Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.24, NO.6, Dec. 2014 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2014.24.6.1117

블록 암호 LEA에 대한 차분 오류 공격*

박 명 서,[1†] 김 종 성[1,2‡]
[1]국민대학교 금융정보보안학과, [2]국민대학교 수학과

Differential Fault Analysis of the Block Cipher LEA*

Park Myungseo,[1†] Kim Jongsung[1,2‡]
[1]Dept. of Financial Information Security, Kookmin University,
[2]Dept. of Mathematics, Kookmin University

요 약

차분 오류 공격(Differential Fault Analysis)은 블록 암호 알고리즘의 안전성 분석에 널리 사용되는 부채널 기법 중 하나이다. 차분 오류 공격은 대표적인 블록 암호인 DES, AES, ARIA, SEED와 경량 블록 암호인 PRESENT, HIGHT 등에 적용되었다[1,2,3,4,5,6]. 본 논문에서는 최근 주목 받고 있는 국내 경량 블록 암호 LEA(Lightweight Encryption Algorithm)에 대한 차분 오류 공격을 최초로 제안한다. 본 논문에서 제안하는 LEA에 대한 차분 오류 공격은 300개의 선택적 오류 주입 암호문을 이용하여 2^{35}의 시간 복잡도로 128 비트 마스터 키 전체를 복구한다. 본 연구의 실험 결과, Intel Core i5 CPU, 메모리 8 GB의 일반 PC 환경에서 수집한 오류 주입 암호문을 이용하여, 평균 40분 이내에 마스터 키를 찾을 수 있음을 확인하였다.

ABSTRACT

Differential Fault Analysis(DFA) is widely known for one of the most powerful method for analyzing block cipher. it is applicable to block cipher such as DES, AES, ARIA, SEED, and lightweight block cipher such as PRESENT, HIGHT. In this paper, we introduce a differential fault analysis on the lightweight block cipher LEA for the first time. we use 300 chosen fault injection ciphertexts to recover 128-bit master key. As a result of our attack, we found a full master key within an average of 40 minutes on a standard PC environment.

Keywords: Side-channel attacks, Differential fault analysis, LEA

I. 서 론

IT 환경이 발전하면서 기존의 폐쇄적이고 고정적인 유선 네트워크 기반의 컴퓨팅 환경이 노트북, 스마트폰, 테블릿 PC 등과 같이 휴대 가능한 소형의 무선 네트워크 기반의 컴퓨팅 환경으로 바뀌었다. 하지만 소형 장비에 따른 제한적인 컴퓨팅 파워는 기존의 블록 암호 AES[8], ARIA[9], SEED[10] 등을 통한 보안 적용을 어렵게 만들었다. 이러한 문제를 극복하기 위해 저전력, 저비용의 경량 암호에 대한 연구가 지속적으로 이루어졌다. 그 결과, HIGHT[11], PRESENT[12], KATAN/KTANTAN[13], LEA[7] 등의 경량 암호 알고리즘이 제안되었다.

국내에서 개발된 LEA는 2013년 12월 한국정보통신기술협회(TTA)의 표준으로 지정되었으며, 현재까지 활발한 기술 보급이 이뤄지고 있다. LEA는 128, 192, 256 비트의 마스터 키를 사용하는 128

접수일(2014년 10월 30일), 게재확정일(2014년 11월 24일)
* 이 논문은 2013년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2013R1A1A2059864).
† 주저자, pms91@kookmin.ac.kr
‡ 교신저자, jskim@kookmin.ac.kr(Corresponding author)

비트 블록 암호로써, 마스터 키의 길이에 따라 총 24, 28, 32 라운드로 이뤄진다. 라운드 함수는 ARX(Addtion, Rotation, XOR) 연산만으로 구성되어 제한적인 컴퓨팅 환경의 스마트폰, 스마트 카드(Smart Card), RFID(Radio Frequency IDentification), 스마트 그리드(Smart Grid) 등에 사용이 적합하다.

부채널 분석(Side Channel Analysis)은 암호 알고리즘이 탑재된 보안 장비가 작동할 때 발생하는 전력신호, 전자파, 소리 등의 부가적인 정보를 이용하는 공격 방법이다. 대표적인 부채널 분석 방법은 전력 분석(Power Analysis), 시차 공격(Timing Attack), 오류 주입 공격(Fault Attack)이 있다. 차분 오류 공격은 기존의 암호 알고리즘 안전성 분석 방법인 차분 공격에 부채널 분석 방법 중 하나인 오류 주입 공격을 결합한 공격 방법이다. 1997년 Biham과 Shamir는 최초로 블록 암호 DES에 차분 오류 공격을 적용하였다[1]. 그 이후 AES, ARIA, SEED, PRESENT, HIGHT 등 여러 블록 암호에 적용되었다[2,3,4,5,6].

본 논문에서는 128 비트의 마스터 키를 사용하는 LEA에 대한 차분 오류 공격을 처음으로 제안하였다. 본 논문에서 제안하는 차분 오류 공격에서는 약 3300개의 오류 주입을 통해 공격에 필요한 오류 주입 암호문 300개를 선택적으로 획득하고, 이를 통해 시간 복잡도 2^{35}으로 마스터 키를 유일하게 복구할 수 있다. 또한, 192, 256 비트 마스터 키를 사용하는 LEA의 마스터 키를 복구하기 위해서는 동일한 오류 주입 암호문의 개수와 2^{99}, 2^{163}의 시간 복잡도를 요구한다.

본 논문의 2장은 블록 암호 LEA에 대한 키 스케줄과 암·복호화 방법에 대해 설명하고, 3장에서는 제안하는 LEA 차분 오류 공격을 위한 사전 준비로써 오류 주입 가정과 특정 위치 오류 주입 암호문 획득 방법에 대해 설명한다. 4장에서는 LEA의 차분 오류 주입 공격에 대해 설명하고, 실제 마스터 키 획득 실험 수행 결과를 설명한다. 마지막으로 5장에서 결론을 맺는다.

