SEED에 대한 오류 분석 공격

하재철*, 김창균**, 문상재**, 박일환***

나사렛대학교 정보과학부, 경북대학교 전자공학과, 국가보안기술연구소

A Fault Analysis Attack on SEED

JaeCheol Ha*, ChangKyun Kim**, SangJae Moon**, IlHwan Park***

*Division of Information Science, Korea Nazarene Univ.

**Dept. of Electronics Engineering, Kyungpook National Univ.

***National Security Research Institute

요 약

오류분석 공격은 암호시스템에 오류를 주입한 후 그 출력 결과를 분석하여 비밀 키를 찾아내는 물리적 공격 방법으로서 RSA, ECC를 포함한 공개 키 시스템을 비롯하여 DES, AES와 같은 대칭 키 암호시스템에도 공격이 시도되고 있다. 본 논문에서는 기존 DES 공격에 사용된 오류 주입의 가정만 있으면 국내 표준 블록 암호 알고리듬인 SEED 역시 오류 주입 공격이 가능함을 증명한다. 또한, 오류 주입 공격에 의해 SEED의 라운드 키 두개만 공격되면 원 암호 키가 모두 노출될 수 있음을 검증한다.

I. 서 론

오류 분석 공격(fault analysis attack) 혹은 오류 주입 공격(fault insertion attack)이라 불리는 물리적 공격 방법은 RSA 암호 방식에 대한 공격 방법으로 처음 소개되었다[1,2]. 이러한 오류 분석 공격은 하드웨어의 예상치 못한 결함이나 넓게는 소프트웨어적인 버그 등에 의해서 오류가 발생할 경우에 가능한 공격임을 가정한다. 특히, 단 한번의 오류 주입으로 비밀키를 알아낼 수 있는 CRT 기반의 RSA 암호시스템에 대한 오류 공격이 가장 강력하다고 알려져 있다[3,4,5,6]. 현재까지 오류 주입 공격은 RSA, ECC와 같은 공개키 암호시스템은 물론 DES나 AES와 같은 비밀키 암호시스템에 대해서 지속적이고 다양한 방법으로 이루어지고 있다[7,8].

SEED는 대칭 암호 키를 사용하여 블록 단위로 메시지를 암호화하여 처리하는 블록 암호 알고리즘이다[9]. 본 논문에서는 1999년 국내 표준으로 지정된 블록 암호 알고리듬 SEED에 대해 오류 주입 공격을 시도해 보고 위협적인 요소가 있음을 증명한다. 공격에 사용된 가정들은 기존의 DES나 AES 공격에서 세웠던 가정을 크게 벗어나지 않는다. 또한, SEED에서 일부 라운드 키가 공격되면 원래 암호 키를 찾아낼 수 있음을 이론적으로 검증한다.

II. SEED의 개요

SEED는 블록 단위로 메시지를 처리하는 대칭 키 블록 암호알고리즘으로서 고정된 128비트 평문을 같은 길이의 128비트 암호문으로 바꾸는 암호 알고리듬이다. 많은 블록 알고리듬이 Feistel 구조로 설계되어 있는데 Feistel 구조란 각각 t비트인 L_0, R_0 블록으로 이루어진 2t비트 평문 블록 (L_0, R_0)이 r라운드($r \geq 1$)를 거쳐 암호문 (L_r, R_r)으로 변환되는 반복 구조를 말한다. 암호 알고리듬의 라운드 함수란 원 암호 키 K로부터 유도된 각 서브키 K_i (또는, 라운드 키)를 입력으로 $L_i = R_{i-1}$, $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$로 바꾸어 주는 함수를 말한다.

SEED 알고리즘은 이와 같은 Feistel 구조로 되어 있으며 128비트의 평문 블록단위당 128비트 키로부터 생성된 16개의 64비트 라운드 키를 입력으로 사용하여 총 16라운드를 거치면서 128비트 암호문 블록을 출력한다.

1) SEED의 전체 구조

그림 1은 SEED 알고리즘의 전체구조를 도식화한 것이다.

