情報保護學會論文誌
第 22 卷 第 5 號, 2012. 10

입·출력 차분 특성을 이용한 오류 주입 공격에 강인한 AES 구현 방안*

박 정 수,[1†] 최 용 제,[2] 최 두 호,[2] 하 재 철[1‡]
[1]호서대학교, [2]한국전자통신연구원

A Secure AES Implementation Method Resistant to Fault Injection Attack Using Differential Property Between Input and Output*

Jeong-Soo Park,[1†] Yong-Je Choi,[2] Doo-Ho Choi,[2] Jae-Cheol Ha[1‡]
[1]Hoseo University, [2]ETRI

요 약

비밀 키가 내장된 암호 장치에 대한 오류 주입 공격은 공격자가 암호화 연산 시 오류를 주입하여 암호 시스템의 키를 찾아내는 공격이다. 이 공격은 AES와 같은 암호 시스템에서 한 바이트의 오류 주입으로도 비밀 키 전체를 찾아낼 수 있을 정도로 매우 위협적이다. 본 논문에서는 AES 암호 시스템에서 입·출력값의 차분을 검사하는 방법으로 오류 주입 공격을 방어하는 새로운 오류 검출 기법을 제안한다. 또한, 제안 방법이 기존의 공격 대응 방법들과 비교하여 오류 탐지 능력이 우수하고 구현에 필요한 추가적인 오버헤드가 적어 효율적임을 컴퓨터 시뮬레이션을 통해 확인하였다.

ABSTRACT

The fault injection attack has been developed to extract the secret key which is embedded in a crypto module by injecting errors during the encryption process. Especially, an attacker can find master key of AES using injection of just one byte. In this paper, we proposed a countermeasure resistant to the these fault attacks by checking the differences between input and output. Using computer simulation, we also verified that the proposed AES implementation resistant to fault attack shows better fault detection ratio than previous other methods and has small computational overheads.

Keywords: AES, Differential Fault Analysis Attack, Fault Attack Countermeasure

I. 서 론

비밀 키가 내장된 하드웨어 칩을 이용하여 정보보호를 위한 암호 알고리듬을 수행할 때에는 여러 가지 물리적인 공격에 주의해야 한다. 특히, 암호 칩의 구현상 취약점을 이용하는 오류 주입 공격은 1997년 Boneh 등에 의해 처음으로 소개되었으며 암호 연산을 수행하는 과정에서 하드웨어 칩에 오류를 주입하여 비밀 키를 찾아내는 공격이다[1]. 그 후 Biham과 Shamir는 블록 암호 시스템 대해서 오류 공격이 가능함을 제안하였고[2] 이것을 정상 암호문과 오류가 주입된 오류 암호문을 차분하여 분석한다고 해서 차분 오류 분석(Differential Fault Analysis, DFA) 공격이라고 불렀다.

특히, 국제 표준 블록 암호 알고리듬인 AES[3]에

접수일(2012년 8월 17일), 수정일(2012년 9월 28일), 게재확정일(2012년 9월 28일)
* 본 연구는 방송통신위원회 및 한국방송통신전파진흥원의 방송통신기술개발사업의 일환인 SCARF 프로젝트로 수행하였음. [부채널 공격 방지 원천기술 및 안전성 검증기술 개발]
† 주저자, sizeplay@nate.com
‡ 교신저자, jcha@hoseo.edu

대하여 DFA 공격 연구도 많이 이루어졌는데, Piret와 Quisquater는 2쌍의 정상-오류 암호문 쌍만을 이용하여 비밀 키를 찾는 방법을 제안하였고[4], Giraud는 한 비트 오류 또는 키 스케줄링에서 한 바이트 오류를 주입하여 전체 키를 찾아내는 방법을 제안하였다[5]. 2008년, Kim과 Quisquater가 키 스케줄링 상에 오류를 주입함으로써 모두 8개의 정상-오류 암호문 쌍을 이용하여 키를 찾아내는 방법을 제안하였다[6]. 최근에는 단지 1개의 정상-오류 암호문 쌍만 있으면 2^8번의 비밀 키 전수 조사(exhaustive search)과정을 거쳐 비밀 키를 찾아낼 수 있음이 밝혀졌다[7].

AES에 대한 여러 가지 오류 주입 공격 방법이 제안되면서 공격을 방어할 수 있는 대응 기법들도 제안되었다. 먼저, 암호화된 데이터를 다시 복호화하여 비교하는 동시 오류 검출(CED) 방법[8]이 제안된 바 있다. 이후에도 패리티 비트를 이용한 대응 방법[9, 10], S-Box의 입·출력 연관성을 검사하는 방법[11, 12], CRC를 이용하는 방법[13] 등이 제안되었다. 그러나 기존의 오류 주입 공격 방어 대책들은 방어 기법을 수행하는데 많은 시간이 추가되어야 할 뿐만 아니라 오류 탐지율도 만족스럽지 못한 경우가 많았다.

