Journal of The Korea Institute of Information Security & Cryptology(JKIISC) ISSN 1598-3986(Print)
VOL.23, NO.3, June 2013 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2013.23.3.371

블록 암호 ARIA에 대한 오류 주입 공격 대응 방안*

김 형 동,* 하 재 철†
호서대학교

A Countermeasure Against Fault Injection Attack on Block Cipher ARIA*

Hyung-Dong Kim,† Jae-Cheol Ha‡
Hoseo University

요 약

정보보호 장치를 이용하여 데이터의 기밀성을 제공할 경우, 내부에 저장된 비밀 키를 사용하여 암호화를 수행한다. 그러나 최근 공격자가 암호화 연산 시에 오류를 주입하는 방법을 사용하여 내부의 비밀 키를 찾아낼 수 있는 오류 주입 공격 방법들이 제시되었다. 특히, 오류 주입 공격으로 블록 암호 ARIA를 공격할 경우 약 33개 정도의 오류 암호문만 있으면 비밀 키를 공격할 수 있다. 본 논문에서는 ARIA 암호 시스템에서 입·출력 정보들간의 차분 상태를 검사하는 방법으로 오류 주입 공격을 방어하는 오류 주입 탐지 방안을 제안한다. 제안 방법은 오류 탐지 능력이 우수할 뿐 아니라 연산 오버헤드가 적어 매우 효율적임을 시뮬레이션을 통해 확인하였다.

ABSTRACT

An encryption algorithm is executed to supply data confidentiality using a secret key which is embedded in a crypto device. However, the fault injection attack has been developed to extract the secret key by injecting errors during the encryption processes. Especially, an attacker can find the secret key of block cipher ARIA using about 33 faulty outputs. In this paper, we proposed a countermeasure resistant to the these fault injection attacks by checking the difference value between input and output informations. Using computer simulation, we also verified that the proposed countermeasure has excellent fault detection rate and negligible computational overhead.

Keywords: Fault Injection Attack, ARIA, Fault Attack Countermeasure

I. 서 론

정보보호용 디바이스를 이용하여 암호 알고리즘을 수행할 때에는 칩 내부에 저장된 비밀 키를 이용하여 연산을 수행한다. 그러나 최근 암호용 칩이 동작할 때 물리적인 공격 방법을 이용하여 비밀 키를 찾아내는 공격 방법들이 제안되었다. 특히, 1997년 Boneh 등에 의해 처음으로 소개된 오류 주입 공격(fault injection attack)은 암호 알고리즘을 수행하는 과정에서 공격자가 하드웨어 칩에 고의적으로 오류를 주입하여 비밀 키를 찾아내는 공격 기법이다[1]. 또한, Biham과 Shamir는 블록 암호 시스템 대한 오류 주입 공격을 처음 제안하였는데[2] 이 공격은 입력 평문에 대한 정상적인 암호문과 오류가 주입된 오류 암호문을 서로 차분하여 분석한다고 해서 차분 오류 분석(Differential Fault Analysis, DFA) 공격이라고 불린다.

특히, 국제 표준 블록 암호 알고리즘인 AES[3]에 대한 오류 주입 공격 연구가 많이 이루어졌는데, Piret와 Quisquater는 2쌍의 정상 암호문과 바이트

접수일(2013년 3월 26일), 수정일(2013년 5월 3일), 게재확정일(2013년 5월 7일)
* 이 논문은 2012년도 호서대학교의 재원으로 학술연구비 지원을 받아 수행된 연구임.(2012-0263)
† 주저자, karuceace@nate.com
‡ 교신저자, jcha@hoseo.edu(Corresponding author)

단위 오류에 의한 오류 암호문만을 이용하여 비밀 키를 찾는 방법을 제안하였다[4]. 또한, 2008년에는 Kim과 Quisquater에 의해 키 스케쥴링 연산 중 바이트 오류를 주입하여 얻은 총 8개의 정상-오류 암호문 쌍을 이용하여 비밀 키를 찾아내는 방법이 제안되었다[5]. 최근에는 공격자가 1개의 정상-오류 암호문 쌍만 얻을 수 있으면 2^8번의 비밀 키 전수 조사를 통해 비밀 키를 찾아낼 수 있음이 밝혀졌다[6].

국내의 표준 블록 알고리즘인 ARIA[7]에 대한 오류 주입 공격은 2008년 Li 등에 의해 처음 제안되었는데 이들은 평균 45개의 바이트 오류를 주입하여 ARIA의 128비트 비밀 키를 복구하였다[8]. 또한, 2011년에는 4개의 오류 암호문만을 이용하여 $O(2^{32})$의 계산 복잡도로 키를 찾아 내는 방법이 제안되었다[9]. 최근에는 Park 등에 의해 약 33개의 바이트 오류를 주입하여 비밀 키를 찾아내는 오류 주입 공격 방법이 새로이 제안되기도 하였다[10].

AES에 대한 오류 주입 공격을 방어하는 방법들로는 동시 오류 검출(CED) 방법[11], 패리티 비트를 이용한 대응 방법[12, 13], S-Box의 입·출력 연관성을 검사하는 방법[14, 15], CRC를 이용하는 방법[16], 입·출력 차분 특성을 이용한 대응 방법[17] 등이 제안되었다. 그러나 ARIA 알고리즘에 대한 오류 주입 공격 대응 방법 연구에 있어서는 AES에 사용되었던 방법을 유사하게는 응용할 수는 있겠지만 아직 구체적인 방법은 제시된 것은 없었다.

따라서 본 논문에서는 ARIA에 대한 오류 주입 공격 특성을 분석하고 이를 효과적으로 방어할 수 있는 대응 기법을 제안하고자 한다. 제안 방법은 암·복호화 과정이나 키 확장 과정에서 입력과 출력의 차분 연관성을 검사하여 오류 주입 여부를 검출하는 방법으로서 기존의 바이트 단위의 오류 주입 공격을 모두 방어할 수 있음을 시뮬레이션을 통해 검증하였다.

II. ARIA에서의 오류 주입 공격

2.1 ARIA 블록 암호 알고리듬

블록 암호 ARIA는 128비트 입·출력 메시지를 사용하고 128, 192, 256비트로 가변되는 키를 사용하는 블록 암호화 알고리즘이다. ARIA는 암호화 과정과 복호화 과정이 동일한 Involution SPN 구조를 가지며 라운드 키 덧셈(AddRoundKey), 치환 계층(SubstLayer), 확산 계층(Diffusion Layer)으로 구성되어 있다.

