Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.26, NO.1, Feb. 2016 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2016.26.1.5

저메모리 기반의 산술 마스킹에서 불 마스킹 변환 알고리즘*

김 한 빛,[1†] 김 희 석,[2,3] 김 태 원,[1] 홍 석 희[1‡]
[1]고려대학교, [2]한국과학기술정보연구원, [3]과학기술연합대학원대학교

An Algorithm for Switching from Arithmetic to Boolean Masking with Low Memory*

HanBit Kim,[1†] HeeSeok Kim,[2,3] TaeWon Kim,[1] SeokHie Hong[1‡]
[1]Korea University, [2]Korea Institute of Science and Technology Information,
[3]University of Science and Technology

요 약

전력 분석 공격은 공격자가 암호 알고리즘이 수행되는 동안 발생하는 전력 신호를 분석하여 비밀정보를 알아내는 분석 기법이다. 이러한 부채널 공격의 대응기법으로 널리 알려진 방법 중 하나는 마스킹 기법이다. 마스킹 기법은 크게 불 마스킹 형태와 산술 마스킹 형태의 두 종류로 나뉜다. 불 연산자와 산술 연산자를 사용하는 암호 알고리즘의 경우, 연산자에 따라 마스킹의 형태를 변환하는 알고리즘으로 마스킹 기법을 적용 가능하다. 본 논문에서는 기존의 방식보다 더 적은 비용의 저장 공간을 이용하는 산술 마스킹에서 불 마스킹 변환 알고리즘을 제안한다. 제안하는 변환 알고리즘은 마스킹의 최하위 비트(LSB)의 경우 불 마스킹과 산술 마스킹이 같음을 이용하여 변환하려는 비트 크기와 같은 크기만큼 저장 공간을 사용하여 참조 테이블을 구성한다. 이로 인해 기존의 변환 알고리즘과 비교해 성능 저하 없이 더 적은 비용으로 변환 알고리즘을 설계할 수 있다. 추가로 제안하는 기법을 LEA에 적용하여 기존의 기법보다 최대 26.2% 성능향상을 보였다.

ABSTRACT

Power analysis attacks are techniques to analyze power signals to find out the secrets when cryptographic algorithm is performed. One of the most famous countermeasure against power analysis attacks is masking methods. Masking types are largely classified into two types which are boolean masking and arithmetic masking. For the cryptographic algorithm to be used with boolean and arithmetic masking at the same time, the converting algorithm can switch between boolean and arithmetic masking. In this paper we propose an algorithm for switching from boolean to arithmetic masking using storage size at less cost than ones. The proposed algorithm is configured to convert using the look-up table without the least significant bit(LSB), because of equal the bit of boolean and arithmetic masking. This makes it possible to design a converting algorithm compared to the previous algorithm at a lower cost without sacrificing performance. In addition, by applying the technique at the LEA it showed up to 26 percent performance improvement over existing techniques.

Keywords: Side-channel, Arithmetic to Boolean masking, Masking, Countermeasure

Received(09. 15. 2015), Modified(12. 15. 2015), Accepted(12. 15. 2015)

* "본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT연구센터육성 지원사업의 연구결과로 수행되었음" (IITP-2015-R0992-15-1017)

† 주저자, luz_damoon@naver.com
‡ 교신저자, shhong@korea.ac.kr(Corresponding author)

I. 서 론

수학적으로 안전성이 검증된 암호 알고리즘이라 할지라도 물리적인 구현 과정에서 설계자가 고려하지 못한 정보의 누출이 발생할 수 있다. 1999년 Kocher에 의해 처음으로 제안된 부채널 공격(Side Channel Attack)은 이러한 정보의 누출을 통해 비밀정보를 알아내는 공격 기법이다[1]. 대표적인 부채널 공격 기법으로는 소요시간 공격(Timing Attack)[2], 전력분석 공격(Power Analysis Attack)[3], 전자기파 공격(Electromagnetic Attack)[4][5] 등이 있다. 부채널 공격이 현실적인 위협 요소라는 것은 국외 보안하드웨어 안전성 검증 항목인 공통평가기준(Common Criteria, CC인증)[6], CMVP의 FIPS 140-3(Federal Information Processing Standard)[7], 신용/직불 카드의 국제표준규격인 EMV 인증[8] 등의 보안 필수 평가 항목으로 지정되어 있는 것에서 확인할 수 있다. 부채널 공격에 대한 안전성이 중요해짐에 따라 암호 설계자들은 수학적 안전성과 더불어 부채널 공격에도 안전한 암호 알고리즘의 설계를 고려하여야 한다.

부채널 공격에 대해 안전성을 보장하기 위한 방법으로 마스킹 기법이 있다[9][10]. Messerges가 처음 제안한 마스킹 기법은 암호 알고리즘이 연산될 때 발생하는 중간 값(Intermediate Values)을 랜덤하게 만들어 공격자에게 필요한 정보의 누출을 막는 대응기법(Countermeasure)이다[11]. 대표적인 마스킹 기법의 형태로는 불 마스킹(Boolean Masking)과 산술 마스킹(Arithmetic Masking)이 있다[12]. 불 마스킹은 배타적 논리합(XOR, Exclusive Or)을 이용한 $x' = x \oplus r$ 형태로 중간 값의 노출을 방지하며, 산술 마스킹은 모듈러 덧셈, 뺄셈을 이용한 $A \equiv x \pm r (\mathrm{mod} n)$ 형태 또는 곱셈을 이용한 $A \equiv x \times r (\mathrm{mod} n)$ 형태 등의 대수 연산을 통해 중간 값을 랜덤하게 만든다. 만약 암호 알고리즘에 불 연산자와 산술 연산자가 사용된다면, 연산자에 따라 두 마스킹 형태를 변환하는 알고리즘이 필요하다.