II. 블록 암호 LEA

2.1 LEA의 암·복호화

LEA 암호 알고리즘은 128 비트 블록단위로 암·복호화하는 경량 블록 암호이다[7]. 128, 192, 256 비트의 마스터 키를 사용할 수 있으며, 각 마스터 키의 길이에 따라 LEA-128, LEA-192, LEA-256 으로 구분한다. LEA의 라운드 함수는 S-box를 사용하지 않고, 32 비트 단위의 법덧셈($\boxplus$) 및 법뺄셈($\boxminus$), 좌·우측 비트 순환이동(ROL, ROR), XOR($\oplus$) 연산만으로 구성되어 있어 경량 구현이 가능하다.

Fig. 1. 은 i 라운드 함수($i=1,2,\cdots$)를 나타낸 것으로 128 비트의 중간 변수 $X^i(X^i[0], X^i[1], X^i[2], X^i[3])$과 i 라운드 키 $RK^i(rk^i[0], rk^i[1], rk^i[2], rk^i[3], rk^i[4], rk^i[5])$을 입력하여 라운드 키 XOR, 법덧셈, 비트 순환이동 과정을 거친 후 128 비트 출력 $X^{i+1}(X^{i+1}[0], X^{i+1}[1], X^{i+1}[2], X^{i+1}[3])$을 얻는다.

LEA-128 암호 알고리즘의 암호화 과정은 초기 평문 128 비트와 192 비트의 1 라운드 키를 입력으로 1 라운드 함수의 출력을 얻는다. 2 라운드는 1 라운드의 128 비트 출력 값과 192 비트의 2 라운드 키를 입력하여 128 비트 출력 값을 얻는다. 이러한 수행을 총 24 라운드 거친 후 나온 128 비트 출력을 암호문으로 얻는다. 복호화 과정은 암호화의 역 연산을 통해 이루어진다. 평문 대신 암호문 128 비트를 입력으로 하고, 암호화에 사용되었던 라운드 키를 역으로 주입한다. 또한 법덧셈을 법뺄셈 연산으로 처리하여 평문 128 비트를 출력으로 얻는다. LEA-192과 LEA-256의 라운드 수는 각각 28, 32 라운드이며, 암·복호화 과정은 LEA-128과 동일하다.

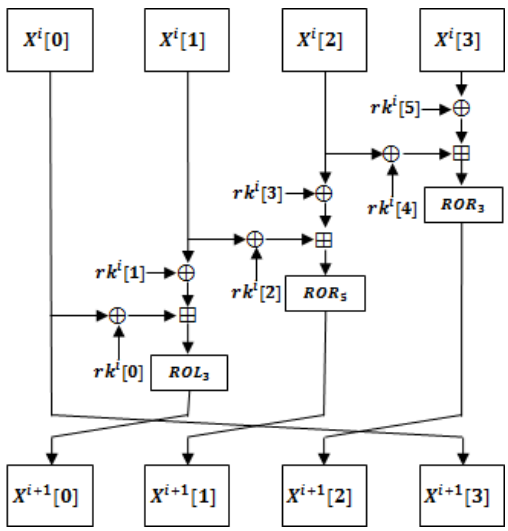

Fig. 1. An encryption round function of LEA

2.2 LEA의 키 스케줄

Table 1. 은 LEA-128, LEA-192, LEA-256의 암호화 키 스케줄을 나타낸 것이다. LEA-128의 암호화 키 스케줄의 경우 먼저 128 비트의 마스터 키 $K(K[0], K[1], K[2], K[3])$를 내부 변수 T $(T[0], T[1], T[2], T[3])$에 대입한다. 그 다음 상수 δ와 법덧셈을 한 후 비트순환이동한 T를 통해 1 라운드 키 $RK^1(rk^1[0], rk^1[1], rk^1[2], rk^1[3], rk^1[4], rk^1[5])$를 생성하고, 다음 2 라운드 키를 생성하기 위한 입력으로 사용한다. 각 i 라운드 출력으로 생성된 T를 통해 i 라운드 키 $RK^i(rk^i[0]=T[0], rk^i[1]=rk^i[3]=rk^i[5]=T[1], rk^i[2]=T[2], rk^i[4]=T[3])$을 얻는다. LEA-192, LEA-256의 암호화 키 스케줄도 전체 구조는 동일하나 내부 변수 및 라운드 키 결정 방법에 차이가 있다. LEA-192의 암호화 키 스케줄은 192 비트의 내부변수 T를 통해 192 비트의 i 라운드 키를 생성한다. 또한 LEA-256의 암호화 키 스케줄에서는 256 비트의 내부변수 T를 통해 192 비트의 i 라운드 키를 생성한다. 여기서 각 라운드 키 생성 시 내부변수 T를 중복 사용하지 않는다는 점이 LEA-128 키 스케줄과의 차이점이다. 이러한 차이는 본 논문의 공격 과정 중 마스터 키 복구에서 중요한 부분으로 작용한다.

Table 1. Key schedules of LEA-128, LEA-192, LEA-256

○Key schedule of LEA-128

$T=(T[0], T[1], T[2], T[3]) \leftarrow K=(K[0], K[1], K[2], K[3])$
for $i=0$ to 23 do
$T[0]=ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 4]))$
$T[1]=ROL_3(T[1] \boxplus ROL_{i+1}(\delta[i \bmod 4]))$
$T[2]=ROL_6(T[2] \boxplus ROL_{i+2}(\delta[i \bmod 4]))$
$T[3]=ROL_{11}(T[3] \boxplus ROL_{i+3}(\delta[i \bmod 4]))$
$RK^i=(T[0], T[1], T[2], T[1], T[3], T[1])$
end for

○Key schedule of LEA-192

$T=(T[0], T[1], \cdots, T[5]) \leftarrow K=(K[0], K[1], \cdots, K[5])$
for $i=0$ to 27 do
$T[0]=ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 6]))$
$T[1]=ROL_3(T[1] \boxplus ROL_{i+1}(\delta[i \bmod 6]))$
$T[2]=ROL_6(T[2] \boxplus ROL_{i+2}(\delta[i \bmod 6]))$
$T[3]=ROL_{11}(T[3] \boxplus ROL_{i+3}(\delta[i \bmod 6]))$
$T[4]=ROL_{13}(T[4] \boxplus ROL_{i+4}(\delta[i \bmod 6]))$
$T[5]=ROL_{17}(T[5] \boxplus ROL_{i+5}(\delta[i \bmod 6]))$
$RK^i=(T[0], T[1], T[2], T[3], T[4], T[5])$
end for

