情報保護學會論文誌
第20卷 第4號, 2010. 8

블록 암호 SEED에 대한 차분 오류 공격*

정 기 태,[1†] 성 재 철,[2‡] 홍 석 희[1]
[1]고려대학교 정보보호연구원, [2]서울시립대학교 수학과

A Differential Fault Attack on Block Cipher SEED*

Kitae Jeong,[1†] Jaechul Sung,[2‡] Seokhie Hong[1]
[1]Center for Information Security Technologies, Korea University
[2]Department of Mathematics, University of Seoul

요 약

차분 오류 공격(DFA)은 블록 암호의 안전성 분석에 널리 사용되는 부채널 공격 기법으로서, 대표적인 블록 암호인 DES, AES, ARIA, SEED 등에 적용되었다. 기제안된 SEED에 대한 DFA는 라운드 16의 입력값에 영구적인 에러를 주입한다는 가정을 이용한다. 본 논문에서는 SEED의 라운드 차분의 특성 분석과 덧셈 차분의 특성을 이용하여 SEED에 대한 DFA를 제안한다. 본 논문에서 사용하는 공격 가정은 특정 레지스터에 1-비트 오류를 주입한다는 것이다. 이 공격을 이용하여 약 2^{32}번의 간단한 산술 연산으로 SEED의 라운드 키 및 마스터 키를 복구할 수 있다. 이는 일반적인 PC에서 수 초 내에 가능함을 의미한다.

ABSTRACT

A differential fault attack(DFA) is one of the most efficient side channel attacks on block ciphers. Almost all block ciphers, such as DES, AES, ARIA, SEED and so on., have been analysed by this attack. In the case of the known DFAs on SEED, the attacker induces permanent faults on a whole left register of round 16. In this paper, we analyse SEED against DFA with differential characteristics and addition-XOR characteristics of the round function of SEED. The fault assumption of our attack is that the attacker induces 1-bit faults on a particular register. By using our attack, we can recover last round keys and the master key with about 2^{32} simple arithmetic operations. It can be simulated on general PC within about a couple of second.

Keywords: Differential fault attack, Block cipher, SEED

I. 서 론

블록 암호의 안전성 분석 기법은 차분 공격[2] 등과 같이 알고리즘 자체의 이론적인 취약점을 이용하는 기법과, 부채널 공격(side channel attack) 등과 같이 암호 시스템의 실질적인 구현 과정에서 얻어지는 정보들을 이용하는 기법으로 분류된다.

부채널 공격은 암호 알고리즘을 구현하였을 때 발생하는 연산 시간, 전력, 전자기파, 오류 등의 부가적인 정보를 이용하는 공격 방법으로서, 오류 주입 공격(fault attack), 시차 공격(timing attack), 전력 분석 공격(power attack) 등이 있다. 최초의 부채널 공격은 시차 공격으로서 Kocher에 의해 공개키 암호의 분석 방법으로 제안되었다[7]. 이 공격은 큰 수의 지수승 연산 시 지수의 비트가 1인 경우 곱셈이 수행되고 0인 경우 곱셈이 수행되지 않아 발생하는 시간의 차이를 이용하여 비트의 값을 추측하는 공격 기

접수일(2010년 1월 7일), 수정일(2010년 4월 16일), 게재확정일(2010년 5월 12일)
* 이 논문은 2010년도 정부(교과부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2010-0000261).
† 주저자, kite@cist.korea.ac.kr
‡ 교신저자, jcsung@uos.ac.kr

법이다. 이 시차 공격이 제안된 이후 전력 분석 공격, 오류 주입 공격 등의 다양한 기법들이 제안되었다.

오류 주입 공격[4]은 공격 대상 알고리즘에 전력 변화, 강제 클럭킹 등을 이용하여 오류를 발생시키고 이를 이용하여 키 정보를 얻는 공격 기법이다. 이 공격은 공격자의 능력과 발생하는 오류의 유형에 따라 일시적(transient) 오류 주입 공격과 영구적(permanent) 오류 주입 공격으로 나뉜다. 일시적 오류 주입 공격에서는 알고리즘 수행 시 순간적인 클록 값 변경 혹은 불규칙적인 전력 공급 등을 이용하여 특정 지점에서만 오류가 발생한다. 따라서 이 오류는 한시적으로만 공격 대상 알고리즘에 영향을 준다. 이와 달리, 영구적 오류 주입 공격에서는 알고리즘 수행 전에 레지스터 값을 특정한 값으로 고정하거나 레지스터를 파괴함으로서 영구적인 오류를 발생시킨다. 따라서 이 오류는 계속 공격 대상 알고리즘에 영향을 준다. 이 공격 기법은 단순히 암호 연산 시 발생하는 시간을 체크하는 시간 공격이나 전력을 체크하는 전력 분석 공격 보다는 좀 더 능동적인 공격 형태이다.

