情報保護學會論文誌
第16卷 第3號, 2006. 6

저메모리 환경에 적합한 마스킹기반의 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 Masking-based ARIA Countermeasure for Low Memory Environment Resistant to Differential Power Attack*

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

본 논문에서는 국가표준 암호인 ARIA에 효율적인 마스킹 기법을 제안하였다. 4개의 SBOX에 대해 각각 마스킹을 적용하는 기존의 마스킹 기법과 달리, 본 논문에서는 1개의 테이블만을 사용한 마스킹 구현기법을 제안하고, 실험을 통하여 1차 DPA 공격에 안전함을 확인하였다. 제안하는 역원테이블 방법을 이용하면 한 번의 역원 테이블을 만드는데 많은 시간이 필요하지만 마스킹을 여러 번 수행하는 경우에는 테이블을 반복적으로 이용하게 되므로 연산속도를 크게 개선할 수 있다.

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 propose a new masking technique as a countermeasure against first order DPA attack and apply it to the ARIA. The proposed method is suitable for low memory environment. By using this countermeasure, we verified that it is secure against first order DPA attack. In addition, our method based on precomputation of inverse table can reduce the computational cost as increasing the number of S-BOX masking.

Keywords : *Differential Power Analysis(DPA), Block Cipher ARIA, Masking Countermeasure*

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[4-8], 랜덤 지연시간 삽입[5], 새로운 하드웨어 논리 스타일 사용[10,11] 등이 제안되었으나, 가장 활발히 연구가 되고 있는 분야가

접수일: 2006년 4월 6일; 채택일: 2006년 6월 1일
* 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었음.
† 주저자, ydilly@hotmail.com
‡ 교신저자, jcha@kornu.ac.kr

마스킹을 사용한 대응방법이다. 마스킹 기법은 암호 연산의 중간값과 전력소모량간의 연결고리를 제거하기 위하여, 임의의 난수를 사용하여 데이터를 변환하는 방법이다. 최근 마스킹을 적용한 AES를 효율적으로 공격할 수 있는 2차 DPA 공격결과가 발표되었으나, 랜덤화 등 추가적인 대응방법을 적용하기 위해서는 마스킹은 반드시 적용되어야 한다. 현재까지 마스킹 기법에 대한 연구는 AES를 대상으로 이루어졌다. AES에 대한 마스킹 기법을 ARIA에 적용할 수는 있으나, 실제 구현시에는 ARIA의 특성을 고려하는 것이 필요하다. 마스킹 적용시 가장 중요한 부분이 비선형 연산인 SBOX이다. AES는 1개의 SBOX를 사용하지만, ARIA의 경우 4개의 SBOX를 사용하므로 많은 메모리가 필요하며, 마스킹 테이블 생성시 많은 시간이 소모된다. 따라서, 저메모리 환경 및 입력 데이터마다 서로 다른 마스크를 사용하는 경우에는 마스킹된 ARIA를 구현하는 것이 어렵다. 본 논문에서는 이러한 문제점을 해결하기 위해 최근 1차 DPA 공격에 취약함이[3] 밝혀진 국내 표준 암호알고리즘인 ARIA를 저메모리 환경의 스마트카드에서 구현이 가능하도록 마스킹 기법을 제안하고 구현하였으며, 실험을 통하여 1차 DPA공격에 대한 안전성을 검증하였다. 본 논문의 구성은 다음과 같다. II장에서 ARIA의 구조 및 1차 DPA 공격결과를 설명하고, III장에서 저메모리 환경에 적합한 마스킹된 ARIA 구현기법을 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 k비트씩 순환이동
- $A^{\ggg k}$ A의 각 비트를 오른쪽으로 k비트씩 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 계층 : 두 종류의 비선형 함수를 사용하여 입력값을 치환한다.
- Diffusion 계층 : 16×16이진 행렬을 사용하여 S-BOX출력을 확산한다.

2.2.1 S-BOX 계층

ARIA의 SBOX는 $S_1, S_2, S_1^{-1}, S_2^{-1}$ 로 이루어져 있다. 각각의 SBOX는 8비트 입출력을 가지며 다음과 같은 함수에 의해 연산이 이루어진다.

$$S_i : GF(2^8) \rightarrow GF(2^8)$$

$$S_1 : x \rightarrow A \cdot x^{-1} \oplus a \quad (1)$$

$$A = \begin{pmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{pmatrix} \text{ and } a = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$S_2 : x \rightarrow B \cdot x^{247} \oplus b \quad (2)$$

$$B = \begin{pmatrix} 0&1&0&1&1&1&1&0 \\ 0&0&1&1&1&1&0&1 \\ 1&1&0&1&0&1&1&1 \\ 1&0&0&1&1&1&0&1 \\ 0&0&1&0&1&1&0&0 \\ 1&0&0&0&0&0&0&1 \\ 0&1&0&1&1&1&0&1 \\ 1&1&0&1&0&0&1&1 \end{pmatrix} \text{ and } b = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

