Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.26, NO.6, Dec. 2016 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2016.26.6.1431

LEA에 대한 마스킹 기반 부채널분석 대응기법에 관한 분석

김 창 균,†‡ 박 제 훈, 한 대 완, 이 동 훈
한국전자통신연구원 부설연구소

Investigation of Masking Based Side Channel Countermeasures for LEA

ChangKyun Kim,†‡ JaeHoon Park, Daewan Han, Dong Hoon Lee
The Attached Institute of ETRI

요 약

ARX 구조를 가지는 블록암호알고리즘에 마스킹 대응기법을 적용할 경우 연산별 마스킹 방식의 차이로 인하여 방식 간 안전한 변환이 반드시 필요하다. 그러나 마스킹 변환 시 발생되는 많은 연산량으로 인하여 ARX 기반 알고리즘에 마스킹 대응기법을 적용하는 것은 AES와 같이 하나의 마스킹 방식을 적용하는 알고리즘보다 상대적으로 비효율적이라고 알려져 있다. 본 논문에서는 현재까지 제안된 다양한 마스킹 변환 기법을 이용하여 1차 부채널분석에 안전한 LEA를 설계하고 32비트 플랫폼에 구현한다. 이를 바탕으로 대응기법의 예상되는 이론적 연산량과 실제 측정한 연산량간 발생하는 차이점에 대해 구현관점에서 살펴본다. 아울러 T-test를 활용하여 본 논문에서 구현한 대응기법이 실제 안전한지를 실험적으로 검증한다.

ABSTRACT

In case of ARX based block cipher algorithms with masking countermeasures, there is a need for a method to convert between Boolean masking and arithmetic masking. However, to apply masking countermeasures to ARX based algorithms is less efficient compared to masked AES with single masking method because converting between Boolean and arithmetic masking has high computation time. This paper shows performance results on 32-bit platform implementations of LEA with various masking conversion countermeasures against first order side channel attacks. In the implementation point of view, this paper presents computation time comparison between actual measurement value and theoretical one. This paper also confirms that the masked implementations of LEA are secure against first order side channel attacks by using a T-test.

Keywords: Masking countermeasure, Side channel attack, LEA

I. 서 론

부채널분석은 하드웨어 암호모듈 분석 시 수학적 기반을 둔 여타 분석기법보다 매우 위협적이며 실효적인 분석기법으로 알려져 있다[1]. 따라서 스마트카드와 같은 임베디드 디바이스에 암호알고리즘을 구현할 경우 부채널분석에 대한 대응기법은 반드시 고려해야 할 요소기술로 인식되고 있다.

부채널분석이 소개된 이후 다양한 방법의 대응기법이 연구되고 있으며, 그 중 마스킹 기반 대응기법은 가장 많이 사용되는 대응기법 중 하나이다. 특히 랜덤 노이즈와 같이 하드웨어 단계의 대응기법이 지원되지 않는 구현환경에서 마스킹 대응기법은 매우 효율적인 기법으로 사용될 수 있다.

대부분의 블록암호알고리즘은 테이블 참조 연산, 불(Boolean) 연산, 산술(arithmetic) 연산 등으로

Received(09. 27. 2016), Modified(12. 07. 2016), Accepted(12. 10. 2016)
† 주저자, kimck@nsr.re.kr
‡ 교신저자, kimck@nsr.re.kr(Corresponding author)

이루어져 있다. 블록암호알고리즘에 마스킹 대응기법을 적용할 경우 연산별 마스킹 대응기법이 다르게 적용되어 서로 간 안전한 변환이 필요하다. 특히 LEA[2]와 같이 ARX(Addition, Rotation, Xor) 구조를 가지는 알고리즘에서는 안전한 마스킹 변화기법이 반드시 고려되어야 할 요소기술이다. 그러나 초창기 제안되었던 마스킹 변환 기법[3,4]는 상호 마스킹 변환 시 많은 연산이 소요되어 경량화, 고속화를 목적으로 하는 ARX 기반 알고리즘에 마스킹 대응기법을 적용하는데 가장 큰 걸림돌로 작용하고 있다.

위와 같은 문제점을 해결하기 위하여 Goubin이 제안한 마스킹 변환 기법에 룩업 테이블(look-up table)을 이용하여 속도를 향상시킨 다양한 대응기법이 제안되었다[5-7]. 최근에는 앞서 설명한 기법과 달리 마스킹 변환이 필요 없는 Secure Addition 기반의 마스킹 기법이 제안되었다[8-10].

본 논문에서는 현재까지 제안된 다양한 마스킹 변환 기법을 구현해보고, 예상되는 이론적 연산량과 실제 측정한 연산량에 어떠한 차이점이 발생되는지를 구현관점에서 살펴본다. 이를 바탕으로 여러 가지 마스킹 변환 기법의 효율성을 비교 분석한다. 또한 다양한 마스킹 변환 기법을 LEA에 적용하여 성능을 비교 분석해 본다. 그리고 LEA는 32비트 플랫폼에서 효율적으로 동작할 수 있도록 설계된 알고리즘이므로 32비트 플랫폼을 대상으로 위 대응기법을 구현했을 때 어느 정도의 성능 감소가 발생되지는 살펴본다. 아울러 본 논문에서 구현한 대응기법이 실제 안전한지를 검증하기 위하여 암호모듈의 부채널분석에 대한 안전성 검증방안으로 연구되고 있는 T-test를 활용하여 검증한다.

본 논문의 구성은 다음과 같다. 2장에서는 지금까지 제안된 마스킹 변환 기법에 대해 살펴보고, 3장에서는 2장에서 설명한 대응기법을 활용한 LEA에 대한 마스킹 기반 대응기법을 소개한 후 4장에서 구현 결과를 분석한다. 5장에서는 논문에서 소개한 LEA에 대한 마스킹 기반 대응기법의 안전성 분석 결과를 설명하고, 마지막으로 간단한 요약과 함께 6장에서 결론을 맺는다.

II. 마스킹 기반 대응기법

본 장에서는 부채널분석을 대응하기 위한 불-산술 상호 마스킹 변환 기법에 대해 설명한다. 다음은 기본적인 불 마스킹과 산술 마스킹 수식이다.