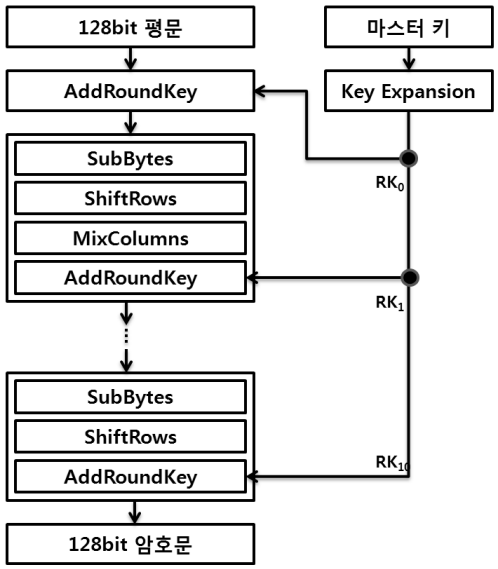
따라서 본 논문에서는 AES에서 오류 주입 공격을 효과적으로 방어할 수 있는 대응 기법을 제안하고자 한다. 제안 방법은 라운드 레벨이나 알고리듬 레벨에서 입력 값과 출력 값의 차분을 검사하여 오류 주입 여부를 검사하는 방법이다. 제안 방법에서는 S-Box에 대한 입·출력 차분 테이블을 사전에 작성하여 암호 알고리듬이 수행될 때의 오류 주입 여부를 판별하는 데 활용한다. 제안 방법은 기존의 오류 주입 공격들 특히, 바이트 단위의 오류 주입 공격은 모두 방어할 수 있음을 시뮬레이션을 통해 검증하였다.

II. AES에서의 오류 주입 공격 대응책

2.1 AES 알고리듬

AES는 국제 표준 대칭 키 암호 알고리즘으로서 128, 192, 256비트의 다양한 키 길이를 사용하며 평문 메시지를 128비트 단위 블록으로 나누어 암호화를 수행한다. 암호화 과정은 [그림 1]과 같이 SubBytes, ShiftRows, MixColumns, AddRoundKey 함수로 이루어져 있으며, 키의 길이에 따라 10, 12, 14라운드를 수행한 후 최종 암호문을 출력한다. 각 라운드가 수행되는 동안 128비트의 입력 메시지는 4×4 형태의 바이트 단위로 처리되는데 각 함수를 수행한 후의 중간 상태를 스테이트(state)라고 한다. 각 스테이트는 아래의 4가지 함수가 수행된 이후의 16개 바이트 값을 가지게 된다.

(1) SubBytes(SB): 스테이트 값을 각 바이트 단위로 S-Box를 이용하여 치환하는 연산
(2) ShiftRows(SR): 스테이트 값을 행(row) 단위로 왼쪽 순환 시프트하는 연산
(3) MixColumns(MX): 스테이트 값을 열(column) 단위로 행렬과 곱하는 연산
(4) AddRoundKey(AK): 각 라운드의 키와 스테이트 값을 XOR하는 연산



[그림 1] AES 암호화 과정

2.2 AES 차분 오류 주입 공격

AES와 같은 블록 암호 시스템에 대한 오류 주입 공격은 크게 두 가지로 분류되는데 하나는 키 스케줄링 과정에서 오류를 주입하는 형태이고, 다른 하나는 라운드 과정을 수행하는 동안 오류를 주입하는 방법이다. 오류 주입 공격에 대한 개념이 제안된 이후 AES에 대한 현실성 있는 오류 주입 공격은 2003년 Piret과 Quisquater에 의해 제안되었는데 최소 2개의 정상-오류 암호문 쌍을 얻을 수 있으면 128비트 비밀 키 추출이 가능하다[4]. 이 공격에서는 8라운드 시작

전에 바이트 오류를 주입한 후 그 오류가 9, 10라운드로 확산하는 성질에 기반을 두어 출력된 오류 암호문을 분석하였다.

또한, Giraud는 알고리듬 연산 시 한 비트 오류가 발생하거나, 키 스케줄링 과정의 한 바이트 오류가 발생할 경우를 가정하여 비밀 키를 찾아내는 방법을 제시하였다[5]. 이 공격에 의하면, AES의 라운드 처리 과정에서 한 비트 오류를 주입한 50개의 오류 암호문만 있으면 비밀 키를 추출할 수 있었다. 그러나 이 공격에서의 오류 주입 가정은 매우 고난도의 오류 주입 기술을 전제로 하므로 현실적인 공격 방법으로 보기에는 무리가 있다. 이 논문에서 사용한 다른 오류 주입 모델로 바이트 단위의 오류 모델이 있는데, 이 경우에는 약 250개의 오류 암호문을 이용하여 비밀 키를 추출할 수 있음을 보였다. 또한, Kim과 Quisquater는 AES 키 스케줄링 상에 바이트 오류를 주입하는 방법으로 8개의 정상-오류 암호문 쌍을 이용하여 비밀 키를 찾아내는 방법을 제안하기도 하였다[6].

