情報保護學會論文誌
第16卷 第2號, 2006. 4

랜덤 마스킹 기법을 이용한 DPA 공격에 안전한 ARIA 구현*

유 형 소,[1†] 하 재 철,[2‡] 김 창 균,[3] 박 일 환,[3] 문 상 재[1]
[1]경북대학교, [2]나사렛대학교, [3]국가보안기술연구소

A Secure ARIA implementation resistant to Differential Power Attack using Random Masking Method*

HyungSo Yoo,[1†] JaeCheol-Ha,[2‡] ChangKyun Kim,[3] IlHwan Park,[3] SangJae Moon[1]
[1]Kyungpook National University, [2]Korea Nazarene University,
[3]National Security Research Institute

요 약

ARIA는 128비트 블록암호알고리즘으로, 2004년 국가표준(KS)으로 선정되었다. 현재 많은 연구가 진행되고 있는 DPA 공격에 ARIA가 취약함이 발견되었다. 따라서 본 논문에서는 1차 DPA 공격에 의한 대응방법으로 가장 많은 연구가 이루어지고 있는 마스킹 기법을 설명하고 국내표준 암호알고리즘인 ARIA에 적용하였다. 마스킹이 적용된 ARIA를 AVR 기반의 8비트 프로세서를 사용하는 스마트카드에 소프트웨어로 구현하였으며, 실험을 통하여 1차 DPA 공격에 안전함을 확인하였다.

ABSTRACT

ARIA is a 128-bit block cipher, which became a Korean Standard in 2004. According to recent research, this cipher is attacked by first order DPA attack. In this paper, we explain a masking technique that is a countermeasure against first order DPA attack and apply it to the ARIA. And we implemented a masked ARIA for the 8 bit microprocessor based on AVR in software. By using this countermeasure, we verified that it is secure against first order DPA attack.

Keywords : *ARIA, 마스킹, 1차 DPA공격, 스마트카드*

Ⅰ. 서 론

최근 국내에서는 금융IC카드, 행정기관IC카드 등이 스마트카드로 대체되고 있으며, 차세대 주민등록증으로 스마트카드로의 대체가 진행되고 있다. 하지만, 1999년 Kocher에 의해 DPA 공격이[1] 처음으로 제안된 이후, 많은 연구자들에 의해 대응방법을 고려하지 않고 암호알고리즘을 구현한 하드웨어 암호장치가 취약함이 밝혀졌다. 따라서 DPA 공격에 안전하기 위한 대응방법이 활발히 연구되고 있다. 지금까지 DPA 공격에 대한 대응방법으로 암호알고리즘의 수행도중 생성되는 중간값과 전력소모량간의 의존성을 제거하기 위하여 마스킹[4, 5], 랜덤 지연시간 삽입[5], 새로운 하드웨어 논리 스타일 사용[6,7] 등이 제안되었으나, 가장 활발히 연구가 되고 있는 분야가 마스킹을 사용한 대

접수일: 2006년 2월 23일; 채택일: 2006년 3월 29일
* 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었음.
† 주저자, ydilly@hotmail.com
‡ 교신저자, jcha@kornu.ac.kr

응방법이다. 마스킹 기법은 암호연산의 중간값과 전력 소모량간의 연결고리를 제거하기 위하여, 임의의 난수를 사용하여 데이터를 변환하는 방법이다. 현재까지 마스킹 기법에 대한 이론적인 연구는 많이 이루어졌으나, 실제 스마트카드에 구현하여 효율성 및 안전성을 분석한 논문은 없다. 본 논문에서는 최근 1차 DPA 공격에 취약함이[3] 밝혀진 국내 표준 암호알고리즘인 ARIA를 대상으로 마스킹 기법을 적용하였다. 또한 1차 DPA공격에 안전함을 이론적으로 분석하였으며, 실험을 통하여 검증하였다. 본 논문의 구성은 다음과 같다. II장에서 ARIA의 구조 및 1차 DPA 공격결과를 설명하고, III장에서 블록암호를 구성하는 각 연산들에 대한 마스킹 기법을 소개한다. IV장에서 마스킹을 적용한 ARIA를 구현하고, 마스킹이 적용된 ARIA의 이론적 안전성 분석 및 스마트카드에 대한 1차 DPA 공격을 통한 실험적결과를 보여준다.

II. ARIA에 대한 1차 DPA 공격

1. ARIA 개요

ARIA는 2004년 국가산업표준(KS)으로 선정된 블록암호알고리즘으로, 128비트 블록크기와 가변키(128, 192, 256비트)크기를 가지는 Involutional SPN 구조이다. ARIA는 스마트카드 등 저전력, 저성능의 플랫폼 및 ASIC, 32비트 프로세서 등의 고성능 플랫폼에서 동작할 수 있도록 개발되었다.

2. 표기 및 주요함수

2.1 표기

- $\oplus$ 배타적 논리합 연산
- S_i S-BOX
- ek_i i번째 라운드 키
- $A^{\lll k}$ A의 각 비트를 왼쪽으로 k 비트씩 순환이동
- $A^{\ggg k}$ A의 각 비트를 오른쪽으로 k 비트씩 순환이동
- || 연접

2.2. 주요함수

ARIA의 각 라운드는 다음과 같이 세 부분으로 구성되며, 마지막 라운드에서는 Diffusion layer가 생략되며, AddRoundKey가 수행된다.

- **AddRoundKey** : 평문과 라운드키를 XOR연산을 사용하여 결합한다.
- **S-BOX layer** : 두 종류의 비선형 함수를 사용하여 입력값을 치환한다.
- **Diffusion layer** : 16×16이진 행렬을 사용하여 S-BOX출력을 확산한다.

3. ARIA 구조

ARIA는 키 크기에 따라 12, 14, 16라운드로 구성되며, 그림 1과 같이 암/복호 함수, 라운드키 생성 함수로 이루어져 있다. ARIA는 암복호 구조가 동일한 Involutional SPN 구조로 평문 입력 128비트를 암호문 출력 128로 변환한다.

4. ARIA에 대한 1차 DPA 공격 결과

[3]에서는 대응방법이 적용되지 않은 ARIA에 대한 1차 DPA 공격을 수행하여 스마트카드에 저장된 암호키를 찾아 낼 수 있었다. 그림 2와 같이 1라운드의 S-BOX출력에 대해 DPA 공격을 적용한 실험에서 3000개의 전력신호샘플을 이용하여 라운드키를 찾을 수 있었다. 그림 3은 올바른 키를 추측했을 때의 상관계수이며, 그림 4는 S-BOX의 입력키로 추측이 가능한 256가지 모두에 대한 상관계수를 비교하였다. 그림 3, 4에서 볼 수 있듯이 올바른 키 추측 시 500[time(clock)]에서 상관계수가 가장 높은 값(0.65)을 얻을 수 있었으며, 틀린 키일 경우에는 0~0.15사이의 상관계수를 얻을 수 있었다. 상관계수를 구하는 구체적인 방법은 본 논문의 4장 2절에 기술하였으며, 그림 4에서 검정색은 올바른 키 추측 시의 파형이며 옅은 회색은 틀린 키 추측 시의 파형이다.