- 불 마스킹 : $x' = x \oplus r_x$
- 산술 마스킹 : $A = x - r_x \bmod 2^k$

여기서 x는 마스킹될 원래의 값을 의미하며, x' (A)는 난수 r_x로 마스킹된 값을 의미한다. 또한 k는 처리되는 데이터의 비트 크기를 뜻한다.

2.1 산술-불 상호 마스킹 변환 기법

산술-불 마스킹 변환 기법은 Messerges에 의해 처음으로 제안[3]되었으며, 이후 그에 대한 안전성 및 효율성 향상에 대한 연구들이 진행되고 있다[4-10]. 본 절에서는 현재까지 제안된 기법 중 안전하다고 알려진 상호 마스킹 변환 기법에 대해 설명한다.

2.1.1 불-산술 마스킹 변환 기법

Messerges가 제안한 불-산술 마스킹 변환 기법[3]의 취약성이 밝혀진 이후 Goubin은 Fig. 1과 같이 새로운 방식의 불-산술 마스킹 변환 기법을 제안하였다[4].

Fig. 1에서 불 마스킹된 x'과 난수 r_x을 입력하면 산술 마스킹된 A를 얻을 수 있고, 이를 위해서는 일곱 번의 기본연산(다섯 번의 XOR, 두 번의 뺄셈)이 필요하다. 여기서 기본연산이란 XOR, AND, OR, 뺄셈, 덧셈, 쉬프트 등을 의미한다. 위 불-산술 마스킹 변환 기법은 다음에 설명할 산술-불 마스킹 변환 기법에 비해 연산량이 매우 적고 개선의 여지가 거의 없어 대부분의 마스킹 변환 기법에서 참조하고 있다.

Input : x', r_x	
Output : A	
1. $\Gamma = \gamma$	5. $\Gamma = \Gamma \oplus r_x$
2. $T = x' \oplus \Gamma$	6. $A = x' \oplus \Gamma$
3. $T = T - \Gamma$	7. $A = A - \Gamma$
4. $T = T \oplus x'$	8. $A = A \oplus T$

Fig. 1. Boolean to arithmetic masking conversion algorithm

2.1.2 Goubin 산술-불 마스킹 변환 기법

[4]에서 제안된 산술-불 마스킹 변환 기법은 데이터의 비트 크기만큼 반복되는 구조를 가지고 있다(이

하 Goubin 기법). 결국 알고리즘 내부 루프로 인해 불-산술 마스킹 변환 기법에 비해 매우 많은 연산량을 가진다.

Goubin 기법은 내부 루프 때문에 데이터의 비트 크기에 비례하여 연산량이 증가하므로 k 비트 데이터를 처리할 경우 $(5k+5)$번의 기본연산(XOR $(2k+4)$번, AND $(2k+1)$번, shift k번)이 필요하다. 따라서 32 비트 데이터의 경우 165번의 기본연산이 소요된다.

Input : A, r_x
Output : x'

1. $\Gamma=\gamma$
2. $T=2\Gamma$
3. $x'=\Gamma\oplus r_x$
4. $\Omega=\Gamma\wedge x'$
5. $x'=T\oplus A$
6. $\Gamma=\Gamma\oplus x'$
7. $\Gamma=\Gamma\wedge r_x$
8. $\Omega=\Omega\oplus\Gamma$
9. $\Gamma=T\wedge A$
10. $\Omega=\Omega\oplus\Gamma$
11. for $i=1$ to $k-1$ do
11.1 $\Gamma=T\wedge r_x$
11.2 $\Gamma=\Gamma\oplus\Omega$
11.3 $T=T\wedge A$
11.4 $\Gamma=\Gamma\oplus T$
11.5 $T=2\Gamma$
12. $x'=x'\oplus T$

Fig. 2. Goubin method

2.1.3 CT 산술-불 마스킹 변환 기법

Coron은 Goubin이 제안한 산술-불 마스킹 변환 기법의 연산량을 개선하기 위해 룩업 테이블을 이용한 방법을 제안하였다[5].

[6]은 Fig. 3과 같이 [5]에서 제안된 마스킹 변환 기법에 취약점을 개선하고 연산 효율성을 높인 수정된 방법을 소개하였다(이하 CT 기법). 알고리즘의 기본 개념은 $x'=(A+r_x)\oplus r_x$를 k 비트 단위의 $(A+r_x)$ 값과 연산 시 발생하는 캐리(carry)를 미리 계산해 두고, 변환 단계에서 미리 계산된 값을 이용하는 것이다. 결국, CT 기법은 k 비트 크기의 룩업 테이블 계산 단계와 변환 단계에 의해 연산량이 결정된다.

O. Neiβe 역시 CT 기법과 유사한 방법을 제시하였다[7]. CT 기법과 차이점은 캐리 값을 사전 연산 단계에서 마스킹하지 않고 변환 단계에서 마스킹하여 메모리 공간을 줄일 수 있다는 점이다. 이 방법 역시 [6]에서 효율성이 개선된 방법이 소개되었다. 하지만 본 논문에서는 [7]에서 제안된 기법이 CT 기법과 연산량이 거의 같아 더 이상 다루지 않는다.

Input : A, r_x
Output : x'

[Pre-computation]
1. Generate random number k-bit s, $((n-1)\cdot k)$-bit γ
2. $\Gamma=\sum_{i=1}^{k-1}2^{i\cdot k}\cdot\gamma \bmod 2^{n\cdot k}$
3. $\gamma'=\gamma\| s$
4. for $A=0$ to 2^k-1 do
4.1 $T[A]=(A+\gamma' \bmod 2^{n\cdot k})\oplus s$

1. $A=A-(s\|s\|...\|s)\bmod 2^{n\cdot k}$
2. $A=A-\Gamma \bmod 2^{n\cdot k}$
3. for $i=0$ to $n-1$ do
3.1 $A=A_h\|A_l$, $r_x=r_{xh}\|r_{xl}$ (k-bit A_l, r_{xl})
3.2 $A=A+r_{xl}\bmod 2^{(n-i)\cdot k}$
3.3 $A=A_h\|0+T[A_l]\bmod 2^{n\cdot k}$
3.4 $x_i'=A_l\oplus r_{xl}$
3.5 $A=A_h$, $r_x=r_{xh}$
4. $x'=(x_0'\|x_1'\|...\|x_{n-1}')\oplus(s\|s\|...\|s)$

Fig. 3. CT method

2.1.4 Debraize 산술-불 마스킹 변환 기법

Debraize는 [5,7]에서 제안되었던 대응기법의 안전성과 효율성에 대한 문제점을 지적하고 룩업 테이블 기반의 새로운 방법을 제안하였다(이하 Debraize 기법)[6].