1997년 Biham과 Shamir는 오류 주입 공격을 대칭키 암호 시스템에 최초로 적용하여 블록 암호 DES를 분석하였다[3]. 차분 오류 공격(differential fault attack, DFA)라 불리는 이 공격은 기존의 차분 공격을 오류 주입 공격에 결합하여 DES 뿐만 아니라 AES, Triple- DES, ARIA 등 대부분의 블록 암호에 적용되었다[5,6,10]. 하지만 블록 암호에 대한 차분 오류 분석의 관건은 공격 가정의 현실성과 공격 복잡도이다. 공격 가정의 현실성은 어떻게 공격 가정을 보다 현실에 맞게 가정의 조건을 약화시킬 수 있는 것인가의 문제이고, 공격 복잡도 문제는, 만약 복구하려는 키 길이가 s라면, 공격에 필요한 복잡도를 얼마나 2^s 보다 적게 할 수 있을 것인가의 문제이다.

블록 암호 SEED는 국내에서 개발된 16-라운드 Feistel 구조를 갖는 128-비트 블록 암호로서, AES, Camellia와 더불어 ISO/IEC 표준 블록 암호이다[8,9]. 기제안된 SEED에 대한 이론적인 공격은 7-라운드 축소된 버전에 대한 차분 공격이 유일하고, 이 공격의 공격 복잡도는 2^{127}이다[12].

SEED의 DFA에 대한 안전성은 2004년에 처음 제안되었다[1,13]. 이 공격은 라운드 16의 왼쪽 64-비트 입력값에 오류를 주입하여 모두 0이나 혹은 공격자가 알고 있는 값으로 강제로 초기화하여 나오는 결과값으로부터 라운드 16의 라운드 키를 찾아내는 공격이다.

본 논문에서는 [1,13]에서 제안된 공격과 달리, 특정 레지스터에 일시적인 1-비트 오류를 주입한다는 가정을 이용한다. 즉, SEED의 라운드 함수 F의 내부 함수 G 함수가 반복 수행하도록 설계된 경우, 라운드 16의 마지막 G 함수의 입력 레지스터에 1-비트 랜덤 오류를 발생시킨다. 그 결과, 평균 15개의 오류를 주입하면 라운드 16의 라운드 키를 간단한 산술 연산 2^{32}번으로 복구할 수 있음을 보인다. 이 공격을 이용하여, 일반적인 PC 환경에서 수 초 내에 키를 찾을 수 있다. 또한 라운드 15에 같은 방법을 적용한다면, 마스터 키를 약 30개의 오류 주입으로 수 초 내에 찾을 수 있다.

본 논문은 다음과 같이 구성되어 있다. 먼저, 2장에서는 블록 암호 SEED의 구조를 소개한다. 3장에서는 라운드 16의 특정 레지스터에 1-비트 오류 주입을 가정한 DFA를 제안한 후, 4장에서 라운드 키로부터 마스터 키를 복구하는 방법을 제안한다. 마지막으로 5장에서 결론을 맺는다.

II. 블록 암호 SEED

128-비트 블록 암호 SEED는 [그림 1]과 같이 128-비트 마스터 키를 사용하고 16-라운드 Feistel 구조를 갖는다. 128-비트 평문 P를 (L_0, R_0), 라운드 i의 라운드 키를 RK_i라고 한다면 (L_i, R_i)는 라운드 i의 출력값이다. 그러면 암호문은 라운드 16의 출력값인 (L_{16}, R_{16})이 된다. 여기서 L_i, R_i, RK_i는 64 비트이다. 또한, 각각의 64-비트 값은 다음과 같이 2개의 32-비트 값으로 표기한다.

