情報保護學會論文誌
第 22 卷 第 1 號, 2012. 2

PRESENT-80/128에 대한 향상된 차분 오류 공격*

박 세 현,[1†] 정 기 태[1], 이 유 섭[1], 성 재 철,[2] 홍 석 희[1‡]
[1]고려대학교 정보보호연구원, [2]서울시립대학교 수학과

Improved Differential Fault Analysis on Block Cipher PRESENT-80/128*

Sehyun Park,[1†] Kitae Jeong,[1] Yuseop Lee,[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 공격은 부채널 공격 기법 중 하나로 DES, AES, ARIA, SEED 등 대표적인 블록 암호 안전성 분석에 널리 사용되었다. PRESENT는 80/128-비트 비밀키를 사용하는 31-라운드 SPN 구조의 64-비트 블록 암호이다. 기제안된 PRESENT에 대한 차분 오류 공격들은 8~64개의 오류를 주입하여 80/128-비트 비밀키를 복구하였다. 본 논문에서는 이를 개선하여 PRESENT-80에 대해 2개의 오류를 주입하여 평균 1.7개의 비밀키를 남기고, PRESENT-128에 대해 3개의 오류를 주입하여 $2^{22.3}$개의 비밀키 후보를 구한다. 이 후 전수 조사를 통해 비밀키를 유일하게 복구한다. 이 공격은 기제안된 PRESENT의 차분 오류 공격보다 더 적은 수의 오류를 주입하여 효과적으로 비밀키를 복구한다.

ABSTRACT

A differential fault analysis(DFA) is one of the most important side channel attacks on block ciphers. Most block ciphers, such as DES, AES, ARIA, SEED and so on., have been analysed by this attack. PRESENT is a 64-bit block cipher with 80/128-bit secret keys and has a 31-round SP-network. So far, several DFAs on PRESENT have been proposed. These attacks recovered 80, 128-bit secret keys of PRESENT with 8~64 fault injections. respectively. In this paper, we propose an improved DFA on PRESENT-80/128. Our attack can reduce the complexity of exhaustive search of PRESENT-80(resp. 128) to on average 1.7(resp. $2^{22.3}$) with 2(resp. 3) fault injections, From these results, our attack results are superior to known DFAs on PRESENT.

Keywords: Side channel analysis, Diffential fault analysis, block cipher PRESENT

I. 서 론

최근, 암호 알고리즘 동작 과정에서 발생하는 부채널 정보를 이용하여 암호 알고리즘의 안전성을 분석하는 공격 기법들이 제안되고 있다. 이 분석 기법들은 기존의 차분 공격이나 선형 공격과 같이 암호 알고리즘 자체의 취약점을 이용하는 것이 아니라 암호 알고리즘이 수행되는 동안 발생하는 각종 부채널 정보들을 이용하여 타깃 알고리즘의 안전성을 분석한다. 차분 오류 공격(DFA)은 기존의 차분 공격에 부채널 공격 방법인 오류 주입 공격을 결합한 공격 방법이다. 1997년 Biham과 Shamir는 최초로 블록 암호 DES를

접수일(2011년 9월 14일), 게재확정일(2011년 12월 29일)
* 본 연구는 지식경제부 IT R&D 사업의 일환으로 수행하였음(유비쿼터스 환경에서의 정보보호 서비스를 위한 프라이버시 강화 암호 기술 개발)
† 주저자, c13441@naver.com
‡ 교신저자, shhong@korea.ac.kr

〔표 1〕 PRESENT80/128에 대한 공격 결과 비교

참고 문헌	타깃 알고리즘	오류 주입 위치	오류 주입 수	후보 비밀키(개수)
[7]	PRESENT-80	암호화 과정	평균 40~50개	2^{16}
[8]	PRESENT-80	키 스케줄	약 64개	2^{29}
[9]	PRESENT-80	암호화 과정	평균 8개	$2^{14.7}$
	PRESENT-128	암호화 과정	평균 16개	$2^{21.1}$
본 논문	PRESENT-80	암호화 과정	2개	평균 1.7
	PRESENT-128	암호화 과정	3개	평균 $2^{22.3}$

대상으로 차분 오류 공격을 적용하였다 [1]. 이 후, AES, Triple-DES, ARIA, SEED 등 대표적인 블록 암호의 안전성 분석에 적용되었다 [2,3,4,5].

PRESENT[6]는 31-라운드 SPN 구조의 64-비트 블록 암호로서 80/128-비트 비밀키를 사용한다. RFID(Radio Frequency IDentification)나 USN(Ubiqutious Sensor Network)와 같이 매우 제한적인 하드웨어 환경에 적합하도록 설계되었다. PRESENT에 대한 기존의 차분 오류 공격은 CHINACRYPT'09에서 PRESENT-80에 대해 라운드 29/30의 입력 레지스터에 평균 40~50개의 랜덤한 1-니블 오류를 주입함으로서 2^{16}개의 후보 비밀키를 복구할 수 있음을 보였고 [7], CIS'10에서는 PRESENT-80의 키스케줄에 약 64개의 랜덤한 1-니블 오류를 주입하여 2^{29}개의 후보 비밀키를 복구하였다 [8]. Zhao 등은 PRESENT-80과 PRESENT-128에 대해서 각각 평균 8개, 16개의 랜덤한 1-니블 오류를 주입하여 $2^{14.7}$개, $2^{21.1}$개의 후보 비밀키를 복구할 수 있음을 보였다 [9].