본 논문에서는 산술 마스킹에서 불 마스킹으로 변환 하는 두 가지 종류의 알고리즘을 제안한다. 첫째는 사전연산 테이블을 이용한 알고리즘이며, 둘째는 전수 저장 테이블을 이용한 알고리즘이다. 두 알고리즘은 기존 변환 알고리즘과는 다르게 1-bit의 테이블 저장 공간을 줄임으로 더 적은 비용으로 변환 알고리즘을 구성할 수 있다. 제안하는 알고리즘에서 연산 도중 발생하는 중간 값의 마스킹 유무를 분석하여 알고리즘의 안전성을 검증한다. 이를 국산 암호 알고리즘인 LEA에 적용하여 1차 마스킹 LEA를 구성한다. 마지막으로 기존의 알고리즘과 비교를 통해 제안하는 알고리즘의 성능을 확인한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 불 마스킹과 산술 마스킹 변환 알고리즘을 살펴본다. 3장에서는 새로운 산술 마스킹에서 불 마스킹 변환 알고리즘을 제안한다. 4장에서는 제안하는 알고리즘의 부채널 분석의 안전성을 검증한다. 5장에서는 제안하는 알고리즘을 LEA 암호 알고리즘에 적용한다. 마지막으로 6장에서는 성능 비교와 결론을 맺는다.

II. 불 마스킹과 산술 마스킹간에 상호 마스킹 변환 알고리즘

ARX(Addition, Rotation, XOR)구조로 구성된 암호 알고리즘은 산술 마스킹으로 연산되는 덧셈 과정과 불 마스킹 형태로 연산되는 순환(Rotation), XOR 연산이 하나의 라운드에 존재한다. 이 경우 불 마스킹과 산술 마스킹의 상호 마스킹 변환 알고리즘을 통해 마스킹 기법의 구현이 가능하다. 다시 말해 덧셈과정에서는 산술 마스킹의 형태로 연산을 진행하고, 순환 및 XOR 과정은 불 마스킹으로 연산을 진행한다. 이러한 암호 알고리즘에 마스킹 기법을 적용할 때 추가비용(Overhead)은 불 마스킹과 산술 마스킹의 상호 마스킹 변환 알고리즘이 큰 요소를 차지한다.

CHES 2001에서 Goubin에 의해 제안된 불 마스킹과 산술 마스킹의 상호 변환 알고리즘은 임의의 비트 크기에 대해 불 마스킹과 산술 마스킹의 상호

Algorithm. 1. Goubin's B2A Algorithm

Input : (x', r) $x = x' \oplus r$

Output : (A, r) $A = x - r$

$\Gamma \leftarrow$ **rand()**

1. $T \leftarrow x' \oplus \Gamma$
2. $T \leftarrow T - \Gamma$
3. $T \leftarrow T \oplus x'$
4. $\Gamma \leftarrow \Gamma \oplus r$
5. $A \leftarrow x' \oplus \Gamma$
6. $A \leftarrow A - \Gamma$
7. $A \leftarrow A \oplus T$

변환이 가능하다[13]. [Alg. 1]는 Goubin이 제안한 불 마스킹에서 산술 마스킹 변환(B2A, Boolean to Arithmetic Masking Conversion) 알고리즘이다. Goubin의 B2A 알고리즘은 임의의 비트단위에 대해 한 번의 루틴만으로 변환이 가능하다. 연산의 시간 복잡도(Time Complexity)는 $O(1)$이다.

그러나 산술 마스킹에서 불 마스킹 변환(A2B, Arithmetic to Boolean Masking Conversion) 알고리즘의 경우 리플-올림수 가산기(ripple-carry adder)를 기반으로 알고리즘이 구성된다[14]. 리플-올림수 가산기 기반이란 하나의 비트를 보고 이 비트에 올림수가 발생하는지 확인, 다음 비트에 올림수를 전파하는 일련의 과정을 반복하여 비트를 처리하는 기법이다. 즉, 이 연산은 상위 비트의 A2B 변환을 위해 하위 비트에서 생성되는 올림수가 필요하다. [Alg. 2]은 Goubin이 제안한 A2B 알고리즘이다. A2B 알고리즘에서 주목해 볼 점은 10~15번째 연산에서 변환되는 비트크기(K)에 따라 루틴이 반복된다는 점이다. 이 경우 시간 복잡도는 $O(K)$이다. 이는 상대적으로 B2A 알고리즘보다 더 많은 추가연산을 필요로 한다. 이러한 추가연산을 줄이기 위해 참조 테이블(LUT, Look-up Table)을 만들어 사용한다[15]. 참조 테이블을 이용할 경우 테이블 크기에 따라 시간 복잡도가 결정되는데 테이블 크기가 k일 경우, 시간 복잡도는 $O(\frac{K}{k})$로 볼 수 있다. 하지만 테이블 크기가 커지면 테이블을 저장하는데 필요한 메모리가 늘어나므로 설계자는 저장 공간과 속도 사이에 적절한 균형을 유지해야 할 것이다.

Algorithm. 2. Goubin's A2B Algorithm

Input : (A,r) $A=x-r$
Output : (x',r) $x=x'\oplus r$

$\Gamma \leftarrow$ **rand()**
1. $T \leftarrow 2\Gamma$
2. $x' \leftarrow \Gamma \oplus r$
3. $\Omega \leftarrow \Gamma \wedge x'$
4. $x' \leftarrow T \oplus A$
5. $\Gamma \leftarrow \Gamma \oplus x'$
6. $\Gamma \leftarrow \Gamma \wedge r$
7. $\Omega \leftarrow \Omega \oplus \Gamma$
8. $\Gamma \leftarrow \Gamma \oplus A$
9. $\Omega \leftarrow \Omega \oplus \Gamma$
10. **for** i = 1 to K-1 **do**
11. $\Gamma \leftarrow T \wedge r$
12. $\Gamma \leftarrow \Gamma \oplus \Omega$
13. $T \leftarrow T \wedge A$
14. $T \leftarrow 2\Gamma$
15. **end for**
16. $x' \leftarrow x' \oplus T$

Algorithm. 3. Debrize's Table T generation

1. Generate a random k-bit r and a random bit ρ
2. **for** $A=0$ to 2^k-1 **do**
$T[\rho \| A] = (A+r) \oplus (\rho \| r)$
$T[\rho \oplus 1 \| A] = (A+r+1) \oplus (\rho \| r)$
3. **end for**
4. **return** T, r and ρ

Debraize에 의해 제안된 산술 마스킹에서 불 마스킹 변환 알고리즘은 사전에 연산된 참조 테이블을 이용한 A2B 알고리즘이다[16]. [Alg. 3]와 [Alg. 4]는 Debraize가 제안한 참조 테이블 생성 알고리즘과 A2B 변환 알고리즘이다. [Alg. 3]으로 참조 테이블을 생성하고 생성된 테이블을 사용하여 [Alg. 4]를 수행한다. 마스킹의 변환 과정은 테이블 생성 과정에서 정의한 r로 마스킹 값을 갱신하고(Step. 1), 정의된 k-bit 크기만큼 A, R을 나누어 변환을 진행하는 일련의 과정을 반복한다(Step. 3~8). 알고리즘에서 주목해 볼 점은 [Alg. 3]에 2번째 줄이다. 테이블 생성 과정에서 k-bit와 1-bit가 연접된 결과를 저장하는, 다시 말해 $(k+1)$-bit를 저장한다.