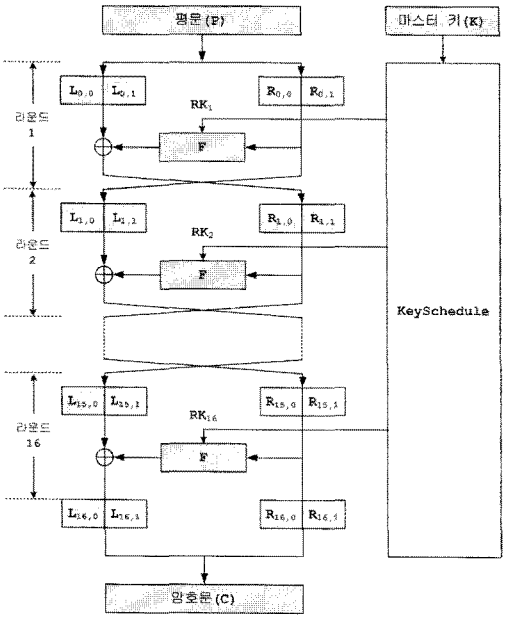
- $L_i = (L_{i,0}, L_{i,1})$, $R_i = (R_{i,0}, R_{i,1})$.
- $RK_i = (RK_{i,0}, RK_{i,1})$.

SEED의 라운드 함수 F는 [그림 2]와 같이 64-비트의 입출력을 갖는 함수로 변형된 3-라운드 MISTY 구조이다. 라운드 함수 F에 사용된 G 함수는 [그림 3]과 같은 32-비트의 입출력을 갖는 함수로 차분 및 선형 공격의 측면에서 좋은 특성을 지닌 함수이고, 라운드 키가 사용되지 않는다. 여기서 덧셈은 법 2^{32}에서의 덧셈을 의미한다. 또한, G 함수는 다음과 같이 표기된다.

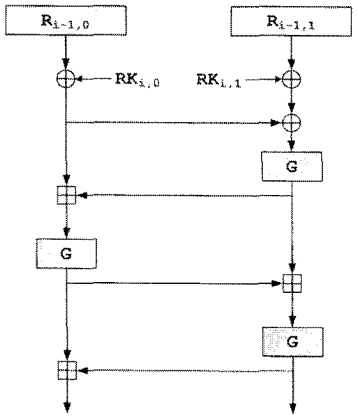
$$G(X) = DL \circ SL(X).$$

여기서 SL은 4개의 8-비트 값 (X_3, X_2, X_1, X_0)를 병

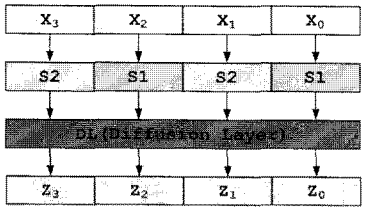
렬적으로 S-box 연산을 하는 것이고, 2개의 S-box $S1$과 $S2$가 사용된다. 그리고 DL은 diffusion layer로서 32×32 이진 행렬이다.



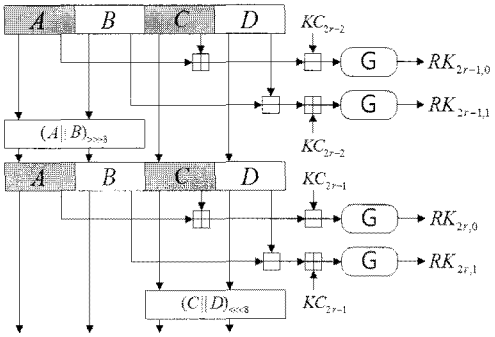
〔그림 1〕 SEED 전체 구조도



〔그림 2〕 라운드 함수 F



〔그림 3〕 G 함수



〔그림 4〕 키스케줄

SEED의 키스케줄은 〔그림 4〕와 같이 128-비트 비밀키 $K=(A,B,C,D)$를 64 비트씩 나누어 이들을 교대로 8-비트 로테이션 연산을 수행한 후, 결과의 4 워드들에 대한 간단한 산술 연산과 G 함수를 적용하여 라운드 키를 생성한다. 먼저, A와 C가 덧셈 연산 후 다시 각 라운드 상수 KC와 다시 뺄셈 연산 후 G 함수 출력으로 처음 32-비트 라운드 키를 생성하고, B와 D가 뺄셈 연산 후 라운드 상수 KC와 덧셈 연산 한 후 G 함수 출력으로 다음 32-비트 라운드 키를 생성한다. 또한 각 32-비트 레지스터 A, B, C, D는 홀수 라운드에서는 앞의 A, B 부분이 묶여서 8-비트 로테이션 연산을 수행하고, 짝수 라운드에서는 C, D 부분이 묶여서 8-비트 로테이션 연산이 수행된다.