본 논문에서는 PRESENT-80/128에 대한 향상된 차분 오류 공격을 제안한다. PRESENT-80에 대해서는 라운드 28의 입력 레지스터에 2개의 랜덤한 2-바이트 오류를 주입하여 평균 1.7개의 후보 비밀키를 복구하며, PRESENT-128에 대해서는 3개의 랜덤한 2-바이트 오류를 주입하여 평균 $2^{22.3}$개의 후보 비밀키를 계산한 후 전수 조사를 통해 비밀키를 유일하게 복구한다. 〔표 1〕은 기제안된 공격 결과와 본 논문에서 제안한 공격 결과를 비교한 것이다.

본 논문은 다음과 같이 구성되어 있다. 먼저, 2장에서는 블록 암호 PRESENT에 대해 간략히 소개한다. 3장에서 오류 주입 가정과 차분 확산 특성을 설명한 후, 4장에서는 PRESENT에 대한 차분 오류 공격을 제안한다. 마지막으로 5장에서 결론을 맺는다.

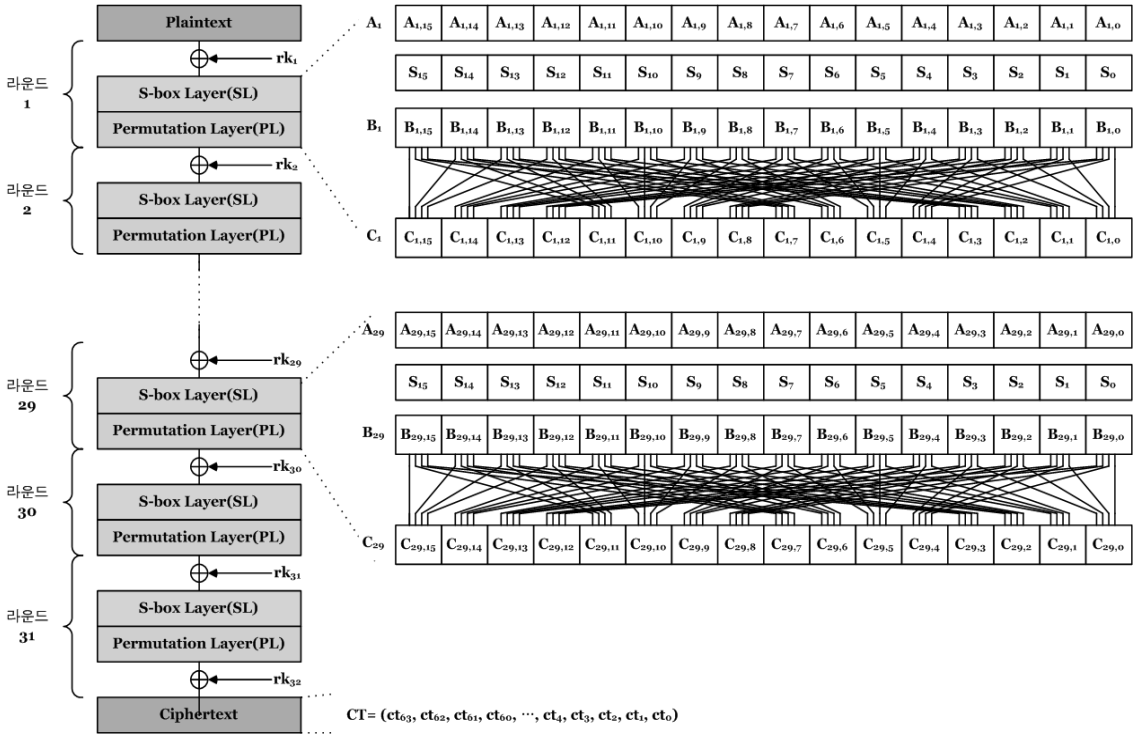
II. 블록 암호 PRESENT

PRESENT는 31-라운드 SPN 구조를 가지는 64-비트 블록 암호로서 80/128-비트 비밀키를 사용한다.

64-비트 평문 P는 16개의 니블 $P_j(0 \le j \le 15)$를 연접한 값으로 표시한다. 라운드 함수는 〔그림 1〕과 같이 AddRoundKey, S-box Layer, Permutation Layer로 구성된다. i 번째 라운드에서 S-box Layer의 입력값과 출력값을 각각 A_i, B_i로 표시하고, Permuation Layer의 출력값을 C_i로 표시한다. 각각의 니블은 〔그림 1〕과 같이 $X_{i,j}$ $(X=\{A,B,C\}, 0 \le i \le 31, 0 \le j \le 15)$로 표시한다.

- AddRoundKey : 64-비트 입력값과 라운드 키를 XOR한 64-비트 값을 출력
- S-box Layer(SL) : 64-비트 입력값을 16개의 4×4 S-box로 연산한 후 연접한 64-비트 값을 출력
- Permutation Layer(PL) : 64-비트 입력값을 단순 비트 치환하여 64-비트 값을 출력

PRESENT-80의 키스케줄은 80-비트 비밀키 K를 입력 받아 32개의 64-비트 라운드 키를 생성한다. 이를 위해 비밀키 K의 상위 64 비트를 라운드 1의 라운드 키 rk_1으로 추출한다. 그리고 K를 갱신하고 상위 64 비트를 라운드 2의 라운드 키 rk_2로 추출한다. 이러한 방법을 반복하여 32개의 라운드키를 생성한다. K를 갱신 하는 방법은 다음과 같다. 여기서, $K^1 = K$이고 Ctr은 라운드 상수를 의미한다. 예를 들어, 라운드 2의 라운드 키 rk_2는 K^2의 상위 64 비트로 생성되며 K^2는 $K^1(=K)$으로부터 다음 과정을 거쳐 계산된다.