일반적으로 SBOX 연산은 식 (1), (2)에 의해 사

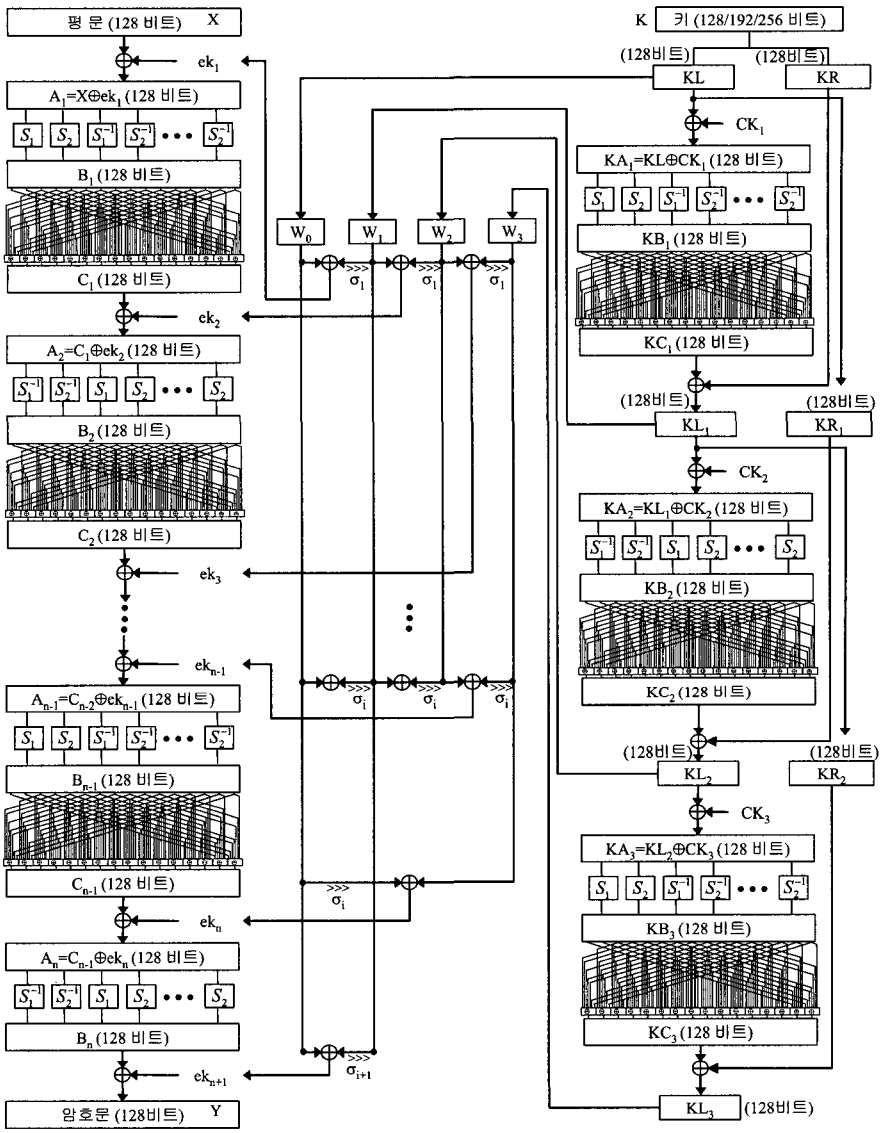


그림 1. ARIA 구조

전 계산된 테이블을 참조하여 이루어진다. ARIA는 짝수 라운드와 홀수 라운드에서 서로 다른 순서로 SBOX 연산을 수행한다.

2.2.2 Diffusion 계층

ARIA의 Diffusion 계층은 16x16 이진 행렬로 구성되어 있으며 다음 수식 및 행렬에 의해 연산이 이루어진다.

$$D : GF(2^8)^{16} \rightarrow GF(2^8)^{16}$$

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{15} \end{pmatrix} = \begin{pmatrix} 0&0&0&1&1&0&1&0&1&1&0&0&0&1&1&0 \\ 0&0&1&0&0&1&0&1&1&1&0&0&1&0&0&1 \\ 0&1&0&0&1&0&1&0&0&0&1&1&1&0&0&1 \\ 1&0&0&0&0&1&0&1&0&0&1&1&0&1&1&0 \\ 1&0&1&0&0&1&0&0&1&0&0&1&0&0&1&1 \\ 0&1&0&1&1&0&0&0&0&1&1&0&0&0&1&1 \\ 1&0&1&0&0&0&0&1&0&1&1&0&1&1&0&0 \\ 0&1&0&1&0&0&1&0&1&0&0&1&1&1&0&0 \\ 1&1&0&0&1&0&0&1&0&0&1&0&0&1&0&1 \\ 1&1&0&0&0&1&1&0&0&0&0&1&1&0&1&0 \\ 0&0&1&1&0&1&1&0&1&0&0&0&0&1&0&0 \\ 0&0&1&1&1&0&0&1&0&1&0&0&1&0&1&0 \\ 0&1&1&0&0&0&1&1&0&1&0&1&1&0&0&0 \\ 1&0&0&1&0&0&1&1&1&0&1&0&0&1&0&0 \\ 1&0&0&1&1&1&0&0&0&1&0&1&0&0&1&0 \\ 0&1&1&0&1&1&0&0&1&0&1&0&0&0&0&1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{pmatrix}$$

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 구조이다.

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 2는 ARIA 1, 2 라운드에 대한 전력파형으로, AddRoundKey 연산, SBOX 연산, Diffusion연산이 각각 수행되는 구간을 알 수 있다. 그림 3과 같이 1라운드의 S-BOX출력에 대해 DPA 공격을 적용한 실험에서 3000개의 전력신호샘플을 이용하여 라운드키를 찾을 수 있었다. 그림 4는 올바른 키를 추측했을 때의 상관계수이며, 그림 5는 S-BOX의 입력키로 추측이 가능한 256가지 모두에 대한 상관계수를 비교하였다. 그림 4, 5에서 볼 수 있듯이 약 올바른 키 추측 시 500[time(clock)]에서 절대값이 최대인 상관계수를 얻을 수 있었으며, 틀린 키일 경우에는 0~0.15사이의 상관계수를 얻을 수 있었다. 그림 5에서 검정색은 올바른 키 추측 시의 파형이며 엷은 회색은 틀린 키 추측 시의 파형이다.