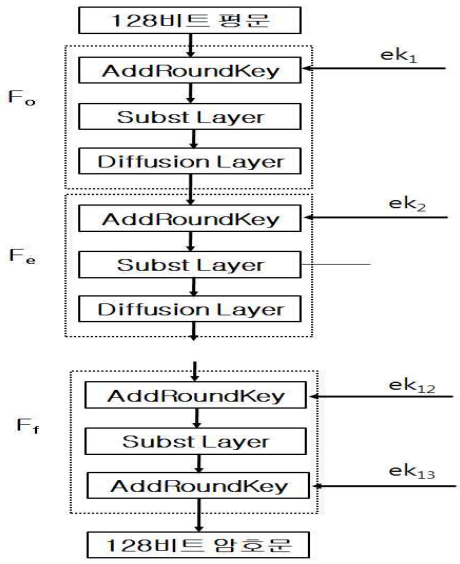
(1) AddRoundKey(ARK) : 128비트 라운드 키를 라운드 입력 128비트와 비트별 XOR

(2) SubstLayer(SL) : 두 유형의 치환 계층이 있으며 각각은 2종의 1바이트 입·출력 S-Box와 그 역변환으로 구성

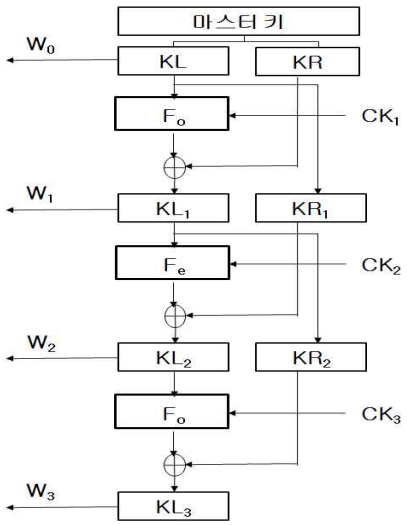
(3) DiffusionLayer(DL) :16×16 Involution 이진 행렬을 사용한 바이트간의 확산 함수로 구성

ARIA 라운드 함수는 비밀 키의 비트 길이에 따라 12, 14, 16 라운드를 반복하여 수행한다. 본 논문에서는 설명의 편의를 위해 128비트 키를 사용하여 12 라운드를 수행하는 ARIA-128을 중심으로 기술하고자 한다. 라운드 함수는 홀수 라운드 함수, 짝수 라운드 함수 그리고 최종 라운드 함수로 구성되며 최종 라운드 함수에는 확산 함수를 사용하지 않고 라운드 키 덧셈 함수를 두 번 사용한다. 32비트 치환 함수는 8비트 입·출력 S-Box들로 구성이 되는데 모두 2개의 S-Box(S_1, S_2)와 그 역치환 S-Box(S_1^{-1}, S_2^{-1})가 사용된다. 치환 함수는 유형 1($S_1 \| S_2 \| S_1^{-1} \| S_2^{-1}$)과 유형 2($S_1^{-1} \| S_2^{-1} \| S_1 \| S_2$)로 구성되어 있다. [그림 1]은 ARIA 알고리즘의 암호화 과정을 구체적으로 나타낸 것이다.

ARIA에서의 키 확장은 초기화 과정과 라운드 키 생성 과정으로 구성되어 있다. 초기화 과정에서는



[그림 1] ARIA 암호화 과정

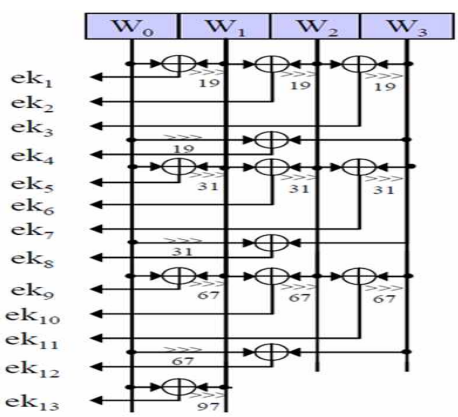


〔그림 2〕 키 확장의 초기화 과정

128비트의 마스터 비밀 키 MK를 입력받아 256비트까지 0으로 패딩한 후 $KL\|KR$로 구성한다. 그 후 한 라운드를 F함수로 하는 3라운드 Feistel 암호를 수행하여 다음과 같이 4개의 128비트 값 W_0, W_1, W_2, W_3를 생성한다. 여기서 CK_i는 π^{-1}의 유리수 부분을 이용하여 생성한 128비트 상수를 나타낸다. ARIA의 키 확장 초기화 과정을 나타낸 것이 〔그림 2〕이다.

$$W_0 = KL, \qquad W_1 = F_o(W_0, CK_1) \oplus KR$$
$$W_2 = F_e(W_1, CK_2) \oplus W_0$$
$$W_3 = F_o(W_2, CK_3) \oplus W_1$$

라운드 키 생성 과정에서는 초기화 과정에서 생성한 W_0, W_1, W_2, W_3를 조합하여 각 라운드 키를 생성하는데 암호 키가 128비트인 경우 12라운드를 수행하게 되고 따라서 128비트로 구성된 13개의 라운드 키를 생성하게 된다. 키 확장의 라운드 키 생성 과정을 나타낸 것이 〔그림 3〕이다. 여기서 $W_i^{\gg j}(0 \le i \le 3,\ 0 \le j \le 127)$는 128비트 W_i를 j비트만큼 오른쪽으로 순환 이동함을 의미한다.



〔그림 3〕 키 확장의 라운드 키 생성

2.2 ARIA에 대한 차분 오류 주입 공격

ARIA에 대한 차분 오류 공격은 2008년에 Li 등에 의해 처음 제안되었다〔8〕. 이 공격에서는 ARIA 암호 알고리즘 연산과정 중 11라운드 Diffusion Layer 연산 전에 바이트 단위의 오류를 주입하여 내부의 비밀 키를 찾아내는 공격 방법이다. 컴퓨터 시뮬레이션 결과 약 45개의 오류 암호문을 얻을 수 있으면 128비트의 비밀 키를 복구할 수 있었다. 또한 2011년에는 암·복호화 과정에서 10라운드의 입력 레지스터에 2개의 바이트를 각각 주입하여 암·복호화 과정의 12라운드의 키 ek_{13}과 ek_1을 복구하였다. 그리고 라운드 키와 키 확산 특성을 이용하여 $O(2^{32})$의 계산 복잡도로 비밀 키를 찾아내었다〔9〕. 최근에는 Park 등에 의해 11라운드에서 8라운드 사이에 있는 Diffusion Layer 연산 전에 바이트 오류를 넣은 오류 암호문을 이용하여 비밀 키를 복구할 수 있음을 보였다〔10〕. 이 공격에서는 약 33개의 오류 암호문을 얻을 수 있으면 충분히 비밀 키를 복구할 수 있음을 실험적으로 검증하였고, 키 확장 과정에서 생성되는 라운드 키에 바이트 오류를 주입해도 동일한 모델로 공격될 수 있음을 보였다.

따라서 현재까지 ARIA에 대한 차분 오류 주입 공격은 모두 바이트 단위의 오류를 이용하여 시도되었으며 암호화 과정 혹은 복호화 과정 그리고 키 확산 과정에서도 오류를 주입하여 공격을 시도할 수 있음을 보이고 있다. 반면 ARIA에 대한 오류 주입 공격을 방어하기 위한 구체적인 대응 알고리즘은 개발된 것이 없다.