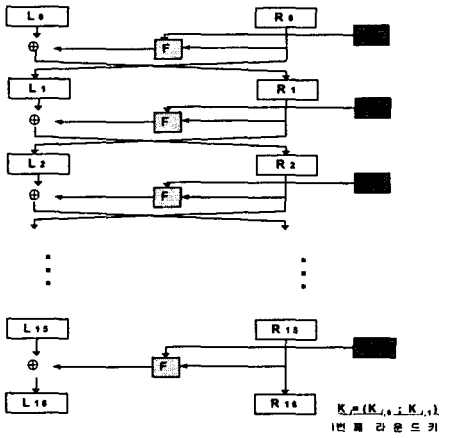

그림 1 : SEED 전체 구조도

SEED에 사용되는 F 함수는 수정된 64비트 Feistel 형태로 구성된다. F 함수는 각 32비트 블록 2개(C, D)를 입력으로 받아, 32비트 블록 2개(C′, D′)를 출력한다. 즉, 64비트 블록(C, D)와 64비트 라운드 키 $K_i=(K_{i,0};K_{i,1})$를 F 함수의 입력으로 받아 64비트 블록(C′, D′)을 출력한다. 그림 2는 i번째 라운드의 F 함수의 구조를 나타낸 것이다.

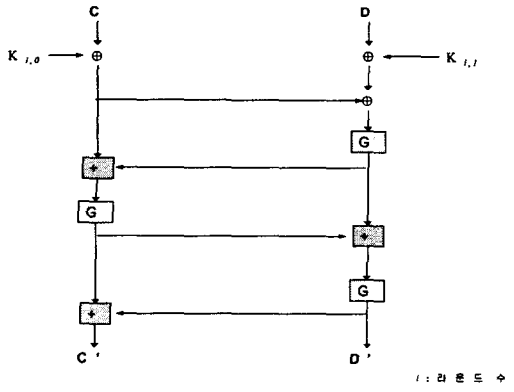

그림 2 : F-함수 구조도

F 함수에 사용된 함수 중 중요한 것이 G 함수인데 G 함수는 다음과 같은 연산을 수행하며 이를 나타낸 것이 그림 3이다. G 함수에 사용된 두 종류의 S-box는 8비트 입력(즉, 0~255)을 받아 8비트 출력(즉, 0~255)을 내는 함수로서 비선형성을 제공하는 중요한 역할을 하는 함수이다.

$$Y_3=S_2(X_3),\ Y_2=S_1(X_2),$$

$$Y_1=S_2(X_1),\ Y_0=S_1(X_0),$$

$$Z_3=(Y_0\&m_3)\oplus(Y_1\&m_0)\oplus(Y_2\&m_1)\oplus(Y_3\&m_2)$$
$$Z_2=(Y_0\&m_2)\oplus(Y_1\&m_3)\oplus(Y_2\&m_0)\oplus(Y_3\&m_1)$$
$$Z_1=(Y_0\&m_1)\oplus(Y_1\&m_2)\oplus(Y_2\&m_3)\oplus(Y_3\&m_0)$$
$$Z_0=(Y_0\&m_0)\oplus(Y_1\&m_1)\oplus(Y_2\&m_2)\oplus(Y_3\&m_3)$$
$$(m_0=0\text{xfc}, m_1=0\text{xf3},\ m_2=0\text{xcf},\ m_3=0\text{x3f})$$

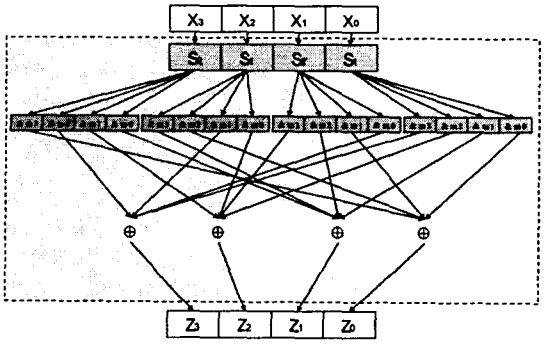

그림 3 : G 함수

2) 라운드 키 생성과정

SEED에서의 라운드 키 생성과정 중 첫 번째는 128비트 암호 키를 64비트씩 좌우로 나누어 이들을 교대로 8비트씩 좌/우로 회전 이동한다. 그 후 출력 결과의 4워드들에 대한 간단한 산술연산과 함께 G 함수를 사용하여 각각의 라운드 키를 생성한다. 각 라운드에서 라운드 키는 생성하는 방법을 구체화하면 아래와 같다.