〔그림 1〕 PRESENT 전체 구조도

1. $K^i = [k_{79}, k_{78}, \cdots, k_1, k_0]$.
2. $[k_{79}k_{78}\cdots k_1k_0] = [k_{18}k_{17}\cdots k_0k_{79}\cdots k_{20}k_{19}]$.
3. $[k_{79}k_{78}k_{77}k_{76}] = [S(k_{79}k_{78}k_{77}k_{76})]$.
4. $[k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15} \oplus Ctr]$.
5. $K^{i+1} = [k_{79}, k_{78}, \cdots, k_1, k_0]$.

PRESENT-80과 유사하게 PRESENT-128의 키스케줄은 128-비트 비밀키 K를 입력 받아 32개의 64-비트 라운드 키를 생성한다. 먼저, 비밀키 K의 상위 64 비트를 라운드 1의 라운드 키 rk_1으로 추출하고, K를 갱신하고 상위 64 비트를 라운드 2의 라운드 키 rk_2로 추출한다. K를 갱신하는 방법은 다음과 같다. PRESENT에 대한 더욱 자세한 사항은 〔6〕을 참조하라.

1. $K^i = [k_{127}, k_{126}, \cdots, k_1, k_0]$.
2. $[k_{127}k_{126}\cdots k_1k_0] = [k_{66}k_{65}\cdots k_{68}k_{67}]$.
3. $[k_{127}k_{126}k_{125}k_{124}] = [S(k_{127}k_{126}k_{125}k_{124})]$.
4. $[k_{123}k_{122}k_{121}k_{120}] = [S(k_{123}k_{122}k_{121}k_{120})]$.
5. $[k_{66}k_{65}k_{64}k_{63}k_{62}] = [k_{66}k_{65}k_{64}k_{63}k_{62} \oplus Ctr]$.
6. $K^{i+1} = [k_{127}, k_{126}, \cdots, k_1, k_0]$.

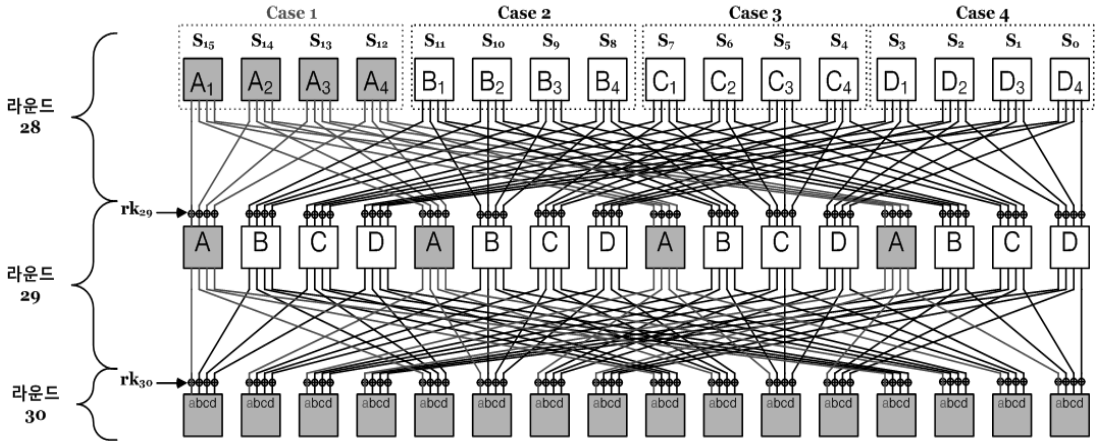
III. 오류 주입 가정 및 차분 확산 특성

본 절에서는 논문에서 사용하는 오류 주입 가정과 이에 따른 차분 확산 특성에 대해 소개한다.

3.1 오류 주입 가정

기 제안된 PRESENT에 대한 차분 오류 공격의 오류 주입 가정은 1-니블 랜덤 오류 주입을 가정하고 있다. 즉, 공격자는 원하는 라운드의 입력 레지스터의 1-니블에 랜덤한 오류를 주입할 수 있다고 가정하고 있다. 본 논문에서는 2-바이트 랜덤 오류 주입 가정을 적용한다. 즉, 레지스터를 2-바이트씩 분리하여 이 중 어느 하나의 구간에 오류가 주입된다고 가정한다. 이러한 가정에서는 기제안된 가정처럼 1-니블 랜덤 오류가 주입되는 경우도 동일한 공격이 가능하다. 즉, 각 구간중 어느 한 구간에 1~4개의 니블 오류가 주입된 경우 모두 본 공격을 적용할 수 있다. 본 논문에서는 사용하는 오류 주입 가정은 다음과 같다.

- 공격자는 라운드 28의 입력 레지스터에 랜덤한 2-바이트 오류를 주입할 수 있다. 이 때, 오류



〔그림 2〕 라운드 28의 입력 레지스터에 차분 발생 시 라운드 29/30의 차분 확산 특성

주입이 가능한 경우는 〔그림 2〕에서처럼 총 4가지 (Case 1, Case 2, Case 3, Case 4)이다.