III. 기존 마스킹 기법

1. 마스킹 개요

마스킹 기법은 실제 스마트카드에 구현된 암호알고리즘 수행시 소모되는 전력소모량과 공격자의 추정 모델간의 상관관계를 제거하는 방법으로, 비밀 키에 대한 정보를 유출하지 않도록 알고리즘을 변경하는 것이다. 이것은 알고리즘의 중간값을 랜덤값(마스크)을 이용하여 공격자가 예측할 수 없도록 랜덤화시키는 방법에 의해 이루어지는 것이다. 마스킹 기법은

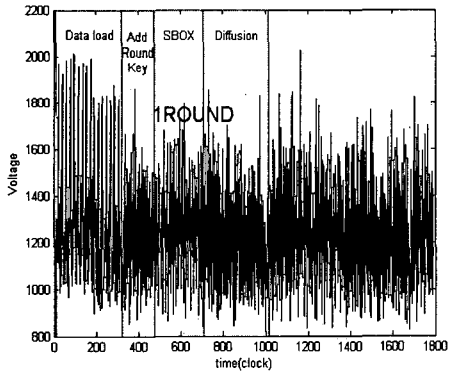


그림 2. ARIA 1라운드 전력파형

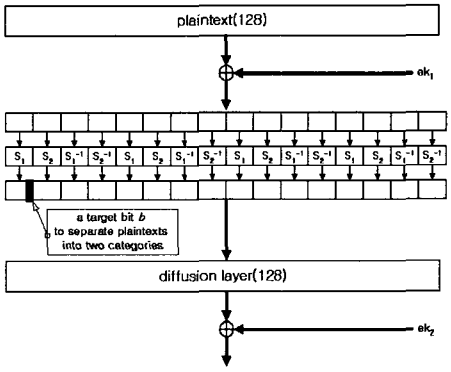


그림 3. 1차 DPA 공격 대상

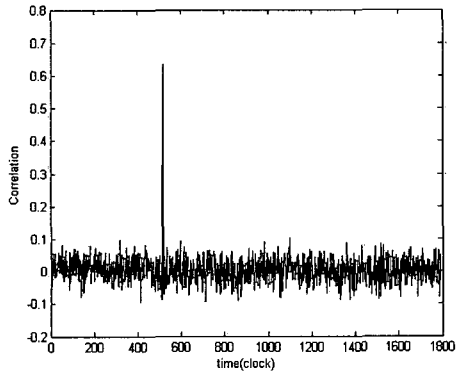


그림 4. SBOX2 올바른 키

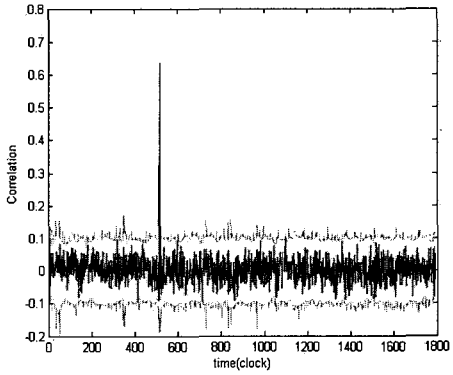


그림 5. SBOX2 모든 키(256)

DPA 공격을 방어하기 위해 필수적인 대응방법이다. 최근 고차 DPA 공격에 의해 마스킹 기법만을 사용한 대응방법에 대한 효율적인 공격이 가능함이 알려졌으나, 랜덤화 등 추가적인 대응방법을 적용하기 위해서는 마스킹 기법이 먼저 구현되어야 하므로, 효율적인 마스킹 기법에 대한 연구가 활발히 이루어지고 있다. 현재까지 제안된 마스킹 기법으로 가산 마스킹[4], multiplicative 마스킹[5,6], 그리고 SBOX의 역원 계산을 변경하는 방법[7] 등이 있으나, multiplicative 마스킹의 경우 zero-value 공격에 의해 취약점이 발견[6,8]되었으며, SBOX의 역원 계산을 변경하는 방법은 하드웨어 구현을 고려한 경우로 소프트웨어 구현의 경우에는 적합하지 않다. 따라서, 소프트웨어 구현으로 가장 많이 사용되는 방법은 가산 마스킹 기법이다. 이번 장에서는 기존의 가산 마스킹 기법의 구현방법에 대해 살펴본다.

2. 가산 마스킹 구현방법

가산 마스킹은 암호알고리즘 수행시의 중간값을 랜덤값(마스크)과 XOR 연산을 이용하여 결합하여 실제 스마트카드의 전력소모량과 공격자의 추정모델 간의 상관관계를 제거하는 방법으로, 랜덤값은 공격자가 예측할 수 없도록 독립적으로 균일하게 분포된 값이어야 하며, 암호알고리즘 수행시마다 다른 값을 사용하여야 한다. 일반적으로 가산 마스킹을 적용하여 새로운 테이블을 계산하는 방법은 다음과 같다[4].

$$\text{MaskedSubBytes}(x+m) = \text{SubBytes}(x) + m' \quad (1)$$

Algorithm 1 Computation of Masked SubBytes
입력 : m, m' 출력 : MaskedSubBytes(x+m) = SubBytes(x)+m'
1 : for i = 0 to 255 do 2 : MaskedSubBytes(i+m) = Subbytes(i)+m' 3 : end for 4 : Return(MaskedSubBytes)

알고리즘 1에서 1개의 SBOX와 마스킹된 SBOX를 저장하기 위하여 512bytes의 RAM 또는 256 bytes의 ROM(SBOX 저장), 256bytes의 RAM을 필요로 한다. ARIA의 경우 4개의 SBOX를 사용하므로 필요한 메모리 양은 4배로 증가하게 되어, 2Kbytes의 RAM 또는 1Kbytes의 ROM과 1Kbytes의 RAM을 필요로 한다. 고차 DPA 공격을 방어하기 위해 각각의 입력바이트에 대해 서로 다른 마스크를 사용할 경우, 사용되는 수만큼의 MaskedSubBytes()테이블을 새로 계산하여야 한다. 예를 들어, 16개의 입력바이트 모두에 대해 서로 다른 마스크를 사용할 경우, 16개의 마스킹 된 테이블이 필요하다. 따라서, 마스킹 테이블 계산에 많은 시간이 소모되며, 메모리 크기 또한 크게 늘어나게 된다. 기존의 가산 마스킹 연산을 이용한 마스킹된 테이블 생성 및 이를 이용한 ARIA의 마스킹된 SBOX 연산은 그림 6 및 식(2)~(5)와 같다.