Debraize 기법의 특징은 사전 연산 단계에서 메모리 참조 시 발생할 수 있는 누설 정보를 제거하기 위해 1비트 난수로 메모리 주소를 마스킹하는 점과, 기존 [5,7]에서 제안된 기법의 변환 단계를 수정하여 연산 효율성을 높인 점이다.

2.2 Secure Addition 마스킹 기법

불 마스킹된 두 개의 데이터를 더하기 위해서는 불-산술 마스킹 변환, 덧셈, 산술-불 마스킹 변환의 3단계를 거쳐야 한다. 다음에서 설명할 Secure Addition (이하 SA 마스킹 기법)은 마스킹 변환의 번거로움을 줄이고 연산량을 개선시키고자 제안된 방법이다. 즉,

Input : A, r_x
Output : x'

〔Pre-computation〕
1. Generate random number k-bit s, 1-bit ρ
2. for $A=0$ to 2^k-1 do
2.1 $T[\rho\|A]=(A+s)\oplus(\rho\|s)$
2.2 $T[(\rho\oplus1)\|A]=(A+s+1)\oplus(\rho\|s)$

1. $A=A-(s\|s\|...\|s)\bmod 2^{n\cdot k}$
2. $\beta=\rho$
3. for $i=0$ to $n-1$ do
3.1 $A=A_h\|A_l$, $r_x=r_{xh}\|r_{xl}$ (k-bit A_l, r_{xl})
3.2 $A=A+r_{xl}\bmod 2^{(n-i)\cdot k}$
3.3 $\beta\|x_i'=T[\beta\|A_l]$
3.4 $x_i'=x_i'\oplus r_{xl}$
3.5 $A=A_h$, $r_x=r_{xh}$
4. $x'=(x_0'\|x_1'\|...\|x_{n-1}')\oplus(s\|s\|...\|s)$

Fig. 4. Debraize method

SA 마스킹 기법은 불 마스킹된 두 개의 데이터를 입력받아 불 마스킹된 덧셈 결과를 출력하는 방법이다.

2.2.1 KRJ SA 마스킹 기법

Karroumi는 기존의 불-산술 상호 마스킹 변환의 번거로움과 연산량을 개선하기 위해 불 마스킹된 두 개의 데이터를 입력받아 불 마스킹된 덧셈 결과를 안전하게 출력할 수 있는 SA 마스킹 기법을 처음으로 제안하였다(이하 KRJ 기법)〔8〕.

Input : x', y', r_x, r_y
Output : z'

1. $C=\gamma$	10. $B=\Omega\ll1$
2. $T=x'\wedge y'$	11. $C=C\ll1$
3. $\Omega=C\oplus T$	12. $z'=x'\oplus y'$
4. $T=x'\wedge r_y$	13. $r_z=r_x\oplus r_y$
5. $\Omega=\Omega\oplus T$	14. $T=C\wedge z'$
6. $T=y'\wedge r_x$	15. $\Omega=\Omega\oplus T$
7. $\Omega=\Omega\oplus T$	16. $T=C\wedge r_z$
8. $T=r_x\wedge r_y$	17. $\Omega=\Omega\oplus T$
9. $\Omega=\Omega\oplus T$	
18. for $i=2$ to $k-1$ do	
18.1 $T=B\wedge z'$	18.4 $B=B\oplus T$
18.2 $B=B\wedge r_z$	18.5 $B=B\ll1$
18.3 $B=B\oplus\Omega$	
19. $z'=z'\oplus B$	20. $z'=z'\oplus C$

Fig. 5. KRJ method

KRJ 기법의 기본 개념은 AND, XOR, double (shift)로 이루어진 기본적 덧셈 알고리즘(AND-XOR-and-double 기법)에 마스킹 기법을 적용한 것이다. 〔4〕에서 제안한 불-산술 상호 마스킹 변환 기법을 이용할 경우 2번의 불-산술 마스킹 변환과 1번의 산술-불 마스킹 변환, 덧셈이 필요하여 총 $(5k+21)$의 기본연산이 소요되지만, KRJ 기법은 $(5k+8)$의 기본연산이 소요되어 연산량을 줄일 수 있다.

2.2.2 VG SA 마스킹 기법

Vadnala는 KRJ 기법의 연산 효율성을 향상시키고자 룩업 테이블을 이용한 SA 마스킹 기법을 제안하였다(이하 VG 기법)〔9〕.

본 논문에서는 기존 VG 기법의 연산 효율성을 높이기 위해 k 비트씩 n번 처리되는 x', y', z'을 루프 밖에서 한 번에 연산될 수 있도록 수정하였다 (Fig. 6 참조). VG 기법의 기본 개념은 $z'=(x+y)\oplus r_x\oplus r_y$를 k비트 단위의 $(x\oplus s_1)+(y\oplus s_2)$ 값과 연산 시 발생하는 캐리를 미리 계산하여 변환

Input : x', y', r_x, r_y
Output : z'

〔Pre-computation〕
1. Generate random number k-bit s_1, s_2, 1-bit ρ
2. for $A=0$ to 2^k-1 do
2.1 for $B=0$ to 2^k-1 do
2.1.1 $C=(A\oplus s_1)+(B\oplus s_2)$
2.1.2 $T[\rho\|A\|B]=C\oplus(\rho\|s_1)$
2.1.3 $T[(\rho+1)\|A\|B]=(C+1)\oplus(\rho\|s_1)$

1. $\beta=\rho$
2. $x'=x'\oplus(s_1\|s_1\|...\|s_1)\oplus r_x$
3. $y'=y'\oplus(s_2\|s_2\|...\|s_2)\oplus r_y$
2. for $i=0$ to $n-1$ do
2.1 $\beta\|z_i'=T[\beta\|x_i'\|y_i']$
3. $z'=(z_1'\|z_2'\|...\|z_{n-1}')\oplus r_x\oplus r_y$
4. $z'=z'\oplus(s_1\|s_1\|...\|s_1)$

Fig. 6. VG method

단계에서 이용하는 것이다.