한편, 2009년에는 Tunstall 등에 의해 한 쌍의 정상 암호문과 오류 암호문으로 비밀 키를 찾는 방법이 제안되었다[7]. 즉, 공격자는 8라운드 앞단에 바이트 단위의 오류를 주입하여 얻은 오류 암호문과 번의 전탐색을 수행하면 비밀 키를 찾아낼 수 있었다. 현재까지 알려진 바로는 이 공격이 라운드 과정을 수행하는 동안 오류를 주입하는 공격 방법으로서는 가장 적은 수의 암호문을 필요로 하는 공격 기법이다. 이외에도 AES 오류 주입 공격의 다른 형태로 키 스케줄링이나 라운드 함수를 수행하는 중간값에 오류를 넣는 것이 아니라 명령어에 오류를 주입하여 키를 찾아내는 방법이 있다[14, 15]. 이러한 오류 주입 공격은 라운드 함수를 수행하는 과정에서 카운트 값 등을 변경하는 방법으로 알고리듬의 수행 라운드를 축소함으로써 쉽게 비밀 키를 찾아낼 수 있었다.

2.3 AES 오류 주입 공격 대응 방법

지금까지의 AES에 대한 오류 주입 공격은 주로 라운드 함수 연산을 수행하는 도중에 오류를 주입한 후 출력되는 오류 암호문에 대한 분석을 주로 실시하였다. 그리고 AES에 대한 오류 주입 공격 대응 방법들도 키 스케줄링에 대한 대응책보다는 대부분 라운드 연산 과정에 대한 공격 대응책이 주류를 이루고 있다. AES에 대한 대표적인 오류 주입 공격 대응 기법으로는 암호화된 데이터를 다시 복호화하여 원 메시지와 비교하는 동시 오류 검출(Concurrent Error Detection, CED) 방법이 있다. 그러나 이 대응 방법은 시간적 오버헤드가 많이 증가하여 오류가 탐지되기까지의 시간이 오래 걸린다는 단점이 존재한다[8].

다른 오류 주입 공격 대응 방법으로는 패리티 비트를 검사하는 방법이 있다. 그러나 이 방식은 결과 값들에 대한 패리티를 검출하는 방식이다 보니 한 비트에 대한 에러 검출에는 용이하나 많은 비트의 에러 검출에는 취약한 특성이 있다. 현재, AES에 대한 오류 주입 공격은 한 번의 바이트 정도의 오류 주입으로도 비밀 키를 찾아내는 수준이므로 패리티 비트를 이용한 대응 방법으로는 공격을 방어하기에 불충분하다[9, 10]. 그리고 S-Box에 대한 패리티 비트의 추가 때문에 테이블 메모리가 256*9비트 정도 필요하며, 패리티 처리 단위가 비트라서 바이트 단위의 오류 검출 방법보다 오히려 시간적 비용이 많이 든다. 또한, Natale 등에 의해 제안된 16개의 S-Box에 대한 입·출력 연관성을 검사하는 방법이 있지만, 오류 검출률이 낮은 것이 단점이다[11].

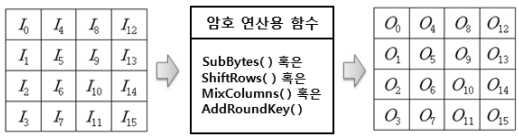
Yen과 Wu에 의해 제안된 $(n+1, n)$ CRC를 이용한 대응 방안[13]에서는 n을 4, 8, 16을 사용하게 되는데, n을 큰 값으로 설정하면 사용하는 패리티의 수가 줄어들게 되므로 오류 검출 확률이 줄어들게 되고, 작은 값일수록 사용하는 패리티의 수가 늘어나므로 오류 검출 확률이 커지게 된다. 그러나 패리티의 수가 늘어날수록 구현 오버헤드가 커지기 때문에 대응 방안 구현 시 이를 고려하여 설계해야 한다. 이 방식에서는 3가지의 구현 형태를 제안하였는데, 특히 테이블을 사용하지 않는 대응책을 구현하면 많은 시간적, 공간적 오버헤드가 요구된다.

III. 제안하는 오류 주입 공격 대응 방안

제안하는 오류 주입 공격 방어책은 AES에서 사용하는 각 함수의 차분 값을 연산이 진행되는 동안 계산하고 연산이 끝난 후 이를 검사함으로써 오류 주입 여부를 검출하는 기법이다[17]. 즉, AES 연산 함수인 SubBytes, ShiftRows, MixColumns 그리고 AddRoundKey의 입·출력 차분 값을 암호화 연산을 하는 동안 계산해 나가는 것이다. 제안 방식에서는 AES 암호 입·출력의 기본 단위가 바이트라는 점과 오류 주입 공격 성공 모델이 바이트 단위의 랜덤 오류라는 점 때문에 오류 검사 단위를 바이트로 사용하였다. 또한, 제안 방식은 알고리듬 전체에 대한 오류 검

출뿐만 아니라 특정 라운드에서만 오류를 검출할 수 있도록 하는 다양성을 가지고 있으며, 오류 주입 검사의 모든 연산이 XOR로만 이루어져 계산상의 효율성을 높이고자 한다.