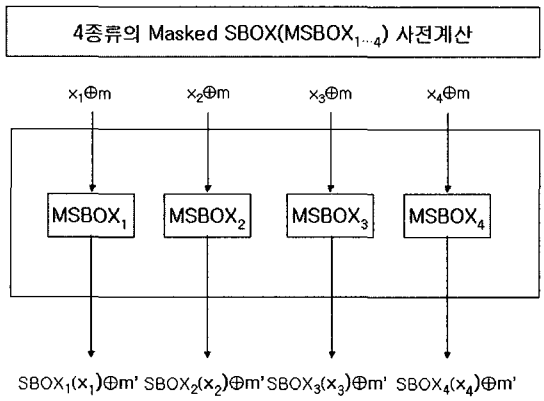


그림 6. 기존 가산 마스킹 SBOX 연산

$$MaskedS_1 : (x \oplus m) \rightarrow A \cdot (x \oplus m)^{-1} \oplus a \quad (2)$$

$$MaskedS_2 : (x \oplus m) \rightarrow B \cdot (x \oplus m)^{247} \oplus b \quad (3)$$

$$MaskedS_3 : (x \oplus m) \rightarrow (A^{-1} \cdot (x \oplus m \oplus a))^{-1} \quad (4)$$

$$MaskedS_4 : (x \oplus m) \rightarrow (B^{-1} \cdot (x \oplus m \oplus b))^{1/247} \quad (5)$$

식(2)~(5)는 알고리즘 1에 의해 사전계산된 마스킹된 테이블을 다음 식을 이용하여 한 번의 테이블 참조만으로 연산을 수행할 수 있다.

$$MSBOX_1(x \oplus m) = SBOX_1(x) \oplus m'$$

IV. 저메모리 환경에 적합한 마스킹 기반 ARIA 대응기법 구현

II장에서 살펴본 바와 같이 ARIA의 각 라운드는 선형 연산인 AddRoundKey, Diffusion 계층, 그

리고 비선형연산인 SBOX계층으로 이루어져 있다. 선형 연산에 대한 마스킹 적용은 어려움없이 적용할 수 있으나, 비선형 연산은 세심한 주의가 필요하다. III장에서 살펴본 것처럼, 지금까지 AES에 적용하기 위해 제안된 가산 마스킹, multiplicative 마스킹 그리고 대수 연산에 기반한 마스킹 중, 소프트웨어 구현에 가장 적합한 방법은 가산 마스킹 기법이다. 하지만, 기존의 가산 마스킹 구현기법을 4종류의 SBOX를 사용하는 ARIA에 적용할 경우에는 많은 양의 메모리가 필요하게 되고, 각 입력 바이트마다 서로 다른 마스크를 사용할 경우에는 처리속도는 더욱 떨어진다. 본 논문에서는 이러한 문제점을 해결하기 위해 역원테이블 1개만을 이용하여 마스킹을 적용할 수 있는 방법을 제안한다.

1. 역원 테이블만을 이용한 SBOX 연산

2.2.1절에서 살펴본 바와 같이 ARIA는 S_1, S_2, S_1^{-1}, S_2^{-1} 4종류의 SBOX를 사용하며, 각각의 SBOX는 256바이트의 테이블을 사전 계산하여 메모리에 저장되어 모두 1Kbytes의 메모리(RAM 또는 ROM)가 필요하다. 따라서 마스킹을 적용시 4종류의 마스킹된 SBOX가 생성이 되어 추가적으로 1Kbytes의 메모리가 더 필요하게 되어 저메모리 환경에서는 구현이 어렵게 된다. 본 논문에서는 이러한 문제점을 해결하기 위해 1개의 테이블만을 사용하여 마스킹을 구현할 수 있는 기법을 제안하였다. $GF(2^8)$에서 $x^{247}=x^{-8}=C\cdot x^{-1}$ (C : 8×8 이진 행렬)로 표현이 가능하다. 이는 모든 선형함수는 선형방정식으로 표현할 수 있는 성질을 이용한 것이다. 따라서 각각의 SBOX는 다음과 같이 역원계산만을 사용하여 연산이 가능하다.

$$S_i : GF(2^8)\rightarrow GF(2^8)$$

$$S_1 : x\rightarrow A\cdot x^{-1}\oplus a \tag{6}$$

$$A=\begin{pmatrix}10001111\\11000111\\11100011\\11110001\\11111000\\01111100\\00111110\\00011111\end{pmatrix} \text{ and } a=\begin{pmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{pmatrix}$$

$$S_2 : x\rightarrow (B\cdot C)\cdot x^{-1}\oplus b \tag{7}$$

$$B\cdot C=\begin{pmatrix}01010111\\00111111\\11101101\\11000011\\01000011\\11001110\\01100011\\11110110\end{pmatrix} \text{ and } b=\begin{pmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{pmatrix}$$

$$S_3 : x\rightarrow (A^{-1}\cdot (x\oplus a))^{-1} \tag{8}$$

$$A^{-1}=\begin{pmatrix}00100101\\10010010\\01001001\\10100100\\01010010\\00101001\\10010100\\01001010\end{pmatrix} \text{ and } a'=\begin{pmatrix}1\\0\\1\\0\\0\\0\\0\\0\end{pmatrix}$$

식(8)은 식(8-1)과 같이 표현할 수 있다.

$$\begin{aligned}&(A^{-1}\cdot (x\oplus a))^{-1} \\ &=\{(A^{-1}\cdot x)\oplus(A^{-1}\cdot a)\}^{-1} \\ &=\{(A^{-1}\cdot x)\oplus a'\}^{-1}\end{aligned} \tag{8-1}$$

이는 식(6)의 SBOX 연산이 역원계산과 선형변환(affine 변환)의 합성함수로 이루어져 있으며, 이의 역함수는 선형변환과 역원 계산의 순서로 이루어지는 것을 보여주기 위한 것이다.

$$S_4 : x\rightarrow ((B\cdot C)^{-1}\cdot (x\oplus b))^{-1} \tag{9}$$

$$(BC)^{-1}=\begin{pmatrix}00011000\\00100110\\00001010\\11100011\\11101100\\01101011\\10111101\\10010011\end{pmatrix} \text{ and } b'=\begin{pmatrix}0\\0\\1\\1\\0\\1\\0\\0\end{pmatrix}$$

식(9)는 식(9-1)과 같이 표현할 수 있다.

$$\begin{aligned}&((B\cdot C)^{-1}\cdot (x\oplus b))^{-1} \\ &=\{((B\cdot C)^{-1}\cdot x)\oplus((B\cdot C)^{-1}\cdot b)\}^{-1} \\ &=\{((B\cdot C)^{-1}\cdot x)\oplus b'\}^{-1}\end{aligned} \tag{9-1}$$

2. 저메모리 환경에 적합한 마스킹 기반 ARIA 대응 기법 구현

본 논문에서는 각 연산의 입출력에 가변 마스크를

사용한 additive 마스킹을 적용하였다. ARIA의 각 라운드는 AddRoundKey, SubBytes, Diffusion의 3가지 연산으로 이루어져 있으며, 각 연산에 대해 다음과 같이 마스킹을 적용하였다.