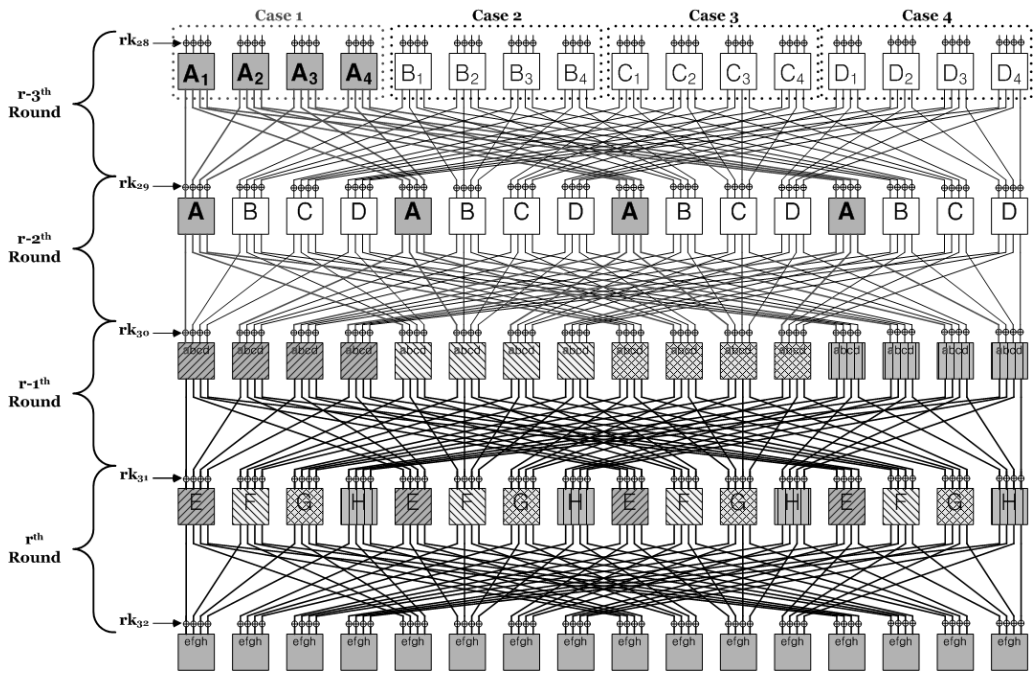
- 공격자는 임의의 평문에 대해 동일한 비밀키를 사용하여 올바른 암호문과 오류가 발생한 암호문을 얻을 수 있다.
- 공격자는 오류 주입 위치와 오류 값을 알지 못한다.

3.2 차분 확산 특성

기제안된 1-니블 랜덤 오류 주입 가정에서는, 타깃 라운드 i의 입력 레지스터에 오류가 주입될 경우 1개의 S-box에만 영향을 준다. 그러나 본 논문에서의 오류 주입 가정을 적용할 경우, 라운드 28에서 최소 1개에서 최대 4개의 S-box에 영향을 주게 된다. 즉, 기제안된 공격보다 더 많은 active S-box가 발생할 수 있으므로 이를 이용하여 후보 비밀키 수를 더 감소시킬 수 있다. 본 소절에서는 라운드 28의 입력 레지스터 중 상위 2개 바이트에 차분이 발생한 경우를 예로 든다 (〔그림 2〕의 Case 1). 라운드 28의 입력 레지스터 중 상위 2개 바이트에 발생한 차분은 라운드 28의 $S_{15},S_{14},S_{13},S_{12}$ (〔그림 2〕의 A_1,A_2,A_3,A_4)에 입력되며 PL을 거쳐 라운드 29의 S_{15},S_{11},S_7,S_3 (〔그림 2〕의 A)에만 영향을 준다. 라운드 29의 S_{15},S_{11},S_7,S_3에 입력된 차분은 PL을 통과해 라운드 30의 각 S-box의 가장 왼쪽 비트 (〔그림 2〕의 a)에만 영향을 준다. 다른 3가지 경우 (Case 2, Case 3, Case 4)도 유사한 차분 확산 특성을 보이기 때문에 위에서 나타난 2가지 특성을 일반화하여 다음을 얻을 수 있다.

- PRESENT의 차분 확산 특성
  1. 라운드 28의 $S_{4i-1},S_{4i-2},S_{4i-3},S_{4i-4}$에 입력된 차분은 라운드 29의 $S_{4i-1},S_{4i-5},\ S_{4i-9},S_{4i-13}$ 입력 차분에만 영향을 준다 $(1 \le i \le 4)$.
  2. 라운드 29의 $S_{4i-1},S_{4i-5},S_{4i-9},S_{4i-13}$에 입력된 차분은 라운드 30의 각 S-box 입력 차분 중 우측 i번째 비트에만 영향을 준다 $(1 \le i \le 4)$.
  3. 그러므로, 라운드 28의 $S_{4i-1},S_{4i-2},S_{4i-3},S_{4i-4}$에 입력된 차분은 라운드 30의 각 S-box 입력 차분 중 우측 i번째 비트에만 영향을 준다 $(1 \le i \le 4)$.

하지만, 라운드 28에 2-바이트 랜덤 차분이 입력되더라도 라운드 30의 모든 S-box에 차분이 입력되는 것은 아니다. 라운드 29에서 S-box 출력 차분의 해밍 웨이트에 따라 라운드 30에서 각 S-box의 차분 입력 유무가 결정된다. 예를 들어, 라운드 29에서 4개의 S-box 출력 차분이 $1000_{(2)},1100_{(2)},1110_{(2)},1111_{(2)}$이라고 가정하면 라운드 30의 10개 S-box에는 차분이 입력되고 6개 S-box에는 차분이 입력되지 않는다. 실험을 통해 라운드 28에 2-바이트 랜덤 차분이 발생한 경우 라운드 30의 active S-box의 수를 확인한 결과, 평균 8개의 S-box에 차분이 입력되었다. 따라서, Case 1에 차분이 발생한 경우 라운드 30의 각 S-box 입력 차분 $\triangle A_{30,i}$는 평균 $2^{-3}\left(=\frac{2}{16}\right)$ 확률로 0 또는 $8(=1000_{(2)})$이 될 것이다 $(0 \le i \le 15)$. 다른 3가지 경우도 유사한 차분 특성을 보이기 때문에 $\triangle A_{30,i}$는 다음 식을 만족하게 된다 $(0 \le i \le 15)$.