2.2.3 CGTV SA 마스킹 기법

[10]에서는 KRJ 기법과 달리 Kogge-Stone 올림수 예견 덧셈기 기반의 마스킹 변환 기법을 제안하였다(이하 CGTV 기법). Goubin 기법이나 KRJ 기법의 경우 $O(k)$의 연산 복잡도를 가지지만 CGTV 기법은 $O(\log k)$ 의 연산 복잡도를 가지기 때문에 더욱 효율적이다.

CGTV 기법의 총 연산량은 $k=2^n$인 연산환경에서 $(28n+4)$를 가진다. 만약 32비트 데이터를 가정할 경우 CGTV 기법과 KRJ 기법의 연산량은 각각 144, 168의 기본연산이 소요되어 CGTV 기법이 KRJ 기법에 비해 효율적이라 할 수 있다.

Fig. 7에서 SecXor, SecAnd, SecShift는 마스킹 기반의 안전한 XOR, AND, Shfit 연산을 뜻하지만 상세한 내용은 본 논문에서 생략한다([10] 참조).

Input : x', y', r_x, r_y
Output : z'

1. Generate random number k-bit s_1, s_2
2. $P' = SecXor(x', y', r_y)$
3. $G' = SecAnd(x', y', r_x, r_y, u)$
4. $G' = G' \oplus r_x$ 5. $G' = G' \oplus u$
6. for $i=1$ to $n-1$ do
6.1 $H = SecShift(G', r_x, s_1, 2^{i-1})$
6.2 $U = SecAnd(P', H, r_x, s_1, s_2)$
6.3 $G' = SecXor(G', U, s_2)$
6.4 $H = SecShift(P', r_x, s_1, 2^{i-1})$
6.5 $P' = SecAnd(P', H, r_x, s_1, s_2)$
6.6 $P' = P' \oplus r_x$ 6.7 $P' = P' \oplus s_2$
7. $H = SecShift(G', r_x, s_1, 2^{n-1})$
8. $U = SecAnd(P', H, r_x, s_1, s_2)$
9. $G' = SecXor(G', U, s_2)$
10. $z' = SecXor(y', x', r_x)$
11. $z' = z' \oplus 2G'$ 12. $z' = z' \oplus 2r_x$
13. $z' = z' \oplus r_x$

Fig. 7. CGTV method

III. LEA에 대한 마스킹 기반 대응기법

3.1 블록암호알고리즘 LEA

LEA는 국내에서 개발된 128비트 블록암호알고리즘으로 스마트카드와 같은 경량, 저전력 환경에 적합하도록 설계되었고, 32비트 플랫폼에서 효율적으로 동작할 수 있도록 개발되었다[2]. 또한 128비트 블록 크기와 가변키(128, 192, 256비트) 크기를 가지고 데이트를 암·복호할 수 있으며, 각각의 라운드 수는 24, 28, 32이다. 라운드 함수는 32비트 Addition, Rotation, XOR 등으로 구성된 ARX 구조이다. 먼저 LEA 알고리즘 설명을 위해 본 논문에서 사용하는 표기들은 다음과 같다.

- $X_i[j]$: i번째 라운드에 입력되는 j번째 워드
- $RK_i[j]$: i번째 라운드 키의 j번째 워드
- $\oplus$: XOR(eXclusive OR)
- $\boxplus$: 32비트 모듈러 덧셈
- ROL_i : 32비트 비트열의 i비트 좌측 순환이동
- ROR_i : 32비트 비트열의 i비트 우측 순환이동

3.2 마스킹 기반 대응기법

본 논문에서는 LEA에 대한 마스킹 기반 대응기법 설계 시 1차 부채널분석만 고려하였다. Fig. 8은 마스킹 대응기법이 적용된 암호화 과정의 라운드 함수이다. 마스킹 대응기법의 일반적인 구현방법과 동일하게 라운드 함수의 입·출력 마스크 값(m_0)을 동일하게 유지하여 구현 효율성을 높이도록 설계하였다[11-12]. 32비트 모듈러 덧셈에 산술-불 상호 마스킹 변환 기법과 SD 기법을 선택적으로 적용하는 것을 제외하면 모든 부분이 동일한 구조를 가진다. 라운드 함수에서 사용되는 모든 마스크 값(m_i)은 32비트 난수발생기에 의해 생성된 난수이며, 라운드 함수의 입·출력 마스크 값을 동일하게 유지하기 위하여 아래와 같은 마스크 보정 값을 사전에 계산해 두고 사용한다.

$m_{f1} = ROL_9(m_1 \oplus m_2)$, $m_{f2} = ROR_5(m_1 \oplus m_2)$, $m_{f3} = ROR_3(m_1 \oplus m_2)$