III. 제안하는 오류 주입 공격 대응 방안

제안하는 오류 주입 공격 방어 대책의 기본 개념은 연산 과정에서 입력과 출력 사이의 연관성을 검사하여 오류 주입 여부를 검출하는 기법이다. 원래 이에 관한 기본 개념은 논문 〔17〕에서 AES를 대상으로 제안된 바 있으나 본 논문에서는 이를 ARIA에 적용하여 암

호화 및 키 확장 단계까지 확장하여 제안하였다. 먼저 ARIA-128에서 사용하는 함수의 입력과 출력은 128비트이므로 16개의 바이트로 구성되어 있다. 이때 입력(출력) 16개의 바이트를 서로 XOR 연산을 수행한 값을 "입력(출력) XOR 바이트"라 하고, 또한 이 두 값을 다시 XOR한 최종 검사 값을 "함수의 차분 바이트"라 부르기로 한다. 예로서, 128비트 입력 I와 출력 O에 대한 XOR 바이트 I_F, O_F 그리고 함수 F의 차분 바이트 D_F는 다음과 같이 나타낼 수 있다. 여기서 $DB(A)$는 16바이트의 A값을 바이트 단위로 XOR한 것을 의미한다. 즉, $A = a_0 \| a_1 \| \cdots\cdots \| a_{15}$일 때 $DB(A) = \bigoplus_{i=0}^{15} a_i$를 의미한다.

$$I = I_0 \| I_1 \| \cdots\cdots \| I_{15},\quad I_F = DB(I) = \bigoplus_{i=0}^{15} I_i \tag{1}$$

$$O = O_0 \| O_1 \| \cdots\cdots \| O_{15},\quad O_F = DB(O) = \bigoplus_{i=0}^{15} O_i \tag{2}$$

$$D_F = I_F \oplus O_F \tag{3}$$

3.1 암호화 과정에서의 오류 검출

ARIA의 암호화 과정에서 오류를 검출하는 방법은 알고리즘의 입력에 대한 XOR 바이트와 출력 암호문에 대한 XOR 바이트를 구하여 차분 값을 검사하는 것이다. 따라서 암호화 과정 전체에서 발생하는 입력 XOR 바이트와 출력 XOR 바이트 간의 차분 값을 구하기 위해서는 먼저 각 함수의 차분 값을 살펴보아야 한다.

먼저 라운드 내의 함수 중 AddRoundKey 함수의 입력과 출력의 XOR 바이트는 아래와 같다. AddRoundKey 함수는 입력 값의 각 바이트에 라운드 키를 XOR하여 더하는 과정이므로 입력 XOR 바이트 값 I_{ARK}과 출력 XOR 바이트 값 O_{ARK}을 서로 차분한 결과 D_{ARK}는 식 (6)과 같이 라운드 키를 바이트별로 XOR한 것과 같다. 여기서 $ek_{r,i}$는 r라운드 키의 i번째 바이트를 의미한다.

$$I_{ARK} = DB(I) = I_0 \oplus I_1 \oplus \cdots I_{15} \tag{4}$$

$$\begin{aligned} O_{ARK} &= DB(O) = O_0 \oplus O_1 \oplus \cdots \oplus O_{15} \\ &= (I_0 \oplus ek_{r,0}) \oplus (I_1 \oplus ek_{r,1}) \cdots \oplus (I_{15} \oplus ek_{r,15}) \end{aligned} \tag{5}$$

$$\begin{aligned} D_{ARK} &= I_{ARK} \oplus O_{ARK} \\ &= ek_{r,0} \oplus \cdots \oplus ek_{r,15} = \bigoplus_{i=0}^{15} ek_{r,i} \end{aligned} \tag{6}$$

또한, 치환 계층 SubstLayer의 입력과 출력의 차분은 아래와 같다. SubstLayer인 경우 입력 16바이트에 대한 XOR값을 구하여 한 바이트를 구한다. 그리고 출력의 16바이트를 각각 XOR하여 한 바이트를 구한다. 따라서 SubstLayer 함수의 차분 바이트는 아래와 같이 쓸 수 있다.

$$I_{SL} = DB(I) = I_0 \oplus I_1 \oplus \cdots I_{15} \tag{7}$$

$$O_{SL} = DB(O) = O_0 \oplus O_1 \oplus \cdots O_{15} \tag{8}$$

$$D_{SL} = I_{SL} \oplus O_{SL} \tag{9}$$

그런데 SubstLayer 함수의 차분 바이트 D_{SL}는 다음과 같이 구할 수도 있다.

$$\begin{aligned} D_{SL} &= I_{SL} \oplus O_{SL} \\ &= (I_0 \oplus I_1 \oplus \cdots I_{15}) \oplus (O_0 \oplus O_1 \oplus \cdots O_{15}) \\ &= (I_0 \oplus O_0) \oplus (I_1 \oplus O_1) \oplus \cdots (I_{15} \oplus O_{15}) \\ &= DSL_{I_0} \oplus DSL_{I_1} \oplus \cdots DSL_{I_{15}} \end{aligned} \tag{10}$$

그러므로 SubstLayer 함수가 오류 주입 없이 제대로 수행되었는가를 확인하기 위해서는 4개의 S-Box의 입력 I_s와 출력 O_s 값의 차분 값 ($DSL_{I_s} = I_s \oplus O_s,\ 0 \le I_s \le 255$)을 사전에 저장해 두어야 한다. 여기서 DSL_{I_s}는 입력이 I_s인 S-Box의 입력과 출력의 차분 값을 의미한다. 즉, 하나의 S-Box 입력으로 가질 수 있는 값은 0에서 255이므로 〔그림 4〕와 같이 사전에 차분 테이블을 계산해 둔다. 그리고 알고리즘내의 SubstLayer 함수에 오류가 주입되었는지 여부를 확인하기 위해 식 (9)의 값을 구하고 차분 테이블을 이용하여 식 (10)의 값을 구한다. 만약 식 (9)의 값과 식 (10)의 값이 동일하면 오류가 없는 것으로 판단하고 두 차분 바이트 값이 같지 않으면 오류가 주입된 것으로 볼 수 있다.