〔그림 3〕 PRESENT에 대한 차분 오류 공격

$$\triangle A_{30,i}(0 \le i \le 15) \in \begin{cases} \{0,8(=1000_{(2)})\} \\ \{0,4(=0100_{(2)})\} \\ \{0,2(=0010_{(2)})\} \\ \{0,1(=0001_{(2)})\} \end{cases} \quad (1)$$

IV. PRESENT-80/128에 대한 차분 오류 공격

본 절에서는 PRESENT-80/128에 대해 2, 3개의 랜덤한 2-바이트 오류를 각각 이용하여 마지막 2개 라운드 키(rk_{31},rk_{32}) 후보를 계산한 후, PRESENT-80/128의 키스케줄을 이용하여 비밀키를 복구하는 방법을 소개한다.

4.1 라운드 키 (rk_{31},rk_{32}) 복구 방법

*Step*1. **오류가 발생하지 않은 데이터 수집**

오류가 발생하지 않은 알고리즘을 이용하여 평문 P에 대한 암호문 C를 얻는다.

*Step*2. **오류가 발생한 데이터 수집**

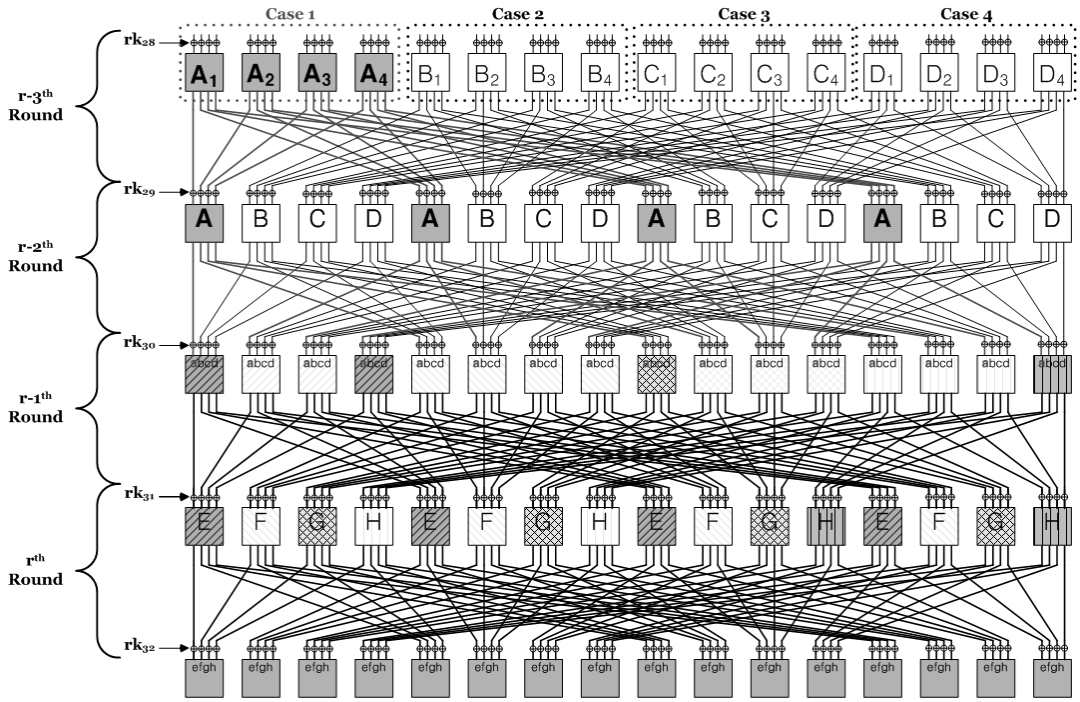
평문 P의 암호화 과정에서 라운드 28의 입력 레지스터에 2-바이트 랜덤 오류를 주입한 j개의 암호문 C^j를 획득한다. PRESENT-80의 경우 (C^1, C^2), PRESENT-128의 경우 (C^1, C^2, C^3)를 얻는다.

*Step*3. **라운드 30의 입력 차분 $\triangle A_{30,i}$ 계산** $(0 \le i \le 15)$

(C, C^j)에 대해 128-비트 라운드 키 (rk_{31},rk_{32})를 추측하면 다음과 같이 $\triangle A_{30,i}$를 계산할 수 있다 $(0 \le i \le 15)$.

$$\triangle A_{30,i} =$$
$$SL^{-1}[PL^{-1}\{SL^{-1}(PL^{-1}(C \oplus rk_{32})) \oplus rk_{31}\}] \oplus$$
$$SL^{-1}[PL^{-1}\{SL^{-1}(PL^{-1}(C^j \oplus rk_{32})) \oplus rk_{31}\}]$$

하지만, (rk_{31},rk_{32})를 한 번에 모두 추측할 경우 계산 복잡도가 너무 커진다. 따라서, 아래와 같이 (rk_{31},rk_{32})를 각각 16-비트씩 32-비트 단위로 4개의 Key Set으로 구분한 후 각 Key Set에 대해서 $\triangle A_{30,i}$를 독립적으로 계산한다 $(0 \le i \le 15)$. Key Set Ⅰ, Ⅱ, Ⅲ, Ⅳ는 〔그림 3〕에서 $(E,e),(F,f),(G,g),(H,h)$ 부분과 XOR되는 (rk_{31},rk_{32})를 각각 의미하며, 다음과 같이 표기할 수 있다. 여기서, rk_r^k는 r번째 라운드 키의 $k(0 \le k \le 63)$번째 비트를 의미하며, i,j의 범위는 $(1 \le i \le 4, 1 \le j \le 16)$이다.