① 128비트 입력키를 32비트씩 4개로 나눔

(A, B, C, D)

② $K_{1,0}$ = G(A+C-KC_0)

$K_{1,1}$ = G(B-D+KC_0)

(단, KC_0 : 1 라운드 상수)

③ A|| B =(A|| B)>>8

④ $K_{2,0}$ = G(A+C-KC_1)

$K_{2,1}$ = G(B-D+KC_1)

(단, KC_1 : 2 라운드 상수)

⑤ C|| D =(C|| D)<<8

⑥ $K_{3,0}$ = G(A+C-KC_2)

$K_{3,1}$ = G(B-D+KC_2)

(단, KC_2 : 3 라운드 상수)

⑦ 16라운드 키를 생성할 때까지 위 과정 반복

즉, 주어진 128비트 암호 키 K=A|| B || C || D를 32비트 레지스터 A, B, C, D로 나눈다. 각 라운드 키 K_i=($K_{i,0}$; $K_{i,1}$)는 다음과 같은 방식으로 생성한다:

```
for( i=1; i<=16; i++) {
    Ki,0 ← G(A+C-KCi);
    Ki,1 ← G(B-D+KCi);
    if( i%2==1 ) A||B ← (A||B)>>8;
    else         C||D ← (C||D)<<8;
}
```

이 과정을 도시한 것이 그림 4이다.

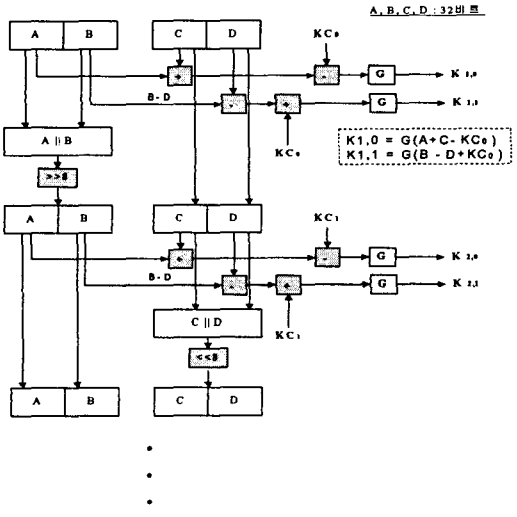

그림 4 : 라운드 키 생성과정 구조도

Ⅲ. SEED에 대한 오류 주입 공격

암호 알고리듬을 공격하는데 필요한 첫 번째 가정은 암호 알고리듬을 알고 있는 공격자가 평문에 대한 암호문을 얻을 수 있고 역으로 암호문에 대한 복호문을 얻을 수 있다는 것이다. 공격의 주요 내용은 15라운드 후의 왼쪽 64비트에 오류를 주입하여 모두 "0"이나 혹은 공격자가 알고 있는 값으로 강제로 초기화하여 나오는 결과 값으로부터 암호 키를 찾아내는 것이다. 이 가정은 Biham과 Shamir의 DES 공격에서 영구적 오류를 발생시켜 사용할 수 있는 방법이다. 여기에서는 레지스터에 들어오거나 나가는 데이터 라인을 차단하거나 비트별로 물리적 방법으로 파괴하는 오류를 가정할 수 있다.

1) 16라운드 키에 대한 공격

오류 주입 공격은 15번째 라운드의 L_{15}를 모두 0이나 혹은 알고 있는 값으로 만들 수 있다는 것으로 시작한다. 1차적인 공격의 목표는 16라운드 키 $K_{16,0}$ 와 $K_{16,1}$을 찾는 것이다.

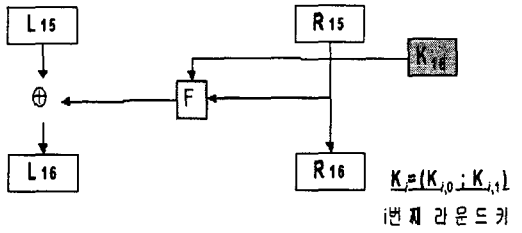

그림 5 : 16 라운드 키의 공격

☐ 1단계 : 공격자는 L_{15}를 모두 알고 있는 값으로 초기화하고 그 결과를 얻었다고 가정하자. L_{16}과 R_{16}은 이미 알고 있으므로 F 함수의 입력과 출력을 알 수 있다. 따라서 16라운드 키를 찾는 문제는 F 함수의 입·출력을 알 때 라운드 키를 찾아내는 문제로 귀결된다.