다음으로 Diffusion Layer에 대한 함수의 차분 바이트를 확인해 볼 필요가 있다. Diffusion Layer의 확산 함수는 16×16 Involution 이진 행렬을 사용한 단순한 바이트 단위 이동에 불과하므로 입력 값과 출력 값에 대한 함수의 차분 바이트 값은 항상 0이 된다. 즉, 입력 16바이트를 XOR하고 출력 16바이트를 XOR한 두 값을 차분해도 값의 변화는 없다.

$$D_{DL} = 0 \tag{11}$$

S_1			S_1^{-1}			S_2			S_2^{-1}		
입력 (I_s)	출력 (O_s)	차분 (DSL_{I_s})	입력 (I_s)	출력 (O_s)	차분 (DSL_{I_s})	입력 (I_s)	출력 (O_s)	차분 (DSL_{I_s})	입력 (I_s)	출력 (O_s)	차분 (DSL_{I_s})
00	63	63	00	52	52	00	e2	e2	00	30	30
01	7C	7D	01	09	08	01	4e	4f	01	68	69
:	:	:	:	:	:	:	:	:	:	:	:
FF	16	E9	FF	7d	82	FF	81	7e	FF	60	9F

〔그림 4〕 S-Box 차분 테이블

이와 같은 사실을 이용하여 한 라운드를 기준으로 "라운드 차분 바이트"를 구하여 보자. 위에서 살펴본 바와 같이 Diffusion Layer의 확산 함수의 차분 바이트는 0이 되고 AddRoundKey 함수의 차분 바이트는 16바이트의 라운드 키를 XOR한 결과가 됨을 알 수 있다. 또한, SubstLayer 함수 의 차분 바이트는 16개의 입력에 따른 4개의 사전 테이블에 있는 차분 값(DSL_{I_s})을 XOR한 결과가 된다. 또한, 따라서 3개의 함수가 있는 한 라운드내의 입력과 출력에 대한 "라운드 차분 바이트"는 아래 식과 같음을 알 수 있다. 여기서 I_{RND}(O_{RND})는 라운드 레벨에서의 입력(출력) 데이터에 대한 바이트 단위의 XOR한 결과이다.

$$\begin{aligned} D_{RND} &= I_{RND} \oplus O_{RND} \\ &= D_{ARK} \oplus D_{SL} \oplus D_{DL} \\ &= (\bigoplus_{i=0}^{15} DSL_{I_i}) \oplus (\bigoplus_{i=0}^{15} ek_{r,i}) \end{aligned} \quad (12)$$

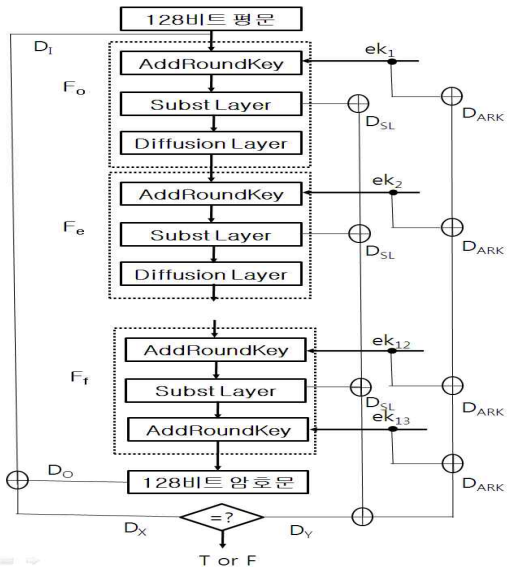
따라서 ARIA의 라운드 레벨에서는 한 라운드 시작하기 전의 입력 XOR 바이트 값(I_{RND})을 구하고 라운드 함수를 수행 후 출력 값에 대한 출력 XOR 바이트(O_{RND})를 구한다. 그리고 SubstLayer의 각 S-Box 입력에 해당하는 16바이트의 차분 값 ($\bigoplus_{i=0}^{15} DSL_{I_i}$)과 16바이트의 라운드 키 값을 XOR한 값($\bigoplus_{i=0}^{15} ek_{r,i}$)을 구한다. 그리고 이 두 값을 XOR 연산으로 차분한 값이 "라운드 차분 바이트" 값이 되고 ($I_{RND} \oplus O_{RND}$) 값과 동일하면 오류가 주입되지 않은 정상적인 라운드를 수행한 것으로 간주한다. 제안하는 오류 주입 검출 방법을 ARIA 암호화 과정의 알고리즘 레벨로 확장시켜 적용한 것이 〔그림 5〕이다.

〔그림 5〕에서 보면 평문(입력)과 암호문(출력)의 차분 바이트 값은 다음과 같이 표현할 수 있다.

$$\begin{aligned} D_X &= D_I \oplus D_O \\ &= (\bigoplus_{i=0}^{15} I_i) \oplus (\bigoplus_{i=0}^{15} O_i) \end{aligned} \quad (13)$$

다른 한편으로 암호화 알고리즘 레벨의 차분 값이 맞는지를 확인하는 값은 아래와 같이 구할 수 있다. 여기서 $i(0 \le i \le 15)$는 중간 암호 값의 각 바이트를 의미하며 $r(1 \le r \le 12)$은 라운드를 의미하며 DSL_{r,I_i}는 r번째 라운드의 S-Box 입력 I_i에 대한 차분 테이블 값을 의미한다.

$$\begin{aligned} D_Y &= \bigoplus_{r=1}^{12} ((\bigoplus_{i=0}^{15} DSL_{r,I_i}) \oplus (\bigoplus_{i=0}^{15} ek_{r,i})) \oplus (\bigoplus_{i=0}^{15} ek_{13,i}) \\ &= \bigoplus_{r=1}^{12} (\bigoplus_{i=0}^{15} DSL_{r,I_i}) \oplus \bigoplus_{r=1}^{13} (\bigoplus_{i=0}^{15} ek_{r,i}) \end{aligned} \quad (14)$$



〔그림 5〕 제안하는 오류 주입 공격 대응책(암호화 알고리즘 레벨)

따라서 ARIA 암호 알고리즘이 정상적으로 수행된다면 알고리즘의 차분 바이트 값인 식 (13)의 값과 식 (14)의 값이 동일할 것이고 오류가 주입된 경우에는 서로 다른 값을 갖게 되어 오류임을 탐지하고 암호문을 출력하지 않는다. 제안하는 ARIA 오류 주입 공격 방어 대책을 수행하는 암호화 과정을 단계별로 기술하면 〔그림 6〕과 같다.

3.2 키 확장 과정에서의 오류 검출

ARIA에서는 128비트 마스터 키 MK가 입력되면 이를 이용하여 12라운드에 사용될 키를 생성하게 된다. 키 확장 단계는 초기화 단계와 라운드 키 생성 단계로 나누어지며 최종적으로 생성되는 각 라운드 키는 128비트로 구성되어 있다. 초기화 과정과 라운드 키 생성 과정은 생성 구조와 사용 연산자가 서로 달라 오류의 주입 여부를 검출하는 방법도 달리 설계하여야 한다.