III. SEED에 대한 DFA

본 절에서는 SEED에 대한 DFA를 소개한다. 우선 라운드 16의 특정 레지스터에 오류 주입을 가정한 후, 라운드 16의 라운드 키를 복구하는 방법을 소개한다.

3.1 오류 주입 가정

블록 암호에 대한 DFA의 일반적인 공격 가정은 특정 레지스터에 오류를 주입하는 것이다. Feistel 구조를 갖는 블록 암호의 경우, 마지막 라운드의 입력 레지스터에 오류를 주입한다. 그래서 오류가 발생하지 않은 알고리즘을 이용하여 얻은 결과값과 오류가 발생한 알고리즘을 이용하여 얻은 결과값의 차분을 이용하여, 오류가 주입된 active S-박스의 위치를 찾아서 해당 active S-box에 관여한 부분키의 후보를 찾는다. 이 과정을 반복 수행하여 가능한 라운드 키의 후

보를 최대한 줄인 후 라운드 키를 복구한다.

SEED의 라운드 함수 F의 구조는 라운드 함수 내에 G 함수가 3회 반복되는 구조이다. 이러한 구조를 갖는 블록 암호를 하드웨어 등으로 구현하는 경우 라운드 함수 F를 한 번에 구현하기에는 무리가 따르므로, 라운드 함수 내의 G 함수만을 구현하여 반복적으로 수행하도록 설계한다. 따라서 일반적인 블록 암호에서 라운드 함수의 입력 레지스터에 오류를 주입하는 가정과 마찬가지로, SEED의 경우 G 함수의 입력 레지스터에 오류를 주입하는 가정은 현실적으로 가능하다. 본 논문에서는 라운드 16의 라운드 함수에서 3번의 G 함수 입력 중 마지막 G 함수의 입력 레지스터에 1-비트 오류를 주입하는 것을 가정한다.

3.2 라운드 16의 라운드 키 복구 방법

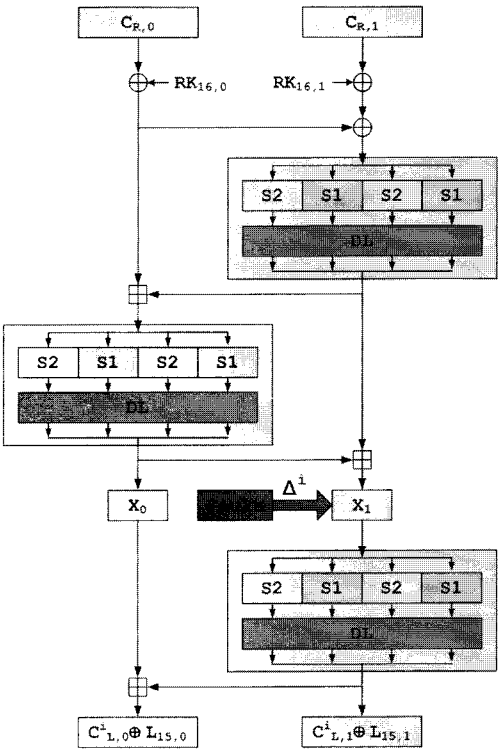
오류 발생 여부에 따라 평문·암호문 쌍을 다음과 같이 표기한다. 여기서, $C_L=(C_{L,0},C_{L,1})$, $C_R=(C_{R,0},C_{R,1})$으로 표기한다.

- (P,C): 오류가 발생하지 않은 알고리즘을 이용하여 얻은 평문·암호문 쌍 $(C=(C_L,C_R))$.
- (P,C^i): i번째 오류가 발생한 알고리즘을 이용하여 얻은 평문·암호문 쌍 $(C^i=(C_L^i,C_R^i))$.

모든 C^i에 대한 라운드 16의 라운드 함수 F의 입력값은 [그림 1]에 의해 $C_R^i=C_R$이고 C_L^i의 값만 오류에 의해 달라진다. 라운드 16의 라운드 함수 F는 [그림 5]와 같다. 여기서 Δ^i는 i번째 오류를 의미한다. 즉, i번째 오류 주입 후 3번째 G 함수의 입력값 $X_1^i=X_1\oplus\Delta^i$이다. 오류가 발생하지 않은 3번째 G 함수의 출력값은 $C_{L,1}\oplus L_{15,1}$이고, 오류 Δ^i가 발생하였을 경우의 출력값은 $C_{L,1}^i\oplus L_{15,1}$이다. 이 두 값으로부터 $L_{15,1}$의 실제 값은 모르지만 두 출력값의 차분 γ^i는 다음과 같다.