○Key schedule of LEA-256

$T=(T[0], T[1], \cdots, T[7]) \leftarrow K=(K[0], K[1], \cdots, K[7])$
for $i=0$ to 31 do
$T[6i \bmod 8]=ROL_1(T[6i \bmod 8] \boxplus ROL_i(\delta[i \bmod 8]))$
$T[6i+1 \bmod 8]=ROL_3(T[6i+1 \bmod 8] \boxplus ROL_{i+1}(\delta[i \bmod 8]))$
$T[6i+2 \bmod 8]=ROL_6(T[6i+2 \bmod 8] \boxplus ROL_{i+2}(\delta[i \bmod 8]))$
$T[6i+3 \bmod 8]=ROL_{11}(T[6i+3 \bmod 8] \boxplus ROL_{i+3}(\delta[i \bmod 8]))$
$T[6i+4 \bmod 8]=ROL_{13}(T[6i+4 \bmod 8] \boxplus ROL_{i+4}(\delta[i \bmod 8]))$
$T[6i+5 \bmod 8]=ROL_{17}(T[6i+5 \bmod 8] \boxplus ROL_{i+5}(\delta[i \bmod 8]))$
$RK^i=(T[6i \bmod 8], T[6i+1 \bmod], T[6i+2 \bmod 8],$
end for

III. LEA에 대한 차분 오류 공격을 위한 사전 준비

본 장에서는 제안하는 공격에 사용되는 오류 주입 가정과 실제 공격을 위해 필요한 특정 위치 오류 주입 암호문의 획득 방법에 대해 설명한다.

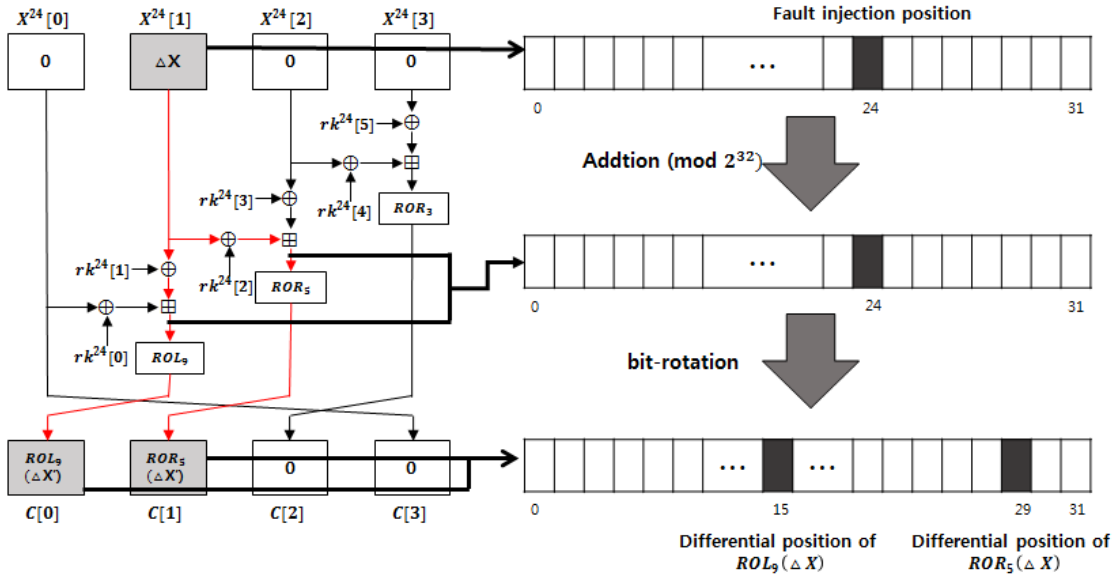

Fig. 2. Ciphertext bit-positions affected by the fault injection

3.1 오류 주입 가정 및 오류 위치 결정

본 논문에서 제안하는 LEA에 대한 차분 오류 공격에 사용되는 오류 주입은 32 비트 랜덤 오류 주입을 가정한다. 즉, 공격자는 원하는 라운드의 32 비트 입력 레지스터에 랜덤한 한 비트 오류를 주입할 수 있다. 일반적으로 블록 암호의 라운드 함수는 입력에 대한 출력을 얻는 구조를 반복 수행하도록 구현되어 있기 때문에 각 라운드의 입력 레지스터에 오류를 주입하는 것은 현실적으로 가능하다[1,2,3,4,5,6]. 이러한 가정을 통해 공격자는 선택된 평문 P에 대한 오류가 주입되지 않은 정상적인 암호문 C와 오류가 주입된 암호문 C'을 얻을 수 있다. 본 공격에서 실제 공격자는 24 라운드 특정 입력 레지스터에 오류를 주입할 수 있지만 랜덤하게 주입된다는 가정 때문에 정확한 비트 위치를 선택할 수 없다. 하지만 출력되는 정상 암호문 C와 오류가 주입된 암호문 C'에 대한 차분 $C \oplus C'$의 확인 과정을 통해 24 라운드의 입력 레지스터의 오류가 주입된 비트의 위치를 알 수 있다.

예를 들어, Fig. 2. 와 같이 랜덤 오류 주입을 통해 $X^{24}[1]$의 24번째 비트에 오류가 주입되었다면, 암호문의 첫 번째, 두 번째 워드 $C[0]$와 $C[1]$의 15번째 비트(24번째 비트에 ROL_9의 결과), 3번째 비트(24번째 비트에 ROR_5의 결과) 각각에 반드시 오류가 발생한다. 반대로 $C[0]$과 $C[1]$의 15, 29번째 비트에 각각 오류가 발생하였다면, $X^{24}[1]$의 24번째 비트에 오류가 발생하였다는 것을 알 수 있다. 만약 연속적 2 비트 이상에 차분이 발생한다면, 그 이유는 24번째 비트에 주입된 오류를 통해 라운드 함수의 법덧셈 연산 중 캐리가 발생할 수 있기 때문이다. 암호문에 1 비트의 차분이 생기면 역 비트순환이동 후 차분 위치가 오류 주입 위치가 되고, 2 비트 이상의 차분이 생기면 역 비트순환이동 후 하위 차분 비트 위치가 오류 주입 위치가 된다. Table 2. 는 24 라운드의 32 비트 입력 레지스터에 오류를 주입했을 때 확산되는 암호문의 위치를 나타낸다.