제안 방식의 설명을 위해 각 함수의 입력 값과 출력 값의 위치를 확인할 필요가 있는데 〔그림 2〕는 이를 도시한 것이다. 여기서 I_i와 O_i는 한 바이트를 나타낸다. 그러나 본 논문에서는 검사의 효율성을 위해 각 입·출력에 대한 차분 값을 16바이트로 모두 유지하는 것이 아니다. 즉, 입력 값 16바이트를 바이트 단위로 모두 XOR하여 한 바이트를 만들고, 출력 값 16바이트 역시 바이트 단위로 XOR하여 한 바이트를 만들어 이 두 값을 다시 XOR로 차분한 한 바이트를 최종 검사 값으로 한다.



〔그림 2〕 AES 각 함수의 입·출력 값

3.1 SubBytes 함수에서의 차분 바이트

SubBytes인 경우 입력 16바이트에 대한 XOR값을 구하여 한 바이트를 구한다. 그리고 출력의 16바이트를 각각 XOR하여 한 바이트를 구한다. 이때 입력(출력) 값을 XOR한 것을 "입력(출력) 차분 바이트"라 하고, 또한 이 두 값을 XOR한 최종 검사 값을 "함수의 차분 바이트"라 부르기로 한다. 예로서 SubBytes 함수의 입·출력 차분 바이트를 정의하면 식 (1), (2)와 같다. 그리고 SubBytes 함수의 차분 바이트는 식 (3)과 같이 쓸 수 있다(단, $0 \le i \le 15$).

$$I_{SB} = SBI_0 \oplus SBI_1 \oplus \cdots SBI_{15} = \bigoplus_{i=0}^{15} SBI_i \tag{1}$$

$$O_{SB} = SBO_0 \oplus SBO_1 \oplus \cdots SBO_{15} = \bigoplus_{i=0}^{15} SBO_i \tag{2}$$

$$D_{SB} = I_{SB} \oplus O_{SB} = \bigoplus_{i=0}^{15} SBI_i \oplus \bigoplus_{i=0}^{15} SBO_i \tag{3}$$

따라서 SubBytes 함수의 차분 바이트 D_{SB}는 다음과 같이 구할 수도 있다.

$$\begin{aligned} D_{SB}' &= (SBI_0 \oplus SBO_0) \oplus (SBI_1 \oplus SBO_1) \\ &\quad \oplus \cdots (SBI_{15} \oplus SBO_{15}) \\ &= \bigoplus_{j=0}^{15} (SBI_j \oplus SBO_j) \\ &= DSB_0 \oplus DSB_1 \oplus \cdots DSB_{15} = \bigoplus_{j=0}^{15} DSB_j \end{aligned} \tag{4}$$

따라서 SubBytes 함수가 오류 공격 없이 수행되었는가를 확인하기 위해서는 S-Box의 입력 SBI_s와 출력 SBO_s값의 차분 값($DSB_s = SBI_s \oplus SBO_s$)을 사전에 저장해 두어야 한다. 즉, SubBytes 입력으로 가질 수 있는 값은 0에서 255이므로 〔그림 3〕과 같은 차분 테이블을 미리 계산해 둔다. 따라서 오류 주입 공격의 방어를 위해서는 알고리듬 내의 SubBytes 함수를 수행한 후 식 (3)의 값을 구하고 테이블을 이용하여 식 (4)의 값을 구한다. 식 (3)의 값과 식 (4)의 값은 다음 식에 의해 동일함이 증명되며 이 두 값이 일치하면 오류가 없는 것으로 판단한다.

$$\begin{aligned} D_{SB} &= I_{SB} \oplus O_{SB} \\ &= (SBI_0 \oplus SBI_1 \oplus \cdots SBI_{15}) \\ &\quad \oplus (SBO_0 \oplus SBO_1 \oplus \cdots SBO_{15}) \\ &= (SBI_0 \oplus SBO_0) \oplus (SBI_1 \oplus SBO_1) \\ &\quad \oplus \cdots (SBI_{15} \oplus SBO_{15}) \\ &= D_{SB}' \end{aligned} \tag{5}$$

입력(SBI_s)	출력(SBO_s)	차분(DSB_s)
0	63	63
1	7C	7D
:	:	:
FF	16	E9

〔그림 3〕 S-Box 차분 테이블

3.2 ShiftRows와 MixColumns 함수에서의 차분 바이트

다음으로 ShiftRows와 MixColumns에 대한 함수의 차분 바이트를 확인해 볼 필요가 있다. ShiftRows 함수는 스테이트 값의 단순한 바이트 단위 이동에 불과하므로 입력 값과 출력 값에 대한 함수의 차분 바이트 값은 0이 된다. 즉, 입력 16바이트를 XOR하고 출력 16바이트를 XOR한 두 값을 차분해도 값의 변화는 없다.