3.3 이론적인 연산량 비교

앞서 설명한 6가지 대응기법을 마스킹 덧셈에 적

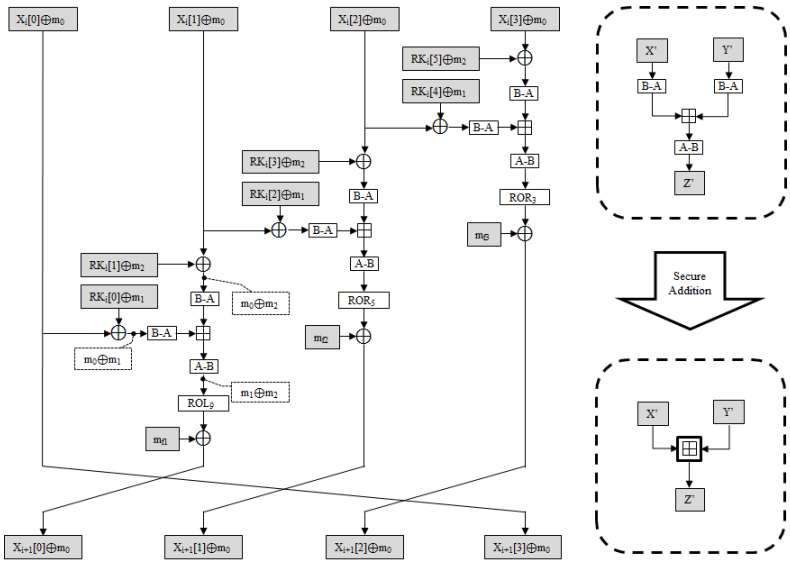


Fig. 8. i^{th} encryption round of LEA with masking countermeasures

용 시 이론적인 연산량을 비교해 보았다. 여기서 마스킹 덧셈은 두개의 불 마스킹된 라운드 입력 32비트와 두 개의 32비트 라운드 키간 XOR부터 불 마킹된 32비트 모듈러 덧셈 출력까지를 마스킹 덧셈으로 가정하였다. 그리고 연산량은 기본연산을 기준으로 계산했으며 모든 기본연산은 1이 소요된다고 가정하였다. 마지막으로 구현환경에 따라 대응기법의 연산량이 달라질 수 있는데 본 논문에서는 32비트 플랫폼으로 가정하였다.

① : Goubin method
② : CT method ($k=4$)
③ : CT method ($k=8$)
④ : Debraize method ($k=4$)
⑤ : Debraize method ($k=8$)
⑥ : KRJ method
⑦ : VG method ($k=4$)
⑧ : CGTV method

Table 1. Computational time for a masked addition on a 32-bit platform

	①	②	③	④	⑤	⑥	⑦	⑧
Pre-com.	0	35	515	128	2,048	0	2,816	0
Masked Add	181	60	40	59	39	168	39	144

Table 1과 같이 마스킹 덧셈의 경우 사전연산을 제외하면 $k=8$인 Debraize 기법과 VG 기법이 가장 효율적인 것으로 나왔다.

Table 1 결과를 바탕으로 앞서 설계한 마스킹 128비트 LEA에 적용하였다. 라운드 키 생성에 대한 연산량을 고려할 때 상수값의 순환이동은 사전에 계산해 둔 값을 사용한다고 가정하였다.

대응기법별 마스킹 128비트 LEA의 연산량은 Table 2와 같이 나왔다. 사전연산과 라운드 키 생성, 암호화를 모두 고려했을 때 $k=8$인 CT 기법이 대응기법 미적용 대비 7.77배로 가장 효율적인 것으로 나타났다.

Table 2. Computational time for a masked 128-bit LEA on a 32-bit platform

None	①	②	③	④	⑤	⑥	⑦	⑧
480	13,368	4,691	3,731	4,712	5,192	12,432	5,960	10,704
1.00	27.85	9.77	7.77	9.82	10.82	25.90	12.42	22.30

IV. 마스킹 LEA 구현 및 비교

4.1 대응기법 구현환경

32비트 ARM 프로세서(ARM7TDMI)와 전용

컴파일러(ADS 1.2)를 이용하여 앞서 설명한 다양한 마스킹 변환과 SA 마스킹 기법을 128비트 LEA에 적용하여 구현하였으며, 전체 라운드에 마스킹을 적용하였다.

본 논문에서는 대응기법 구현 시 코드 크기는 고려하지 않고 속도에 초점을 두어 구현하였으며, 코드 컴파일 시 최적화 옵션은 -O0(minimum)로 설정하였다. -O2(maximum)로 설정할 경우 속도는 향상될 수 있지만 의도하지 않은 잠재적인 부채널분석 취약점이 존재할 수 있기 때문에 본 논문에서는 이 옵션을 고려하지 않았다.

속도 최적화를 위해 라운드 키를 포함한 주요 함수는 loop unrolling 기법과 매크로 함수로 구현하였다. 그리고 메모리 참조 연산을 최소화하기 위해 빈번하게 사용되는 메모리 값은 특정 변수에 저장하여 레지스터로 연산될 수 있도록 고려하였다. 마지막으로 함수 호출로 인한 속도저하를 최소화하기 위해 Fig 7과 같이 SecXor, SecAnd 등은 함수로 구현하지 않고 알고리즘 내부에서 처리될 수 있도록 구현하였다. 다음에 설명할 구현 결과는 프로세서의 clock cycle 기준이며, 난수생성 시간은 구현 결과에서 제외하였다.

4.2 대응기법 구현 결과 분석

구현 결과는 크게 마스킹 덧셈, 암호화, 라운드 키 생성 등이 포함된 LEA 전체로 나누어 비교하였다. 마스킹 덧셈 구현 결과, CGTV 기법이 대응기법 미적용 대비 15.54배로 가장 효율적인 것으로 나타났다. 통상 SC100과 같은 보안용 코어의 경우 캐시기능이 없기 때문에 메모리 참조 연산에 많은 시간이 소비된다. 결국 룩업 테이블을 사용하는 기법들은 메모리 참조 연산을 많이 이용하기 때문에 비록 한 개의 기본연산이라도 많은 clock cycle이 소요되어 예상보다 느린 결과를 나타낸 것이라 판단할 수 있다. 또한 k비트씩 처리하기 위한 부가적인 연산들도 속도 저하의 한 요인으로 작용한 것으로 보인다.

반면 Goubin, KRJ, CGTV 기법은 기본연산을 레지스터간 연산으로 구현할 수 있어 (일부 기법은 룩업 테이블 기반 기법보다 느리긴 하지만) 룩업 테이블에 비해 상대적으로 높은 연산 효율성을 가지는 것으로 나타났다.