먼저 초기화 과정에서 오류 주입 여부를 검출하는 방법을 제안한다. 초기화 과정에서는 마스터 비밀 키 MK를 입력받아 256비트까지 0으로 패딩한 후 $KL \| KR$로 구성한다. 그 후 한 라운드를 F함수로 하는 256비트 입·출력을 갖는 3라운드 Feistel 암호를 수행하여 4개의 128비트 값 W_0, W_1, W_2, W_3를 생성한다.

$$\begin{aligned} &W_0 = KL,\ W_1 = KL_1 = F_o(W_0, CK_1) \oplus KR \\ &W_2 = KL_2 = F_e(W_1, CK_2) \oplus W_0 \\ &W_3 = KL_3 = F_o(W_2, CK_3) \oplus W_1 \end{aligned} \tag{15}$$

여기서 각 $W_i(0 \le i \le 3)$간의 차분 바이트 값을 구하면 다음과 같다. 단, 여기서 DSL_r는 Feistel 구조의 r번째 라운드의 S-Box 입력들과 출력들에 대한 차분 바이트를 의미한다.

$$\begin{aligned} &DB(W_1 \oplus KL \oplus KR) \\ &= DB(W_1 \oplus W_0 \oplus KR) = DB(CK_1 \oplus DSL_1) \end{aligned} \tag{16}$$

$$DB(W_2 \oplus W_1) = DB(CK_2 \oplus DSL_2 \oplus W_0) \tag{17}$$

$$DB(W_3 \oplus W_2) = DB(CK_3 \oplus DSL_3 \oplus W_1) \tag{18}$$

위 식이 성립하는 이유는 암호화 단계에서 언급한 바와 같이 확산 함수는 차분 바이트에 영향을 미치지 않지만 S-Box 함수는 두 W_i 값들의 차분에 영향을 미치기 때문이다.

여기서 입력 성분인 KL과 KR의 XOR 바이트와 출력인 W_i 값들과의 XOR 바이트를 차분한 값을 계산해 본다. 즉, 위의 식 (16), (17), (18)을 좌변은 좌변끼리 우변은 우변끼리 XOR 연산을 하면 다음과 같이 쓸 수 있다.

$$\begin{aligned} &DB(KL \oplus KR \oplus W_3) \\ &= DB(\left(\bigoplus_{r=0}^{3} CK_r\right) \oplus \left(\bigoplus_{r=0}^{3} DSL_r\right)) \oplus W_0 \oplus W_1) \end{aligned} \tag{19}$$

이를 다시 정리하여 모든 초기화 확산 출력 값 W_0, W_1, W_2, W_3이 포함된 검증식을 유도하면 아래와 같다. 단, 여기서 W는 확장 결과 값 W_0, W_1, W_2, W_3를 검사하기 위해 128비트 단위로 축약한 변수로서 $W = W_0 \oplus W_1 \oplus W_2 \oplus W_3$이다.

입력 : 평문(I), 라운드 키(ek)
출력 : 정상 암호문(O) 혹은 오류 메시지(Error)

단계 0 : (사전 준비 단계) 각 S-Box 입·출력 값에 대한 차분 값을 미리 계산하여 테이블에 저장한다.

단계 1 : 입력 평문에 대한 XOR 바이트를 구한다. ($D_I = \bigoplus_{i=0}^{15}(I_i)$)

단계 2 : 각 라운드를 수행하면서 SubBytes 함수의 차분 바이트와 각 라운드 키에 대한 XOR 바이트를 구한다.

$$D_Y = \bigoplus_{r=1}^{12}\left(\bigoplus_{i=0}^{15} DSL_{r,I_i}\right) \oplus \bigoplus_{r=1}^{13}\left(\bigoplus_{i=0}^{15} ek_{r,i}\right)$$

단계 3 : 출력 암호문에 대한 XOR 바이트를 구하고 ($D_O = \bigoplus_{i=0}^{15}(O_i)$), D_I 값과 XOR를 하여 차분 바이트 D_X를 구한다. ($D_X = D_I \oplus D_O$)

단계 4 : D_X와 D_Y 값이 동일한지 비교하여 동일하면 암호문을 출력한다.
그렇지 않으면 오류 메시지를 출력하고 종료한다.

〔그림 6〕 오류 주입 공격에 대응하는 ARIA 암호화 알고리즘

$$DB(KL \oplus KR \oplus W) = DB(\left(\overset{3}{\underset{r=0}{\oplus}} CK_r\right) \oplus \left(\overset{3}{\underset{r=0}{\oplus}} DSL_r\right)) \oplus W_2) \quad (20)$$

결국, 입력인 KL과 KR 그리고 초기화 과정의 최종 출력인 W_i의 차분 바이트는 상수 CK의 XOR 바이트와 각 치환계층에서 사용되는 S-Box 입력과 출력의 차분 바이트를 모두 구하여 식 (20)이 성립하는지 확인함으로써 오류 검출 여부를 확인할 수 있다.

- 입력 XOR 바이트 : $D_I = DB(KL \oplus KR)$
- 출력 XOR 바이트 : $D_O = DB(W)$
- 초기화 과정의 차분 바이트 : $D_X = D_I \oplus D_O$
- 오류 검사식 :

$$D_Y = DB(\left(\overset{3}{\underset{r=0}{\oplus}} CK_r\right) \oplus \left(\overset{3}{\underset{r=0}{\oplus}} DSL_r\right)) \oplus W_2)$$

다음으로 라운드 키 생성 과정에서 오류 주입 여부를 검출하는 방법을 고려해 본다. ARIA의 라운드 키 생성 과정은 초기화 과정에서 생성된 W_0, W_1, W_2, W_3로부터 식 (20)과 같이 모두 13개의 라운드 키를 생성하게 된다.

$$ek_1 = (W_0) \oplus (W_1^{\gg 19}), ek_2 = (W_1) \oplus (W_2^{\gg 19}),$$
$$ek_3 = (W_2) \oplus (W_3^{\gg 19}), ek_4 = (W_3) \oplus (W_0^{\gg 19}),$$
$$ek_5 = (W_0) \oplus (W_1^{\gg 31}), ek_6 = (W_1) \oplus (W_2^{\gg 31}),$$
$$ek_7 = (W_2) \oplus (W_3^{\gg 31}), ek_8 = (W_3) \oplus (W_0^{\gg 31}),$$
$$ek_9 = (W_0) \oplus (W_1^{\gg 67}) ek_{10} = (W_1) \oplus (W_2^{\gg 67}),$$
$$ek_{11} = (W_2) \oplus (W_3^{\gg 67}) ek_{12} = (W_3) \oplus (W_0^{\gg 67}),$$
$$ek_{13} = (W_0) \oplus (W_1^{\gg 97}) \quad (21)$$

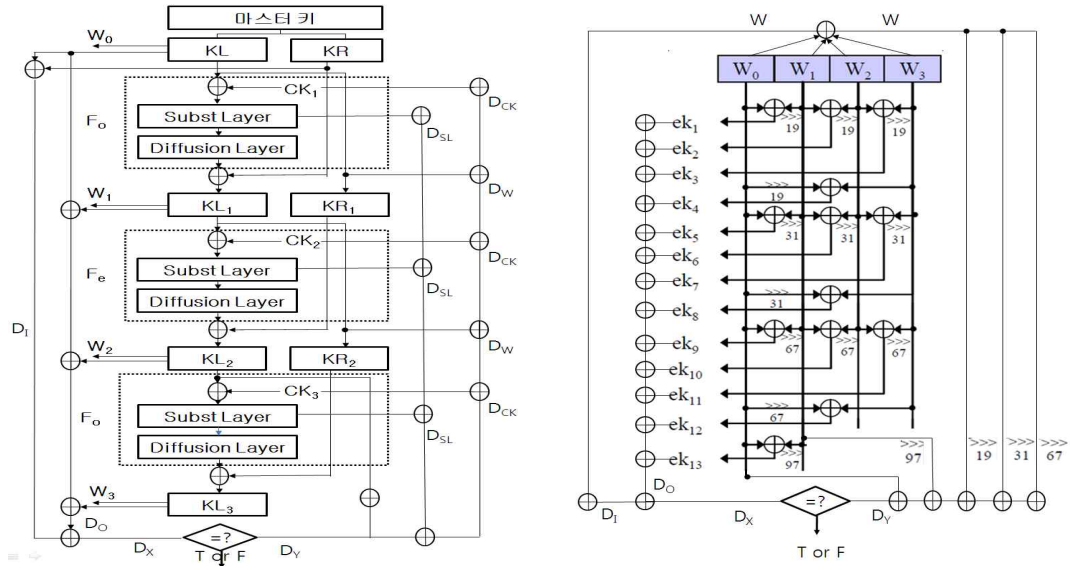
따라서 위 식의 모든 라운드 키를 XOR하면 다음과 같이 나타낼 수 있다.