$$D_{SR} = 0 \tag{6}$$

MixColumns 함수에서의 차분 바이트 값을 계산

하기 위해 먼저 한 열(column)에 대한 차분 바이트를 계산해 볼 필요가 있다.

$$\begin{bmatrix} MXO_0 \\ MXO_1 \\ MXO_2 \\ MXO_3 \end{bmatrix} = \begin{bmatrix} 2\,3\,1\,1 \\ 1\,2\,3\,1 \\ 1\,1\,2\,3 \\ 3\,1\,1\,2 \end{bmatrix} \begin{bmatrix} MXI_0 \\ MXI_1 \\ MXI_2 \\ MXI_3 \end{bmatrix} \quad (7)$$

따라서 한 열에 대해서 입력 차분과 출력 차분을 구해 보면 같음을 알 수 있다.

$$\begin{aligned} & MXO_0 \oplus MXO_1 \oplus MXO_2 \oplus MXO_3 \\ & = MXI_0 \oplus MXI_1 \oplus MXI_2 \oplus MXI_3 \end{aligned} \quad (8)$$

결국, 식 (8)과 같이 한 열에 대한 차분 바이트 값이 0이 되듯이 다른 열에 대해서도 분석해 보면 MixColumns 함수의 차분 바이트는 항상 0이 됨을 알 수 있다.

$$D_{MX} = 0 \quad (9)$$

3.3 AddRoundKey 함수에서의 차분 바이트

마지막으로 AddRoundKey 함수의 차분 바이트를 계산해 본다. AddRoundKey 함수는 스테이트의 각 바이트에 라운드 키를 XOR하여 더하는 과정이므로 입력 차분 바이트 값과 출력 차분 바이트 값을 차분한 결과는 라운드 키를 바이트별로 차분한 것과 같다.

$$I_{AK} = AKI_0 \oplus AKI_1 \oplus \cdots AKI_{15} = \bigoplus_{i=0}^{15} AKI_i \quad (10)$$

$$\begin{aligned} O_{AK} &= AKO_0 \oplus AKO_1 \\ &\quad \oplus \cdots AKO_{15} = \bigoplus_{i=0}^{15} AKO_i \\ &= (AKI_0 \oplus RK_0) \oplus (AKI_1 \oplus RK_1) \\ &\quad \oplus \cdots (AKI_{15} \oplus RK_{15}) = \bigoplus_{i=0}^{15} (AKI_i \oplus RK_i) \end{aligned} \quad (11)$$

$$\begin{aligned} D_{AK} &= I_{AK} \oplus O_{AK} \\ &= RK_0 \oplus RK_1 \oplus \cdots RK_{15} = \bigoplus_{i=0}^{15} RK_i \end{aligned} \quad (12)$$

3.4 한 라운드 내에서의 차분 바이트

이와 같은 사실을 이용하여 한 라운드를 기준으로 "라운드 차분 바이트"를 구하여 보자. 위에서 살펴본 바와 같이 ShiftRows와 MixColumns의 차분 바이트는 0이 되고 SubBytes의 차분 바이트는 16개의 입력에 따른 사전 테이블 값을 XOR한 결과가 된다. 또한, AddRoundKey 함수의 차분 바이트는 16바이트의 라운드 키를 XOR한 결과가 됨을 알 수 있다. 따라서 4개의 함수가 있는 한 라운드내의 입력과 출력에 대한 "라운드 차분 바이트"는 아래 식과 같음을 알 수 있다. 여기서 I_{RND}(O_{RND})는 입력(출력) 데이터 RI_i(RO_i)의 바이트를 바이트 단위로 XOR한 결과이다.

$$D_{RND} = I_{RND} \oplus O_{RND} = \bigoplus_{i=0}^{15} RI_i \oplus \bigoplus_{i=0}^{15} RO_i \quad (13)$$

$$\begin{aligned} D_{RND}' &= D_{SB} \oplus D_{SR} \oplus D_{MX} \oplus D_{AK} \\ &= (\bigoplus_{i=0}^{15} DSB_i) \oplus (\bigoplus_{i=0}^{15} RK_i) \end{aligned} \quad (14)$$

따라서 AES의 라운드 레벨에서는 한 라운드 시작하기 전의 입력 차분 바이트 값(I_{RND})을 구하고 라운드 함수를 수행 후 출력 값에 대한 출력 차분 바이트(O_{RND})를 구한다. 그리고 식 (14)과 같이 〔그림 3〕에서 입력에 해당하는 16바이트의 차분 값($\bigoplus_{i=0}^{15} DSB_i$)과 16바이트의 라운드 키 값을 차분한 값($\bigoplus_{i=0}^{15} RK_i$)을 구한다. 그리고 식 (13)의 값과 식 (14)의 값이 동일하면 오류가 주입되지 않은 정상적인 알고리듬 수행으로 간주한다.