마스킹 암호화는 라운드 키 생성과 사전연산을 제외한 암호화 구간의 속도를 측정하였다. 마스킹 암호화의 속도는 Fig. 10과 같이 마스킹 덧셈의 속도 비율을 그대로 따르고 있음을 알 수 있다. 참고로 CGTV 기법의 SecXor, SecAnd 등을 함수로 구현한 경우 잦은 함수 호출로 인하여 마스킹 덧셈과 암호화의 속도 비율이 각각 65.63과 25.35로 나왔다.

128비트 마스킹 LEA 전체에 대한 속도를 측정한 결과, CGTV 기법이 대응기법 미적용 대비 4.67배로 가장 빠른 것으로 나타났다. 이는 앞서 설명한 구현 시 발생될 수 있는 다양한 요인(레지스터간 연산, 룩업 테이블 미사용 등)에 의한 것으로 판단된다. 또한 CGTV 기법과 같이 SA 마스킹 기법은 마스킹 변환 시 발생하는 함수 호출을 생략할 수 있어 마스킹 변환이 필요한 대응기법보다 상대적으로 효율적인 것으로 나타났다.

마지막으로 모든 키 크기에 대한 속도 비율도 측정해 보았다. Fig. 12와 같이 192, 256비트 역시 128비트와 속도 비율이 비슷했으며, 모든 기법의 속도 비율이 128비트에 비해 조금 더 개선되는 것을 알 수 있었다.

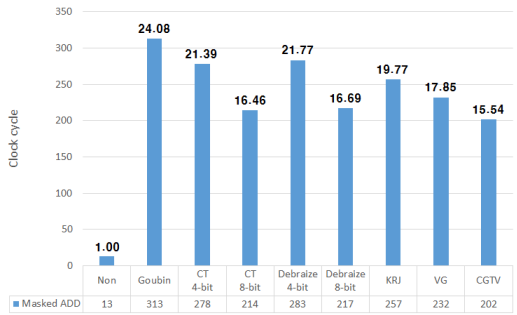


Fig. 9. Masked addition

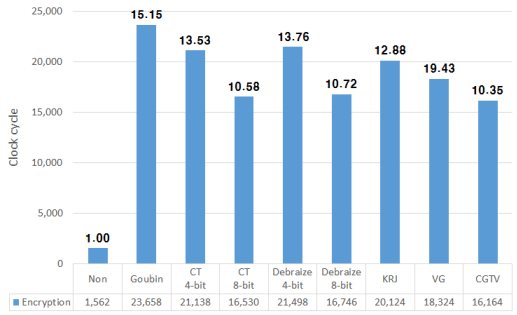
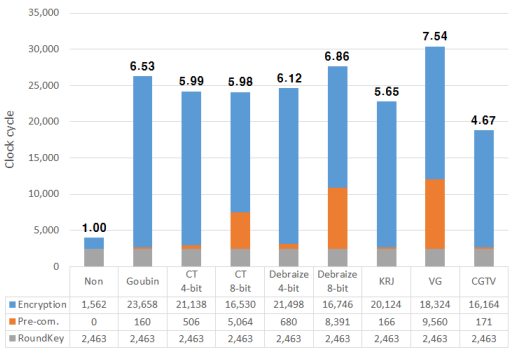
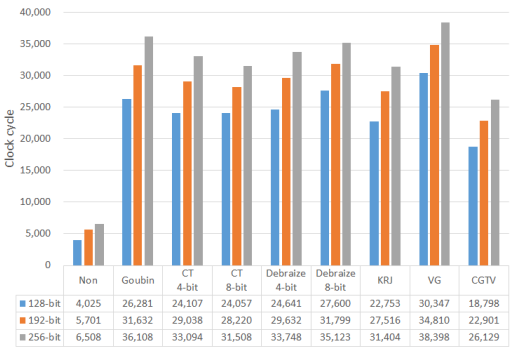


Fig. 10. Masked encryption



	Non	Goubin	CT 4-bit	CT 8-bit	Debraize 4-bit	Debraize 8-bit	KRJ	VG	CGTV
Encryption	1,562	23,658	21,138	16,530	21,498	16,746	20,124	18,324	16,164
Pre-com.	0	160	506	5,064	680	8,391	166	9,560	171
RoundKey	2,463	2,463	2,463	2,463	2,463	2,463	2,463	2,463	2,463

Fig. 11. 128-bit Masked LEA



	Non	Goubin	CT 4-bit	CT 8-bit	Debraize 4-bit	Debraize 8-bit	KRJ	VG	CGTV
128-bit	4,025	26,281	24,107	24,057	24,641	27,600	22,753	30,347	18,798
192-bit	5,701	31,632	29,038	28,220	29,632	31,799	27,516	34,810	22,901
256-bit	6,508	36,108	33,094	31,508	33,748	35,123	31,404	38,398	26,129

Fig. 12. 128/192/256-bit Masked LEA

V. 안전성 분석

5.1 T-test 개요

공통평가(CC, Common Criteria)에서는 스마트카드와 같은 하드웨어 제품의 안전성 평가 시 부채널분석을 필수 평가항목으로 포함하고 있다. 하지만 평가기관의 기술력 차이, 세부 평가항목에 대한 비정형화, 낮은 재현성 등 여러 문제점을 가지고 있다. 최근 이러한 문제점을 해결하기 위해 T-test와 같은 수학적 검증식에 기반한 안전성 검증 방법이 소개되었다[13-17].

T-test는 서로 다른 두 모집단이 통계적으로 유의미한 차이가 있는지를 검증하는 방법으로 현재 다양한 분야에서 활용되고 있으며, 검증 수식은 다음과 같다.

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}}$$

위 수식에서 μ는 평균, s는 표준편차, n은 샘플수를 뜻하며, t-value는 두 집단의 평균이 얼마만큼의 신뢰도를 가지고 유의미한 차이가 있는지를 나타낸다.

5.2 T-test를 이용한 LEA 안전성 분석 절차

LEA에 대한 T-test는 DATA-SET 생성 과정과 이를 이용한 검증 과정으로 이루어진다.

먼저 DATA-SET 생성은 다음과 같다.

a. 마스터키 설정 - 검증자 임의의 고정된 값
b. $I_1, I_2, ..., I_n$ 랜덤 평문을 이용하여 n번의 암호연산 수행

생성한 DATA-SET을 이용한 검증 절차는 다음과 같다.