일반적으로 디바이스에 지원되는 메모리 처리 단위는 8-bit의 char형과 32-bit의 int형 등이 있다.

Algorithm. 4. Debrize's Conversion Algorithm

Input : (A,R) such that $x=A+r \bmod 2^{n \cdot k}$, r, ρ generated during precomputation phase
Output : $x'=(x'_0 \| ... \| x'_i \| ... \| x'_{n-1}) \oplus (r \| ... \| r \| ... \| r)$

1. $A \leftarrow A-(r \| ... \| r \| ... \| r) \bmod 2^{n \cdot k}$
2. $\beta \leftarrow \rho$
3. **for** $i=0$ to $n-1$ **do**
4. Split A into $A_h \| A_l$ and R into $R_h \| R_l$, such that A_l and R_l have size k
5. $A \leftarrow A+R_l \bmod 2^{(n-i) \cdot k}$
6. $\beta \| x'_i \leftarrow T[\beta \| A_l]$
7. $x'_i \leftarrow x'_i \oplus R_l$
8. $A \leftarrow A_h$ and $R \leftarrow R_h$
9. **end for**
10. **return** $x'=(x'_0 \| ... \| x'_i \| ... \| x'_{n-1}) \oplus (r \| ... \| r \| ... \| r)$

위와 같은 현실의 구현환경에서는 기존의 기법과 같은 1-bit 늘어난 결과를 저장하는데 적합하지 않다. 따라서 사전 연산의 결과 값을 효율적으로 저장하는 알고리즘의 설계가 필요하다.

III. 제안하는 알고리즘

본 장에서는 참조 테이블을 이용한 변환 기법을 기반으로 기존보다 더 적은 비용의 저장 공간을 이용하는 산술 마스킹에서 불 마스킹 변환 알고리즘을 제안한다. 제안하는 변환 알고리즘의 핵심 아이디어는 마스킹된 값의 최하위 비트(LSB, Least Significant Bit)의 경우 불 마스킹과 산술 마스킹이 같음을 이용하여 변환하려는 비트크기와 같은 크기의 저장 공간으로 참조 테이블을 구성함이다. 제안하는 알고리즘은 크게 2가지 버전(Version)으로 나눌 수 있다. 첫 번째는 사전연산(Pre-computation)을 통해 테이블을 생성하여 변환 알고리즘에 이용한다. 두 번째는 마스킹 값의 모든 가능한 경우를 전부 ROM에 저장하여 사전 연산을 배제하는 방법이다. 첫 번째의 경우 사전 연산으로 테이블을 생성하는 비용이 필요하지만, 연산에 사용되는 마스킹 값만으로 테이블을 구성하므로 작은 저장 공간을 사용한다. 두 번째의 경우 모든 경우의 수를 ROM에 저장하여 사전 연산 비용이 필요 없지만, 상대적으로 큰 테이블이 필요하다.

제안하는 마스킹 기법의 설명을 위해 표기법을 다음과 같이 정의한다.

- A : 산술 마스킹 상태 값
- B : 불 마스킹 상태 값
- x : 비밀 값 (연산 도중 노출이 일어나면 부채널 공격에 취약)
- R : 마스킹 값
- c : 올림수 비트 (A2B 연산 도중 리플-올림수 가산기에 의해 발생하는 올림수 비트, 연산도중 노출이 일어나면 부채널 공격에 취약)
- γ : 올림수 마스킹 값
- c_γ : 불 마스킹된 올림수 비트 ($c_\gamma = c \oplus \gamma$)
- A_h, A_l : 상위비트열, 하위비트열의 내부 상태 값
- R_h, R_l : 상위비트열, 하위비트열의 마스킹 값
- $\boxed{\ }$: 난수 값 (공격자가 예측하지 못함)

3.1 사전연산 테이블을 이용한 산술 마스킹에서 불 마스킹 변환 알고리즘

〔Alg. 5〕는 첫 번째 변환 알고리즘에 사용되는 테이블을 생성하는 알고리즘이다. 주목해 볼 점은 중간 연산 값인 tmp가 $A_LUT[i]$에 저장 될 때 LSB를 제외하고 저장되는 점이다. 실제 구현으로 예를 들면 $k=8$인 경우, 내부 상태변수인 tmp는 int형을 취하지만 저장되는 테이블 $A_LUT[i]$은 char형을 취함으로 Debraize등이 제안한 기존의 테이블 참조 알고리즘들에서 1-bit를 추가로 저장하여 생기는 공간의 비효율성을 줄일 수 있다.

〔Alg. 6〕는 사전 연산된 테이블을 이용한 산술 마스킹에서 불 마스킹 변환 알고리즘이다. 알고리즘은 산술 마스킹 상태의 A를 사전에 생성한 테이블의 변환하려는 크기인 k-bit 만큼 나누어 불 마스킹으로 변환한 후 B_i에게 저장하는 일련의 과정을 반복한다.

이 과정에서 생성되는 올림수 비트 c를 부채널 공격으로부터 안전하게 설계하기 위해 9~11줄의 과정이 필요하다. 각각의 연산 과정은 4장에 안전성 검증에서 확인할 수 있듯 모두 마스킹으로 보호되어 있다. 제안하는 알고리즘의 연산과정을 자세하게 설명하면 다음과 같다.