III. 마스킹 기반 ARIA 대응기법 구현

대부분의 블록암호 알고리즘은 테이블 참조 연산, 대수 연산, 산술 연산 등 다양한 연산으로 이루어져 있다. 본 장에서는 기존에 제시된 마스킹 기법[4]을 설명하고, 이를 바탕으로 마스킹된 ARIA 구현에 대해 설명할 것이다.

1. 기존 마스킹 대응방법

1.1. 테이블 참조 연산

테이블 참조 연산은 입력 x에 대해 출력 $y = T[x]$

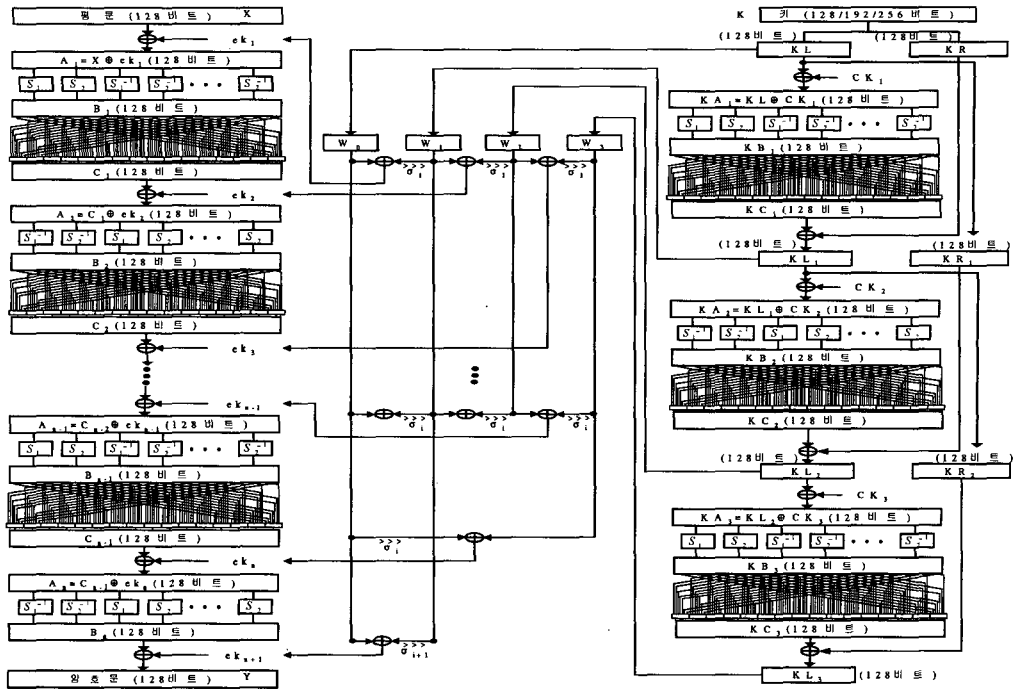

그림 1. ARIA 구조

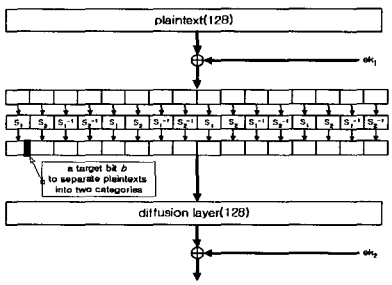

그림 2. 1차 DPA 공격 대상

를 결과로 얻는 연산이다. 테이블 참조 연산에 대한 마스킹을 수행하기 위해서는 테이블 자체를 새로운 테이블로 변경, 즉 마스킹된 테이블로 만들어야 한다. 테이블을 마스킹하기 위한 가장 쉬운 방법은 입력 마스크 m과 출력 마스크 m'을 사용한, 가산(additive) 마스킹 방법이다.

$$T'[x] = T[x \oplus m] \oplus m'$$

실제 스마트카드 구현에서는 처리속도를 향상시키기 위해서 랜덤 마스크 m과 m'을 암호 알고리즘 수행 전에 미리 생성하고, 마스킹된 테이블을 계산한 후 RAM에 저장하여 사용한다.

1.2. 비트별 대수 연산

- **XOR** : 마스킹된 데이터를 피연산자로 하는 XOR 연산은 단순히 두 개의 피연산자에 대한 XOR를 그대로 수행하면 된다. 즉 $x' = x \oplus r_x$, $y' = y \oplus r_y$ 라고 했을 때 $z' = x' \oplus y' = x \oplus y \oplus r_x \oplus r_y$가 되고 새로운 마스크 $r_z = r_x \oplus r_y$가 된다.
- **AND** : AND연산의 경우 마스킹된 데이터를 처리하는 경우 조금 복잡한 과정을 거친다. $x' = x \oplus r_x$, $y' = y \oplus r_y$일 경우 $z' = x' \wedge y'$가 되며, $r_z = (r_x \wedge y') \oplus (r_y \wedge x') \oplus (r_x \wedge r_y)$가 된다. 이 때 중간과정에서 데이터 x 또는 y가 드러나게 되는데, 이를 방지하기 위해 별도의 마스크를 사용하여야 한다.
- **OR** : OR연산은 AND와 유사한 과정을 거친다. $x' = x \oplus r_x$, $y' = y \oplus r_y$일 경우 $z' = x' \vee y'$가 되며, $r_z = (r_x \wedge \overline{y'}) \oplus (r_y \wedge \overline{x'}) \oplus (r_x \wedge r_y)$가 된다. 이 때 중간과정에서 데이터 x 또는 y가 드러나게 되는데, 이를 방지하기 위해 별도의 마스크를 사용하여야 한다.

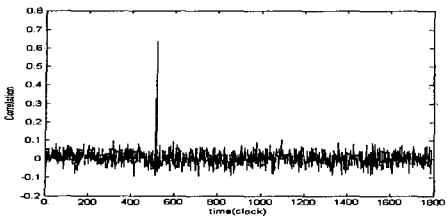

그림 3. SBOX 올바른 키

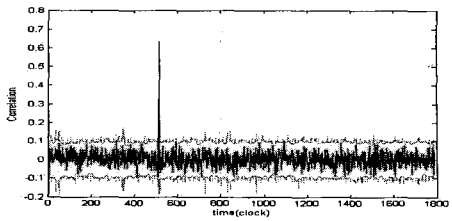

그림 4. SBOX 모든 키(256)

1.3. 쉬프트 및 로테이트 연산

마스크된 데이터에 대한 쉬프트 및 로테이트 연산은 간단하게 계산할 수 있다. 단순히 마스크된 데이터 자체를 쉬프트 시키면 되고, 새로운 마스크 역시 마스크 자체를 쉬프트 시킨 값이 된다. 즉 마스크된 데이터를 $x' = x + r_x$, 마스크를 r_x라고 했을 때 n비트 쉬프트 시킨 값은 $x' \gg n$, $r_x \gg n$이 된다.