$$\gamma^i=(C_{L,1}\oplus L_{15,1})\oplus(C_{L,1}^i\oplus L_{15,1})=C_{L,1}\oplus C_{L,1}^i.$$

$G(X)=DL\circ SL(X)$에서 DL^{-1}는 선형이므로, 라운드 16의 마지막 G 함수의 출력 차분 γ^i를 이용하여 $\beta^i=(\beta_3^i,\beta_2^i,\beta_1^i,\beta_0^i)=DL^{-1}(\gamma^i)$는 쉽게 계산된다. 또한 1-비트 오류가 발생한다고 가정했으므로, 32-비트 β^i



[그림 5] 라운드 16의 오류 주입

중 한 바이트만 0이 아니고 나머지 세 바이트는 0이 될 것이다. 이는 S-box의 출력 차분을 알 수 있음을 의미하고, 0이 아닌 입력 차분 바이트는 한 비트만 1이므로 $2^j(0\le j\le 7)$임을 의미한다. 따라서 S-box의 차분 테이블을 다음과 같이 미리 선계산 하면 쉽게 X의 후보를 찾을 수 있다. 여기서 입력 차분 ID는 8개이고 출력 차분 OD는 255개이다. SEED의 S-box 설계 특성에 의해 각 테이블의 원소의 개수는 0, 2, 4 중 하나이다. 또한 하나의 ID에 대해, 원소의 개수가 0개인 것이 128개 존재하고 2인 것이 126개, 4인 것이 1개 존재한다.

- $S1[ID][OD]=\{x|S1(x\oplus ID)\oplus S1(x)=OD\}$.
- $S2[ID][OD]=\{x|S2(x\oplus ID)\oplus S2(x)=OD\}$.

따라서 평균적으로 1개의 후보가 나온다고 볼 수 있고, ID가 8개 가능하므로 평균 8개의 후보가 존재한다. 이는 하나의 오류를 통해 X_1의 값 중 한 바이트의 값의 후보를 평균 8개 얻을 수 있음을 의미한다. 그러나 이 8개의 후보 중 어느 것이 정확한 값인지는 하나의 오류를 통해서는 알 수 없다. 하지만 오류를 더

주입하여 같은 바이트의 다른 위치에 오류가 발생시킨다면 이 오류에 대한 X_1의 한 바이트의 후보와 이전 후보 중 겹치는 것은 거의 유일하므로 정확한 값을 얻을 수 있게 된다. 실험 결과, 8~20개의 오류를 이용하면 X_1의 값을 거의 유일하게 결정할 수 있는 것으로 나타났다.

X_0는 앞에서 복구한 X_1와 덧셈-XOR 차분의 특성을 이용하면 구할 수 있다. 오류가 발생하지 않았을 경우 라운드 16의 왼쪽 32-비트 출력값은 $C_{L,0} \oplus L_{15,0}$이고, 오류 Δ^i가 발생하였을 경우의 출력값은 $C^i_{L,0} \oplus L_{15,0}$이다. 따라서 라운드 16의 왼쪽 32-비트 출력값의 XOR 차분 δ^i는 다음과 같다.

$$\delta^i = (C_{L,0} \oplus L_{15,0}) \oplus (C^i_{L,0} \oplus L_{15,0}) = C_{L,0} \oplus C^i_{L,0}.$$

또한 앞서 구한 X_1을 이용하면 $L_{15,1} = G(X_1) \oplus C_{L,1}$이므로 다음과 같이 쉽게 계산된다. $G(X_1)$의 값을 t라고 하면, Δ^i에 의해 마지막 G 함수의 출력값은 $t \oplus C_{L,1} \oplus C^i_{L,1}$가 된다. 이를 이용하여 식 (1)을 얻을 수 있다.

$$(X_0 + t) \oplus (X_0 + (t \oplus C_{L,1} \oplus C^i_{L,1})) = \delta^i. \quad (1)$$