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

마스킹된 SubBytes : III장에서 살펴본 바와 같이 일반적으로 SubBytes는 기존의 S-BOX에 마스크 m과 m′를 적용한 가산 마스킹 기법으로 구현한다.

$$S'(X\oplus m)=S(X)\oplus m' \tag{10}$$

본 논문에서는 기존의 SBOX에 마스킹을 적용하는 대신 역원테이블에 대해 가산 마스킹을 적용하여, 4개의 SBOX에서 공통으로 사용할 수 있도록 하였다.

$$InvTable'(X\oplus m)=InvTable(X)\oplus m' \tag{11}$$

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

마스킹된 데이터에 대한 SBOX 연산은 알고리즘 2를 사용하여 계산된 역원 테이블과 선형연산(affine 변환)을 사용하여 이루어진다. 4개의 SBOX에 대한 마스킹은 수식 (12)~(13)에 의해 연산이 이루어지며, 그림 7과 같이 표현할 수 있다.

$$\begin{aligned}&MaskedS_1:(x\oplus m)\rightarrow A\cdot(x\oplus m)^{-1}\oplus a \\ &=A\cdot(x^{-1}\oplus m')\oplus a=A\cdot(y')\oplus a\end{aligned} \tag{12}$$

$$\begin{aligned}&MaskedS_2:(x\oplus m)\rightarrow(B\cdot C)\cdot(x\oplus m)^{-1}\oplus b \\ &=(B\cdot C)\cdot(x^{-1}\oplus m')\oplus b=(B\cdot C)\cdot(y'')\oplus b\end{aligned} \tag{13}$$

$$\begin{aligned}&MaskedS_3:(x\oplus m)\rightarrow\{A^{-1}\cdot(x\oplus m\oplus a)\}^{-1} \\ &=\{(A^{-1}\cdot(x\oplus a))\oplus(A^{-1}\cdot m)\}^{-1} \\ &=\{(A^{-1}\cdot(x\oplus a))\oplus(A^{-1}\cdot m)\oplus(A^{-1}\cdot m)\oplus m\}^{-1} \\ &=\{(A^{-1}\cdot(x\oplus a))\oplus m\}^{-1} \\ &=(x'\oplus m)^{-1}\end{aligned} \tag{14}$$

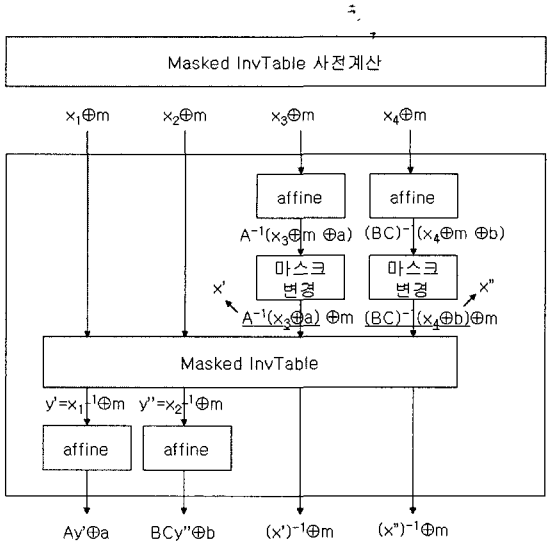


그림 7. 제안 마스킹 SBOX 연산

$$\begin{aligned}&MaskedS_4:(x\oplus m)\rightarrow\{(B\cdot C)^{-1}\cdot(x\oplus m\oplus b)\}^{-1} \\ &=\{((B\cdot C)^{-1}\cdot(x\oplus b))\oplus((B\cdot C)^{-1}\cdot m)\}^{-1} \\ &=\left\{\begin{array}{l}((B\cdot C)^{-1}\cdot(x\oplus b))\oplus((B\cdot C)^{-1}\cdot m)\\ \oplus((B\cdot C)^{-1}\cdot m)\oplus m\end{array}\right\}^{-1} \\ &=\{(B\cdot C)^{-1}\cdot(x\oplus b)\oplus m\}^{-1} \\ &=(x''\oplus m)^{-1}\end{aligned} \tag{15}$$

MaskedS1과 MaskedS2 연산은 식(12),(13)과 같이 알고리즘 2에 의해 사전 계산된 마스킹된 역원 테이블 참조연산과 행렬곱셈으로 연산이 가능하다. 즉 마스킹된 데이터 $x\oplus m$에 대한 역원계산은 역원테이블($InvTable'(x\oplus m)=InvTable(x)\oplus m$)을 이용하여 이루어지며, (식(12),(13)에서는 편의상 $(x\oplus m)^{-1}=x^{-1}\oplus m$으로 표기), 역원연산의 출력에 선형변환 연산이 적용된다. 하지만, MaskedS3와 MaskedS4는 선형연산(affine mapping)후에 역원 테이블 참조가 이루어지기 때문에 중간에 마스크를 변경하는 추가 연산이 필요하다. 즉, MaskdedS3의 경우 선형연산의 결과값이 식(14)에서와 같이 $(A^{-1}\cdot(x\oplus a))\oplus(A^{-1}\cdot m)$가 되어 역원 테이블을 그대로 이용할 수가 없다. (역원 테이블을 이용하기 위해서는 $x'\oplus m$와 같은 형태의 입력이 필요). 따라서, 식(14)의 결과값을 $x'\oplus m$ 와 같은 형태로 변형하는 과정이 추가로 요구된다. 먼저, 식(14)에 마스크m을 XOR 연산을 통하여 추가한 후, $A^{-1}\cdot m$을 제거하기 위해서 $A^{-1}\cdot m$와의 XOR 연산을 수행한다. 최종적인 MaskedS3의 출력값은