1.4. 모듈러 덧셈 및 곱셈 연산

모듈러 덧셈 및 곱셈 연산은 산술 연산이므로, 산술적으로 마스킹된 데이터에 대해서는 다음과 같이 간단히 수행할 수 있다. 산술적으로 마스킹된 데이터 $x' = x + r_x$, $y' = y + r_y$가 주어졌을 때 모듈러 덧셈 연산은 $z' = (x' + y') \bmod 2^{32}$으로 간단히 계산되며, 이 때 새로운 마스크는 $r_z = r_x + r_y \bmod 2^{32}$이 된다. 모듈러 곱셈 연산은 $z' = (x'y') \bmod 2^{32}$으로 계산되며, 마스크는 $r_z = (r_x y' + r_y x' + r_x r_y) \bmod 2^{32}$이 된다. 대수적으로 마스킹된 데이터에 대해 산술 연산을 수행하기 위해서는 이에 대한 적절한 변환이 필요하다[8].

1.5. 선형변환 연산

반복적이지 않은 선형변환 연산에서, 마스킹된 데이터는 $z' = LT[x'], r_z = LT[r_x]$로 간단히 계산될 수 있다.

2. 마스킹 기반 ARIA 대응기법 구현

ARIA의 각 라운드는 AddRoundKey, SBOX, Diffusion의 3가지 연산으로 이루어져 있다. 마스킹을 적용한 ARIA 구현시 주의해야 할 사항은 마스킹을 적용한 후, 중간값은 노출시키지 않으면서 최종 암호문은 마스킹을 적용하기 전과 동일한 값을 가지도록 해야 하는 것이다. 이를 위해, 본 논문에서는 각 연산의 입출력에 가변 마스크를 사용한 additive 마스킹을 적용하였다. 랜덤 마스크는 스마트카드 내부의 난수발생기를 이용하거나, 프로그램의 난수발생기 함수를 이용하여 생성할 수 있으나, 본 논문에서는 외부에서 생성된 마스크를 데이터와 같이 입력으로 받아 실험을 수행하였다. 각 연산별 마스킹 방법은 다음과 같다. 마스킹을 적용한 연산 수행시 중간에 마스크가 제거되지 않도록 주의하여야 하며, 마스크값의 변화과정을 지속적으로 관찰하여야 한다.

마스킹된 AddRoundKey : AddRoundKey연산은 마스크를 변화시키지 않으므로 마스킹과 관련하여 특별하게 고려하지 않아도 된다.

마스킹된 SubBytes : SubBytes는 3장에서 언급한 마스킹 연산중 테이블 참조 연산에 대한 기법을 사용한다. 따라서 기존의 S-BOX에 마스크 m과 m'를 적용하여 마스킹하여, 다음과 같은 마스킹된 새로운 S-BOX를 계산하였다.

$$S'(x \oplus m) = S(x) \oplus m' \quad (1)$$

마스킹에서 입력과 출력 마스크는 동일한 값 또는 서로 다른 값을 사용할 수 있으나, 동일한 값을 사용할 경우에는 구현방법에 따라 1차 DPA공격에 취약할 수 있으므로 구현시 주의를 기울여야 한다. 또한, 각 암호연산시마다 고정된 마스크를 사용할 경우 원래의 S-BOX $S(x)$과 마스킹된 S-BOX $S(x) \oplus m'$ 사이에 높은 상관관계가 있으므로 DPA 공격을 방어할 수 없다. 따라서 각각의 암호 연산시마다 가변 마스크를 사용하여야 한다.

Algorithm 1 Masked SBOX 계산
입력 : x, m, m' 출력 : MaskedSBOX(x+m) = SBOX(x)+m'
1 : for i = 0 to 255 do 2 : MaskedSBOX(i+m) = SBOX(i)+m' 3 : end for 4 : Return(MaskedSBOX)

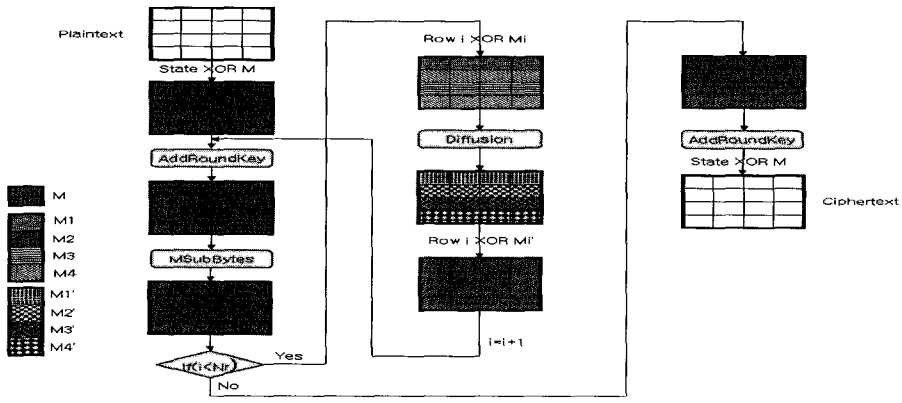

그림 5. ARIA 마스킹 구조

```
Mask_ARIA_Crypt_Round()
{
Data_Masking;
Addroundkey;
MSBOX;
for(i=1 ; i < Nr ; i++) {
      Masking_State;
      Diffusion;
      Remove_Masking_State;
     Addroundkey;
     MSBOX;
}
Addroundkey;
Remove_Data_Masking;
}
```

그림 6. 마스킹 ARIA Pseudo 코드

마스킹된 Diffusion Layer : Diffusion계층은 선형연산이므로 3장에서 언급된 선형변환에 대한 마스킹 기법을 적용하였다. 마스킹된 S-BOX의 결과값과 마스크에 대한 확산 연산을 별도로 수행한 후 결과값을 결합할 수 있다. 따라서, 마스크에 대한 사전계산을 통해 연산량을 줄일 수 있다. Diffusion 연산시 주의할 사항은 Diffusion연산 도중 마스크값이 제거되지 않도록 해야 한다. 이는 ARIA의 Diffusion연산이 SBOX출력의 배타적 논리합 연산으로 이루어져 있기 때문으로 동일한 마스크 값을 적용한 두 개의 SBOX출력을 배타적 논리합 연산을 할 경우, 마스크 값이 제거되어 SBOX의 출력값이 드러나게 된다. 따라서, 본 논문의 구현에서는, 이를 방지하기 위하여, 중간값의 각 행마다 다른 마스크를 적용하였으며, Diffusion연산 후에 다시 제거하였다. 본 논문에서는 입력 1블럭(128비트)에 대해 마스킹 연산이 수행되는 과정을 수식적인 표현을 통해 중간값이 드러나지 않고 최종 암호문 이 원래의 암호문과 동일하다는 것을 확인(부록 1)하고 실험을 수행하였다. 마스킹 구조 및 마스킹 ARIA의 Pseudo 코드는 그림 5, 6과 같다. 그림 5의 "State XOR M" 부분이 Pseudo 코드의 Data Masking에 해당하며, "Row i XOR Mi", "Row i XOR Mi'"가 각각 Masking_State와 Remove_Masking_State에 해당한다. MSBOX는 위의 마스킹된 SubBytes에서 설명한 방법으로 알고리즘 수행전에 미리 계산하여 RAM에 저장된다.