식 (1)은 덧셈과 XOR의 결합에서 발생하는 차분식을 나타낸 것이다. 이에 대한 연구는 [11]에 잘 정리되어 있다. [11]의 표기법을 사용하여 식 (1)이 성립할 확률을 $DP^+((0, C_{L,1} \oplus C^i_{L,1}) \rightarrow \delta^i)$로 표기한다. 주어진 t, $t \oplus C_{L,1} \oplus C^i_{L,1}$, δ^i에 대해 만족하는 X_0의 후보를 계산할 수 있다. 가능한 X_0의 후보는 2^{32}개이므로 하나의 오류에 대해 만족하는 후보를 간단한 산술 연산으로 구할 수 있다. 그러나 각각의 δ^i에 대해 식 (1)이 성립하는 X_0의 후보의 개수는 $DP^+((0, C_{L,1} \oplus C^i_{L,1}) \rightarrow \delta^i)$의 확률에 따라 많이 존재할 수 있다. 그러므로 다른 오류를 이용하여 후보의 개수를 더 줄일 수 있다. 실험 결과, 8개의 오류를 사용하면 평균 4개(최소 2개, 최대 32개)의 후보를 얻을 수 있는 것으로 나타났다.

위의 방법을 통해 X_0를 계산한 후, $L_{15,0} = X_0 \oplus G(X_1)$도 쉽게 계산 가능하다. 라운드 16의 라운드 키 RK_{16}을 식 (2)와 식 (3)에 의해 순차적으로 구할 수 있다.

$$RK_{16,0} = C_{R,0} \oplus (G^{-1}(X_0) - (X_1 - X_0)). \quad (2)$$

$$RK_{16,1} = G^{-1}(X_1 - X_0) \oplus C_{R,1} \oplus RK_{16,0}. \quad (3)$$

위에서 설명한 SEED에 대한 DFA의 공격 과정을 요약하면 다음과 같다.

1. [오류가 발생하지 않은 데이터 수집] 오류가 발생하지 않은 알고리즘을 이용하여 평문 P에 대한 암호문 $C=(C_L, C_R)$을 얻는다.
2. [오류가 발생한 데이터 수집] 라운드 16의 3번째 G 함수의 입력 레지스터에 1-비트 오류 Δ^i를 n번 주입한 후, 해당 오류에 대한 암호문 $C^i = (C^i_L, C^i_R)$를 얻는다 $(i=1,\cdots,n)$.
3. [X_1의 후보값 저장] S-box의 입출력 차분 테이블을 이용하여 X_1의 후보값들을 저장한다.
4. [X_0의 후보값 저장] 식 (1)을 이용하여 X_0의 후보값들을 계산하고 저장한다.
5. [라운드 키 RK_{16}의 후보 찾기] 단계 3과 4를 통해 얻은 후보 $X=(X_0, X_1)$에 대해, 식 (2)와 식 (3)을 이용하여 RK_{16}의 후보값들을 계산한다.

3.3 필요한 오류 주입 수 및 계산 복잡도 분석

단계 1과 단계 2는 데이터 수집 단계이므로 필요한 계산량은 거의 없다. 단계 3에서 X_1을 유일하게 복구하려면, 라운드 16의 3번째 G 함수에서 4개의 S-box가 서로 다르게 최소한 2번 이상 active가 되도록 오류가 발생해야만 한다. 만약 공격자가 오류의 위치를 지정할 수 있다면, 8번의 오류 주입으로 X_1의 값을 거의 유일하게 결정할 수 있다. 하지만 랜덤한 위치에 오류가 주입된다면 같은 비트에 오류가 발생할 수도 있고, 4개의 S-box 중 한 곳에 몰릴 수도 있으므로 더 많은 수의 오류가 주입되어야 할 것이다.

실험 결과, 평균 15개의 오류가 주입되었을 경우 모든 S-box가 서로 다르게 2번 이상 active가 되는 것으로 나타났다. 단계 3에서는 S-box의 입력 차분에 대한 출력 차분 테이블을 미리 구성할 수 있으므로 무시할 정도의 계산량으로 X_1을 복구할 수 있다.

단계 4는 단계 3에서 얻은 X_1의 후보에 대해 모든 가능한 2^{32}개의 X_0에 대한 간단한 산술 연산을 통해 후보 데이터를 얻을 수 있다. 이는 일반 PC에서 수 초 내에 결과를 얻을 수 있고, 단계 3에서와는 달리 덧셈과 XOR의 차분 특성으로 인해 2~32개의 후보가 존재한다. 실험 결과, 평균 4개의 후보가 발생하는 것으

로 나타났다.

단계 3과 단계 4에서 얻은 후보들(평균 4개)에 대해 식 (2)와 식 (3)을 이용하면 쉽게 같은 개수의 RK_{16}의 후보를 계산할 수 있다.