☐ 2단계 : 그림 2의 F 함수에서 입·출력 C, D, C', D'을 알 때 키 $K_{16,0}$ 와 $K_{16,1}$을 찾기 위해서는 G함수의 역함수 G^{-1}( )를 찾으면 된다. 즉, F 함수에서 mod 2^{32}에 관한

덧셈의 역함수는 뺄셈으로 간단히 처리하므로 G의 출력을 알고 입력을 찾아낼 수 있다면 최종적으로 16라운드 키를 찾을 수 있다.

□ 3단계 : G 함수의 역함수를 찾아보자. 즉, G 함수의 출력을 알고 입력을 찾아보는 것인데 S 함수의 역함수는 함수 테이블에 의해 쉽게 찾을 수 있다. 결국, $Z_3|Z_2|Z_1|Z_0$는 S함수의 출력 $S_2(X_3)|S_1(X_2)|S_2(X_1)|S_1(X_0)$에 의해 결정되는데 공격 핵심은 Z_0의 마지막비트 Z_{00}가 $S_{20}(X_3)|S_{10}(X_2)|S_{20}(X_1)|S_{10}(X_0)$의 출력값에 의해 결정된다는 점이다. 물론 S 함수의 각 비트는 상수 값과 & 연산을 수행한 후 다시 ⊕에 의해 결정된다. 따라서 각 S박스의 출력 중 각 워드의 최하위 비트 4비트 $S_{20}(X_3)|S_{10}(X_2)|S_{20}(X_1)|S_{10}(X_0)$만 임의로 조사하면 Z_{00}값을 만드는 비트를 구할 수 있다. 이때 올바른 Z_{00}를 출력하는 4비트 값의 개수는 모두 8종류이다. 나머지 8종류는 올바른 Z_{00}을 출력할 수 없게 된다. 또, 선택된 8종류의 4비트 중에서 Z_{10}을 만드는 $S_{20}(X_3)|S_{10}(X_2)|S_{20}(X_1)|S_{10}(X_0)$ 값을 찾으면 입력값의 범위가 4개로 줄어든다. 이어서 Z_{20}을 만드는 것은 2개, Z_{30}를 만드는 것을 구하면 Z_{00}, Z_{10}, Z_{20}, Z_{30}을 만드는 하나의 $S_{20}(X_3)|S_{10}(X_2)|S_{20}(X_1)|S_{10}(X_0)$를 구할 수 있다. 동일한 방법으로 Z값을 모두 알고 있다면 S 함수들의 출력 값을 모두 알 수 있고 S 함수의 출력을 구하면 G 함수의 입력 값을 모두 구할 수 있다. 따라서 G 함수의 역함수를 구할 수 있게 된다. 이와 같이 G 함수의 역함수를 찾아낼 수 있는 가장 큰 이유는 Z_{ij}의 j번째 비트는 $S_{kj}(X_l)$의 j번째 4비트만으로 결정된다는 점이다. 그리고 이 4비트가 4개의 Z_{ij} 값을 결정하므로 이를 만족하는 입력 값의 범위를 줄일 수 있고 결국 4비트의 S 함수 출력 비트를 구할 수 있다. 최종적으로 G 함수의 출력 4비트로부터 입력 4비트를 구할 수 있어 역함수를 구할 수 있다.

□ 4단계 : G의 역함수를 구할 수 있고 mod 2^{32}에 대한 역함수를 알고 있으므로 F 함수의 입력과 출력을 알면 16라운드 키를 알 수 있다.

2) 라운드 키 차분을 이용한 암호 키 공격

오류를 주입한 후 평문에 대해 암호문을 얻어 16라운드 키를 구할 수 있다고 가정하면 역으로 암호문에 대한 복호문을 얻는 과정에서 1라운드 키를 얻을 수 있다. 이 경우 16라운드 키에 사용된 암호문을 1라운드 키를 찾아내는데 사용할 필요는 없으며 키의 입력 순서가 역순인 점만 이용하면 되므로 임의의 암호문을 복호해도 위에서와 같은 방법으로 1라운드 키를 얻을 수 있다.

다음으로 1라운드 키와 16라운드 키를 알고 있을 때 128 비트의 암호 키를 찾아보자. 그림 4에서 보는 바와 같이 각 라운드 키는 128비트 암호 키를 64비트씩 좌우로 나누어 이들을 교대로 8비트씩 좌/우로 회전 이동한 후, 결과의 4워드들에 대한 간단한 산술연산과 G 함수를 적용하여 생성한다.