IV. 마스킹 기반 ARIA에 대한 1차 DPA 공격

1. DPA 실험환경

마스킹이 적용된 ARIA에 대한 DPA 공격을 수행하기 위하여, 그림 7, 8과 같이 AVR 기반의 8비트 마이크로프로세서가 장착된 IC카드, LeCroy사의 LC584 디지털 오실로스코프, Infineon사의 스

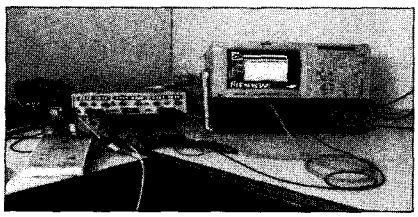

그림 7. 실험환경

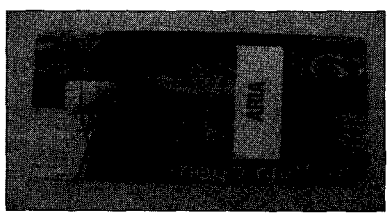

그림 8. 스마트카드

마트카드 리더기를 사용하였다. CrossStudio for AVR을 사용하여 마스킹된 ARIA를 8비트 마이크로 프로세서에 적합한 형태로 컴파일하였으며, 측정된 전력소모량에 대한 DPA 분석을 위하여 MATLAB으로 구현한 부채널 공격 TOOLBOX를 사용하였다.

2. 1차 DPA 공격 방법

마스킹된 ARIA를 AVR기반의 마이크로프로세서를 사용하는 스마트카드의 EEPROM에 주입한 후 1차 DPA 공격을 다음과 같이 수행하였다. 1차 DPA 공격을 위해서는 스마트카드내부에 구현된 암호알고리즘 및 스마트카드의 하드웨어 구조를 알고 있다고 가정한다. 본 논문에서는 DPA공격 방법중 가장 효율성이 높은 상관계수를 이용하는 방법을 사용하였다. 그림 3, 4, 및 9, 10은 공격자가 추측한 키와 암호연산이 수행되는 시간에서의 전력소모량간의 상관계수를 나타낸 것이다. 상관계수를 이용하여 공격자의 추정모델과 실제 스마트카드에서 수집한 전력소모량 사이의 상관관계를 계산하는 방법은 다음과 같이 이루어진다.

단계 1 : 스마트카드에서 동일한 키를 사용하여 S개의 서로 다른 입력 데이터를 암호화하는 동안의 전력 소모량을 측정한다. 이 때, 측정된 전력파형을 $P_{1...S,1...T}$로 표기한다. T는 스마트카드에서 암호를 수행할 때 기록되는 샘플 수이며, 디지털 오실로스코프의 설정에 의해 결정된다.

단계 2 : 입력 평문과 추측한 부분키를 사용하여 중간값을 계산하고, 계산된 중간값에 따라 전력소모량을 추정한다. 추정된 중간값 행렬 $I_{1...K,1...S}$를 얻을 수 있다.이 때, K는 가능한 부분키의 수이며 8비트일 경우 256개의 가능한 수가 존재한다. 스마트카드에서 실제로 사용되는 부분키 k_c는 K개의 가능한 부분키중 하나이다. 따라서, $I_{k_c,1...S}$는 S개의 암호 연산을 수행할 때 실제로 수행된 값이다. 결론적으로 $P_{1...S,k_c}$값은 $I_{k_c,1...S}$ 에 의존한다. t_c는 공격대상 중간값이 수행되는 순간의 시간이다.

단계 3 : 각각의 $I_{k,s}$에 대해서 추정 전력 소모량 $H_{k,s}$를 결정한다. 이 때 $H_{1...K,1...S}$의 절대값은 중요하지 않다. 단지 $H_{1...K,1...S}$를 구성하는 값들간의 상대적인 차이가 관련이 있다. 추정 전력 소모량을 계산하기 위해 일반적으로 사용되는 모델은 논리 1을 회로에 저장할 때가 논리 0을 저장할 때보다 더 많은 에너지를 필요로 한다는 사실에 기반한다. 이 모델은 적용하기가 쉬울 뿐만 아니라 현실적으로 대부분의 하드웨어 구현에 있어서 정확히 맞아 떨어진다. 특히 CMOS를 사용하여 구현된 회로가 이러한 특성을 가지고 있다.

단계 4 : 추정 전력 소모량을 결정한 후, 공격자는 추정 전력 소모량 $H_{1...K,1...S}$와 전력 파형 $P_{1...S,1...T}$의 상관도를 계산하여 정확한 부분키 k_c를 찾아낼 수 있다. 이 때 상관계수가 높은 것이 찾고자 하는 부분키이다.

상관도를 측정하는 방법으로 피어선 상관계수(pearson correlation coefficient)가 있다. 피어선 상관계수는 두 변수들간의 선형 관계를 결정하는 보편적인 측정방법이다. 두 변수 X, Y간의 상관도를 나타내는 일반적인 수식이다.

$$\rho(X,Y) = \frac{E(XY)-E(X)E(Y)}{\sqrt{Var(X)\,Var(Y)}} = \frac{Cov(X,Y)}{\sqrt{Var(X)\,Var(Y)}} \quad (2)$$

한편, 피어선 상관계수(r)의 정의는 다음과 같으며, r은 두 변수들 사이의 상관도를 S개의 샘플에 기반하여 추정한다.

$$r(x_1,...,x_S,y_1,...,y_S) = \frac{\sum_{s=1}^{S}(x_s-\bar{x})(y_s-\bar{y})}{\sqrt{\sum_{s=1}^{S}(x_s-\bar{x})^2}\sqrt{\sum_{s=1}^{S}(y_s-\bar{y})^2}} \quad (3)$$

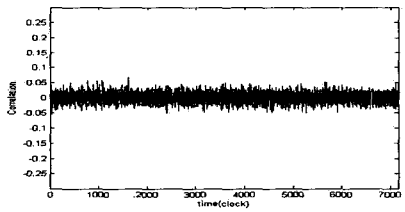

그림 9. 마스킹 SBOX에 대한 올바른 키

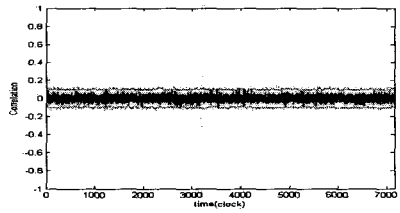

그림 10. 마스킹 SBOX에 대한 모든 키(256)

두 벡터 $H_{k=fixed,1...S}$와 $P_{1...S,t=fixed}$ 간의 피어선 상관계수를 계산하면, 상관계수행렬 $R_{1...K,1...T}$가 생성된다. 이러한 상관계수들에 의해 상관도 $\rho_{1...K,1...T}$를 추정할 수 있다. $P_{1...S\forall t\neq t_c}$과 $H_{\forall k\neq k_c,1...S}$는 거의 상관관계가 없기 때문에 상관도 $\rho_{\forall k\neq k_c,\forall t\neq t_c}$는 $\rho_{\forall k=k_c,\forall t=t_c}$에 비해 상당히 작은 값을 가진다. 차분전력분석공격에서 부분키 k_c를 찾기 위해서 필요한 샘플의 수는 $\rho_{\forall k=k_c,\forall t=t_c}$에 의존한다. 즉 $\rho_{\forall k=k_c,\forall t=t_c}$의 값이 크면 클수록 더 적은 수의 샘플이 필요하다.