〔그림 4〕 (rk_{31},rk_{32}) 후보 수가 많이 남게 되는 경우

Key Set Ⅰ :

$\{rk_{31}^{16i-1}, rk_{31}^{16i-2}, rk_{31}^{16i-3}, rk_{31}^{16i-4}, rk_{32}^{4j-1}\}$

$\Rightarrow \triangle A_{30,k}(12 \le k \le 15)$ 체크

Key Set Ⅱ :

$\{rk_{31}^{16i-5}, rk_{31}^{16i-6}, rk_{31}^{16i-7}, rk_{31}^{16i-8}, rk_{32}^{4j-2}\}$

$\Rightarrow \triangle A_{30,k}(8 \le k \le 11)$ 체크.

Key Set Ⅲ :

$\{rk_{31}^{16i-9}, rk_{31}^{16i-10}, rk_{31}^{16i-11}, rk_{31}^{16i-12}, rk_{32}^{4j-3}\}$

$\Rightarrow \triangle A_{30,k}(4 \le k \le 7)$ 체크.

Key Set Ⅳ :

$\{rk_{31}^{16i-13}, rk_{31}^{16i-14}, rk_{31}^{16i-15}, rk_{31}^{16i-16}, rk_{32}^{4j-4}\}$

$\Rightarrow \triangle A_{30,k}(0 \le k \le 3)$ 체크.

*Step*4. **라운드 28의 오류 발생 위치를 가정해 식 (1)을 만족하는 후보** (rk_{31},rk_{32}) **저장**

오류 주입 가정에 의해 공격자는 오류 발생 위치를 알 수 없다. 따라서, 오류 발생이 가능한 4가지 경우 (Case 1, Case 2, Case 3, Case 4)를 모두 고려한다. 먼저, 오류 발생 위치를 Case 1로 가정했을 때 라운드 30의 S-box 입력 차분 $\triangle A_{30,i}(0 \le i \le 15)$는 $\{0,8(=1000_{(2)})\}$이 된다. 한 개의 S-box에 입력 차분이 $\{0,8\}$일 확률은 $2^{-3}\left(=\frac{2}{16}\right)$이 되며 16개의 모든 S-box에 대해 입력 차분이 $\{0,8\}$일 확률은 $2^{-48}(=(2^{-3})^{16})$이 된다. (Case 2, Case 3, Case 4)에 오류가 발생한 경우에도 $\triangle A_{30,i}(0 \le i \le 15)$가 $\{0,4\}$, $\{0,2\}$, $\{0,1\}$을 만족할 확률은 각각 2^{-48}이 된다. 따라서, 한 개의 암호문 쌍 (C,C^1)을 이용하면 2^{128}개의 라운드 키 (rk_{31},rk_{32}) 후보를 $2^{82}(=2^{128}\cdot 2^{-48}\cdot 4)$개로 줄일 수 있을 것으로 기대된다. PRESENT-80의 경우 두 개의 암호문 쌍 (C,C^1), (C,C^2)를 이용할 수 있으므로 (rk_{31},rk_{32})의 후보를 $2^{36}(=2^{128}\cdot(2^{-46})^2)$개로 줄일 수 있을 것으로 기대되며, PRESENT-128의 경우 세 개의 암호문 쌍 (C,C^1), (C,C^2), (C,C^3)을 이용할 수 있으므로 (rk_{31},rk_{32})의 후보를 $2^{-10}(=2^{128}\cdot(2^{-46})^3)$확률로 유일하게 복구할 수 있을 것으로 기대된다.

4.2 비밀키 복구

본 소절에서는 앞에서 계산한 후보 128-비트 라운드 키 (rk_{31},rk_{32})를 이용하여 PRESENT-80과 PRESENT-128의 비밀키를 복구하는 방법을 소개한다.

4.2.1 PRESENT-80 비밀키 복구

앞 소절에서 소개한 방법에 따라 두 개의 암호문 쌍 (C,C^1), (C,C^2)를 이용하면 후보 (rk_{31},rk_{32})의 수의 기댓값은 2^{36}개다. 키스케줄로부터 rk_{31}은 $K^{31}=[k_{79},\cdots,k_0]$의 상위 64-비트 값인 $[k_{79},\cdots,k_{16}]$이 된다. 그리고 나서 K^{31}은 Update Process를 거쳐 K^{32}가 되고 rk_{32}는 K^{32}의 상위 64-비트 값이 되는데 rk_{32}를 K^{31}에 관해 표현하면 다음과 같다.

$$rk_{32}=[S(k_{18},k_{17},k_{16},k_{15}),k_{14},k_{13},\cdots,k_0,k_{79},k_{78},\cdots,k_{39},(k_{38},k_{37},k_{36},k_{35}\oplus Ctr)].$$

여기에서, (rk_{31},rk_{32}) 모두 K^{31}의 48-비트 값 $(k_{16},k_{17},k_{18},k_{35},\cdots,k_{79})$를 포함하므로 다음 3개의 식이 성립한다. 여기서, '?'는 임의의 1-비트 값을 의미한다.