$(A^{-1} \cdot (x \oplus a) \oplus m)^{-1}$ 이 된다. 즉 선형연산의 결과값에 마스킹을 적용한 후 역원 테이블 참조 연산을 수행한 결과이다. 이 때 주의해야 할 점은 $A^{-1} \cdot m$와 m의 연산 순서인데 $A^{-1} \cdot m$를 먼저 수행할 경우, 중간에 마스크가 제거되어 공격이 가능하다. 따라서 m을 먼저 추가한 후에 $A^{-1} \cdot m$를 제거해야 한다. 식(15)도 같은 방법으로 연산이 가능하다. 제안한 마스킹 구현기법은 4개의 SBOX가 1개의 역원 테이블을 공유하므로 256bytes의 RAM으로도 구현이 가능하다.

마스킹된 Diffusion Layer : Diffusion계층은 선형연산이므로, 마스킹된 S-BOX의 결과값과 마스크에 대한 확산 연산을 별도로 수행한 후 결과값을 결합할 수 있다. 따라서, 마스크에 대한 사전계산을 통해 연산량을 줄일 수 있다. 단, Diffusion연산 도중 마스크 값이 중첩된 배타적 논리합 연산에 의해 제거될 수 있으므로, 중간값의 각 행마다 다른 마스크를 적용하였다. 마스킹 구조 및 마스킹 ARIA의 Pseudo 코드는 그림 8, 9과 같다. 그림 8의 "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. MSubBytes는 위의 마스킹된 InvTable에서 설명한 방법으로 알고리즘 수행전에 미리 계산하여 RAM에 저장된 테이블과 행렬 곱셈을 이용하여 계산된다.

V. 마스킹 기반 ARIA에 대한 1차 DPA 공격

1. DPA 실험환경

마스킹이 적용된 ARIA에 대한 DPA 공격을 수행하기 위하여, AVR기반의 8비트 마이크로프로세서가 장착된 IC카드, LeCroy사의 LC584 디지털 오실로스코프, Infineon사의 스마트카드 리더기를 사용하였다. CrossStudio for AVR을 사용하여 마스킹된 ARIA를 마이크로프로세서에 적합한 형태로 컴파일하였으며, 측정된 전력소모량에 대한 DPA 분석을 위하여 MATLAB으로 구현한 부채널 공격 TOOLBOX를 사용하였다.

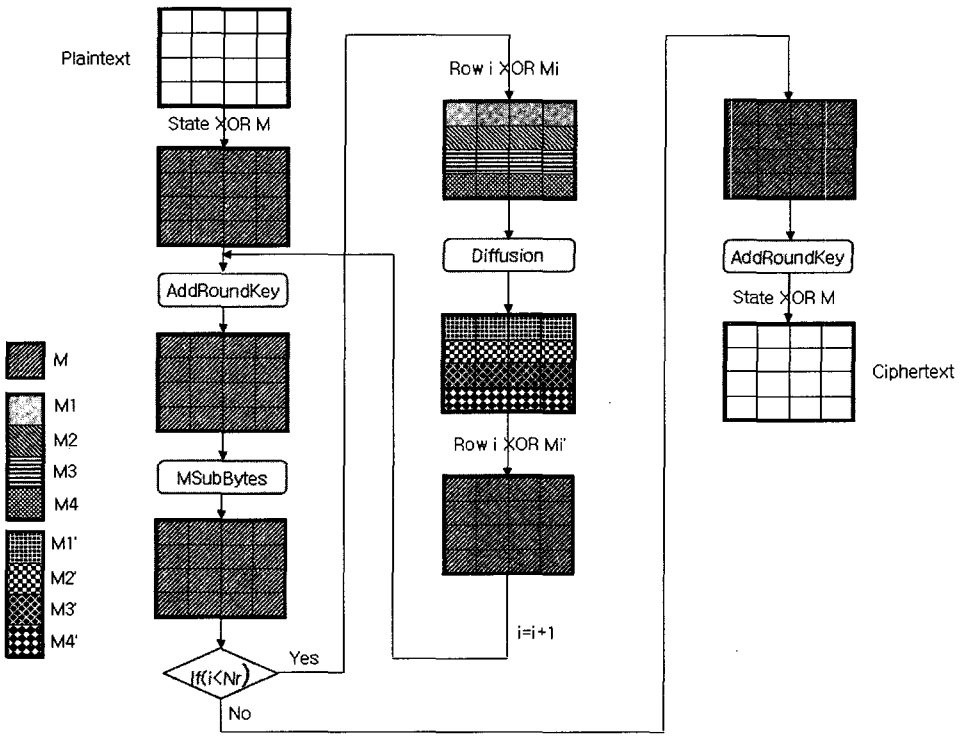


그림 8. ARIA 마스킹 구조

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

그림 9. 마스킹 ARIA Pseudo 코드

2. 1차 DPA 공격 방법

마스킹된 ARIA를 ATmega163 마이크로프로세서를 사용하는 스마트카드의 EEPROM에 주입한 후 상관도를 이용한 1차 DPA 공격을 다음과 같이 수행하였다. 1차 DPA 공격을 위해서는 스마트카드 내부에 구현된 암호알고리즘 및 스마트카드의 하드웨어 구조를 알고 있다고 가정한다.