산술 마스킹에서 불 마스킹으로 변환 알고리즘은 산술 마스킹 상태의 하위 비트열 A_l을 불 마스킹 상태 B_i로 변환하고, 이 때 발생된 올림수 비트 c를 상위 비트열(아직 변환 연산이 일어나지 않은 비트열) A_h에 더하여 $(x+c-\boxed{R})$ 라는 값을 얻어야한다. 그러나 연산 도중 올림수 비트 c가 마스킹 없이 노출되면 부채널 분석에 취약하다. 즉, 우리는 마스킹된 $c \oplus \boxed{\gamma}$를 연산에 사용하여 $(x-\boxed{R})+(c \oplus \boxed{\gamma})$ 이

Algorithm 5. AtoB Table Generating

Input : -

Output : AtoB Table $A\_LUT[\,] \in Z_{2^k}$, $r \in Z_{2^k}$, $\gamma \in \{0,1\}$

1. Generate a random k-bit r and a random 1-bit γ
2. **for** i from 0 to (2^k-1) **do**
3. $tmp \leftarrow i+r \mod 2^{k+1}$
4. $tmp \leftarrow tmp \oplus (\gamma \| r)$
5. tmp is stored in $A\_LUT[i]$ without LSB
4. **end**

Algorithm 6. Proposal AtoB Algorithm

Input : ($A \in Z_{2^w}$, $R \in Z_{2^w}$, $r \in Z_{2^k}$, $\gamma \in \{0,1\}$)
such that $x = A + R \mod 2^w$,
r, γ generated during table generation
Output : ($B \in Z_{2^w}$)
such that $x = B \oplus R$

1. **for** i from 0 to $\lceil \frac{w}{k} \rceil - 2$ **do**
2. Split R into $R_h \| R_l$
with $R_h \in Z_{2^{w-((i+1)\cdot k)}}$, $R_l \in Z_{2^k}$
3. $A \leftarrow A - r \mod 2^{(w-i\cdot k)}$
4. $A \leftarrow A + R_l \mod 2^{(w-i\cdot k)}$
5. Split A into $A_h \| A_l$
with $A_h \in Z_{2^{w-((i+1)\cdot k)}}$, $A_l \in Z_{2^k}$
6. $tmp \leftarrow A\_LUT[A_l] \| A_l^{LSB}$
7. Split tmp into $c_\gamma \| B_i$
with $c_\gamma \in \{0,1\}$, $B_i \in Z_{2^k}$
8. $B_i \leftarrow (B_i \oplus R_l) \oplus r$
9. $A_h \leftarrow A_h + c_\gamma \mod 2^{w-((i+1)\cdot k)}$
10. $sign \leftarrow [c_\gamma \cdot (-2)] + 1 \mod 2^{w-((i+1)\cdot k)}$
11. $A_h \leftarrow A_h + (sign \cdot \gamma) \mod 2^{w-((i+1)\cdot k)}$
12. $A \leftarrow A_h$, $R \leftarrow R_h$
13. **end**
14. $i \leftarrow \lceil \frac{w}{k} \rceil - 1$
15. $A \leftarrow A - r \mod 2^{(w-i\cdot k)}$
16. $A \leftarrow A + R \mod 2^{(w-i\cdot k)}$
17. $B_i \leftarrow A\_LUT[A] \| A^{LSB} \mod 2^k$
18. $B_i \leftarrow (B_i \oplus R) \oplus r$
19. **return** $B_i \| B_{i-1} \| \ldots \| B_0$

란 값을 얻고, 그 결과에서 $\oplus\boxed{\gamma}$를 제거하는 과정이 필요하다.

〔Alg. 6〕의 9번째 줄에서 연산의 중간 값은 $(x - \boxed{R}) + (c \oplus \boxed{\gamma})$ 이다. 10번째 줄에서 $sign$이라는 값을 생성하고 이를 통해 11번째 줄에서 γ를 제거한다. 즉, $(x - \boxed{R}) + (c \oplus \boxed{\gamma}) + (sign \cdot \gamma) = (x + c - \boxed{R})$ 의 형태로 변형된다. 수식의 변화는 진리표인 〔Table 1〕를 보면 명백하다.

Table 1. Algorithm. 5. Line 9. Truth-table

c	γ	$sign$	$(c \oplus \boxed{\gamma}) + (sign \cdot \gamma)$
0	0	+1	0
0	1	-1	0
1	0	-1	1
1	1	+1	1

3.2 전수 저장 테이블을 이용한 산술 마스킹에서 불 마스킹 변환 알고리즘

〔Alg. 7〕,〔Alg. 8〕은 두 번째 방법인 전수 저장 테이블을 만드는 알고리즘과 그 테이블을 이용한 변환 알고리즘이다. 〔Alg. 7〕의 경우 암호기기가 동작할 때마다 실행되지 않고 알고리즘의 결과만이 ROM에 저장되어 있음을 가정한다. 전수 저장 테이블의 경우 상대적으로 큰 저장 공간을 필요로 하지만 임의의 마스킹 난수에 대해 사전연산 없이 변환을 진행할 수 있다. 따라서 구현 환경에 따라 적절한 알고리즘을 선택할 수 있다. 작동원리는 상기 설명한 알고리즘과 동일하다.

IV. 안전성 검증

본 장에서는 제안하는 마스킹 기법의 안전성을 확인하기 위해 알고리즘 수행 과정에서 나타날 수 있는 모든 중간 값을 확인하고 난수 값으로 마스킹 되어있는지 확인한다.

〔Alg. 5〕, 〔Alg. 7〕의 경우 테이블의 만드는 과정이다. 테이블의 경우 가능한 모든 경우 $(0,\ldots,2^k-1)$에 대해 연산을 진행하므로 비밀 정보의 누출이 없다.