$$(rk_{31,2},rk_{31,1},rk_{31,0},?)=S^{-1}(rk_{32,63},rk_{32,62},rk_{32,61},rk_{32,60})$$
$$(rk_{31,22},rk_{31,21},rk_{31,20},rk_{31,19})=(rk_{32,3},rk_{32,2},rk_{32,1},rk_{32,0})$$
$$rk_{31,19+i}=rk_{32,i}\ (4\le i\le 44).$$

복구한 (rk_{31},rk_{32}) 중 위의 식을 만족할 확률은 $2^{-47}(=2^{-2}\cdot 2^{-4}\cdot 2^{-41})$이므로 위의 테스트를 통과하는 후보 (rk_{31},rk_{32})의 수의 기댓값은 $2^{-11}(=2^{36}\cdot 2^{-47})$ 확률로 1이다. 즉, 위의 과정을 수행하여 우리는 옳은 (rk_{31},rk_{32})를 복구할 수 있다. 또한, 옳은 (rk_{31},rk_{32})로부터 K^{31}을 계산하고 키스케줄을 이용하여 PRESENT-80의 비밀키 K를 쉽게 복구할 수 있다. 그러므로, 본 논문에서 제안한 공격으로 PRESENT-80의 비밀키를 복구할 수 있다.

4.2.2 PRESENT-128 비밀키 복구

PRESENT-128의 경우 세 개의 암호문 쌍 (C,C^1), (C,C^2), (C,C^3)를 이용하면 2^{-10}의 확률로 (rk_{31},rk_{32})를 유일하게 복구할 것으로 기대된다. 〔표 2〕로부터 rk_{31}은 $K^{31}=[k_{127},\cdots,k_0]$의 상위 64-비트 값인 $[k_{127},\cdots,k_{64}]$이 된다. 그리고 나서 K^{31}은 Update Process를 거쳐 K^{32}가 되고 rk_{32}는 K^{32}의 상위 64-비트 값이 되는데 rk_{32}를 K^{31}에 관해 표현하면 다음과 같다.

$$rk_{32}=[S(k_{66},k_{65},k_{64},k_{63}),S(k_{62},k_{61},k_{60},k_{59}),k_{58},k_{57},\cdots,k_7,k_6,(k_5,k_4,k_3\oplus Ctr)].$$

여기에서, (rk_{31},rk_{32}) 모두 K^{31}의 3-비트 값 (k_{66},k_{65},k_{64})를 포함하므로 다음이 성립한다.

$$(rk_{31,2},rk_{31,1},rk_{31,0},?)=S^{-1}(rk_{32,63},rk_{32,62},rk_{32,61},rk_{32,60})$$

복구한 후보 (rk_{31},rk_{32})가 위의 식을 만족할 확률은 2^{-2}이므로 위의 테스트를 통과하는 후보 (rk_{31},rk_{32})의 수의 기댓값은 $2^{-12}(=2^{-10}\cdot 2^{-2})$확률로 1이다. 즉, 위의 과정을 수행하여 우리는 옳은 (rk_{31},rk_{32})를 복구할 수 있다. 하지만, 옳은 (rk_{31},rk_{32})를 복구하더라도 K^{31}의 3-비트 값인 (k_2,k_1,k_0)에 대한 정보를 얻을 수 없다. 따라서, 3-비트에 대해서는 전수조사가 필요하므로 후보 K^{31}의 수의 기댓값은 2^3이다. 복구한 후보 K^{31}에 대해 키스케줄을 이용하면 2^3개의 비밀키 K를 계산할 수 있으며, 전수 조사를 통해 PRESENT-128의 비밀키 K를 유일하게 복구할 수 있다. 그러므로, 본 논문에서 제안한 공격으로 PRESENT-128의 비밀키를 복구할 수 있다.

4.3 계산 복잡도 분석

*Step3*에서는 128-비트 라운드 키 (rk_{31},rk_{32})를 32-비트 단위 4개의 Key Set으로 구분하여 각각에 대해서 $\triangle A_{30,i}$를 계산하였다 $(0\le i\le 15)$. 각각의 Key Set은 2^{32}개의 후보 라운드 키가 존재하므로, 이 단계의 계산 복잡도는 $O(2^{32})$이다. *Step4*는 *Step3*에서 계산한 결과가 식 (1)을 만족하는지 여부만 확인하기 때문에 이 과정의 계산 복잡도는 *Step3*에 비해 미미하다. 따라서, 4.1 소절에서 소개한 과정의 계산 복잡도는 $O(2^{32})$이다. 4.2 소절은 4.1 소절에서 복구한 후보 (rk_{31},rk_{32})에 대해 비트 단위 비교 연산만 수행한다. 따라서 이 과정의 계산 복잡도도 4.1 소절의 *Step3*에 비해 미미하다. 따라서, 비밀키를 복구하는데 필요한 전체 계산 복잡도는 $O(2^{32})$이다.

4.4 구현 결과

본 논문에서 제안하는 공격을 Intel(R) Core (TM)2 CPU 6400 @2.13GHz, RAM 2GB, Visual Studio 2008 환경에서 시뮬레이션 하였다. PRESENT-80/128에 대해 비밀키 복구 공격을 각

〔표 2〕 실험 결과

알고리즘	실험 횟수	오류 주입수	후보 비밀키 수
PRESENT-80	1,000	2	평균 1.7
PRESENT-128	1,000	3	평균 $2^{22.3}$