Table 2. Ciphertext word-positions affected by the fault injection

Fault injection parameters(32-bit)	Affected ciphertext word-positions
$X^{24}[0]$	$C[0]$, $C[3]$
$X^{24}[1]$	$C[0]$, $C[1]$
$X^{24}[2]$	$C[1]$, $C[2]$
$X^{24}[3]$	$C[2]$

3.2 특정 위치 오류 주입 암호문 획득 방법

LEA에서 오류 주입은 공격자가 32 비트 랜덤 오류 주입 후 암호문을 확인하는 방법으로 원하는 위치에 오류가 주입된 암호문을 획득할 수 있다. 하지만 공격자가 차분 오류 공격을 위해 필요한 특정 비트 위치에 오류가 주입된 암호문이 N개 이상이라면 그보다 많은 양의 오류 주입이 필요하다.

32 비트 랜덤 오류 주입에서 특정 1 비트 위치에 오류가 주입될 확률은 1/32이고, 그 외의 비트 위치에 오류가 주입될 확률은 31/32이다. 이 때 특정 1 비트에 오류가 주입된 암호문 ω개 이상을 획득할 확률 $p = \sum_{r=w}^{n} nCr(\frac{1}{32})^r(\frac{31}{32})^{n-r}$이다($n$ =오류 주입 수행 횟수, ω = 공격에 필요한 최소 오류 주입 암호문 개수). 여기서 n =1100, ω =25이면, 이때 확률 $p_{25} \approx 96/100$이다. 확률 p_{25}를 이용하여 특정 4 비트 위치에 오류가 주입된 암호문을 얻을 확률을 구하면 $(p_{25})^4 \approx 85/100$이다. 즉, 32 비트 랜덤 오류 주입을 1100회 수행하면 약 85%의 확률로 특정 4 비트 위치의 암호문 25개 이상씩 얻을 수 있다. 이를 실제 프로그램 상으로 1000회의 실험을 수행한 결과, 1100회 시행 이내에 특정 4 비트 위치 오류 주입 암호문을 각각 25개 이상 얻는 경우가 총 840~870회 발생하였다. 다시 말하면, 공격자가 원하는 특정 4 비트 위치의 오류 주입 암호문을 각각 25개 이상을 얻기 위해 32 비트 입력 레지스터에 랜덤한 오류 주입을 1100회 시도한다면, 약 84~87%의 확률로 필요한 오류 주입 암호문을 얻을 수 있다.

IV. LEA에 대한 차분 오류 공격을 이용한 마스터 키 복구 과정

본 장에서는 블록 암호 LEA-128의 차분 오류 공격에 대해 설명한다. 또한 본 장의 마지막 절인 4.5절에서는 LEA-128에 대한 차분 오류 공격이 LEA-192, LEA-256에 어떻게 적용되는지 살펴본다. LEA-128에 대한 공격 과정은 크게 24번째 라운드 키 후보 추측과 마스터 키 추측 단계로 나눈다. 24 라운드 키 후보 추측 단계에서는 선택적 오류 주입 암호문들을 통해 라운드 키에 대한 후보를 추측한다. 여기서 알아낸 후보들을 통해 마스터 키 추측 단계에서 마스터 키를 찾아낼 수 있다.

4.1 24 라운드 키 후보 추측

24 라운드 키 후보 추측 과정은 총 3단계로 나누어 이뤄진다. 각 단계에서 사용되는 오류 주입 위치는 Fig. 3. 와 같으며(오류 주입은 1 비트 오류 주입임), 특정 위치에 오류가 주입된 암호문을 통해 대수적 특성식을 세워 키 후보를 줄이는 작업을 수행한다.

1단계. $rk^{24}[0]$에 대한 후보 키 탐색

$rk^{24}[0]$에 대한 후보 키를 탐색하기 위해서 $X^{24}[1]$ 위치에 오류가 주입된 암호문 C'을 사용한다(C : 정상적인 암호문). 이를 통해 차분 $\triangle X^{24}[1]$ $(= X^{24}[1] \oplus X^{24}[1]')$에 관한 식 (3)을 식 (1)과 (2)의 XOR 연산으로 계산한다.

$$X^{24}[1] = (ROR_9(C[0]) \boxminus (X^{24}[0] \oplus rk^{24}[0])) \oplus rk^{24}[1] \quad (1)$$

$$X^{24}[1]' = (ROR_9(C[0]') \boxminus (X^{24}[0] \oplus rk^{24}[0])) \oplus rk^{24}[1] \quad (2)$$

$$\triangle X^{24}[1] = (ROR_9(C[0]) \boxminus (X^{24}[0] \oplus rk^{24}[0])) \oplus ((ROR_9(C[0]') \boxminus (X^{24}[0] \oplus rk^{24}[0])) \quad (3)$$

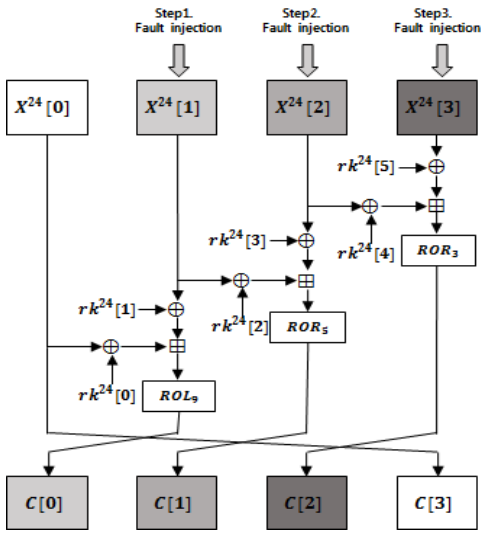

Fig. 3. Three-step fault injection for the 24-th round-key guesses

식 (3)을 통해 차분 $\triangle X^{24}[1]$을 만족하는 $rk^{24}[0]$ 값을 추측하여 후보 키로 저장한다. 여기서 $\triangle X^{24}[1], C[0], C[0]', X^{24}[0](=C[3])$은 이미 알고 있는 값이기 때문에 이러한 계산이 가능하다. 또한, $\triangle X^{24}[1]$의 값은 $C \oplus C'$의 값을 통해 알 수 있다 (3.1절 참조).