① 검증자는 마스터키 길이에 따라 테스트할 중간 라운드 선택
② 다음의 그룹을 이용하여 288번의 테스트 수행
a. Group 생성 : n개 파형이 포함된 그룹
b. 수집된 파형 전체 영역을 합격/불합격 판정에 사용할 수 있지만 효율성을 위해 중간 $N-1, N, N+1$ 라운드만을 사용
③ 중간 라운드 round key addition 테스트
a. 중간 라운드의 round key addition 출력 192 비트를 Subset 생성에 사용
b. 비트 위치를 바꾸어 가면서 비트값에 따라 총 192쌍의 Subset 생성
i. Subset A : i번째 위치의 비트값이 '0'인 파형
ii. Subset B : i번째 위치의 비트값이 '1'인 파형
④ 중간 라운드 모듈러 덧셈 테스트
a. 중간 라운드의 모듈러 덧셈 결과 96 비트를 Subset 생성에 사용
b. 비트 위치를 바꾸어 가면서 비트값에 따라 총 96쌍의 Subset 생성
i. Subset A : i번째 위치의 비트값이 '0'인 파형
ii. Subset B : i번째 위치의 비트값이 '1'인 파형

위 검증 절차를 통해 얻은 결과는 다음과 같은 방법으로 합격·불합격을 판정한다. 여기서 불합격이란 취약점이 있음을 말한다. LEA 연산 시 수집된 소비전력 파형을 이용하여 288번(round key addition 192번 + 모듈러 덧셈 96번)의 비트 테스트를 수행한 후 t-value가 ±4.5 범위 안에 있을 경우 합격으로 판정한다. 이 경우 판정 결과는 99.999% 이상의 신뢰도를 가진다.

5.3 LEA 안전성 분석 실험 결과

안전성 분석을 위해 대응기법이 적용된 것과 적용되지 않은 LEA를 스마트카드(32-bit ARM7TDMI 기반 SC100 CPU)에 구현한 후 40,000개의 소비전력 파형을 수집하였다. 다음으로 LEA 4 라운드 연산 중간값을 이용하여 검증 절차에 따라 T-test 안전성 검증을 위한 Subset을 만들어 검증을 수행하였다.

대응기법이 적용되지 않은 LEA에 대한 실험 결과, 192번의 round key addition 출력에 대한 비트 테스트와 96번의 모듈러 덧셈 출력에 대한 비트 테스트 모두에서 그림과 같이 한계치(±4.5) 이상의 값이 나오는 것을 확인할 수 있다. 또한 검증 기준으로 사용한 비트들이 연산되는 위치에 따라 T-test 결과값의 최대값 위치도 달라지는 것을 확인 할 수 있다. 따라서 부채널분석에 대한 대응기법을 고려하지 않은 LEA 알고리즘은 T-test 안전성 검증을 통과하지 못했다.

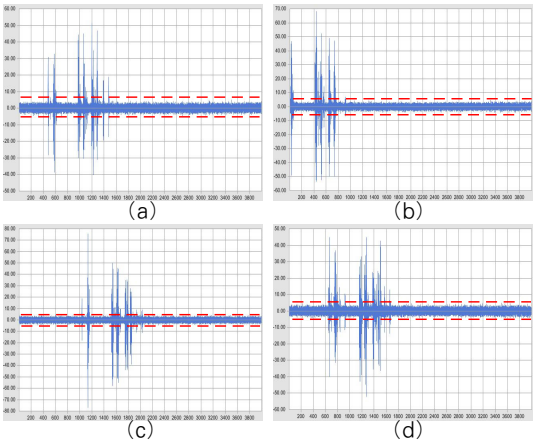


Fig. 13. T-test results on LEA : (a) 1st bit of round key addition output (b) 192th bit of round key addition output (c) 1st bit of addition output (d) 96th bit of addition output

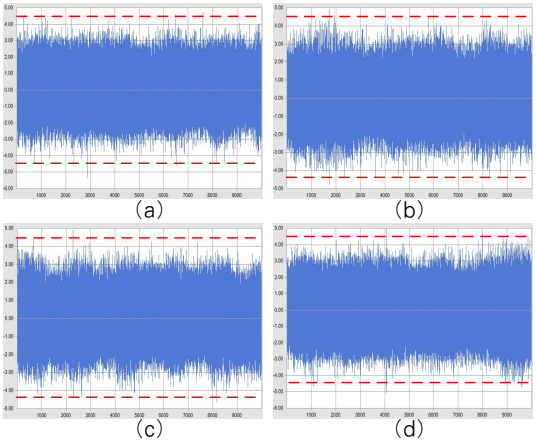


Fig. 14. T-test results on masked LEA : (a) 1st bit of round key addition output (b) 192th bit of round key addition output (c) 1st bit of addition output (d) 96th bit of addition output

다음으로 KRJ 기법을 임의로 선택하여 LEA에 적용한 후 T-test를 수행하였다. Fig. 14와 같이 대부분의 테스트에서 한계치(±4.5) 이하의 값이 나오는 것을 확인할 수 있다. 몇몇의 시점에서 T-test 결과값이 한계치(±4.5) 이상으로 나오는 경우도 있었지만, 이는 재현성이 없으며 알고리즘의 구현 취약성과는 연관성이 없는 무시할 수 있는 결과이다. [14,15]에서도 상기 현상에 대해 무시할 수 있다고 언급하고 있으며, 또 다른 그룹을 대상으로 T-test를 수행한 결과 재현성이 없음을 알 수 있었다. 따라서 부채널분석 대응기법이 적용된 LEA는 T-test 안전성 검증을 통과한 것으로 판단할 수 있다. 또한 다른 대응기법에 대해서도 동일한 결과를 얻을 수 있었다.