단계 1 : 스마트카드에서 동일한 키를 사용하여 S개의 서로 다른 입력 데이터를 암호화하는 동안의 전력 소모량을 측정한다. 이 때, 측정된 전력 파형을 $P_{1...S,1...T}$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 논리1을 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. 식 (16)은 두 변수 X, Y간의 상관도를 나타내는 일반적인 수식이다.

$$\rho(X,Y) = \frac{E(XY)-E(X)E(Y)}{\sqrt{Var(X)\,Var(Y)}} = \frac{Cov(X,Y)}{\sqrt{Var(X)\,Var(Y)}} \quad (16)$$

한편, 피어선 상관계수(r)의 정의는 식 (17)과 같으며, r은 두 변수들 사이의 상관도를 S개의 샘플에 기반하여 추정한다.

$$r(x_1,\cdots,x_S,y_1,\cdots,y_S) = \frac{\sum_{s=1}^{S}(x_s-\bar{x})(y_s-\bar{y})}{\sqrt{\sum_{s=1}^{S}(x_s-\bar{x})^2}\sqrt{\sum_{s=1}^{S}(y_s-\bar{y})^2}} \quad (17)$$

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

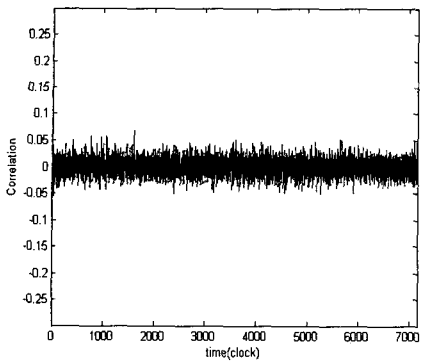


그림 10. 마스킹 SBOX에 대한 올바른 키

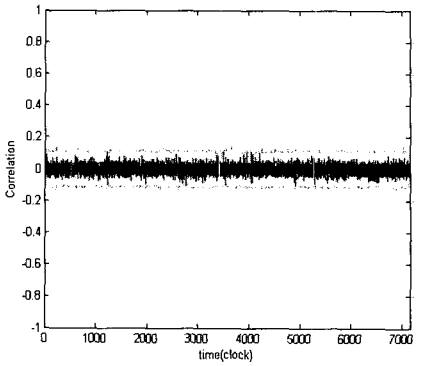


그림 11. 마스킹 SBOX에 대한 모든 키

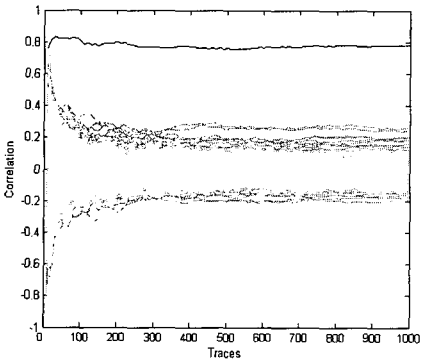


그림 12. 마스킹이 적용되지 않은 ARIA에 대한 DPA 공격에 필요한 전력파형 수

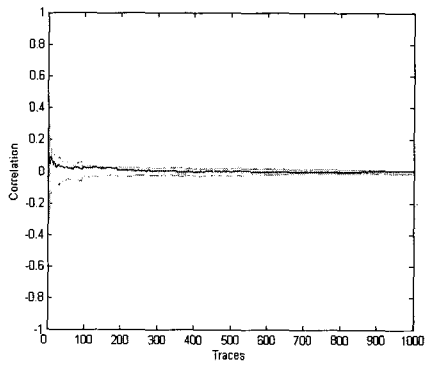


그림 13. 마스킹을 적용한 ARIA에 대한 DPA 공격에 필요한 전력파형 수

계산한 결과 의미 있는 값을 가지는 키를 찾을 수 없었다. 따라서 마스킹이 적용된 ARIA는 1차 DPA 공격에 안전함을 확인할 수 있었다. 그림 10, 11,는 각각 마스킹된 S-BOX에 대해 올바른 키를 추측했을 경우와 그리고 256가지의 모든 가능한 키에 대해 상관계수를 계산한 결과이다.

4. 안전성 및 효율성 분석

제안한 마스킹 방법을 적용한 ARIA에 대한 1차 DPA 공격에 대한 안전성 및 효율성을 분석하였다. 일반적으로 DPA 공격에 대한 안전성은 공격에 필요한 전력파형의 수에 의해 결정되며, 전력파형의 수는 최대 상관계수와 관련이 있다. [9]에서는 최대 상관계수와 공격에 필요한 전력파형의 수의 관계를 다음과 같이 정의하였다. 최대 상관계수가 클수록 필요한 전력파형의 수는 줄어든다.

$$S = 3 + 8\left(\frac{Z_\alpha}{\ln\left(\frac{1+\rho_{max}}{1-\rho_{max}}\right)}\right)^2 \quad (18)$$

Z_α는 상관도 분포에 따라 결정되는 값이며, 실험적으로 결정된다. 본 논문에서는 [9]에서와 같이 $Z_{0.9999}=3.7190$으로 두고 계산하였다. 이론적인 계산결과 $\rho_{max}=0.65$ 일때 공격에 필요한 최소 전력파형의 수는 50개이다. 실험결과 마스킹을 적용하기 전 공격에 성공하기 위한 전력파형의 수는 그림 12에서와 같이 50~70개 정도임을 확인할 수 있다. 반면, 마스킹을 적용한 후에는 전력파형의 수와 상관없이 공격이 이루어지지 않는다. 그림 13에서 위쪽 한 줄은 올바른 키에 대한 상관계수이며, 아래쪽은 틀린 키에 대한 상관계수이다. 이는 식 18의 결과와 일치한다. 따라서 마스킹을 적용한 ARIA는 이론적 및 실험적 결과로 1차 DPA공격에 안전하다.