2단계. $rk^{24}[1] \oplus rk^{24}[2]$에 대한 후보 탐색

$rk^{24}[0]$의 후보 키를 구하는 방식으로는 $rk^{24}[1]$, $rk^{24}[2]$의 후보 키를 구할 수 없다. 그 이유는 $rk^{24}[0]$의 후보 키를 구하는 단계에서는 $X^{24}[0]$의 값을 암호문 $C[3]$을 통해 구할 수 있었지만, $rk^{24}[1], rk^{24}[2]$에서는 이에 해당하는 $X^{24}[1], X^{24}[2]$의 값은 $C[0], C[1]$을 통해 구할 수 없기 때문이다. 그 대신에 $X^{24}[1] \oplus rk^{24}[1]$과 $X^{24}[1] \oplus rk^{24}[2]$을 계산하여 $rk^{24}[1] \oplus rk^{24}[2]$의 후보를 구하고, 24 라운드 키 후보를 구하는데 이용한다. $rk^{24}[1] \oplus rk^{24}[2]$에 대한 후보를 탐색하기 위해서는 $X^{24}[2]$ 위치에 오류가 주입된 암호문 C''와 이전 단계에서 구한 $rk^{24}[0]$의 후보 키를 이용한다.

$$X^{24}[1] \oplus rk^{24}[1] = ROR_9(C[0]) \boxminus (X^{24}[0] \oplus rk^{24}[0]) \quad (4)$$

$$X^{24}[2] = (ROL_5(C[1]) \boxminus ((X^{24}[1] \oplus rk^{24}[2])) \oplus rk^{24}[3] \quad (5)$$

$$X^{24}[2]' = (ROL_5(C[1]'') \boxminus ((X^{24}[1] \oplus rk^{24}[2])) \oplus rk^{24}[3] \quad (6)$$

$$\triangle X^{24}[2] = (ROL_5(C[1]) \boxminus (X^{24}[1] \oplus rk^{24}[2])) \oplus ((ROL_5(C[1]'') \boxminus (X^{24}[1] \oplus rk^{24}[2])) \quad (7)$$

먼저 앞서 구한 $rk^{24}[0]$의 후보를 식 (4)에 대입하여 $X^{24}[1] \oplus rk^{24}[1]$ 값을 구한다. 그 다음 차분 $\triangle X^{24}[2]$에 관한 식 (7)을 식 (5)와 (6)의 XOR 연산으로 계산한다. 식 (7)과 알고 있는 값 $\triangle X^{24}[2], C[1], C[1]''$을 통해 차분 $\triangle X^{24}[2]$을 만족하는 $X^{24}[1] \oplus rk^{24}[2]$을 추측한다. 마지막으로 추측한 $X^{24}[1] \oplus rk^{24}[2]$와 $rk^{24}[0]$을 이용하여 계산한 $X^{24}[1] \oplus rk^{24}[1]$을 XOR하여 계산된 값인 $rk^{24}[1] \oplus rk^{24}[2]$을 후보로 저장한다.

3단계. $rk^{24}[3] \oplus rk^{24}[4]$에 대한 후보 탐색

$rk^{24}[3] \oplus rk^{24}[4]$에 대한 후보를 구하는 방식은 $rk^{24}[1] \oplus rk^{24}[2]$의 후보를 구하는 방식과 같으나 2단계에서 사용된 $X^{24}[1] \oplus rk^{24}[1]$과 같은 역할을 하는 $X^{24}[2] \oplus rk^{24}[3]$은 앞서 구한 $rk^{24}[0]$, $rk^{24}[1] \oplus rk^{24}[2]$을 이용하여 구해야한다. 구하는 방법은 Fig. 4. 에 나타낸 순서와 같다.

먼저 $rk^{24}[0]$ 후보 키를 식 (4)에 대입시켜 그에 해당하는 $X^{24}[1] \oplus rk^{24}[1]$을 구한다(Fig. 4. 의 ①). 그 다음 $rk^{24}[1] \oplus rk^{24}[2]$ 후보와 앞서 2단계에서 구한 $X^{24}[1] \oplus rk^{24}[1]$의 XOR 연산을 통해 $X^{24}[1] \oplus rk^{24}[2]$을 구한다(Fig. 4. 의 ②). 이 값을 식 (8)에 대입하여 $X^{24}[2] \oplus rk^{24}[3]$의 값을 구한다 (Fig. 4. 의 ③).

$$X^{24}[2] \oplus rk^{24}[3] = ROL_5(C[1]) \boxminus (X^{24}[1] \oplus rk^{24}[2]) \quad (8)$$

이와 동시에 $X^{24}[3]$ 위치에 오류가 주입된 암호문 C'''을 통해 차분 $\triangle X^{24}[3]$에 관한 식 (11)을 계산한다.