〔Alg. 6〕, 〔Alg. 8〕에서 나타나는 중간 값의 노출은 각각 〔Table 2〕, 〔Table 3〕과 같다. 여기서 x와 c는 비밀정보에 해당한다. 〔Table 2〕, 〔Table 3〕에서 확인할 수 있듯이 연산도중 나타는

Algorithm. 7. A2B LUT Generating Function

Input : -
Output : AtoB Table $A\_LUT[][][] \in Z_{2^k}$,

1. **for** i from 0 to (2^k-1) **do**
2. **for** j from 0 to (2^k-1) **do**
3. $tmp \leftarrow i + r \mod 2^{k+1}$
4. $tmp_0 \leftarrow tmp \oplus (0 \| r)$
5. $tmp_1 \leftarrow tmp \oplus (1 \| r)$
6. tmp_0, tmp_1 is stored in $A\_LUT[0][i][j]$, $A\_LUT[1][i][j]$ without LSB, respectively
7. **end**
8. **end**

Algorithm. 8. A2B Conversion Algorithm

Input : ($A \in Z_{2^w}$, $R \in Z_{2^w}$)
such that $x = A + R \mod 2^w$
Output : ($B \in Z_{2^w}$)
such that $x = B \oplus R$

1. Generate a random 1-bit γ
2. **for** i from 0 to $\lceil \frac{w}{k} \rceil - 2$ **do**
3. Split R into $R_h \| R_l$
with $R_h \in Z_{2^{w-((i+1)\cdot k)}}$, $R_l \in Z_{2^k}$
4. Split A into $A_h \| A_l$
with $A_h \in Z_{2^{w-((i+1)\cdot k)}}$, $A_l \in Z_{2^k}$
5. $tmp \leftarrow A\_LUT[\gamma][A_l][R_l] \| A_l^{LSB}$
6. Split tmp into $c_\gamma \| B_i$
with $c_\gamma \in \{0,1\}$, $B_i \in Z_{2^k}$
7. $A_h \leftarrow A_h + c_\gamma \mod 2^{w-((i+1)\cdot k)}$
8. $sign \leftarrow [c_\gamma \cdot (-2)] + 1 \mod 2^{w-((i+1)\cdot k)}$
9. $A_h \leftarrow A_h + (sign \cdot \gamma) \mod 2^{w-((i+1)\cdot k)}$
10. $A \leftarrow A_h$, $R \leftarrow R_h$
11. **end**
12. $i \leftarrow \lceil \frac{w}{k} \rceil - 1$
13. $B_i \leftarrow A\_LUT[\gamma][A][R] \| A^{LSB} \mod 2^k$
14. **return** $B_i \| B_{i-1} \| \ldots \| B_0$

모든 비밀정보는 마스킹 값 R, γ, r으로 마스킹 되어 있다. 공격자는 마스킹 값을 알 수 없으므로 연산이 일어나는 과정의 중간 값을 추측할 수 없다. 그 결과 알고리즘은 부채널 분석에 안전하다.

Table 2. Intermediate Values of Algorithm 6

$V_3 \ldots V_{18}$ is intermediate values in computing at A2B algorithm
A ($= x - R$)
c: Carry bit

$V_2 = \boxed{R} \| \boxed{R}$
$V_3 = V_{15} = x - \boxed{R} - \boxed{r}$
$V_4 = V_{16} = x - \boxed{R} - \boxed{r} + \boxed{R_l} = x - \boxed{R_h \| r}$
$V_5 = (x - \boxed{R}) \| (x - \boxed{r})$
$V_6 = V_7 = V_{17} = c \oplus \boxed{\gamma} \| x \oplus \boxed{r}$
$V_8 = V_{18} = (x \oplus \boxed{R_l})$
$V_9 = (x - \boxed{R}) + (c \oplus \boxed{\gamma})$
$V_{10} = -2 \cdot (c \oplus \boxed{\gamma}) + 1$
$V_{11} = x + c - \boxed{r} \| x \oplus \boxed{R}$

Table 3. Intermediate Values of Algorithm 8

$V_3 \ldots V_9$ is intermediate values in computing at A2B algorithm
A ($= x - R$)
c: Carry bit

$V_1 = \boxed{\gamma}$
$V_3 = \boxed{R} \| \boxed{R}$
$V_4 = (x - \boxed{R}) \| (x - \boxed{R})$
$V_5 = V_6 = V_{13} = c \oplus \boxed{\gamma} \| x \oplus \boxed{R}$
$V_7 = (x - \boxed{R}) + (c \oplus \boxed{\gamma})$
$V_8 = -2 \cdot (c \oplus \boxed{\gamma}) + 1$
$V_9 = x + c - \boxed{r} \| x \oplus \boxed{R}$

V. Case Study

2012년 국가보안기술연구소에서 개발된 국산 암호 알고리즘 LEA(Lightweight Encryption Algorithm)[17]는 ARX(Addition, Rotation, XOR)구조의 암호 알고리즘 중 하나이다. 본 장에서는 전수 저장 테이블을 이용한 산술 마스킹에서 불 마스킹 변환 알고리즘을 LEA에 적용한다. 기본 골자는 비트연산자에 해당하는 순환이동(Rotation)과 XOR 연산에 경우 불 마스킹 형태를 취하고, 대수 연산에 해당하는 덧셈 연산은 산술 마스킹 형태를 취함을 기본으로 한다. 이 때 불 마스킹에서 산술 마스킹으로의 변환은 Goubin의 변환 알고리즘을 사용한다.

5.1 LEA(Lightweight Encryption Algorithm)

LEA는 128 비트 데이터 블록을 암호화 하는 블록 암호 알고리즘이다. LEA 라운드 함수는 32 비트 단위의 ARX(Addition, Rotation, XOR) 연산만으로 구성되어 있으며, 이들 연산을 지원하는 32 비트 소프트웨어 플랫폼에서 고속으로 동작한다. 또한 라운드 함수 내부의 ARX 연산 배치는 충분한 안전성을 보장하는 것과 동시에 S-box의 사용을 배제하여 경량 구현이 가능하다. 라운드 함수는 [Fig. 1]과 같으며 총 6개의 32 비트 라운드 키를 사용한다. 세부 사양은 다음과 같다.

- 기본구조: ARX 구조
- 입, 출력크기: 128 비트
- 키 크기: 128 / 192 / 256 비트

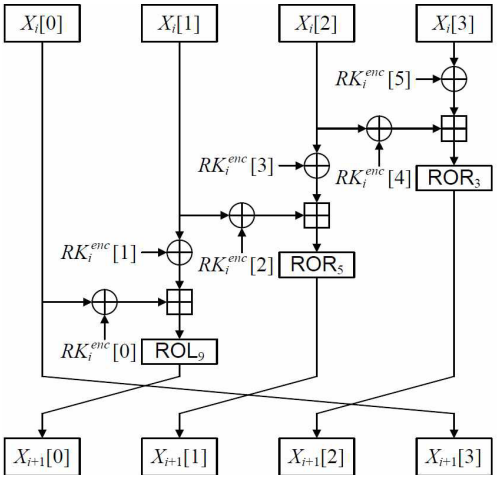

Fig. 1. LEA Encryption Round

- 라운드 키 크기: 32 비트 (총 6개)
- 라운드 수: 24 / 28 / 32 라운드

5.2 키 스케줄 함수

LEA의 암호화 키 스케줄 함수는 〔Alg. 9〕와 같다. LEA 라운드 함수는 192비트의 암호화 라운드 키가 필요하며, 키 스케줄 함수는 4개의 32비트 내부 상태 변수 T가 갱신되어 암호화 라운드 키를 생성한다. 스케줄 함수에 사용되는 상수들 δ는 다음과 같다.