$$\overset{13}{\underset{i=0}{\bigoplus}} ek_i = W \oplus W^{\gg 19} \oplus W^{\gg 31} \oplus W^{\gg 67} \oplus W_1^{\gg 97} \oplus W_0 \quad (22)$$

그러므로 라운드 키 생성 과정의 오류 주입 여부는 다음과 같이 입력인 $W_i(0 \le i \le 3)$와 출력인 $ek_i(0 \le i \le 13)$과의 차분 바이트를 구하여 비교함으로써 검출할 수 있다.

$$DB(\overset{13}{\underset{i=0}{\bigoplus}} ek_i) = DB(W) \oplus DB(W^{\gg 19} \oplus W^{\gg 31} \oplus W^{\gg 67} \oplus W_1^{\gg 97} \oplus W_0) \quad (23)$$

최종적으로 라운드 함수에서 오류 주입 여부를 알아보기 위해서는 입력 XOR 바이트와 출력 XOR 바이트 값을 차분한 것이 W를 오른쪽으로 각각 19번, 31번, 67번씩, W_1을 97번 오른쪽 순환 이동한 값 그리고 W_0를 XOR한 값을 XOR한 것과 같으므로 다음 수식으로 간소화할 수 있다.



〔그림 7〕 키 확장 과정에서의 오류 주입 공격 대응 방안

- 입력 XOR 바이트 : $D_I = DB(W)$
- 출력 XOR 바이트 : $D_O = DB(\bigoplus_{i=0}^{13} ek_i)$
- 라운드 키 생성과정 차분 바이트 : $D_X = D_I \oplus D_O$
- 오류 검사식 :

$$D_Y = DB(W^{\ggg 19} \oplus W^{\ggg 31} \oplus W^{\ggg 67} \oplus W_1^{\ggg 97} \oplus W_0)$$

따라서 D_X와 D_Y 값이 동일하면 오류가 주입되지 않은 경우이며 두 값이 같지 않으면 오류가 주입된 것으로 판단하여 출력을 멈추거나 암호화 연산과 관련이 없는 쓰레기 값을 출력하면 된다. 〔그림 7〕은 키 확장 단계에서 초기화 과정과 라운드 키 생성과정 중 오류 주입시 이를 검출하는 메커니즘을 도식화 한 것이며 〔그림 8〕은 키 확장 과정의 오류 주입 공격 방어 알고리즘을 순차적으로 기술한 것이다.

IV. 시뮬레이션 및 구현시 고려 사항

4.1 오류 주입 공격 방어 알고리즘 시뮬레이션

본 논문에서는 오류 주입 공격에 대응할 수 있는 ARIA 알고리즘을 제안하였는데 오류가 정확히 검출되는지 컴퓨터로 구현하여 시뮬레이션 하였다. 현재까지 ARIA에 대한 오류 공격 대응 방안은 제시된 바가 없어 상대적 비교는 불가능하였다.

시뮬레이션은 키 확장 과정 중에서 초기화 과정과 라운드 키 생성 과정으로 나누어 시뮬레이션 하였고 라운드 키가 정상적으로 생성된 상태에서 암호화 알고리즘에 대해 검증을 실시하였다. 성능 평가를 위한 환경은 다음과 같다.

- CPU : Intel duo CPU 3GHz
- RAM : 3GB
- 테스트 환경 : Windows XP
- 개발 환경 : Visual Studio 6.0

그리고 현재까지 제시된 ARIA에 대한 오류 주입 공격에 대해 제안 방법이 오류를 검출할 수 있는지 확인해 보았다. 〔표 1〕은 공격이 성공할 수 있는 위치에서 발생할 수 있는 경우의 수를 계산하고 오류 검출 정도를 시뮬레이션 한 결과를 나타낸 것이다. 표에서 보는 바와 같이 기존의 공격들은 10라운드나 11라운드의 확산 계층의 이전 바이트에 오류 주입에 의해 시도되었으며 각 라운드에서 발생할 수 있는 오류의 종

〔표 1〕 오류 주입 공격에 대한 검출 성능

공격 방법	오류 주입 위치	경우의 수	검출율 (%)
W. Li 등 〔8〕	11라운드 확산 계층 이전 바이트	16*255 =4080	100
S. Park 등 〔9〕	10라운드 치환 계층 이전 바이트	16*255 =4080	100
J. Park 등 〔10〕	11라운드 확산 계층 이전 바이트	16*255 =4080	100

입력 : 마스터 키(MK)
출력 : 라운드 키(ek) 혹은 오류 메시지(Error)

단계 0 : (사전 준비 단계) 각 S-Box 입·출력 값에 대한 차분 값을 미리 계산하여 테이블에 저장한다.
단계 1 : 마스터 키에 대한 XOR 바이트를 구한다. ($D_I = DB(KL \oplus KR)$)
단계 2 : 초기화 과정의 각 라운드를 수행하면서 W_2가 생성되면 바로 XOR 바이트를 구하고 SubBytes 함수의 차분 바이트와 상수 값에 대한 XOR 바이트를 각각 구하여 다음 검사 값을 만든다.

$$D_Y = DB((\overset{3}{\underset{r=0}{\oplus}} CK_r) \oplus (\overset{3}{\underset{r=0}{\oplus}} DSL_r)) \oplus W_2)$$

단계 3 : W_0, W_1, W_2, W_3 값을 각각 XOR하여 W를 구하고 XOR 바이트를 계산한다. ($D_O = DB(W)$)
단계 4 : D_I와 D_O를 차분한 값이 D_Y 값과 동일한지 비교하여 동일하면 키 라운드 생성 단계로 넘어간다. 그렇지 않으면 오류 메시지를 출력하고 종료한다.
단계 5 : 단계 3에서 계산된 $DB(W)$를 D_I로 둔다
단계 6 : 각 라운드 키를 생성한 후 W 및 W_1에 대한 오른쪽 순환 이동을 수행한 결과의 XOR 바이트를 구한다. 그리고 W_0에 XOR 바이트를 구하여 아래의 검사 값을 구한다.

$$D_Y = DB(W^{\ggg 19} \oplus W^{\ggg 31} \oplus W^{\ggg 67} \oplus W_1^{\ggg 97} \oplus W_0)$$

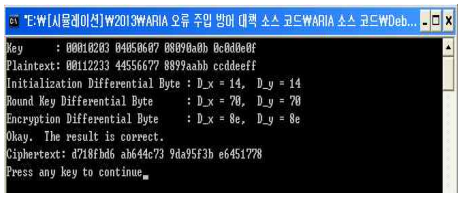
단계 7 : 모든 라운드 키에 대한 XOR 바이트를 구한다. ($D_O = DB(\bigoplus_{i=0}^{13} ek_i)$)
단계 8 : D_I와 D_O를 차분한 값이 D_Y 값과 동일한지 비교하여 동일하면 암호화 단계로 넘어간다. 그렇지 않으면 오류 메시지를 출력하고 종료한다.