3. 실험결과

1라운드 16개의 S-BOX에 대해 각각 DPA 공격을 수행하였다. 1개의 S-BOX는 8비트의 입출력을 가지므로 가능한 키의 수는 256개이다. 각각의 S-BOX에 대해 256개의 모든 키에 대해 상관계수를 계산한 결과 의미 있는 값을 가지는 키를 찾을 수 없었다. 따라서 마스킹이 적용된 ARIA는 1차 DPA 공격에 안전함을 확인할 수 있었다. 그림 9, 10은 각각 마스킹된 S-BOX에 대해 올바른 키를 추측했을 경우와, 그리고 256가지의 모든 가능한 키에 대해 상관계수를 계산한 결과이다.

4. 안전성 및 효율성 분석

이 절에서는 마스킹 기법의 이론적 상관계수 계산, 해밍웨이트 모델을 이용한 시뮬레이션, 그리고 실제 실험결과를 통해 분석하였다. 또한 랜덤 마스킹 기법을 적용하여 실제 구현했을 경우 추가되는 시간과 면적에 대한 비용을 비교 분석하였다. 본 논문에서의 분석은 AES 등 다른 알고리즘의 마스킹 안전성에도 적용할 수 있다.

1) 이론적 상관계수 계산

스마트카드에서 수집한 전력소모량의 파형(ℙ)과 공격자가 추정한 전력모델(H)간의 상관계수는 추측한 키가 정확하고 공격시점이 정확할 때($\rho_{\forall k=k_c,\forall t=t_c}$) 가장 높다. 따라서 DPA 공격에 안전하기 위해서는 공격자가 추정한 전력모델과 실제 스마트카드에서 얻어지는 전력파형간의 상관계수를 줄여 올바른 키와 틀린 키를 추측했을 경우를 구분할 수 없게 만들어야 한다. 마스킹은 공격자가 올바른 키를 추측했을 경우와 틀린 키를 추측했을 경우의 상관계수를 구분할 수 없게 만드는 효과를 가져온다. 올바른 키 및 틀린 키를 추측했을 경우 모두 상관계수가 0이 된다. 이는 마스크값을 모를 경우 공격자가 추정한 모델과 스마트카드의 전력소모량 파형은 서로 독립변수가 된다. 즉 식 (2)에서 E(ℍℙ)=E(ℍ)E(ℙ)가 되어 분자 E(ℍℙ)-E(ℍ)E(ℙ)를 0으로 만들게 되므로 상관계수가 0이 된다. 이는 공격자가 추측한 키와 상관없으며, 따라서 올바른 키를 추측했을 경우와 틀린 키를 추측했을 경우를 구분할 수 없게 된다. 이는 공격자가 마스크값을 알 수 없기 때문으로, 마스크를 추측할 수도 있으나, 마스크의 종류가 256가지로 매 알고리즘 수행시마다 변하기 때문에 많은 전력소모량 파형의 수집이 필요하기 때문에 현실적으로 공격이 불가능하다.

$$\rho(\mathbb{H},\mathbb{P})=\frac{E(\mathbb{H}\mathbb{P})-E(\mathbb{H})E(\mathbb{P})}{\sqrt{Var(\mathbb{H})\,Var(\mathbb{P})}}=0 \qquad (4)$$

2) 추정모델과 전력소비량과의 상관관계 시뮬레이션

본 논문에서는 마스킹 적용전과 적용후의 공격자의 추정모델과 실제 스마트카드에서 수집한 전력소모량 파형간의 상관계수를 시뮬레이션하여 안전성을 분석하였다. 단 여기에서의 가정은 전술한 바와 같이 스마트카드의 전력소모량은 해밍웨이트 모델에 기반했다는 것이다. 이와 같은 가정을 가지고 마스킹을 적용하지 않은 경우와 적용한 경우에 대해 공격자의 추정모델과 실제 스마트카드에서 얻을 수 있는 전력소모량간의 상관관계를 얻기 위한 시뮬레이션을 다음과 같이 수행하였다. 모든 경우에 있어서 평문 $P_{1..N}$과 마스킹을 적용한 경우 마스크 $M_{1...N}$은 MATLAB의 난수발생기를 이용하여 임의로 100,000개를 생성하였다.

가) 마스킹을 적용하지 않은 경우

- 공격자가 만들 수 있는 추정모델 : N개의 평문 $P_{1...N}$과 8비트 입력키(256개)각각에 대해 XOR연산 결과 값을 입력으로 하는 S-BOX의 출력 해밍웨이트 $HW(S(P_{1...N}\oplus K_{0\cdots255}))$

$$\mathrm{H}=\begin{pmatrix} HW(S(P_1\oplus K_0)) & \cdots & HW(S(P_1\oplus K_{255})) \\ HW(S(P_2\oplus K_0)) & \cdots & HW(S(P_2\oplus K_{255})) \\ HW(S(P_3\oplus K_0)) & \cdots & HW(S(P_3\oplus K_{255})) \\ \vdots & & \vdots \\ HW(S(P_N\oplus K_0)) & \cdots & HW(S(P_N\oplus K_{255})) \end{pmatrix} \qquad (5)$$

- 실제 스마트카드에서의 전력소모량 파형 : N개의

평문 $P_{1\cdots N}$과 실제 스마트카드 내부에 저장된 8비트 입력키를 이용하여 S-BOX연산을 수행할 때의 전력소모량 파형($HW(S(P_{1\cdots N}\oplus K))$에 비례)

$$P = \begin{pmatrix} HW(S(P_1\oplus K_{real})) \\ HW(S(P_2\oplus K_{real})) \\ HW(S(P_3\oplus K_{real})) \\ \vdots \\ HW(S(P_N\oplus K_{real})) \end{pmatrix} \quad (6)$$

공격자의 추정모델과 실제 스마트카드에서의 전력소모량 파형에 대한 모델에 대한 상관계수 ρ(H, P)를 식(2)에 따라 계산하면, 스마트카드에서 실제 사용되고 있는 암호키와의 이론적 상관계수는 1이 된다. 물론, 이 값은 잡음, 스마트카드의 다른 전력특성을 고려하지 않은 경우이므로 실제에 있어서는 상관계수가 줄어들 수 있다.