δ〔0〕=0xC3EFE9DB, δ〔1〕=0x44626B02,
δ〔2〕=0x79E27C8A, δ〔3〕=0x78DF30EC,
δ〔4〕=0x715EA49E, δ〔5〕=0xC785DA0A,
δ〔6〕=0xE04EF22A, δ〔7〕=0xE5C40957

Algorithm. 9. LEA-128 Encryption Key Generating Function

Input : 128-bit Secret Key K
Output : 24 Rounds, 192-bit Encryption Round Key $RK_i^{enc}(0 \le i \le 23)$

1. T ← K
2. **for** i = 0 to 23 **do**
3. T〔0〕 ← $ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 4]))$
4. T〔1〕 ← $ROL_3(T[1] \boxplus ROL_{i+1}(\delta[i \bmod 4]))$
5. T〔2〕 ← $ROL_6(T[2] \boxplus ROL_{i+2}(\delta[i \bmod 4]))$
6. T〔3〕 ← $ROL_{11}(T[3] \boxplus ROL_{i+3}(\delta[i \bmod 4]))$
7. RK_i^{enc} ← (T〔0〕,T〔1〕,T〔2〕,T〔1〕,T〔3〕,T〔1〕)
8. **end for**

5.3 마스킹 적용

마스킹이 적용된 LEA에 암/복호화 과정의 전체 흐름은 다음과 같다.

1) 32-bit 난수 M_k를 생성한 후, 키 스케줄 함수로 라운드 키를 생성한다. (다음 소절에서 상세히 설명한다.)
2) 4개의 32-bit 난수 $m_{0,j} j=\{0,1,2,3\}$ 를 생성한 후 입력문과 xor한다.
3) 라운드가 끝난 후, 마스킹 값 $m_{0,j}$를 갱신한다.
4) 암/복호화 과정이 끝난 후, 출력문의 마스킹 값을 벗겨낸다.

위 순서에 따른 전체 구조에 대한 대응기법은 〔Fig. 2〕, 〔Fig. 3〕과 같다.
〔Fig. 2〕, 〔Fig. 3〕에 표기된 MA와 MS는 〔Alg. 10〕과 같이 처리한다.
이 때 i+1번째 암호화 라운드에서 갱신되는 마스킹 값 $m_{i+1,j}$은 다음과 같다.

$$
\begin{aligned}
m_{i+1,0} &= ROL_9((m_{i,0} \oplus M_k) + (m_{i,1} \oplus M_k)) \\
m_{i+1,1} &= ROR_5((m_{i,1} \oplus M_k) + (m_{i,2} \oplus M_k)) \\
m_{i+1,2} &= ROR_3((m_{i,2} \oplus M_k) + (m_{i,3} \oplus M_k)) \\
m_{i+1,3} &= m_{i,0}
\end{aligned} \tag{1}
$$

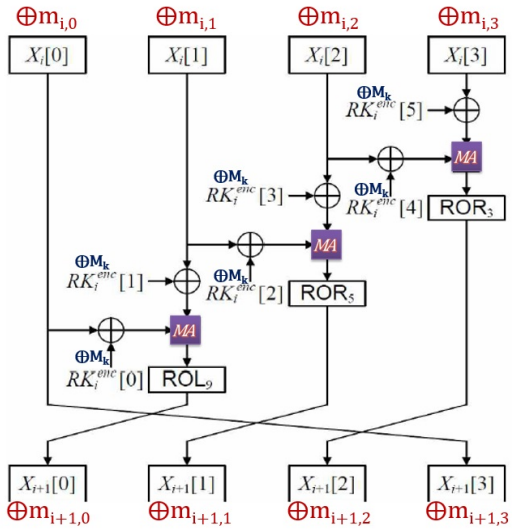

Fig. 2. 1st Masking LEA Encryption Round

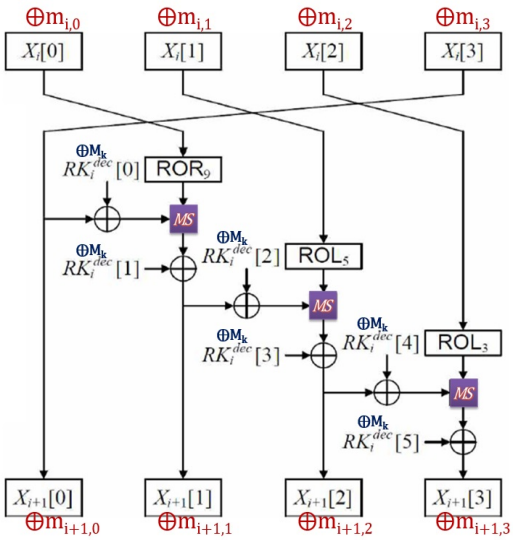

Fig. 3. 1st Masking LEA Decryption Round

Algorithm. 10. Masking Addition / Subtraction Algorithm	
Input : $X'(=X\oplus m_x), Y'(=Y\oplus m_y)$ $X', Y' \in Z_{2^w}$	
Output : $Z'(=(X\pm Y)\oplus(m_x \pm m_x))$ $Z' \in Z_{2^w}$	
1. $tmp_X \leftarrow$ Goubin_B2A(X')	
2. $tmp_Y \leftarrow$ Goubin_B2A(Y')	
3. $tmp_{Z_A} \leftarrow$ $tmp_X \pm tmp_Y$	
4. $tmp_{Z_B} \leftarrow$ Proposal A2B Algorithm(tmp_{Z_A})	
5. **return** tmp_{Z_B}	

i+1번째 복호화 라운드에서 갱신되는 마스킹 값 $m_{i+1,j}$은 다음과 같다.

$$
\begin{aligned}
m_{i+1,0} &= m_{i,3} \qquad (2)\\
m_{i+1,1} &= (ROR_9(m_{i,0}) - (m_{i+1,0}\oplus M_k))\oplus M_k\\
m_{i+1,2} &= (ROL_5(m_{i,1}) - (m_{i+1,1}\oplus M_k))\oplus M_k\\
m_{i+1,3} &= (ROL_3(m_{i,2}) - (m_{i+1,2}\oplus M_k))\oplus M_k
\end{aligned}
$$

키 스케줄링의 경우, 고정된 비밀키에 대한 전력 소모량 만을 측정하므로 DPA 공격을 고려하지 않는다. 따라서 비밀키의 헤밍 웨이트(Hamming Weight)의 노출을 막는 것에 초점을 둔다.