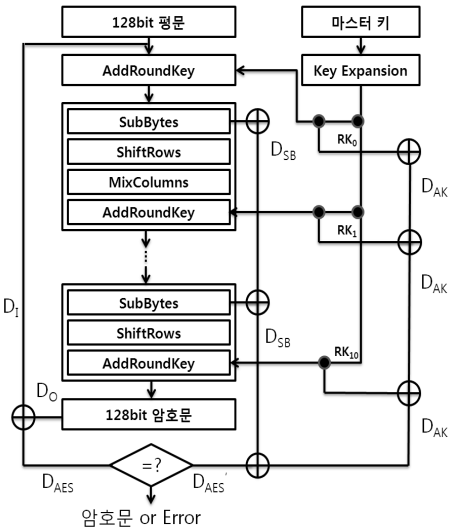
3.5 암호화 과정 전체에서의 오류 주입 검출

제안하는 오류 주입 검출 방법은 위와 같이 함수 레벨이나 라운드 레벨에서 수행할 수도 있지만 AES 알고리듬의 전체 입·출력을 검사하는 알고리듬 레벨에서도 적용할 수 있다. 제안하는 오류 주입 공격 방어 대책을 알고리듬 레벨에서 적용한 것을 도시한 것이 〔그림 4〕이다.

〔그림 4〕에서 보면 평문(입력)과 암호문(출력)의 차분 바이트 값은 다음과 같이 표현할 수 있다. 여기서 PI_i는 평문 입력 16바이트를 의미하며 CO_i는 암호문 출력 16바이트를 의미한다.

$$\begin{aligned} D_{AES} &= D_I \oplus D_O \\ &= (\bigoplus_{i=0}^{15} PI_i) \oplus (\bigoplus_{i=0}^{15} CO_i) \end{aligned} \quad (15)$$

다른 한편으로 알고리듬 레벨의 차분 값이 맞는지를 확인하는 값은 아래와 같이 구할 수 있다. 여기서 i



〔그림 4〕 제안하는 오류 주입 공격 대응책

$(0 \le i \le 15)$는 스테이트의 각 바이트를 의미하며 r $(1 \le r \le 10)$은 라운드를 의미한다.

$$D_{AES}' = \bigoplus_{r=0}^{10} (\bigoplus_{i=0}^{15} RK_{r,i}) \oplus \bigoplus_{r=1}^{10} (\bigoplus_{i=0}^{15} DSB_{r,i}) \quad (16)$$

따라서 AES 암호 알고리듬이 정상적으로 수행되면 식 (15)의 값과 식 (16)의 값이 동일할 것이고 오류가 발생할 경우에는 서로 다른 값을 갖게 되어 오류임을 인지하고 암호문을 출력하지 않는다. 제안하는 AES 오류 주입 공격 방어 알고리듬을 수행하는 전체 과정을 단계별로 기술하면 〔그림 5〕와 같다.

IV. 시뮬레이션 및 비교 분석

본 논문에서는 오류 주입 공격에 대응하는 AES 알고리듬을 제안하였는데 오류가 정확히 검출되는지 컴퓨터로 구현하여 시뮬레이션하였다. 또한, 오류 검출 성능을 다른 방식과 비교하기 위해 16비트 패리티 비트를 이용한 대응 방안〔9〕, 1비트 패리티 비트를 이용하는 방법〔10〕과 CRC를 이용한 대응 방안〔13〕도 같이 구현하였다. 시뮬레이션은 알고리듬 레벨에서 AES 알고리듬을 구현하여 수행하였으며 성능 평가를 위한 환경은 다음과 같다.

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

〔표 1〕은 오류 검출 성능 면에서 각 대응 기법에 대한 오류 검출 확률을 정리한 것으로, 한 바이트에 영향을 미치는 오류를 주입하는 것으로 가정하였다. 그 이유는 현재까지 제시된 AES 오류 주입 공격 중 가장 효율적인 방법은 연산 중간값에 바이트 오류를 주입하는 공격이기 때문이다. 시뮬레이션은 문헌 〔16〕과 동일한 오류 가정하에서 AES의 모든 라운드, 모든 함수, 모든 바이트에 1~8비트 오류를 주입하도록 실시하였다.

예를 들어 3비트 오류를 주입할 경우, 10라운드 동안 40번의 함수 연산이 수행되고, 16개 바이트가 존재하므로, $10*4*16*\binom{3}{8}=35840$번의 오류를 주입하여

입력 : 평문(PI), 마스터 키(K) 혹은 라운드 키(RK)
출력 : 정상 암호문(CO) 혹은 오류 메시지(Error)

단계 0 : (사전 준비 단계) SubBytes 입·출력 값에 대한 차분 값을 미리 계산하여 저장한다.