IV. 라운드 키로부터 마스터 키 복구 방법

앞 절에서 소개한 방법을 이용하여 라운드 16의 라운드 키 $RK_{16}=(RK_{16,0},RK_{16,1})$의 값을 복구한 후, 동일한 방법을 라운드 15에 적용하면 같은 계산 복잡도로 라운드 15의 라운드 키 $RK_{15}=(RK_{15,0},RK_{15,1})$을 복구할 수 있다. 두 라운드의 라운드 키를 구한 후, SEED의 키스케줄 특성을 이용하여 쉽게 마스터 키를 복구할 수 있다.

SEED의 키스케줄에서, 마스터 키는 최초 4개의 32-비트 레지스터 (A,B,C,D)에 입력된 후 16개의 라운드 키가 생성된 뒤 최초의 상태로 복원된다 ([그림 4] 참조). 만약 라운드 16의 라운드 키를 생성하기 전의 (A,B,C,D)의 값을 찾으면, 마스터 키 K는 레지스터 C와 D의 값을 8-비트 로테이션 연산으로 쉽게 계산이 가능하다. 즉, $K=A\|B\|(C\|D)_{\ll 8}$.

따라서 라운드 15, 16의 라운드 키 $RK_{15}=(RK_{15,0},RK_{15,1})$와 $RK_{16}=(RK_{16,0},RK_{16,1})$는 다음과 같이 계산된다. 여기서 KC_{15}와 KC_{14}는 32-비트 상수이고 $A'\|B'=(A\|B)_{\ll 8}$이다.

- $RK_{15,0}=G(A'+C-KC_{14})$.
- $RK_{15,1}=G(B'-D+KC_{14})$.
- $RK_{16,0}=G(A+C-KC_{15})$.
- $RK_{16,1}=G(B-D+KC_{15})$.

x와 y를 레지스터 C와 D의 후보라고 한다면, x와 y에 대한 레지스터 A와 B의 후보는 다음과 같이 계산된다.

- $A(x)=G^{-1}(RK_{16,0})-x+KC_{15}$.
- $B(y)=G^{-1}(RK_{16,1})+y-KC_{15}$.
- $A'(x)=G^{-1}(RK_{15,0})-x+KC_{14}$.
- $B'(y)=G^{-1}(RK_{15,1})+y-KC_{14}$.

위의 식에서 $A'(x)$의 상위 24 비트는 $A(x)$의 하위 24 비트와 같고 $B'(x)$의 상위 24 비트는 $B(x)$의 하위 24 비트와 같으므로 식 (4)와 식 (5)가 성립된다.

$$A(x)\&0x00ffffff=(A'(x)\&0xffffff00)_{\gg 8}. \quad (4)$$

$$B(y)\&0x00ffffff=(B'(y)\&0xffffff00)_{\gg 8}. \quad (5)$$

각각의 x의 후보에 대해, 식 (4)의 판별식으로 약 2^8개의 후보를 얻을 수 있다. 마찬가지로 식 (5)의 판별식으로 약 2^8개의 후보를 얻을 수 있다. 총 2^{16}개의 후보는 A'의 하위 8 비트와 B의 상위 8 비트가 같고 A의 상위 8 비트와 B'의 하위 8 비트가 같다는 성질을 이용하면 쉽게 판별이 가능하므로 마스터 키의 복구가 가능하다. 이러한 마스터 키의 복구는 x와 y의 값을 독립적으로 수행 가능하므로 약 2^{32}번의 산술 연산을 2회 수행하는 복잡도로 공격이 가능하다. 이는 일반적인 PC에서 수 초 내에 계산 가능한 복잡도이다.

V. 결 론

본 논문에서는 SEED에 대한 DFA를 제안하였다. 본 논문에서 제안한 공격을 이용하여 라운드 16과 라운드 15의 3번째 G 함수의 입력 레지스터에 1-비트 오류를 약 30번 주입하여 마스터 키를 수 초 내에 복구할 수 있음을 보였다. 본 공격을 보다 일반화하여 첫 번째, 두 번째 G 함수의 입력 레지스터에 오류를 주입하거나, 라운드 함수의 입력 레지스터에 오류를 주입할 경우에 대한 분석 및 1-바이트 오류 주입에 대한 분석은 향후 연구 과제이다.