128-bit LEA 키 스케줄 함수의 경우 4개의

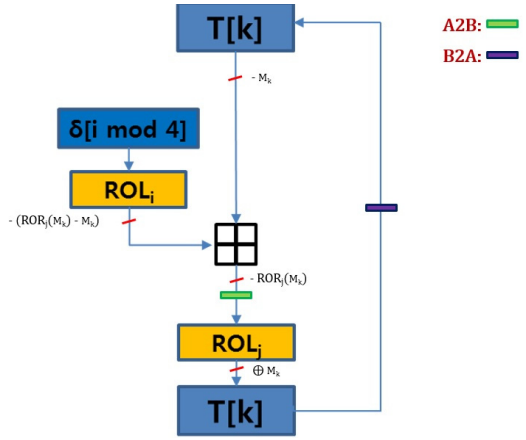

Fig. 4. 1st Masking LEA Key Generation

32-bit 내부 상태 변수 $T[k]k=\{0,1,2,3\}$ 를 갱신해 가며 라운드 키를 생성한다. 이 때 각각의 내부 상태 변수는 독립적으로 연산된다.

〔Fig. 4〕는 키 스케줄링에 대한 마스킹 기법이다. 초기의 T〔k〕는 $-M_k$로 마스킹된 산술 마스킹 상태이며, 덧셈 연산 후 불 마스킹으로 변환된다. 이 때 마스킹의 형태를 유지하기 위해 키 스케줄링 상수 δ는 로테이션 연산 후 $-(ROR_j(M_k)-M_k)$ 의 형태로 마스킹한다. 불 마스킹 상태의 T〔k〕를 라운드 키로 사용하며, 다시 T〔k〕를 갱신하게 위해 산술 마스킹으로 변환한다. 다른 3개의 T에 대해서도 같은 방법으로 마스킹 된 라운드 키를 얻을 수 있다.

VI. 비교 및 결론

본 논문에서는 저장 공간 측면에서 효율적인 산술 마스킹에서 불 마스킹 변환 알고리즘을 구성하고 이를 LEA에 적용시켰다. 〔Table 4, 5〕에서 Table Size는 사용하는 메모리이다. Time, Case Study 경우 연산에 소요된 Clock 수와 괄호안의 숫자는 Debraize를 1로 보고 계산한 상대적 비율이다.

〔Table 4〕는 첫 번째 제안한 알고리즘과 Debraize의 알고리즘을 8-bit 프로세서 기반인 Atmega128에서 시뮬레이션한 결과이다. 두 번째 제안한 알고리즘의 경우 테이블을 저장하는 크기가 저사양 프로세서 기반의 실험환경에 적용하기 적절하지 않아 제외하였다. 본 측정은 AVR studio에서 gcc -O2 컴파일러 환경에서 시뮬레이션 되었다. 제안하는 첫 번째 알고리즘의 경우 Debraize의 알

Table 4. Compare Proposals and Debraize's simulated on Atmega128(8-bit microcontroller)

	Standard	Proposal 1	Debraize
Table Size(RAM)	-	$(2^8)\cdot(2^8\text{-bit})$	$(2\cdot2^8)\cdot(2^9\text{-bit})$
Computing Time(Table gen)	-	7,196 (0.49)	14,639 (1)
Computing Time(conversion only)	-	464 (0.94)	492 (1)
Case Study(LEA)	22,857	331,061 (0.97)	340,589 (1)

Table 5. Compare Proposals and Debraize's simulated on ARM(32-bit microcontroller)

	Standard	Proposal 1	Proposal 2	Debraize
Table Size (RAM)	-	$(2^8)\cdot(2^8\text{-bit})$	-	$(2\cdot2^8)\cdot(2^9\text{-bit})$
Table Size (ROM)	-	-	$(2\cdot2^8\cdot2^8)\cdot(2^8\text{-bit})$	-
Computing Time (Table gen)	-	4,872 (0.44)	-	11,074 (1)
Computing Time (conversion only)	-	214 (0.78)	227 (0.82)	276 (1)
Case Study (LEA)	7,553	47,975 (0.81)	43,507 (0.74)	58,988 (1)

고리즘과 비교해서 테이블 생성시간과 변환연산 시간은 각각 약 50.8%, 5.7% 빠르며 테이블 크기는 1/4배임을 알 수 있다. 그러나 실제 구현에서 Debraiz의 알고리즘은 16-bit의 long형 저장 공간을 사용하며, 제안하는 알고리즘은 8-bit의 저장 공간을 사용한다. 즉, 실제 구현환경에서는 제안하는 알고리즘이 더욱 효율적인 저장 공간에 운영이 가능하다. LEA에 적용의 경우 마스킹이 적용되지 않은 기본 LEA 보다 약 14.48배 추가 연산이 필요하지만, Debraize와 비교해서 약 2.8% 성능 향상을 확인할 수 있다.