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

단계 2 : 라운드 키에 대한 차분 바이트를 구한다. ($D_0 = \bigoplus_{r=0}^{10} (\bigoplus_{i=0}^{15} RK_{r,i})$)

각 라운드를 수행하면서 SubBytes 함수의 차분 바이트와 각 라운드 키에 대한 차분 바이트를 구한 후 D_0와 XOR 한다. $D_1 = \bigoplus_{r=1}^{10} (\bigoplus_{i=0}^{15} DSB_{r,i})$, $D_{AES}' = D_0 \oplus D_1$

단계 3 : 출력 암호문에 대한 차분 바이트를 구하고 ($D_O = \bigoplus_{i=0}^{15} (CO_i)$), D_I 값과 XOR를 한다.($D_{AES} = D_I \oplus D_O$)

단계 4 : D_{AES}'와 D_{AES} 값이 동일한지 비교하여 동일하면 암호문을 출력한다.
그렇지 않으면 오류 메시지를 출력한다.

〔그림 5〕 오류 주입 공격에 강인한 AES 알고리듬

오류 검출 확률을 실험하였다.

Bertoni 등의 패리티 대응 기법[9]은 짝수 패리티를 사용하여 오류를 감지하기 때문에 홀수 비트 오류가 발생하면 100% 오류 감지가 가능하지만, 짝수 비트 오류가 발생할 경우에는 MixColumns 연산에서 오류 확산에 영향을 받아 오류를 감지하지 못하는 경우가 발생한다. Wu 등에 의한 방법도 동일한 방법으로 구현하여 실험한 결과 홀수 비트 오류는 약 12~13%, 짝수 개의 오류는 87% 정도로 짝수 개의 오류에는 매우 취약함을 보였다. Yen과 Wu의 CRC 대응 기법[13]에서 n이 4일 경우에는 모든 오류에 대해 100% 검출이 가능하다는 것을 확인할 수 있었다. 위 3가지 기존 대응 방법을 시뮬레이션한 결과는 문헌[16]에서 실험한 결과와 동일하였다. 이에 반해 제안하는 대응책은 현재까지 가장 현실적인 오류 주입 공격 모델인 바이트 단위의 오류를 100% 검출해 낼 수 있었다.

기존의 방법과 추가되는 오버헤드를 비교해 보면 개발자 환경이나 시스템 구조에 따라 차이를 보일 수 있다. 패리티 비트 검출 방법은 S-Box의 패리티를 저장할 수 있는 256*9비트의 메모리 공간이 필요하다. 또한, CRC 방법 중 Lookup 테이블을 이용하는 방식은 256바이트의 메모리 공간이 필요하다. 본 논문에서는 SubBytes 연산 시 S-Box 테이블을 사용하는 것과 무관하게 차분 테이블이 필요하므로 256바이트의 메모리 공간이 추가로 요구된다.

대응 방법을 소프트웨어 구현하여 수행 시간을 비교 분석해 본 결과 대응책이 없는 AES는 1000번 수행하는데 약 31ms가 소요되었다. 그리고 Bertoni 등의 패리티 대응 기법은 약 71ms 정도, Wu 방법은 94ms 정도의 시간이 소요되었다. CRC 방법은 구현 형태에 따라 많은 차이가 있으며 특히, SubBytes를 테이블에 기반을 두지 않는 방식은 시간적인 오버헤드가 많아 200ms 이상이 소요되었다. 반면 제안하는 대응 방법은 시간적 오버헤드가 거의 없이 평균 32ms 정도가 소요되었다.

특히, 논문에서 제안하는 대응 기법에는 Piret과 Quisquater가 제안한 공격 방법[4]도 적용하여 비밀 키 추출이 가능한지 분석하였다. [그림 6]은 정상적으로 알고리듬이 수행되었을 경우와 8라운드 시작 전에 바이트 오류가 주입되었을 경우를 가정하여 시뮬레이션한 화면이다. 첫 번째 화면은 오류의 주입이 없을 때 암·복호 과정이 정상적으로 이루어지는 것을 나타낸 것이며, 아래 화면은 한 바이트의 오류를 공격 시점에 주입하였을 때 오류 발생을 감지하여 암호문을 출력하지 않는 화면을 나타낸 것이다.

제안 방식을 구현할 경우의 오버헤드를 살펴보면, 먼저 S-Box의 차분 테이블 값을 저장할 수 있는 256 바이트 정도의 영구적 혹은 일시적 메모리 공간이 추가적으로 요구된다. 또한, 시간적으로는 각 S-Box 입력 값에 대한 차분 값(DSB)을 서로 XOR하는 연산과 각 라운드 키 값을 XOR하는 연산 시간이 추가적으로 필요하다. 참고로 암호 알고리듬에서 마스터 키를 사용한 키 스케줄링 과정 없이 라운드 키를 바로 저장하여 사용한다면, 라운드 키 값에 대한 XOR 값도 미리 계산해 둘 수 있어 계산 시간상의 오버헤드는 알고리듬 전체에 비해 아주 작게 된다.