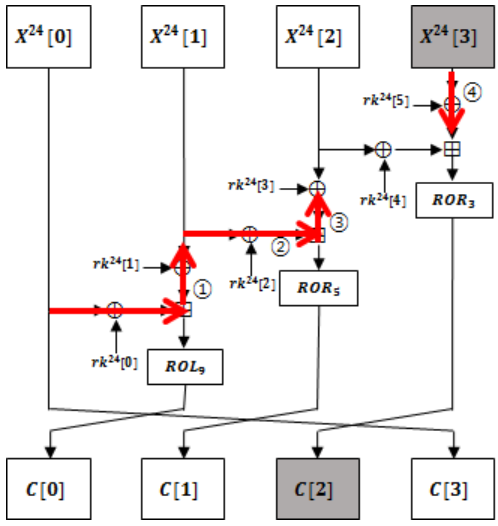

Fig. 4. Round-key search path for the 24-th round

$$X^{24}[3] = (ROL_3(C[2]) \boxminus (X^{24}[2] \oplus rk^{24}[4])) \oplus rk^{24}[5] \quad (9)$$

$$X^{24}[3]' = (ROL_3(C[2]''') \boxminus (X^{24}[2] \oplus rk^{24}[4])) \oplus rk^{24}[5] \quad (10)$$

$$\triangle X^{24}[3] = (ROL_3(C[2]) \boxminus (X^{24}[2] \oplus rk^{24}[4])) \oplus ((ROL_3(C[2]''') \boxminus (X^{24}[2] \oplus rk^{24}[4])) \quad (11)$$

여기서 알고 있는 값 $\triangle X^{24}[3], C[2], C[2]'''$을 이용하여 식 (11)(= 식 (9)⊕식 (10))을 만족하는 $X^{24}[2] \oplus rk^{24}[4]$을 추측한다. 그 다음 앞서 구한 $X^{24}[2] \oplus rk^{24}[3]$와 XOR하여 계산된 값인 $rk^{24}[3] \oplus rk^{24}[4]$을 후보로 저장한다(Fig. 4. 의 ④).

실제 LEA-128 키 스케줄을 살펴보면, 192 비트의 i 라운드 키 $RK^i(rk^i[0], rk^i[1], rk^i[2], rk^i[3], rk^i[4], rk^i[5])$ 중에 $rk^i[1] = rk^i[3] = rk^i[5]$인 특성을 가지고 있기 때문에 총 128 비트의 키를 사용한다. 이러한 특성과 앞의 3단계를 통해 구한 $rk^{24}[0]$, $rk^{24}[1] \oplus rk^{24}[2]$, $rk^{24}[3] \oplus rk^{24}[4]$의 후보들을 통해 24 라운드 키 후보를 생성할 수 있다. 실제 24 라운드 키 후보 생성 방법은 Table 3. 과 같다.

Table 3. Extraction process for the 24-th round-key candidates

- Input
  - ○ Candidates of $rk^{24}[0]$
    - $A_1, A_2, \cdots, A_\alpha$
  - ○ Candidates of $rk^{24}[1] \oplus rk^{24}[2]$
    - $B_1, B_2, \cdots, B_\beta$
  - ○ Candidates of $rk^{24}[3] \oplus rk^{24}[4]$
    - $C_1, C_2, \cdots, C_\gamma$
  - ○ Parameters
    - $T = (T[0], T[1], T[2], T[3], T[4], T[5])$
- Output
  - ○ Candidates of RK^{24}
    - $RK_1^{24}, RK_2^{24}, \cdots, RK_{2^{32} \times \alpha \times \beta \times \gamma}^{24}$

- Extraction phase for the RK^{24}

```
candidates
for i = 1 to α do (1)
  T[0] = A_i
 for j = 1 to β do (2)
   for k = 1 to γ do (3)
     for l = 0 to 2^32 − 1 do (4)
       T[1] = l, T[3] = l, T[5] = l;
       T[2] = B_j ⊕ l, T[4] = C_k ⊕ l;
       RK^24_{i×j×k×(l+1)} ← T
     end for (4)
   end for (3)
 end for (2)
end for (1)
```

위 알고리즘을 통해 총 $2^{32} \times \alpha \times \beta \times \gamma$개의 24 라운드 키 후보를 찾을 수 있다. 여기서 24 라운드 키 후보의 개수는 마스터 키 복구 시 연산량에 큰 영향을 미친다. 따라서 라운드 키 후보 수에 직접적인 영향을 주는 α, β, γ의 수를 적절히 낮추는 작업이 필요하다. 본 논문에서는 α, β, γ를 각각 2개로 구하여 총 2^{35}개의 24 라운드 후보로 마스터 키 추측을 수행하였으며, α, β, γ의 결정 방법은 4.3절에서 설명한다.

4.2 마스터 키 복구 방법 - 마스터 키 추측

본 절에서는 4.1절에서 구한 24 라운드 키 후보를 통해 실제 128 비트의 마스터 키를 찾는 방법에 대해 설명한다. 실제 LEA-128의 키 스케줄을 살펴보면, 마스터 키를 주입한 내부 변수 T와 상수 δ 값들에 대한 법덧셈, 비트순환이동의 반복적인 수행으로 라운드 키를 생성한다. 따라서 마지막 라운드인 24 라운드의 키를 입력하여 기존의 키 스케줄을 역으로 수행하면 모든 라운드 키를 복구할 수 있고, 마스터 키까지 모두 복구 가능하다. 이렇게 복구된 마스터 키가 옳은 키인지 확인하는 과정이 필요하다. 올바른 평문 P에 대한 암호문 C와 복구된 키를 통해 P를 암호화하여 나온 C^*가 일치하는지 확인함으로써 옳은 키 확인이 가능하다. Table 4. 는 마스터 키 복구 과정을 나타낸다.

Table 4. Master key recovery process

- Input
  - ○ Candidates of RK^{24}
    - $RK_1^{24}, RK_2^{24}, \cdots, RK_{2^{32}\times\alpha\times\beta\times\gamma}^{24}$
  - ○ Plaintext and its corresponding ciphertext pair (P, C)
    - $P=(P[0], P[1], P[2], P[3])$
    - $C=(C[0], C[1], C[2], C[3])$
  - ○ Constants used in the key schedule
    - $\delta=(\delta[0], \delta[1], \delta[2], \delta[3])$
  - ○ Inner status parameter
    - $T=(T[0], T[1], T[2], T[3])$
- Output
  - ○ Master key
    - $K=(K[0], K[1], K[2], K[3])$

- Master key recovery algorithm

for $i=1$ to $2^{32}\times\alpha\times\beta\times\gamma$ do (1)