나) 마스킹 을 적용한 경우

마스킹을 적용한 경우는 고정마스크와 가변마스크를 적용한 경우로 구분할 수 있다.

(1) 고정 마스크

- 공격자가 만들 수 있는 추정모델 : N개의 평문 $P_{1\cdots N}$과 8비트 입력키(256개)각각에 대해 XOR연산 결과 값을 입력으로 하는 S-BOX의 출력 해밍웨이트 $HW(S(P_{1\cdots N}\oplus K_{0\cdots 255}))$, 이는 1)마스킹을 적용하지 않은 경우와 동일하다.
- 실제 스마트카드에서의 전력소모량 파형 : N개의 평문 $P_{1\cdots N}$과 마스크 M,실제 스마트카드 내부에 저장된 8비트 입력키를 이용하여 S-BOX연산을 수행할 때의 전력소모량 파형 ($HW(S(P_{1\cdots N}\oplus K)\oplus M)$에 비례)

$$P = \begin{pmatrix} HW(S(P_1\oplus K_{real})\oplus M) \\ HW(S(P_2\oplus K_{real})\oplus M) \\ HW(S(P_3\oplus K_{real})\oplus M) \\ \vdots \\ HW(S(P_N\oplus K_{real})\oplus M) \end{pmatrix} \quad (7)$$

공격자의 추정모델과 실제 스마트카드에서의 전력소모량 파형에 대한 모델에 대한 상관계수 ρ(H, P)를 식(2)에 따라 계산하면, 사용한 마스크의 해밍웨이트 수에 따라 최대 0.17~1사이의 값을 가진다. 또한, 고정된 마스크를 사용할 경우에는, 모든 평문의 암호화 과정에서 동일하게 마스크에 의한 전력소모량이 존재하기 때문에 상관관계 분석에 의해 공격이 가능하다.

(2) 가변 마스크

- 공격자가 만들 수 있는 추정모델 : N개의 평문 $P_{1\cdots N}$과 8비트 입력키(256개)각각에 대해 XOR연산 결과 값을 입력으로 하는 S-BOX의 출력 해밍웨이트 $HW(S(P_{1\cdots N}\oplus K_{0\cdots 255}))$, 이는 1)마스킹을 적용하지 않은 경우와 동일하다.
- 실제 스마트카드에서의 전력소모량 파형 : N개의 평문 $P_{1\cdots N}$과 마스크 M,실제 스마트카드 내부에 저장된 8비트 입력키를 이용하여 S-BOX연산을 수행할 때의 전력소모량 파형($HW(S(P_1\oplus K)\oplus M_1)$, $HW(S(P_2\oplus K)\oplus M_2)$ $\cdots HW(S(P_N\oplus K)\oplus M_N)$에 비례)

$$P = \begin{pmatrix} HW(S(P_1\oplus K_{real})\oplus M_1) \\ HW(S(P_2\oplus K_{real})\oplus M_2) \\ HW(S(P_3\oplus K_{real})\oplus M_3) \\ \vdots \\ HW(S(P_N\oplus K_{real})\oplus M_N) \end{pmatrix} \quad (8)$$

공격자의 추정모델과 실제 스마트카드에서의 전력소모량 파형에 대한 모델에 대한 상관계수 ρ(H, P)를 식(2)에 따라 계산하면, 상관계수의 값은 최대 0.0422가 되며, 따라서, 상관계수를 이용한 DPA 공격이 어렵게 된다. 위 3가지 경우에 대한 상관계수를 비교하면 표 3과 같다.

표 1. 상관계수 비교

구 분	최대 상관계수
마스킹 적용전	1
고정 마스크	0.17~1*
가변 마스크	0.0422

* 마스크의 해밍웨이트수에 의존

이는 마스킹을 적용한 경우 1차 DPA공격에 대한 이론적인 안전성을 제공한다. 실제 실험 결과 마스킹을 적용하기 전, 올바른 키를 추측했을 때 공격자의 추정 전력모델과 실제 수집한 전력파형과의 최대 상관관계는 0.49이었으며, 틀린 키를 추측했을 때는 0.1031이었다. 반면, 마스킹을 적용한 후의 상관관계는 최대 0.076이였으며, 틀린 키를 추측했을 때와 차이가 없었다.

3) 공격에 필요한 전력 파형수

DPA 공격에 대한 안전성은 공격에 필요한 전력파형의 수에 의해 결정된다. [9]에서는 최대 상관계

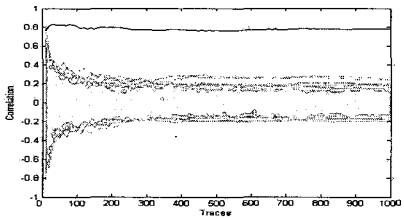

그림 11. 마스킹이 적용되지 않은 ARIA에 대한 DPA 공격에 필요한 전력파형 수

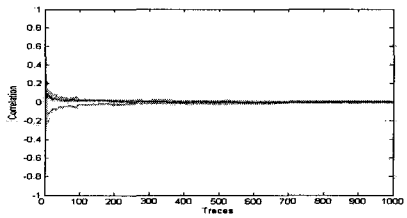

그림 12. 마스킹을 적용한 ARIA에 대한 DPA 공격에 필요한 전력파형 수

수와 공격에 필요한 전력파형의 수의 관계를 다음과 같이 정의하였다.

$$S = 3 + 8\left(\frac{Z_\alpha}{\ln\left(\frac{1+\rho_{\max}}{1-\rho_{\max}}\right)}\right)^2 \quad (9)$$

Z_α는 상관도 분포에 따라 결정되는 값이며, 실험적으로 결정된다. 논문에서는 [9]에서와 같이 $Z_{0.9999} = 3.7190$으로 두고 계산하였다. 계산결과 공격에 필요한 최소 전력파형의 수는 $\rho_{\max} = 0.65$일때 50개이다. 실험결과 마스킹을 적용하기 전 공격에 성공하기 위한 전력파형의 수는 그림11에서와 같이 50개 정도임을 확인할 수 있다. 반면, 마스킹을 적용한 후에는 $\rho_{\max} = 0.076$이 되므로 상관 계수가 너무 작아 전력파형의 수와 상관없이 공격이 이루어지지 않는다. 그림 11, 12에서 검정색은 올바른 키에 대한 상관계수이며, 엷은 회색은 틀린 키에 대한 상관계수이다. 이는 식 9의 결과와 일치한다. 따라서 마스킹을 적용한 ARIA는 이론적 및 실험적 결과로 1차 DPA공격에 안전하다. 하지만, 마스킹 기법만으로는 최근 활발히 연구되고 있는 2차 이상의 고차 DPA공격에 대해 취약하므로, 이에 대한 별도의 대응방법이 추가적으로 필요하다. 효율성 측면에서 마스킹을 적용하기 전과 적용한 후의 메모리 크기, 사이클 수, 프로그램 크기을 비교하면 구현방법에 따라 조금씩 차이가 있으나, 암호 알고리즘 수행시마다 새로운 마스킹 테이블을 계산해서 RAM에 저장해야 하므로 이에 대한 추가적인 메모리가 요구되며, 수행 속도 측면에서도, 마스킹 생성, 그리고 Diffusion계층에서 마스킹 추가 및 제거 연산 등이 추가로 요구되기 때문에, 성능이 떨어진다. 표2에서와 확인할 수 있는 바와 같이 메모리(RAM)가 1,024bytes가 더 요구되며, 처리 속도는 4배정도 느려진다.