참고문헌

[1] 하재철, 김창균, 문상재, 박일환, "SEED에 대한 오류 분석 공격", 한국정보보호학회 동계정보보호학술대회 논문집(CISC-W'04), pp. 39-44, 2004년 12월.

[2] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystem," Journal of Cryptology, Vol. 4, No. 1, pp. 3-72, Springer-Verlag, Jan. 1991.

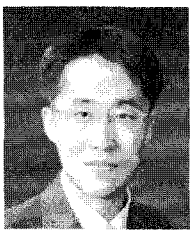
[3] E. Biham and A. Shamir, "Differential Faut Analysis of Secret Key Cryptosystems," Crypto'97, LNCS 1294, pp. 513-525, Springer-Verlag, 1997.

[4] D. Boneh, R. DeMillo and R. Lipton, "On the importance of checking cryptographic protocols for faults," Eurocrypt'97, LNCS

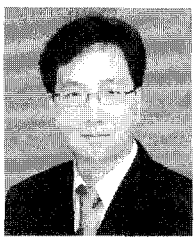
1233, pp. 37-51, Springer-Verlag, 1997.
[5] P. Dusart, G. Letourneux, and O. Vivolo, "Differntial Fault Attack on AES," ACNS'03, LNCS 2846, pp. 293-306, Springer-Verlag, 2003.
[6] L. Hemme, "A Differential Faut Aanalysis against Early Rounds of (Triple)-DES," CHES'04, LNCS 3156, pp. 254-267, Springer-Verlag, 2004.
[7] P. Kocher, "Timing attacks on implementation of Diffie-Hellman," Crypto'96, LNCS 1109, pp. 104-113, Springer-Verlag, 1996.
[8] Korea Information Security Agencry, "A Design and Analysis of 128-bit Symmetric Block Cipher SEED," 1999. Available at http://www.kisa.or.kr/kisa/seed/jsp/seed_1010.jsp
[9] ISO/IEC 18033-3, "Information technology -Security techniques-Encryption algorithms - Part 3: Block Ciphers," 2005.
[10] W. Li, D. Gu, and J. Li, "Differential Fault Analysis on the ARIA Algorithm," Information Science, Vol. 178, Issue. 19, pp. 3727-3737, Elsevier, 2008.
[11] H. Lipmaa and S. Moriai, "Efficient Algorithms for Computing Differential Properties of Addition," FSE'01, LNCS 2355, pp. 336-350, Springer-Verlag, 2002.
[12] H. Yanami and T. Shimoyama, "Differential Cryptanalysis of a Reduced-Round SEED," SCN'02, LNCS 2576, pp. 186-198, Springer-Verlag, 2002.
[13] H. Yoo, C. Kim, J. Ha, S. Moon, and I. Park, "Side Channel Cryptanalysis on SEED," WISA'04, LNCS 3325, pp. 411-424, Springer-Verlag, 2004.

〈著者紹介〉

정 기 태 (Kitae Jeong) 학생회원
2004년 2월: 고려대학교 수학과 학사
2006년 2월: 고려대학교 정보보호대학원 석사
2006년 3월~현재: 고려대학교 정보경영공학전문대학원 박사과정
〈관심분야〉 블록 암호, 스트림 암호의 분석 및 설계



성 재 철 (Jaechul Sung) 종신회원
1997년 8월: 고려대학교 수학과 학사
1999년 8월: 고려대학교 수학과 석사
2002년 8월: 고려대학교 수학과 박사
2002년 8월~2004년 1월: 한국정보보호진흥원 선임연구원
2004년 2월~현재: 서울시립대학교 수학과 부교수
〈관심분야〉 암호 알고리즘 설계 및 분석



홍 석 희 (Seokhie Hong) 종신회원
1995년 2월: 고려대학교 수학과 학사
1997년 2월: 고려대학교 수학과 석사
2001년 8월: 고려대학교 수학과 박사
1999년 8월~2004년 2월: (주) 시큐리티 테크놀로지스 선임연구원
2003년 8월~2004년 2월: 고려대학교 정보보호기술연구센터 선임연구원
2004년 4월~2005년 2월: K.U. Leuven, ESAT/SCD-COSIC 박사후연구원
2005년 3월~2008년 8월: 고려대학교 정보보호대학원 조교수
2008년 9월~현재: 고려대학교 정보경영공학전문대학원 부교수
〈관심분야〉 대칭키·공개키 암호 분석 및 설계, 컴퓨터 포렌식