VI. 결 론

본 논문에서는 지금까지 제안된 다양한 마스킹 변환 기법을 LEA에 적용하여 예상되는 이론적 연산량과 실제 측정한 연산량에 어떠한 차이점이 있는지 구현관점에서 살펴보았다. 분석 결과, 룩업 테이블 기반 마스킹 변환 기법의 경우 사전연산과 메모리 참조 연산 등으로 인해 룩업 테이블을 사용하지 않는 기법에 비해 상대적으로 낮은 연산 효율성을 가지는 것으로 나타났다. 또한 구현 최적화 기술의 적용 유무에 따라 많은 성능 차이를 보였다.

32비트 플랫폼에 CGTV 기법을 이용하여 전체 라운드가 마스킹된 128비트 LEA를 구현했을 때 약 4.67배의 속도 저하만이 발생하는 것을 알 수 있었다. 이는 마스킹 적용 시 통상 약 2배의 성능 저하를 가지는 AES[12]와 비교하여 ARX 구조인 LEA가 불리한 것은 사실이다. 하지만 부채널분석 대응기법 미적용 시 LEA가 AES에 비해 좋은 성능을 가지고 있는 것을 감안한다면 마스킹 기법이 적용된 두 알고리즘의 성능 차이는 미비할 것으로 판단된다.

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," Advances in Cryptology, CRYPTO'99, LNCS 1666, pp. 388-397, 1999.

[2] TTA, "128-bit lightweight block cipher LEA," TTAK.KO-12.0223, Dec. 2013.

[3] T. Messerges, "Securing the AES finalists against power analysis attacks," Fast Software Encryption, FSE'00, LNCS 1978, pp. 150-164, 2001.

[4] L. Goubin, "A sound method for switching between Boolean and arithmetic masking," Cryptographic Hardware and Embedded Systems, CHES'01, LNCS 2162, pp. 3-15, 2001.

[5] J. Coron and A. Tchulkine, "A new algorithm for switching from arithmetic to Boolean Masking," Cryptographic Hardware and Embedded Systems, CHES'03, LNCS 2779, pp. 89-97, 2003.

[6] B. Debraize, "Efficient and provably secure methods for switching from arithmetic to Boolean masking," Cryptographic Hardware and Embedded Systems, CHES'12, LNCS 7428, pp. 107-121, 2012.

[7] O. Neiße and J. Pulkus, "Switching blindings with a view towards IDEA," Cryptographic Hardware and Embedded Systems, CHES'04, LNCS 3156, pp. 230-239, 2004.

[8] M. Karroumi, B. Richard, and M. Joye, "Addition with blinded operands," Constructive Side-Channel Analysis and Secure Design, COSADE'14, LNCS 8622, pp. 41-55, 2014.

[9] P. Vadnala and J. Großschadl, "Faster mask conversion with lookup tables," Constructive Side-Channel Analysis and Secure Design, COSADE'15, LNCS 9064, pp. 207-221, 2015.

[10] J. Coron, J. Großschadl, M. Tibouchi, and P. Vadnala, "Conversion from arithmetic to Boolean masking with Logarithmic complexity," Fast Software Encryption, FSE'15, LNCS 9054, pp. 130-149, 2015.

[11] C. Herbst, E. Oswald, and S. Mangard, "An AES smart card implementation resistant to power analysis attacks," Applied Cryptography and Network Security, ACNS'06, LNCS 3989, pp. 239-252, 2006.

[12] J. Park, T. Kim, H. An, Y. Won, and D. Han, "Side channel attacks on LEA and its countermeasures," *Journal of The Korea Institute of Information Security & Cryptology*, 25(2), pp. 449-456, Apr. 2015.

[13] G. Goodwill, B. Jun, J. Jaffe, and P. Rogatgi, "A testing methodology for side-channel resistance validation," NIST Non-Invasive Attack Testing Workshop, NIAT 2011, 2011.

[14] J. Jaffe and P. Rogatgi, "Efficient side-channel testing for public key algorithms - RSA case study," NIST Non-Invasive Attack Testing Workshop, NIAT 2011, 2011.

[15] G. Becker, J. Cooper, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, and M. Marson, "Test Vector Leakage Assessment (TVLA) methodology in practice," International Cryptographic Module Conference, ICMC'13, 2013.

[16] S. Tobias and M. Amir, "Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations," Cryptographic Hardware and Embedded Systems, CHES'15, LNCS 9293, pp. 495-513, 2015.

[17] ISO17825:2016, "Information technology - Security technique - Testing methods for the mitigation of non-invasive attack classes against cryptographic modules," ISO/IEC, 2016.

〈저자소개〉

사 진

김 창 균 (ChangKyun Kim) 정회원
2001년 2월: 경북대학교 전자전기공학부 졸업
2003년 2월: 경북대학교 전자공학과 석사
2009년 8월: 경북대학교 전자공학과 박사
2004년 11월~현재: 한국전자통신연구원 부설연구소 선임연구원
〈관심분야〉 부채널분석, HW 암호모듈 안전성 분석, 암호알고리즘 구현

사 진

박 제 훈 (JaeHoon Park) 정회원
2004년 2월: 경북대학교 전자전기공학부 졸업
2006년 2월: 경북대학교 전자공학과 석사
2011년 2월: 경북대학교 전자공학과 박사
2011년 1월~2012년 1월: 국방기술품질원 선임연구원
2012년 2월~현재: 한국전자통신연구원 부설연구소 선임연구원
〈관심분야〉 부채널분석, 정보보호시스템 안전성 분석

사 진

한 대 완 (Daewan Han) 정회원
1997년 2월: 서울대학교 수학과 석사
2007년 8월: 서울대학교 수학과 박사
2001년 3월~현재: 한국전자통신연구원 부설연구소 실장, 책임연구원
〈관심분야〉 암호 설계 및 분석, 정보보호시스템 안전성 분석

사 진

이 동 훈 (Dong Hoon Lee) 정회원
1994년 2월: 서울대학교 수학교육과 졸업
1996년 2월: 한국과학기술원 수학과 석사
2000년 2월: 한국과학기술원 수학과 박사
2000년 2월~2002년 3월: (주)퓨처시스템 선임연구원
2002년 4월~현재: 한국전자통신연구원 부설연구소 실장, 책임연구원
〈관심분야〉 응용정수론, 암호이론, 정보보호시스템 안전성 분석