[그림 6] 오류 주입 공격 시뮬레이션 화면

[표 1] 오류 검출 대응 기법 실험 결과

바이트내 오류 비트 수	오류 주입 시도 수	오류 미검출 오류 수 및 확률(%)			
		16비트 패리티 비트[9]	1비트 패리티 비트[10]	CRC[13]	제안 방식
1	5,120	5,120(0%)	666(13.0%)	0(0%)	0(0%)
2	17,920	16,912(94.4%)	15,674(87.5%)	0(0%)	0(0%)
3	35,840	35,840(0%)	4,385(12.2%)	0(0%)	0(0%)
4	44,800	39,760(88.8%)	39,135(87.4%)	0(0%)	0(0%)
5	35,840	35,840(0%)	4,576(12.8%)	0(0%)	0(0%)
6	17,920	14,896(83.1%)	15,666(87.4%)	0(0%)	0(0%)
7	5,120	5,120(0%)	615(12.0%)	0(0%)	0(0%)
8	640	496(77.5%)	563(88.0%)	0(0%)	0(0%)

V. 결 론

본 논문에서는 AES에 대한 오류 주입 공격 기술을 분석하고 주입된 오류를 검출하기 위한 대응 기법들을 살펴보았다. 기존의 방법들은 패리티 비트 등을 활용함으로써 오류 검출 성능이 떨어지거나 대응 알고리듬 구현을 위한 오버헤드가 많이 필요하였다. 그러나 논문에서 제안하는 방식은 S-Box에 대한 입·출력 차분 값만 메모리에 사전 저장한 후 이를 활용하여 알고리듬 전체의 오류를 탐지할 수 있다. 즉, 알고리듬의 입·출력에 대한 차분 바이트 값이 오류없이 유지되고 있는지를 순차적으로 계산함으로써 바이트나 비트 단위의 오류는 모두 검출해 낼 수 있었다. 결론적으로 제안 방식은 안전한 AES를 구현하는데 추가되는 오버헤드는 아주 적지만 오류 검출 성능은 매우 우수한 특성을 나타낸다.

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. Giraud, "DFA on AES," Advanced Encryption Standard-AES'04, LNCS 3373, pp. 27 - 41, 2005.

〔6〕 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.

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

〔8〕 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.

〔9〕 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," IEE Transactions on Computers, vol 52, no. 4, pp. 492-505, 2003

〔10〕 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.

〔11〕 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.

〔12〕 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.

〔13〕 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.

〔14〕 H. Choukri and M. Tunstall, "Round reduction using faults," FDTC'05, pp. 13-24, 2005.

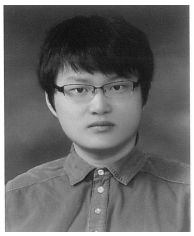
〔15〕 박제훈, 배기석, 오두환, 문상재, 하재철, "AES에 대한 반복문 오류 주입 공격," 한국정보보호학회논문지, 20(6), pp. 59-65, 2010년 12월.

〔16〕 K. Bousselam, G. Di Natale, M. L. Flottes,

and B. Rouzeyre, "Fault Detection in Crypto-Devices," In book "Fault Detection", Wei Zhang (Ed.), ISBN: 978-953-307-037-7, InTech, March 2010.

〔17〕 박정수, 최용제, 최두호, 하재철, "AES에서 입·출력 차분값에 기반한 오류 주입 공격 방어대책," 한국정보보호학회 하계학술대회 논문집, 22(1), pp. 34-38, 2012년 6월.

〈著者紹介〉



박 정 수 (Jeong-Soo Park) 학생회원
2011년 2월: 호서대학교 컴퓨터공학과 졸업
2011년 3월~현재: 호서대학교 대학원 정보보호학과 석사 과정
〈관심분야〉 스마트폰 보안, 부채널 공격, 무선 네트워크 보안

최 용 제 (Yong-Je Choi) 정회원
1996년 8월: 전남대학교 전자공학과 졸업
1999년 2월: 전남대학교 전자공학과 석사
1999년 2월~1999년 8월: 전남대학교 전자통신연구소 인턴연구원
1999년 8월~현재: 한국전자통신연구원 선임연구원
〈관심분야〉 보안프로세서 설계, 부채널 분석 시스템, RFID/USN 보안

최 두 호 (Doo-Ho Choi) 정회원
1994년 2월: 성균관대학교 수학과 졸업
1996년 2월: KAIST 수학과 석사
2002년 2월: KAIST 수학과 박사
2002년 1월~현재: 한국전자통신연구원 선임연구원
〈관심분야〉 암호학, 부채널 분석, RFID/USN 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 부교수
〈관심분야〉 정보보호, 네트워크 보안, 부채널 공격