$T[0]=rk_i^{24}[0]$, $T[1]=rk_i^{24}[1]$,
$T[2]=rk_i^{24}[2]$, $T[3]=rk_i^{24}[4]$

for $j=23$ to 0 do (2)
$T[0]=ROR_1(T[0]) \boxplus ROL_j(\delta[j(\mathrm{mod}4)])$
$T[1]=ROR_3(T[1]) \boxplus ROL_{j+1}(\delta[j(\mathrm{mod}4)])$
$T[2]=ROR_6(T[2]) \boxplus ROL_{j+2}(\delta[j(\mathrm{mod}4)])$
$T[3]=ROR_{11}(T[3]) \boxplus ROL_{j+3}(\delta[j(\mathrm{mod}4)])$
end for (2)

$K[0]=T[0], K[1]=T[1],$
$K[2]=T[2], K[3]=T[3]$

$C^*=\mathrm{LEA\_Encryption}(P, K)$

if ($C=C^*$)
return K
end if

end for (1)

4.3 24 라운드 키 후보 결정 방법

4.1절의 각 단계에서 24 라운드의 키 후보를 구하기 위해서 오류가 주입된 암호문과 오류가 주입되지 않은 암호문에 대한 차분을 이용한다. 효율적인 공격을 위해 오류 주입 수와 위치를 알맞게 결정하는 것이 중요하다. 제안하는 공격에서는 $X^{24}[1], X^{24}[2], X^{24}[3]$의 위치에 오류 주입을 하며, 각 32 비트의 1, 8, 16, 24번째 비트에 오류가 주입된 암호문을 사용한다.

먼저 24번째 비트에 오류가 주입된 암호문에 대한 차분 형태를 통해 하위 8 비트에 대한 라운드 키 후보를 결정할 수 있다. 이 때 주입된 오류를 통해 법덧셈에서 캐리가 발생할 수 있기 때문에 식 (3), (7), (11)의 차분 확인 과정에서 23번째 비트 부분까지의 확인이 필요하다. 24번째 비트에 오류 주입된 암호문 25개를 이용하면 하위 8 비트의 키 후보를 유일하게 결정할 수 있다.[1] 그 다음 16번째 비트에 오류가 주입된 암호문 25개와 유일하게 결정된 하위 8 비트 정보를 이용하여, 16~23번째 비트의 키 후보를 유일하게 결정한다. 8~15번째 비트의 키 후보도 8번째 비트에 오류 주입된 암호문을 이용하여 동일한 방법으로 유일하게 결정한다. 마지막 0~7번째 비트의 키 후보를 결정하기 위해 1번째 비트에 오류 주입된 암호문을 사용한다. 0번째 비트에 오류 주입을 할 경우 법덧셈에 대한 캐리가 일어난 차분을 확인할 수 없기 때문에 많은 후보가 발생하나, 1번째 비트에 오류 주입된 암호문 25개를 사용할 경우 0~7번째 비트의 키 후보가 2개 남는다. 이런 방식으로 각 단계마다 약 $100(=25\times4)$개, 총 300개의 오류 주입 암호문으로 24 라운드 키 정보에 대한 후보를 2^3개($rk^{24}[0]$의 후보 개수 $\alpha=2$, $rk^{24}[1]\oplus rk^{24}[2]$의 후보 개수 $\beta=2$, $rk^{24}[3]\oplus rk^{24}[4]$

1) 이론상으로 오류 주입된 암호문의 개수가 25보다 적은 양으로 키 후보가 유일하게 결정될 수 있지만, 실험 시 랜덤하지 않은 암호문이 선택되는 경우도 발생하므로 이론상의 수치보다 실험치가 높게 나타났다. 또한, 본 실험에서는 성공 확률을 높이기 위해 충분한 오류 주입 암호문 25개를 이용하였다.

의 후보 개수 $\gamma=2$), 24 라운드 키 후보 2^{35}개를 얻을 수 있다.

4.4 구현 결과

본 절에서는 제안하는 LEA에 대한 차분 오류 공격을 실제 프로그램으로 작성하여 수행한 결과를 설명한다. 25개의 올바른 평문/암호문 쌍을 이용하여 $X^{24}[1], X^{24}[2], X^{24}[3]$의 1, 8, 16, 24,번째 비트 위치에 오류가 주입된 암호문 각 25개씩 총 300개의 오류 주입 암호문 쌍을 사용하여 공격을 수행하였다. 특정 위치에 오류 주입된 암호문을 얻기 위해 3.2절의 내용을 이용하였고, 시뮬레이션을 통해 약 3300번의 오류 주입을 통해 약 85%의 확률로 원하는 오류 주입 암호문을 획득할 수 있었다. 실험 환경은 다음과 같다. OS는 Window 7(64bit), CPU는 Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz 3.40GHz, 메모리는 8 GB를 사용하였다.

먼저 24 라운드의 키 정보에 대한 후보를 얻기 위해 올바른 평문/암호문 쌍 25개를 통해 수집한 300개의 오류 주입 암호문으로 1000회의 실험을 한 결과 100%의 확률로 $rk^{24}[0]$(2개), $rk^{24}[1]\oplus rk^{24}[2]$(2개), $rk^{24}[3]\oplus rk^{24}[4]$(2개)의 후보를 획득하였다. 즉 라운드 키 후보 2^{35}개를 얻었다. 이는 공격 프로그램으로 수 초 이내에 이루어진다. 그 다음 얻은 라운드 키 후보와 Table 4. 의 알고리즘을 이용하여 마스터 키 복구를 수행하였다. 여기서 각 라운드 키 후보마다 옳은 키인지 확인하기 위해 역 키 스케줄 1회, LEA 암호화 1회를 수행한다. 마스터 키 검출 프로그램을 통한 모든 라운드 키 후보를 조사하는데 역 키 스케줄 2^{35}회, LEA 암호화 2^{35}회의 연산이 필요하며, 평균적으로 수행 시간 40분 이내에 옳은 마스터 키가 유일하게 복구되었다.