그런데 위에서 살펴본 바와 같이 G 함수의 역함수도 구할 수 있으며 ⊞의 역함수는 ⊟이므로 공격자가 알고 있는 1 라운드 키 $K_{1,0}$와 $K_{1,1}$으로부터 (A+C)와 (B-D)를 각각 구할 수 있다. 따라서 A, B, C, D를 8비트씩 분할하여 바이트 단위로 표현하면 다음과 같은 수식을 얻을 수 있다.

$$A + C = G^{-1}(K_{1,0}) + KC_0 = T_{1,0}$$

$$B - D = G^{-1}(K_{1,1}) - KC_0 = T_{1,1}$$

$$A_3||A_2||A_1||A_0 + C_3||C_2||C_1||C_0 = T_{1,0}$$

$$B_3||B_2||B_1||B_0 - D_3||D_2||D_1||D_0 = T_{1,1}$$

그런데 16라운드 키를 만드는데 사용된 왼쪽 64비트 암호 키는 1라운드 키를 만드는데 사용된 것과 동일하다는 사실을 발견할 수 있다. 즉, 16라운드의 왼쪽 64비트 암호 키는 결국 1라운드에 사용된 암호 키를 8비트씩 8번을 오른쪽 쉬프트를 수행한 값이므로 동일할 수 밖에 없다. 또, 16라운드의 오른쪽 64비트 암호는 1라운드에 사용된 암호 키를 8비트씩 7번을 왼쪽으로 쉬프트를 수행한 결과이기 때문에 오른쪽으로 8비트, 즉 1 바이트 쉬프트시킨 것과 동일하다. 이를 수식으로 표현하면 아래와 같다. 여기서 L은 왼쪽 32비트를 의미하며 R은 오른쪽 32비트를 의미한다.

$$A+L((C||D)>>8) = G^{-1}(K_{16,0})+KC_{16} = T_{16,0}$$

$B\text{-}R((C||D)>>8) = G^{-1}(K_{16,1})\text{-}KC_{16} = T_{16,1}$

$A_3||A_2||A_1||A_0 + D_0||C_3||C_2||C_1 = T_{16,0}$

$B_3||B_2||B_1||B_0 - C_0||D_3||D_2||D_1 = T_{16,1}$

따라서 라운드 키를 서로 차분하면 다음과 같이 수식이 성립하게 된다.

$T_{16,1} - T_{1,1} = D_3||D_2||D_1||D_0 - C_0||D_3||D_2||D_1$

$T_{1,0} - T_{16,0} = C_3||C_2||C_1||C_0 - D_0||C_3||C_2||C_1$

이를 그림으로 도시하면 그림 6과 같다.

$T_{16,1} - T_{1,1}$ =

D_3	D_2	D_1	D_0
C_0	D_3	D_2	D_1

$T_{1,0} - T_{16,0}$ =

C_3	C_2	C_1	C_0
D_0	C_3	C_2	C_1

그림 6 : 라운드 키의 차분

여기서 $T_{1,0}$, $T_{1,1}$, $T_{16,0}$, $T_{16,1}$를 알고 있으므로 다음 절차에 의해 위 수식을 만족하는 암호 키 C와 D를 계산할 수 있다. 첫 번째 수식에서 8비트의 D_0를 임의로 정하면 D_1를 구할 수 있고, D_1을 구하면 D_2를 구할 수 있다. 이와 같이 연쇄적 방법으로 C_0를 구하고 두 번째 수식에서 C_1을 구할 수 있다. 결국 두 번째 수식에서 최종적으로 구할 수 있는 D_0는 첫 번째 수식에서 임의로 정한 D_0와 같게 된다. 따라서 위의 두 식을 만족하는 256종류의 C와 D 값을 구할 수 있다.

위에서 보는 바와 같이 C를 구하면 A를 구할 수 있고 D를 구하면 B를 구할 수 있으므로 결국 256종류의 암호 키를 후보 집합을 구할 수 있다. 이 암호 키 후보 집합들은 1라운드 키와 16라운드 키 조건만 만족하는 암호 키 값들이다.