표 2. 일반 ARIA와 마스킹된 ARIA의 구현 cost 비교

	ARIA	마스킹ARIA
RAM	-	1,024bytes
ROM(S-BOX)	1,024bytes	1,024bytes
마스킹 생성 싸이클수	-	19,144
12라운드 암호 싸이클수	7,920	11,352
1차 DPA에 대한 취약성	취약	안전

V. 결 론

본 논문에서는 마스킹된 ARIA의 이론적 안전성을 분석하고, AVR기반의 8비트 마이크로프로세서를 사용하는 스마트카드에 구현하여 1차 DPA 공격에 의해 안전함을 실험적으로 확인하였다. 마스킹은 아주 적은 비용으로 적용할 수 있는 대응방법으로, 소프트웨어로 블록암호를 구현할 경우, 모든 스마트카드에서 1차 DPA 공격에 대한 대응방법으로 손쉽게 적용이 가능한 방법이다. 논문에서 구현된 ARIA는 DPA 공격을 막기 위한 랜덤 마스킹 기법은 추가적인 구현 비용이 필요하지만 현재의 공격 기술을 충분히 고려해 보면 구현시에는 이와 같은 대응기술이 필히 적용되어야 할 것이다. 본 논문의 기여는 국내 표준 암호 알고리듬인 ARIA에 랜덤 마스킹 기법을 적용해 봄으로써 전력파형과 비밀 키 간의 상관성을 제거함으로써 DPA에 대한 안전성을 유지할 수 있음을 실제 구현을 통하여 검증한 것이다.

참 고 문 헌

[1] Paul Kocher, Joshua Jaffe, and Benjamin Jun, "Differential Power Analysis," *CRYPTO '99*, Springer-Verlag, 1999, pp.388-397

[2] Daesung Kwon et al., "New Block Cipher ARIA," *ICISC 2002*, Springer-Verlag, 2002, pp.541-548

[3] JaeCheol Ha, ChangKyun Kim, Sang-

Jae Moon, IlHwan Park, and HyungSo Yoo, "Differential Power Analysis on Block Cipher ARIA," *HPCC 2005*, Springer-Verlag, 2005, pp.541-548

[4] Thomas S. Messerges, "Power Analysis Attacks and Countermeasures for Cryptographic Algorithms," Ph.D Thesis 2000, pp.541-548

[5] L.Goubin and J.Patarin, " DES and Differential Power Analysis - The Duplication Method," *CHES 1999*, LNCS 1717, pp.158-172, Springer, 1999

[6] K.Tiri, M.Akmal, and I.Verbauwhede, "A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withsatand Differential Power Analysis on Smart Cards," *ESSCIRC2002*, 2002

[7] K.Tiri and I.Verbauwhede, "Securing Encryption Algorithms against DPA at the Logic Level:Next Generation Smart Card Technology," *CHES 2003*, LNCS 2779, pp.125-136, Springer, 2003

[8] Louis Goubin, "A Sound Method for Switching between Boolean and Arithmetic Masking," *CHES 2001*, Springer-Verlag, 2001, pp.3-15

[9] Stefan Mangard, "Hardware Countermeasures against DPA - A Statistical Analysis of Their Effectiveness," *CT-RSA2004*, Springer-Verlag, 2004, pp.222-235

부록 1. 마스킹된 ARIA의 중간값 연산 과정

1. 입력 128비트 :

x_0	x_1	x_2	x_3
x_4	x_5	x_6	x_7
x_8	x_9	x_{10}	x_{11}
x_{12}	x_{13}	x_{14}	x_{15}

2. 입·출력 마스크 연산

$x_0 \oplus m$	$x_1 \oplus m$	$x_2 \oplus m$	$x_3 \oplus m$
$x_4 \oplus m$	$x_5 \oplus m$	$x_6 \oplus m$	$x_7 \oplus m$
$x_8 \oplus m$	$x_9 \oplus m$	$x_{10} \oplus m$	$x_{11} \oplus m$
$x_{12} \oplus m$	$x_{13} \oplus m$	$x_{14} \oplus m$	$x_{15} \oplus m$

3. AddRoundKey연산 후

$x_0 \oplus m \oplus k_0$	$x_1 \oplus m \oplus k_1$	$x_2 \oplus m \oplus k_2$	$x_3 \oplus m \oplus k_3$
$x_4 \oplus m \oplus k_4$	$x_5 \oplus m \oplus k_5$	$x_6 \oplus m \oplus k_6$	$x_7 \oplus m \oplus k_7$
$x_8 \oplus m \oplus k_8$	$x_9 \oplus m \oplus k_9$	$x_{10} \oplus m \oplus k_{10}$	$x_{11} \oplus m \oplus k_{11}$
$x_{12} \oplus m \oplus k_{12}$	$x_{13} \oplus m \oplus k_{13}$	$x_{14} \oplus m \oplus k_{14}$	$x_{15} \oplus m \oplus k_{15}$

4. 마스킹된 SBOX 연산 :

$S(x_0 \oplus k_0) \oplus m$	$S(x_1 \oplus k_1) \oplus m$	$S(x_2 \oplus k_2) \oplus m$	$S(x_3 \oplus k_3) \oplus m$
$S(x_4 \oplus k_4) \oplus m$	$S(x_5 \oplus k_5) \oplus m$	$S(x_6 \oplus k_6) \oplus m$	$S(x_7 \oplus k_7) \oplus m$
$S(x_8 \oplus k_8) \oplus m$	$S(x_9 \oplus k_9) \oplus m$	$S(x_{10} \oplus k_{10}) \oplus m$	$S(x_{11} \oplus k_{11}) \oplus m$
$S(x_{12} \oplus k_{12}) \oplus m$	$S(x_{13} \oplus k_{13}) \oplus m$	$S(x_{14} \oplus k_{14}) \oplus m$	$S(x_{15} \oplus k_{15}) \oplus m$