〔그림 8〕 오류 주입 공격에 대응하는 키 확장 알고리즘

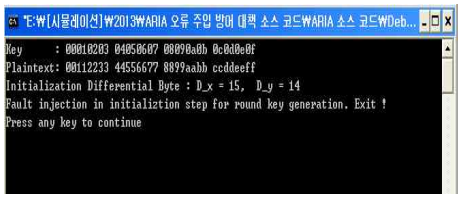
류는 16바이트에 대해 각각 255번의 오류를 가정할 수 있다. 지금까지의 오류 주입 공격이 성공한 유형에 대해서 시뮬레이션 결과 제안 방법은 오류를 100% 검출할 수 있음을 확인하였다.

〔그림 9〕는 정상적으로 알고리즘이 수행되었을 경우와 바이트 오류가 주입되었을 경우를 가정하여 시뮬레이션 한 화면의 예를 보인 것이다.

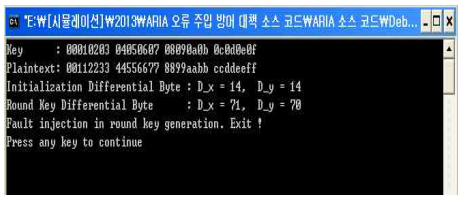
그림의 (a)화면은 오류의 주입이 없을 경우 암호화 과정이 정상적으로 이루어지는 것을 나타낸 것이며, 그림의 (b)화면은 한 바이트의 오류를 키 초기화 시점에 주입하였을 경우 오류 발생을 감지하여 암호문을 출력하지 않는 화면을 나타낸 것이다. 그리고 (c)와 (d)는 라운드 키 생성 시 그리고 암호화 과정에서 바이트 오류를 주입했을 때의 화면이다.



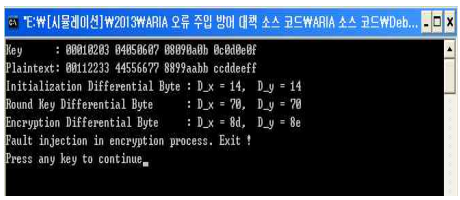
(a) 정상 암호문



(b) 초기화 과정 오류 주입



(c) 라운드 키 생성과정 오류 주입



(d) 암호화 과정 오류 주입

〔그림 9〕 오류 주입 공격 시뮬레이션 화면

4.2 알고리즘의 실제 구현 시 고려사항

제안 방식을 구현할 경우의 오버헤드를 살펴보면, 먼저 4개의 S-Box의 차분 테이블 값을 저장할 수 있는 1024바이트 정도의 영구적 혹은 일시적 메모리 공간이 추가적으로 요구된다. 여기서 고려할 사항은 사전에 계산된 S-box 차분 테이블을 영구적인 메모리에 저장하는 경우는 문제가 되지 않지만 암호를 시작하는 단계에서 차분 값을 구하여 저장해서 사용하는 일시적 메모리인 경우에는 차분 바이트 자체가 오류 없이 생성되었는지 확인할 필요가 있다.

만약, S-Box의 차분 바이트를 사전에 계산 시 오류가 발생하면 암호문 중간에 오류가 발생한 것과 동일한 효과를 낼 수 있으므로 이를 확인하는 것이 좋다. 이 경우 아래와 같은 간단한 계산을 통해 각 S-Box의 차분 바이트가 제대로 생성되었는지 검증할 수 있다. 각 S-Box의 입·출력은 8비트로 표현되는 256가지의 모든 값을 가지므로 그 차분 값 역시 256종류를 모두 가지므로 이들을 차분해 보면 0이 되는 성질을 이용하면 정확한 DSL_{I_s} 값이 생성되었는지 확인할 수 있다.

$$\bigoplus_{s=0}^{255} I_s = 0, \quad \bigoplus_{s=0}^{255} O_s = 0, \quad \bigoplus_{s=0}^{255} DSL_{I_s} = 0 \qquad (24)$$

또한 오류 주입 공격에 대응하는 ARIA 알고리즘을 구현할 경우 추가되는 연산량을 계산해 보면 대부분의 연산이 입·출력 데이터에 대한 XOR 연산이므로 매우 효율적임을 알 수 있다. 따라서 각 S-Box 입력 값에 대한 차분 값을 읽어 XOR하는 시간과 입·출력 정보에 대한 차분 연산 시간 등이 추가적으로 필요하다. 참고로 상수 CK와 관련한 차분 바이트는 미리 계산해 고정적으로 저장해 둘 수 있어 계산 시간상의 오버헤드는 알고리즘 전체에 비해 아주 작게 된다.

〔표 2〕는 제안하는 오류 주입 대응 방법을 구현함에 있어 추가되는 비용을 메모리와 수행 시간 면에서 요약한 것이다. 표에서 보는 바와 같이 ARIA를 원형으로 구현할 경우보다 제안 방식에서는 2배 정도의 메모리 공간이 필요하였다. 수행 시간을 소프트웨어로 측정해 본 결과 일반 ARIA는 약 7.2ms 정도가 소요되었다. 이에 암호화 부분만 대응 기법을 적용하면 약 8.6ms, 키 확장 부분까지 대응 기법을 적용하면 약 9.9ms가 소요되었다. 이 결과는 오류 검출을 위해 동일한 연산을 반복수행하는 기법에 비해서는 매우 효과

〔표 2〕 오류 주입 공격 대응 알고리듬의 추가 비용

구현 방법	원형 ARIA	제안하는 대응 기법	추가 비율(%)
S-Box 테이블용 ROM/RAM	1024 바이트	2048 바이트	200
수행 시간	7.2ms	8.6ms(암호) 9.9ms(전체)	119 138

적임을 알 수 있다. 그러나 이 결과는 단지 소프트웨어로 구현하여 시뮬레이션한 결과이며 하드웨어 장치로 구현할 경우에는 구현 기법이나 개발 환경에 따라 다소 차이가 있을 수 있다.