따라서 소프트웨어로 SEED를 구현하여 256종류의 암호 키를 하나씩 대입해 보면 그 중에서 단 하나의 암호 키만이 공격자가 가지고 있는 평문과 암호문 혹은 암호문과 복호문 쌍을 만들 수 있다. 따라서 256번의 공격에 사용된 입·출력문과 암호 키 후보를 사용한 입·출력문을 비교함으로써 사용된 비밀 암호 키를 찾을 수 있다. 이 경우 한 가지 주의할 점은 소프트웨어로 암호 키를 찾을 때 인위적으로 15라운드 후의 왼쪽 값이 우리가 주입한 오류와 같은 값으로 강제로 설정해 주어야 하는데 이 점은 공격을 시도하는데 제한 요소는 아니다.

Ⅳ. 결 론

SEED 암호 알고리듬은 15라운드 후에 왼쪽 64비트에 공격자가 원하는 오류를 주입할 수 있고 평문에 대한 암호문과 암호문에 대한 복호문을 한 쌍씩 가지고 있을 경우 사용자의 암호 키를 완전히 찾아낼 수 있다. 여기서 주입되는 오류는 영구적인 오류이거나 일시적인 오류도 가능하지만 공격자가 알 수 있는 오류 정보를 주입하기만 하면 된다. 결국, 오류 공격에 대한 SEED 암호 알고리듬의 취약점은 G 함수 부분에 있다고 할 수 있으며 G 함수의 입력과 출력을 알고 있는 가정하에서 역으로 라운드 키를 찾을 수 있다는 점이다. 또한 두개의 라운드의 키를 알면 차분 방식을 이용하여 원래 암호 키를 찾을 수 있었다.

이와 같은 오류 공격 방법은 Feistel 구조의 암호 알고리듬에 적용된다. DES 역시 이와 같은 공격에 취약함을 보이고 있으며 SEED 역시 취약하다. 그러나 단순히 Feistel 구조라고 해서 모두 취약한 것은 아니며 결국 사용되는 F 함수가 어느 정도의 강도d를 가지는지를 고려해 보아야 한다. F 함수의 입력과 출력을 알고 라운드 키를 찾을 수 있는 블록 암호 알고리듬은 이 공격에 취약한 구조가 될 수 밖에 없으므로 F 함수 설계시 이점을 고려할 필요가 있다. 또한 라운드 키 일부가 발견되더라도 원래 암호 키는 찾을 수 없는 구조의 키 생성 함수를 사용하는 것이 안전성면에서 우수하다고 할 수 있다. 향후 이러한 오류공격이 실현화 될 가능성에 대비한 다각도의 대응책 연구가 필요하다.

참고문헌

[1] Bellcore Press Release, "New threat model breaks crypto codes," Sept. 1996 or D. Boneh, R.A. DeMillo, and R.J. Lipton, "On the importance of checking cryptographic protocols for faults," *In Advances in Cryptology - EUROCRYPT '97, LNCS 1233,* PP. 37-51, Springer-Verlag, 1997.

[2] Sergei P. Skorobogatov, Ross J. Anderson "Optical Fault Induction Attacks" *in Cryptographic Hardware and Embedded Systems - CHES2002, LNCS 2523,* pp.

2-12, Springer-Verlag, Aug. 2002

[3] M. Joye, A.K. Lenstra, and J.-J.Quisquater, "Chinese remaindering based cryptosystems in the presence of faults," *Journal of Cryptology, vol. 12, no. 4*, pp. 241-245, 1999.

[4] A.K. Lenstra, "Memo on RSA signature generation in the presence of faults," September 1996.

[5] M. Joye, J.-J.Quisquater, "Attacks on systems using chinese remaindering," *Tech. Report CG-1996/9, UCL Crypto Group, http://www.dice.ucl.ac.be/crypto/techreports.html*

[6] M. Joye, F.Koeune, and J.-J.Quisquater, "Further results on Chinese remaindering," *Tech. Report CG-1997/1, UCL Crypto Group*, Louvain-la-Neuve, March 1997.

[7] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *in Proceedings of Advances in Cryptology - CRYPTO '97*, pp. 513-525, Springer-Verlag, 1997.

[8] P. Dusart, G. Letourneux, O.Vivolo, "Differential Fault Analysis on A.E.S.," *Tech. Report. Laboratoire d'Arithmetique, de Calcul Formel et d'Optimisation*, 2003.

[9] 한국정보통신기술협회, "128비트 블록 암호 알고리즘 SEED 표준", http://www.tta.or.kr/