	304	768

[8] 한국정보보호진흥원, "SEED 알고리즘 상세 명세서"
[9] J. Coron and Louis Goubin "On Boolean and Arithmetic Masking against Differential Power Analysis", CHES'00, LNCS 1965, pp.231-237, 2000.
[10] S. Chari, C. Jutla, J. Rao, P. Rohatgi, "Towards Sound Approaches to Counteract Power-Analysis Attacks," CRYPTO99, Springer-Verlag, pp.398-412, 1999.
[11] C. Clavier, J. Coron, and N. Dabbous, "Differential power analysis in the presence of hardware countermeasures," CHES 2000, Lecture Notes in Computer Science, Vol.1965, pp.252-263, August. 2000.
[12] O. Kommerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," Proc. of the USENIX Workshop on Smartcard Technology, Chicago, pp.9-20, May, 1999.
[13] H. Handschuh and B. Preneel, "Blind Differential Cryptanalysis for Enhanced Power Attacks," Proc.SAC2006, LNCS, Vol.4356, pp.163-173, 2007.
[14] J. Coron, A. Tchulkine, "A New Algorithm for Switching from Arithmetic to Boolean Masking," CHES'03, LNCS 2779 pp.89-97, 2003.
[15] O. Neiße and J. Pulkus1, "Switching Blinding with a View Towards IDEA," CHES'04, LNCS 3156, pp.230-239, 2004.

조 영 인

e-mail : elowey@korea.ac.kr
2006년 한양대학교 수학과(학사)
2009년 고려대학교 정보경영공학과(공학석사)
2009년 9월~현 재 고려대학교 정보경영공학과 박사과정
관심분야 : 암호칩 설계 기술, 부채널 공격, 암호시스템 고속구현 등

김 희 석

e-mail : 80khs@korea.ac.kr
2006년 연세대학교 수학과(학사)
2008년 고려대학교 정보경영공학과(공학석사)
2008년~현 재 고려대학교 정보경영공학과 박사과정
관심분야 : 부채널 공격, 암호시스템 안전성 분석 및 고속구현, 암호칩 설계 기술 등

최 두 호

e-mail : dhchoi@etri.re.kr
1994년 성균관대학교 수학과(학사)
1996년 한국과학기술원 수학과 석사(이학석사)
2002년 한국과학기술원 수학과 박사(이학박사)
2002년 1월~현 재 한국전자통신연구원 정보보호연구본부 선임연구원/팀장
관심분야 : RFID/USN 정보보호 기술, 페어링 기반 암호 이론, 암호시스템 안전성 증명, 비가환군 암호 이론 등

한 동 국

e-mail : christa@kookmin.ac.kr
1999년 고려대학교 수학과(학사)
2002년 고려대학교 수학과(이학석사)
2005년 고려대학교 정보보호학과(공학박사)
2004년 4월~2005년 4월 일본 Kyushu Univ. 방문연구원
2005년 4월~2006년 4월 일본 Future Univ.-Hakodate, Post. Doc.
2006년 6월~2009년 2월 한국전자통신연구원 정보보호연구단 선임연구원
2009년 3월~현 재 국민대학교 수학과 조교수
관심분야 : 암호시스템 안전성 분석 및 고속 구현, 부채널 분석, RFID/USN 정보보호 기술 등

홍 석 희

e-mail : hsh@cist.korea.ac.kr
1995년 고려대학교 수학과(학사)
1997년 고려대학교 수학과(이학석사)
2001년 고려대학교 수학과(이학박사)
1999년 8월~2004년 2월 (주)시큐리티 테크놀로지스 선임연구원
2003년 3월~2004년 2월 고려대학교 시간강사
2004년 4월~2005년 2월 K.U. Leuven 박사 후 연구원
2005년 3월~2008년 8월 고려대학교 정보경영공학전문대학원 조교수
2008년 9월~현 재 고려대학교 정보경영공학전문대학원 부교수
관심분야 : 대칭키 암호 알고리즘, 공개키 암호 알고리즘, 포렌식 등

이 옥 연
e-mail : oyyi@kookmin.ac.kr
1988년 고려대학교 수학과(학사)
1990년 고려대학교 수학과(이학석사)
1996년 8월 Univ. of Kentucky Ph.D.(이학박사)
1999년~2001년 ETRI 선임연구원
2001년~현 재 국민대학교 수학과 부교수
관심분야 : 이동통신 정보보호, 컴퓨터 보안 등