4.5 LEA-192, LEA-256에 대한 차분 오류 공격

본 절에서는 LEA-192, LEA-256에 대한 차분 오류 공격 방법과 복잡도에 대해 설명한다. 기본적인 공격 방법은 LEA-128과 동일하게 마지막 라운드 키에 대한 후보를 결정한 후 마스터 키를 복구한다. LEA-192와 LEA-256은 LEA-128과는 달리 각 라운드에서 사용되는 키가 모두 다르기 때문에 192비트 모두를 추측해야한다. 또한 LEA-128과 LEA-192의 경우 마지막 라운드 키를 통해 모든 라운드 키를 복구할 수 있으나, LEA-256은 마지막 라운드 키 이외에 내부 변수 $T[0], T[1]$을 추가적으로 추측해야 모든 라운드 키 복구가 가능하다. 따라서, LEA-192, LEA-256에 대한 차분 오류 공격은 각각 $2^{35+64}=2^{99}$, $2^{35+64+64}=2^{163}$의 시간 복잡도를 요구한다.

V. 결 론

본 논문에서는 블록 암호 LEA에 대한 차분 오류 공격을 제안하였다. 공격자는 24 라운드 입력 레지스터 32 비트에 랜덤한 1 비트 오류를 주입하여 얻어진 암호문과 정상적인 암호문의 차분 특성을 이용하여 24 라운드 키 후보를 구하고, 이를 통해 옳은 마스터 키를 복구가 가능함을 보였다. 24 라운드 입력 레지스터의 특정 12 비트 위치에 오류를 주입하여 얻은 300개의 오류 주입 암호문을 이용하여 2^{35}의 시간 복잡도로 128 비트 마스터 키를 찾았다. 본 공격에 대한 기본적인 아이디어는 LEA-192, LEA-256에서 동일하게 적용 가능하며 28 라운드, 32 라운드 키 후보의 개수를 각각 2^{99}개로 구할 수 있다. 다만 키 스케줄의 차이로 LEA-192는 2^{99}의 시간 복잡도로 공격이 가능하지만, LEA-256은 2^{163}의 시간 복잡도로 공격이 가능하다. 향후 연구 과제로는 더 적은 수의 오류 주입과 암호문을 통한 최적화 LEA 차분 오류 공격과 대응 기법에 대해 추가적으로 연구할 예정이다.

References

[1] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," Proceedings of Crypto 1997, LNCS 1294, pp. 513-525, Aug. 1997.

[2] J. Blomer and J.-P. Seifert, "Fault based cryptanalysis of the advanced encryption standard (AES)," Proceedings of FC 2003, LNCS 2742, pp. 162-181, Jan. 2003.

[3] W Li. D Cu and J Li, "Differential fault analysis on the ARIA algorithm," Information Sciences, vol. 178, no. 19,

pp. 3727-3739, Oct. 2008.

〔4〕 Kitae Jeong, Jaechul Sung, and Seokhie Hong, "A Differential Fault Attack on Block Cipher SEED," Journal of the Korea Institute of Information Security & Cryptology, 20(4), pp. 17-24, Aug. 2010.

〔5〕 Sehyun Park, Kitae Jeong, Yuseop Lee, Jaechul Sung and Seokhie Hong, "Improved Differential Fault Analysis on Block Cipher PRESENT-80/128," Jonornal of the Korea Institute of Information Security & Cryptology, 22(1), pp. 33-41, Feb. 2012

〔6〕 Yuseop Lee, Jongsung Kim and Seokhee Hong, "A Differential Fault Attack against Block Cipher HIGHT," Jonornal of the Korea Institute of Information Security & Cryptology, 22(3), pp. 485-494, Feb. 2012.

〔7〕 J. Park, D. Hong, D. Kim, D. Kwon and H. Park, "128-Bit Block Cipher LEA," TTAK.KO-12.0223, Dec. 2013.

〔8〕 NIST, "Advanced Encryption Standard," FIPS-197, Nov. 2001.

〔9〕 D. Kwon, J. Kim, S. Park, S. Sung, Y. Sohn, J. Song, Y. Yeom, E. Yoon, S. Lee, J. Lee, S. Chee, D. Han and J. Hong, "New Block Cipher: ARIA," Proceedings of ICISC 2003, LNCS 2971, pp. 443-456, Nov. 2003.

〔10〕 J. Park, S. Lee, J. Kim, and J. Lee, "The SEED Encryption Algorithm," RFC 4009, Dec. 2005.

〔11〕 D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim and S. Chee, "HIGHT: a new block cipher suitable for low-resource device," Proceedings of CHES 2006, LNCS 4249, pp. 46-59, Oct. 2006.

〔12〕 A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," Proceedings of CHES 2007, LNCS 4727, pp. 450-466, Sep. 2007.

〔13〕 C. Canniere, O. Dunkelman and M. Knezevic. "KATAN and KTANTAN – a family of small and efficient hardware-oriented block cipohers," Proceedings of CHES 2009, LNCS 5747, pp. 272-288, Sep. 2009.

〔14〕 Sehyun Park, Kitae Jeong, Yuseop Lee, Jaechul Sung, and Seokhie Hong, "Differential Fault Analysis on Block Cipher ARIA-128," Jonornal of the Korea Institute of Information Security & Cryptology, 21(5), pp. 15-25, Oct. 2011.

〈저자소개〉

박 명 서 (Myungseo Park) 학생회원
2013년 2월: 국민대학교 수학과 졸업
2013년 3월~현재: 국민대학교 금융정보보안학과 석사과정
〈관심분야〉 정보보호, 암호 알고리즘, 이동통신보안

김 종 성 (Jongsung Kim) 종신회원
2000년 8월/2002년 8월: 고려대학교 수학 학사/이학석사
2006년 11월: K.U.Leuven, ESAT/SCD-COSIC 정보보호 공학박사
2007년 2월: 고려대학교 정보보호대학원 공학박사
2007년 3월~2009년 8월: 고려대학교 정보보호기술연구센터 연구교수
2009년 9월~2013년 2월: 경남대학교 e-비즈니스학과 조교수
2013년 3월~현재: 국민대학교 수학과 조교수
2014년 3월~현재: 국민대학교 일반대학원 금융정보보안학과 조교수
〈관심분야〉 정보보호, 암호 알고리즘, 디지털 포렌식