두 번째로 고려할 사항은 본 논문에서 제안하는 대응 알고리즘은 오류를 검사하여 오류가 있으면 키 확장이나 암호화 과정을 중단하는 형식으로 구성을 하였다. 그러나 공격 능력이 우수한 공격자는 2번의 오류를 주입하여 실제 연산 과정에 오류를 주입할 뿐 아니라 두 값을 비교하는 오류 검사 과정을 건너 띄게 하는 2차 공격도 가능하다.

따라서 두 차분 결과 값 D_X와 D_Y를 비교하여 프로그램 종료 여부를 결정하는 것보다 오류가 발생했을 때 쓰레기 값을 출력하는 방법도 고려해 볼 수 있다. 예를 들어 공격자에 의해 오류가 주입되어 두 값을 차분한 결과가 0이 되지 않을 경우 다음 식과 같이 암호 연산과 관련 없는 값을 출력시킬 수도 있다. 여기서 120비트의 랜덤 수를 곱하는 이유는 $(D_X \oplus D_Y)$가 한 바이트인 점을 고려하여 암호문의 크기 128비트와 맞추도록 한 것이다. 식에서 보는 바와 같이 만약 오류가 주입되지 않은 경우에는 $(D_X \oplus D_Y)$는 0이 되고 정상 암호문이 출력될 것이다.

$$((D_X \oplus D_Y) * \text{120비트 랜덤 수}) \oplus \text{오류 암호문} \quad (25)$$

결국, D_X와 D_Y 두 값의 비교를 위해 "if"문과 같은 조건문을 사용하는 검사 방법은 2번의 오류를 주입하는 2차 공격에 취약하지만 위 식과 같이 오류 값 확산 기법을 사용하면 2차 공격이 불가능하므로 물리적 안전성 측면에서 더 유용하다고 할 수 있다.

V. 결 론

본 논문에서는 ARIA에 대한 오류 주입 공격 기술을 분석하고 주입된 오류를 검출하기 위한 대응 방안을 제시하였다. 논문에서 제안하는 방식은 알고리즘의 암호화 과정, 키 확장 시 초기화 과정 그리고 라운드 키 생성 과정에서 입력되는 값과 최종적으로 출력되는 값의 차분 값을 계산한 후 이 값이 사전 계산 테이블 값과 계산 과정의 변수 값들의 조합으로 생성된 차분 값과 비교하는 것이다. 4개의 S-Box에 대한 입·출력 차분 값만 메모리에 사전 저장한 후 이를 활용하면 키 확장 단계 뿐만 아니라 암호화 알고리즘 전체의 오류를 탐지할 수 있었다. 즉, 알고리즘의 입·출력에 대한 차분 바이트 값이 오류 주입 과정 없이 잘 유지되고 있는지를 순차적으로 계산함으로써 바이트나 비트 단위의 오류는 모두 검출해 낼 수 있었다.

참고문헌

〔1〕 D. Boneh, R. DeMillo, and R. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," EUROCRYPTO'97, LNCS 1233, pp. 37-51, 1997.

〔2〕 E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," CRYPTO'97, LNCS 1294, pp. 513-525, 1997.

〔3〕 National Institute of Standards and Technology, "Advanced Encryption Standards," NIST FIPS PUB 197, 2001.

〔4〕 G. Piret and J. Quisquater, "A differential fault attack technique against SPN structures, with application to the AES and KHAZAD," CHES'03, LNCS 2779, pp. 77 - 88, 2003.

〔5〕 C. Kim and J. Quisquater, "New Differential Fault Analysis on AES Key Schedule: Two Faults are enough," CARDIS'08, LNCS 5189, pp. 48-60, 2008.

〔6〕 M. Tunstall and D. Mukhopadhyay, "Differential fault analysis of the advanced encryption standard using a single fault," Cryptology ePrint Archive, Report 2009/575, 2009.

〔7〕 D. S, Kwon, J. S. Kim, S. W. Park, S. H. Sung, Y. K. Sohn, J. H. Song, Y. J. Yeom, E. J. Yoon, S. J. Lee, J. W. Lee, S. T. Chee, A. W. Han, and J. Hong, "New

block cipher ARIA," ICISC'03, LNCS 2971, pp. 432-445, 2003.
〔8〕 W. Li, D. Gu and J. Li, "Differential fault analysis on the ARIA algorithm," Information science, vol. 178, no. 19, pp. 3727-3737, Oct. 2008.
〔9〕 박세현, 정기태, 이유섭, 성재철, 홍석희, " 블록 암호 ARIA-128에 대한 차분 오류 공격," 한국정보보호학회 논문지, 21(5), pp. 15-25, 2011년 10월.
〔10〕 J. H. Park and J. C. Ha, "Improved Differential Fault Analysis on Block Cipher ARIA," WISA'12, LNCS 7690, pp. 82-95, 2012.
〔11〕 R. Karri, K. Wu, P. Mishra, and Y. Kim, "Concurrent error detection of fault-based side-channel cryptanalysis of 128-bit symmetric block ciphers," IEEE Design Automation Conference (DAC'01), pp. 579-584, 2001.
〔12〕 G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard," IEEE Transactions on Computers, vol. 52, no. 4, pp. 492-505, 2003
〔13〕 K. Wu, R. Karri, G. Kuznetsov, and M. Goessel, "Parity based concurrent error detection for the advanced encryption standard," IEEE International Test Conference (ITC'04), pp. 1242-1248, 2004.
〔14〕 M. M. Kermani and A. R. Masoleh, "A Structure-independent Approach for Fault Detection Hardware Implementations of the Advanced Encryption Standard," FDTC'07, IEEE-CS, pp. 47-53, 2007.
〔15〕 G. Di Natale, M. L. Flottes, and B Rouzeyre, "An On-Line Fault Detection Scheme for SBoxes in Secure Circuits," IEEE International On-Line Testing Symposium, pp. 57-62, 2007.
〔16〕 C. H. Yen and B. F. Wu, "Simple Error Detection Methods of Hardware Implementation of Advanced Encryption Standard," IEEE Trans. on Computers, vol. 55, no. 6, pp. 720-731, 2006.
〔17〕 박정수, 최용제, 최두호, 하재철, "입·출력 차분 특성을 이용한 오류 주입 공격에 강인한 AES 구현 방안," 한국정보보호학회 논문지, 22(5), pp. 1009-1017, 2012년 10월.

〈저자소개〉

김 형 동 (Hyung-Dong Kim) 학생회원
2012년 2월: 호서대학교 정보보호학과 졸업
2012년 3월~현재: 호서대학교 대학원 정보보호학과 석사 과정
〈관심분야〉 부채널 공격, 무선 네트워크 보안, 스마트폰 보안

하 재 철 (Jae-Cheol Ha) 종신회원
1989년 2월: 경북대학교 전자공학과 졸업
1993년 2월: 경북대학교 전자공학과 석사
1998년 2월: 경북대학교 전자공학과 박사
1998년 3월~2007년 2월: 나사렛대학교 정보통신학과 부교수
2007년 3월~현재: 호서대학교 정보보호학과 교수
〈관심분야〉 정보보호, 네트워크 보안, 부채널 공격