〔Table 5〕는 32-bit 프로세서 기반의 ARM 시뮬레이터를 통해 첫 번째, 두 번째 제안한 알고리즘과 Debraize 알고리즘을 비교한 표이다. 두 번째 제안한 알고리즘의 경우, 상대적으로 큰 테이블에서 값을 참조해 옴으로 변환시간이 첫 번째 제안하는 알고리즘보다 다소 느리지만 테이블 연산시간이 없는 이점이 있다. 이 결과 실제 1st Masking LEA에 적용했을 때 첫 번째 제안하는 알고리즘보다 9.3%, Debraize와 비교에선 26.2% 성능이 향상된다. 8-bit 프로세서보다 32-bit 프로세서에서 성능이 더 상향된 이유는 다음과 같다. Computing Time(conversion only)에서 실제 줄어든 Clock 수는 두 경우 모두 약 30~40 정도로 비슷한 반면 기본적으로 처리되는 Clock이 8-bit의 경우 400~500 Clock, 32-bit의 경우 200~300 Clock으로 더 빠르다. 이로 인해 줄어든 Clock이 미치는 영향이 더 커져서 비교 퍼센트가 더 크게 나타난 것으로 분석된다.

마스킹 기법은 부채널 공격에 효과적인 대응기법으로 알려져 있다. 그러나 마스킹 기법은 최근의 Glitch 공격 등으로 대표되는 오류주입공격에 매우 취약함이 잘 알려져 있다. 따라서 논문에 제안한 마스킹 기법에 오류주입공격을 탐지하는 대응기법이 적용된 향상된 변환 마스킹 알고리즘에 대한 연구가 필요로 할 것이다.

References

〔1〕 P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," CRYPTO'9, pp.388-397, Springer-Verlag, Dec. 1999.
〔2〕 Kocher, Paul C. "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," Advances in Cryptology—CRYPTO'96. Springer

Berlin Heidelberg, July. 1996.
〔3〕 Örs, Siddika Berna, et al. "Power-Analysis Attack on an ASIC AES implementation," Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on. Vol. 2. IEEE, April. 2004.
〔4〕 Agrawal, Dakshi, et al. "The EM side-channel (s)," Cryptographic Hardware and Embedded Systems-CHES 2002. Springer Berlin Heidelberg, 29-45. Feb. 2003.
〔5〕 Gandolfi, Karine, Christophe Mourtel, and Francis Olivier. "Electromagnetic analysis: Concrete results," Cryptographic Hardware and Embedded Systems—CHES 2001. Springer Berlin Heidelberg, Sep. 2001.
〔6〕 ISO/IEC 15408-1/2/3, "Common Criteria for Information Technology Security Evaluation," Ver. 3.1, Revision 4, Sep. 2012.
〔7〕 "FIPS-140 -3: DRAFT Security Requirements for Cryptographic Modules (Revised Draft)," NIST. 2013-03-07. Retrieved May. 2013.
〔8〕 "Integrated Circuit Card Specifications for Payment Systems," EMVCo. Retrieved 26 March. 2012.
〔9〕 E. Oswald and K. Schramm. "An Efficient Masking Scheme for AES Software Implementations," WISA 2005, LNCS 3786, pp. 292 - 305, Springer, Aug. 2006.
〔10〕 J. Bl¨omer, J. Guajardo, and V. Krummel. "Provably Secure Masking of AES," SAC2004, LNCS 3357, pp. 69 - 83, Springer, Aug. 2005.
〔11〕 Thomas S. Messerges, "Power Analysis Attacks and Countermeasures for Cryptographic Algorithms," Ph.D Thesis, pp.541-548, Feb. 2000.
〔12〕 Jean-Sébastien Coron, Louis Goubin, "On Boolean and Arithmetic Masking against Differential Power Analysis," CHES 2000, Lecture Notes in Computer Science Volume 1965, pp 231-237, Dec. 2000.
〔13〕 Louis Goubin, "A Sound Method for Switching between Boolean and Arithmetic Masking," CHES 2001, Lecture Notes in Computer Science Volume 2162, pp 3-15, July. 2001.
〔14〕 Coron, Jean-Sébastien, et al. "Conversion from arithmetic to boolean masking with logarithmic complexity," Fast Software Encryption. Springer Berlin Heidelberg, Aug. 2015.
〔15〕 Coron, Jean-Sébastien, and Alexei Tchulkine. "A new algorithm for switching from arithmetic to boolean masking," Cryptographic Hardware and Embedded Systems-CHES 2003. Springer Berlin Heidelberg, 89-97, Sep. 2003.
〔16〕 Debraize, Blandine. "Efficient and provably secure methods for switching from arithmetic to boolean masking," Cryptographic Hardware and Embedded Systems - CHES2012. Springer Berlin Heidelberg, 107-121, Sep. 2012.
〔17〕 J. Park et al., "128-Bit Block Cipher LEA,"TTAK.KO-12.0223, Dec. 2013

〈저자소개〉

김 한 빛 (HanBit Kim) 학생회원
2014년 2월: 고려대학교 신소재공학 학사
2014년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 부채널 공격, 부채널 대응기법, 암호시스템 안전성 분석 및 고속구현

김 태 원 (TaeWon Kim) 학생회원
2010년 2월: 광운대학교 수학과 학사
2012년 8월: 고려대학교 정보보호대학원 석사
2012년 8월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 부채널 공격, 스마트 카드 보안, 암호시스템 안전성 분석 및 고속구현

김 희 석 (HeeSeok Kim) 정회원
2006년: 연세대학교 수학과 학사
2008년: 고려대학교 정보보호대학원 공학석사
2011년: 고려대학교 정보보호대학원 공학박사
2011년 9월~2012년 12월: Bristol University 박사후 연구원
2013년~현재: 한국과학기술정보연구원 (KISTI) 첨단연구망정보보호실 선임연구원
2015년~현재: 과학기술연합대학원대학교(UST) 조교수
〈관심분야〉 부채널 공격, 암호시스템 안전성 분석 및 고속구현, 암호칩 설계 기술, 보안관제, 네트워크 보안

홍 석 희 (Seokhie Hong) 종신회원
1995년 : 고려대학교 수학과 학사
1997년 : 고려대학교 수학과 석사
2001년 : 고려대학교 수학과 박사
1999년 8월~2004년 2월 : (주)시큐리티 테크놀로지스 선임연구원
2003년 3월~2004년 2월 : 고려대학교 정보보호기술연구센터 선임연구원
2004년 4월~2005년 2월 : K.U. Leuven ESAT/SCD-COSIC 박사후연구원
2005년 3월~2013년 8월: 고려대학교 정보보호대학원 부교수
2013년 9월~현재 : 고려대학교 정보보호대학원 정교수
〈관심분야〉 대칭키 암호 알고리즘, 공개키 암호 알고리즘, 디지털 포렌식