각 1,000회 실시하였으며, 구현 결과는 〔표 2〕와 같다. PRESENT-80/128에 대해 2개, 3개의 오류를 주입한 결과, 이론적인 수보다 많은 평균 2^{45}개, $2^{21.3}$개의 후보 (rk_{31},rk_{32})가 남았다. 이는 라운드 30/31의 active S-box의 수가 적을 경우 필터링 확률이 줄어들기 때문인 것으로 예측된다. 예를 들어, 〔그림 4〕와 같이 라운드 30/31의 S-box를 E, F, G, H로 표현할 때 F 부분의 차분이 0이라고 가정하면, Key Set Ⅱ에 관여된 모든 후보 라운드 키에 대해 $\triangle A_{30,i}$는 0이 된다 $(8 \le i \le 11)$. 즉, Key Set Ⅱ의 후보 라운드 키는 한 비트도 필터링이 되지 않고 2^{32}개의 라운드 키 후보가 그대로 남게 된다. 또한, H 부분에서 라운드 30, 31의 active S-box의 수가 각각 1개, 2개라고 가정할 경우 Key Set Ⅳ에 대한 필터링 확률은 예상치인 $2^{-12}(=(2^{-3})^4)$보다 적은 약 2^{-7}이었다. 실험 결과, 1개의 암호문 쌍 (C,C^4)을 이용할 경우 라운드 30/31의 active S-box의 수에 따라 $2^{-25} \sim 2^{-51}$의 확률로 (rk_{31},rk_{32})가 필터링 되어 예상보다 많은 후보 (rk_{31},rk_{32})가 남게 되었다. 남은 (rk_{31},rk_{32})에 대해 4.2 소절에서 제안한 비밀키 복구 공격을 실시한 결과, PRESENT-80에 대해서는 2개의 오류를 주입하여 평균 1.7개의 후보 비밀키가 살아남았으며, PRESENT-128에 대해서는 3개의 오류를 주입하여 평균 $2^{22.3}$개의 후보 비밀키가 살아남았다. 살아남은 후보 비밀키는 전수 조사를 통해 옳은 비밀키를 유일하게 복구할 수 있었다.

Ⅴ. 결 론

본 논문에서는 블록 암호 PRESENT-80/128에 대한 향상된 차분 오류 공격을 제안하였다. 공격자는 라운드 28의 입력 레지스터에 랜덤한 2-바이트 오류를 주입한다. PRESENT-80의 경우 2개의 오류를 이용하여 평균 1.7개의 후보 비밀키를 계산할 수 있으며, PRESENT-128의 경우 3개의 오류를 주입하여 평균 $2^{22.3}$개의 후보 비밀키를 계산할 수 있다. 이후 전수 조사를 통해 옳은 비밀키를 유일하게 복구할 수 있다. 본 논문에 제안된 공격은 기제안된 PRESENT-80/128에 대한 차분 오류 공격보다 더 현실적인 가정을 사용함은 물론, 더 적은 오류를 이용하여 비밀키를 유일하게 복구할 수 있다.

참고문헌

〔1〕 E. Biham, A. Shamir, "Differential faut analysis of secret key cryptosystems," Advances in Cryptology, CRYPTO'97, LNCS 1294, pp. 513-525, 1997.

〔2〕 P. Dusart, G. Letourneux, O. Vivolo, "Differntial fault attack on AES," Applied Cryptography and Network Security, ACNS'03, LNCS 2846, pp. 293-306, 2003.

〔3〕 L. Hemme, "A differential faut analysis against early rounds of (triple)-DES," Workshop on Cryptographic Hardware and Embedded Systems, CHES'04, LNCS 3156, pp. 254-267, 2004.

〔4〕 W. Li, D. Gu, J. Li, "Differential fault analysis on the ARIA algorithm," Information Sciences, Vol. 178, no. 19, pp. 3727-3737, Oct. 2008.

〔5〕 정기태, 성재철, 홍석희, "블록 암호 SEED에 대한 차분 오류 공격," 정보보호학회논문지, 제 20권, 제 4호, pp. 17-24, 2010년 8월.

〔6〕 A. Bogdanov, L. Knudsen, G. Leander, C. Paar, "PRESENT: an ultra-lightweight block cipher,". Workshop on Cryptographic Hardware and Embedded Systems, CHES'07, LNCS 4727, pp. 450-466, Springer-Verlag, 2007.

〔7〕 J. Li, D. Gu, "Differential fault analysis on PRESENT," CHINACRYPT'09, pp.3-13. Nov. 2009.

〔8〕 G. Wang, S. Wang, "Differential fault analysis on PRESENT key schedule," CIS'10, pp.362-366. Dec, 2010.

〔9〕 X. Zhao, T. Wang and S. Guo, "Fault propagate pattern based DFA on SPN structure block ciphers using bitwise permutation with Application to PRESENT and PRINTcipher," ePrint 2011-086, Feb. 2011.

〈著者紹介〉

박 세 현 (Sehyun Park) 학생회원
2000년 2월: 육군사관학교 토목공학과 학사
2010년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 대칭키 암호의 분석 및 설계

정 기 태 (Kitae Jeong) 학생회원
2004년 2월: 고려대학교 수학과 학사
2006년 2월: 고려대학교 정보보호대학원 석사
2011년 8월: 고려대학교 정보경영공학전문대학원 박사
2011년 9월~현재: 고려대학교 정보보호연구원 박사후연구원
〈관심분야〉 대칭키 암호의 분석 및 설계

이 유 섭 (Yuseop Lee) 학생회원
2007년 2월: 서울시립대학교 수학과 학사
2006년 3월~현재: 고려대학교 정보보호대학원 석박사 통합과정
〈관심분야〉 스트림 암호, 해쉬 함수의 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~현재: 고려대학교 정보보호대학원 부교수
〈관심분야〉 대칭키·공개키 암호 분석 및 설계, 컴퓨터 포렌식