5. SBOX출력의 각 행에 대한 마스킹

$S(x_0 \oplus k_0) \oplus m \oplus m_1$	$S(x_1 \oplus k_1) \oplus m \oplus m_1$	$S(x_2 \oplus k_2) \oplus m \oplus m_1$	$S(x_3 \oplus k_3) \oplus m \oplus m_1$
$S(x_4 \oplus k_4) \oplus m \oplus m_2$	$S(x_5 \oplus k_5) \oplus m \oplus m_2$	$S(x_6 \oplus k_6) \oplus m \oplus m_2$	$S(x_7 \oplus k_7) \oplus m \oplus m_2$
$S(x_8 \oplus k_8) \oplus m \oplus m_3$	$S(x_9 \oplus k_9) \oplus m \oplus m_3$	$S(x_{10} \oplus k_{10}) \oplus m \oplus m_3$	$S(x_{11} \oplus k_{11}) \oplus m \oplus m_3$
$S(x_{12} \oplus k_{12}) \oplus m \oplus m_4$	$S(x_{13} \oplus k_{13}) \oplus m \oplus m_4$	$S(x_{14} \oplus k_{14}) \oplus m \oplus m_4$	$S(x_{15} \oplus k_{15}) \oplus m \oplus m_4$

6. Diffusion 연산 :

Diffusion 연산은 SBOX 출력 7개의 배타적 논리합 연산으로 이루어진다. 즉, s_i를 SBOX의 i번째 출력이라고 했을 때, Diffusion연산의 첫 번째 출력은 다음과 같이 계산된다.

$$\begin{aligned} mask\_y_0 &= s_3 \oplus s_4 \oplus s_6 \oplus s_8 \oplus s_9 \oplus s_{13} \oplus s_{14} \\ &= S(x_3 \oplus k_3) \oplus m \oplus m_1 \oplus S(x_4 \oplus k_4) \oplus m \oplus m_2 \\ &\quad \oplus S(x_6 \oplus k_6) \oplus m \oplus m_2 \oplus S(x_8 \oplus k_8) \oplus m \oplus m_3 \\ &\quad \oplus S(x_9 \oplus k_9) \oplus m \oplus m_3 \oplus S(x_{13} \oplus k_{13}) \oplus m \oplus m_4 \\ &\quad \oplus S(x_{14} \oplus k_{14}) \oplus m \oplus m_4 \\ &= S(x_3 \oplus k_3) \oplus S(x_4 \oplus k_4) \oplus S(x_6 \oplus k_6) \oplus S(x_8 \oplus k_8) \\ &\quad \oplus S(x_9 \oplus k_9) \oplus S(x_{13} \oplus k_{13}) \oplus S(x_{14} \oplus k_{14}) \oplus m \oplus m \end{aligned}$$

식(1)에서와 같이 마스킹을 적용하기 전 출력에 두 개의 마스크 m, m_1이 배타적 논리합 연산으로 결합되어 있다. 마스킹을 적용하기 전 출력을 y_i라고 했을 때 마스킹을 적용한 후의 각 데이터는 다음과 같이 표현된다.

$y_0 \oplus m \oplus m_1$	$y_1 \oplus m \oplus m_1$	$y_2 \oplus m \oplus m_1$	$y_3 \oplus m \oplus m_1$
$y_4 \oplus m \oplus m_2$	$y_5 \oplus m \oplus m_2$	$y_6 \oplus m \oplus m_2$	$y_7 \oplus m \oplus m_2$
$y_8 \oplus m \oplus m_3$	$y_9 \oplus m \oplus m_3$	$y_{10} \oplus m \oplus m_3$	$y_{11} \oplus m \oplus m_3$
$y_{12} \oplus m \oplus m_4$	$y_{13} \oplus m \oplus m_4$	$y_{14} \oplus m \oplus m_4$	$y_{15} \oplus m \oplus m_4$

7. SBOX출력의 각 행에 대한 마스킹 제거

$y_0 \oplus m$	$y_1 \oplus m$	$y_2 \oplus m$	$y_3 \oplus m$
$y_4 \oplus m$	$y_5 \oplus m$	$y_6 \oplus m$	$y_7 \oplus m$
$y_8 \oplus m$	$y_9 \oplus m$	$y_{10} \oplus m$	$y_{11} \oplus m$
$y_{12} \oplus m$	$y_{13} \oplus m$	$y_{14} \oplus m$	$y_{15} \oplus m$

단계 2는 최초 데이터 입력에 대해서 수행되며, 단계 3에서 7은 1라운드에서 11라운드까지 수행이 된다. 이 때의 중간값은 단계 7. SBOX출력의 각 행에 대한

마스킹 제거 후와 같다. 마지막 라운드는 단계 3, 4만 수행이 되며, 이 때의 중간값은 단계 4. 마스킹된 SBOX 연산 후와 같다. 마지막으로 AddRoundKey 연산이 이루어지며, 이 연산은 마스킹 값에 아무런 영향을 미치지 않는다. 따라서 AddRoundKey연산 후의 값과 마스크를 배타적 논리합 연산을 하게 되면, 원래의 암호문을 얻을 수 있다.

〈著者紹介〉

유 형 소 (HyungSo Yoo) 정회원
1997년 2월: 경북대학교 전자공학과 졸업
1999년 2월: 경북대학교 전자공학과 석사
1999년 3월~현재: 경북대학교 전자공학과 박사과정
〈관심분야〉 정보보호, 암호이론, 부채널공격

하 재 철 (JaeCheol Ha) 종신회원
1989년 2월: 경북대학교 전자공학과 졸업
1993년 8월: 경북대학교 전자공학과 석사
1998년 2월: 경북대학교 전자공학과 박사
1998년 3월~2000년 2월: 나사렛대학교 전자계산소장
1999년 3월~현재: 나사렛대학교 정보통신학과 부교수
2002년 3월~현재 : 한국정보보호학회 이사
〈관심분야〉 정보보호, 네트워크 보안, 스마트카드 보안

김 창 균 (ChangKyun Kim)
2001년 2월: 경북대학교 전자전기공학부 졸업
2003년 2월: 경북대학교 전자공학과 석사
2003년 3월~200년 10월: 경북대학교 전자공학과 박사과정
2004년 11월~현재: 국가보안기술연구소
〈관심분야〉 정보보호기술

박 일 환 (IlHwan Park)
1988년 2월: 고려대학교 수학과 졸업
1990년 2월: 고려대학교 수학과 석사
1996년 2월: 고려대학교 수학과 박사
1996년 5월~1999년 12월: 한국전자통신연구원
2000년 1월~현재: 국가보안기술연구소
〈관심분야〉 정보보호이론

문 상 재 (SangJae Monn) 종신회원
1972년 2월: 서울대학교 공업교육(전자)과 졸업
1974년 2월: 서울대학교 전자공학과 석사
1984년 6월: 미국 UCLA 전자공학과 박사
1984년 7월~1985년 6월: UCLA Postdoctoral 근무
1984년 7월~1985년 6월: 미국 OMNET 컨설턴트
1974년 12월~현재: 경북대학교 공과대학 전자전기컴퓨터공학부 교수
2000년 8월~현재: 경북대학교 이동네트워크 정보보호기술 연구센터 소장
2002년 2월~현재: 한국정보보호학회 명예회장
〈관심분야〉 정보보호, 디지털 통신, 이동 네트워크