효율성 측면에서 마스킹을 적용하기 전, 기존의 마스킹 기법, 그리고 제안한 마스킹 기법을 적용한 후의 메모리 크기, 암호 수행에 필요한 클럭 싸이클 수를 비교하였다. 마스킹을 적용하기 전 ARIA 12라운드를 수행하는데에 7,920 클럭 싸이클이 소요되었으며, 기존 마스킹 기법을 적용한 ARIA의 경우 마스킹 테이블을 생성하는 시간과 암호를 수행하는 시간 모두 31,257 클럭 싸이클이 소모되어 성능이 4배 정도 느려지며, 제안한 마스킹 기법을 적용시에는 거의 10배 정도 느려지게 된다. 하지만, 제안한 마스킹 기법은 구현에 필요한 메모리를 1/4로 줄일 수 있으며, 고차 DPA 공격에 대응하기 위하여 각 입력 바이트마다 서로 다른 마스크를 사용할 경우 또는 각 라운드마다 서로 다른 마스크를 사용하는 경우에는 처리 속도도 1.5배 이상 더 빠르다. 이와 같은 결과는 한번의 역원 테이블을 만드는데 많은 시간이 필요하지만 마스킹을 여러 번 수행하는 경우에는 저장된 테이블을 반복적으로 이용하게 되어 오히려 수행시간이 줄어들게 되는 장점을 가지게 되며 마스킹 횟수가 많을수록 연산속도는 크게 개선된다는 것을 의미한다. 따라서, 제안한 기법은 저메모리 환경에서의 구현에 적합하며, challenge-response프로토콜과 같은 작은 양의 데이터의 연산 또는 각 입력 바이트와 모든 라운드에 새로운 마스크를 적용하는 경우에 효율적으로 사용될 수 있다.

표 2. ARIA와 마스킹된 ARIA비교

	ARIA	기존 마스킹 기법ARIA	제안 마스킹 기법 ARIA
RAM	-	1,024btyes	256bytes
ROM(S-BOX)	1,024bytes	1,024bytes	256bytes
마스킹 테이블 생성 및 12라운드 싸이클수	7,920	31,257	78,723
입력바이트마다 서로다른 마스크를 사용하는 경우의 싸이클수	-	346,793	198,915
각 라운드마다 서로 다른 마스크를 사용하는 경우의 싸이클수	-	248,188	161,355
1차 DPA에 대한 취약성	취약	안전	안전

VI. 결 론

본 논문에서는 저메모리 환경에 적합한 마스킹된 ARIA의 구현기법을 설명하고 이를 실제 AVR기반의 ATmega163 마이크로프로세서를 사용하는 스마트카드에 구현하여 1차 DPA 공격에 의해 안전함을 실험적으로 확인하였다. 마스킹은 가장 적은 비용으로 적용할 수 있는 대응방법으로, 소프트웨어로 블록암호를 구현할 경우, 모든 스마트카드에서 1차 DPA 공격에 대한 대응방법으로 손쉽게 적용이 가능한 방법이다. 하지만, ARIA의 경우 4개의 SBOX를 사용하므로 마스킹 적용시 메모리 요구량이 늘어나게 되어 저메모리 환경에서 구현이 어려운 문제가 있다. 본 논문에서 제안한 구현방법은 기존의 구현방법에 비해 메모리 요구량을 75%(1Kbytes→256Bytes) 줄일 수 있어 저메모리 환경에 적합하다. 또한, 입력 바이트마다 서로 다른 마스크를 사용하는 경우 또는 각 라운드마다 새로운 마스크를 사용하는 경우에는 연산시간면에서도 기존의 마스킹 기법에 비하여 효율적이다. 본 논문의 기여는 저메모리 환경 및 여러 개의 마스크를 사용하는 환경에서 ARIA에 대한 마스킹 기법을 적용할 수 있는 기법을 설계한 것이다.

참 고 문 헌

[1] Paul Kocher, Joshua Jaffe, and Benjamin Jun, "Differential Power Analysis," *in proceedings of Advances in Cryptology -CRYPTO '99*, LNCS 1666, pp.388-397, Springer-Verlag, 1999

[2] Daesung Kwon et al., "New Block Cipher ARIA," *in proceedings of ICISC 2002*, LNCS 2971 ,Springer-Verlag, pp.541-548, 2002

[3] JaeCheol Ha, ChangKyun Kim, SangJae Moon, IlHwan Park, and HyungSo Yoo, "Differential Power Analysis on Block Cipher ARIA," *in proceedings of HPCC 2005*, LNCS 3726, pp.541-548, Springer-Verlag, 2005,

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

[5] Mehdi-Laurent Akkar and Christophe Giraud, "An implementation of DES and AES, secure against some attacks," *in proceedings of CHES2001*, LNCS 2162, pp.309-318, Springer-Verlag, 2001

[6] Elena Trichina, Domenico De Seta, and Lucia Germani, "Simplified Adaptive Multiplicative Masking for AES," *in proceedings of CHES2002*, LNCS 2523, pp.187-197, Springer-Verlag, 2003

[7] Johannes Blomer, Jorge Guajardo, and Volker Krummel, "Provably Secure Masking of AES," *in proceedings of SAC2004*, LNCS 3357, pp.69-83, Springer-Verlag, 2005

[8] Jovan D. Golic and Christophe Tymen. "Multiplicative Masking and Power Analysis of AES," *in proceedings of CHES2002*, LNCS 2523, pp.198-212, Springer-Verlag, 2002

[9] Louis Goubin and Jacques Patarin, " DES and Differential Power Analysis - The Duplication Method," *in proceedings of CHES 1999*, LNCS 1717, pp.158-172, Springer-Verlag, 1999

[10] Kris Tiri, Moonmoon Akmal, and Ingrid Verbauwhede, "A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withsatand Differential Power Analysis on Smart Cards," *in proceedings of ESSCIRC2002*, 2002

[11] Kris Tiri and Ingrid Verbauwhede, "Securing Encryption Algorithms against DPA at the Logic Level : Next Generation Smart Card Technology," *in proceedings of CHES2003*, LNCS 2779, pp.125-136, Springer, 2003

[12] Louis Goubin, "A Sound Method for Switching between Boolean and Arithmetic Masking," *in proceedings of CHES 2001*, LNCS 2162, pp.3-15, Springer-Verlag, 2001,

[13] Stefan Mangard, "Hardware Countermeasures against DPA - A Statistical Analysis of Their Effectiveness," *in proceedings of CT-RSA2004*, LNCS 2964, pp.222-235, Springer-Verlag, 2004.

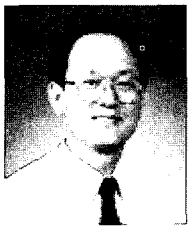
〈著者紹介〉

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~2004년 12월: 나사렛대학교 전자계산소장, 학술정보관장
2005년 1월~2006년 1월: 나사렛대학교 입시학생처장
1998년 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