情報保護學會論文誌
第 21 卷 第 5 號, 2011. 10

블록 암호 ARIA-128에 대한 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

Differential Fault Analysis on Block Cipher ARIA-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 공격(DFA)은 블록 암호의 안전성 분석에 널리 사용되는 부채널 공격 기법으로서, 대표적인 블록 암호인 DES, AES, ARIA, SEED 등에 적용되었다. 2008년 Wei 등은 ARIA-128에 대한 첫 번째 DFA를 제안하였다. 이 공격은 평균 45개의 바이트 오류를 이용하여 128-비트 비밀키를 복구하였다. 본 논문에서는 Wei 등의 공격을 개선한 ARIA-128에 대한 DFA를 제안했다. 본 논문에서 제안하는 공격은 4개의 오류만을 이용하여 $O(2^{32})$의 계산 복잡도로 ARIA-128의 비밀키를 복구할 수 있다.

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. In 2008, Wei et al. proposed the first DFA on ARIA-128. Their attack can recover the 128-bit secrey key by about 45 faulty ciphertexts. In this paper, we propose an improved DFA on ARIA-128. We can recover the 128-bit secret key by only 4 faulty ciphertexts with the computational complexity of $O(2^{32})$.

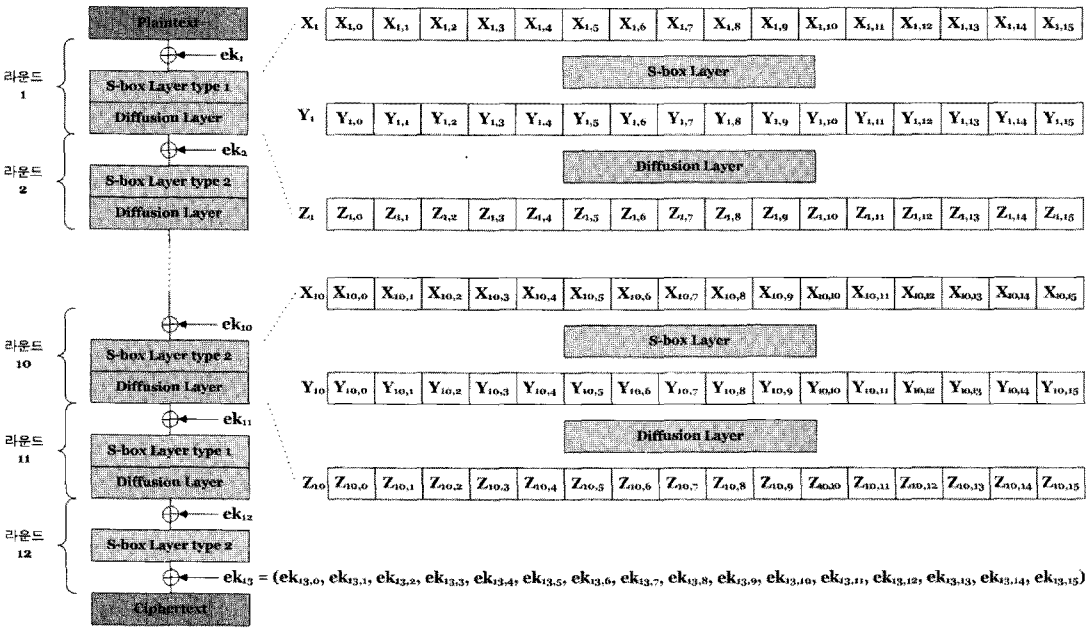
Keywords: Side channel analysis, Differtial fault analysis, block cipher ARIA

I. 서 론

차분 오류 공격(DFA)은 기존의 차분 공격[1]과 오류 주입 공격이 결합된 부채널 공격 기법 중 하나이다. 1997년 DES에 최초로 적용되었으며, 이후 AES, Triple-DES, ARIA, SEED 등 대부분의 블록 암호에 적용되었다[2, 3, 4, 5]. ARIA는 전자 정부 구현 등으로 다양한 환경에 적합한 암호 알고리즘이 필요함에 따라 ETRI 부설 연구소 주도로 학계, 국가정보원 등의 암호 기술 전문가들이 공동으로 개발한 128-비트 블록 암호이다[6]. 이 알고리즘은 경량 환경 및 하드웨어 구현을 위해 최적화된 involutional SPN 구조를 갖는 범용 블록 암호로서, 민간 암호화 알고리즘 SEED와 함께 전자 정부의 대국민행정서비스용으로 보급되고 있으며, 스마트 카드 등의 초경량 환경 및 고성능 서버 환경 등에서 장점을 가지고 있다. 또한 2004년에는 국가표준기본법에 의거, 지식경제부에 의하여 국가표준(KS)으로 지정되었다.

기제안된 ARIA에 대한 DFA는 [7]에서 제안된 ARIA-128에 대한 공격 결과가 유일하다. 이 공격은 ARIA-128에 평균 45개의 바이트 오류를 주입하여

접수일(2010년 12월 29일), 게재확정일(2011년 5월 24일)
* 이 연구에 참여한 연구자(의 일부)는 '2단계bk21사업의 지원비를 받았음
† 주저자, parksh@cist.korea.ac.kr
‡ 교신저자, hsh@cist.korea.ac.kr



〔그림 1〕 ARIA 암호화 과정

ARIA-128의 128-비트 비밀키를 복구한다. 먼저, 라운드 8～11의 각 라운드에 평균 11개의 바이트 오류를 주입한 후, 발생하는 차분 특성을 이용하여 마지막 4개 라운드의 라운드 키를 복구한다. 그리고 복구한 라운드 키와 키스케줄 특성을 이용하여 128-비트 비밀키를 복구한다.

한편, Tunstall 등은 AES-128에 대한 DFA를 제안하였다[8]. 이 공격은 다음과 같은 세 개의 단계로 구성된다. 먼저, AES-128의 라운드 8에 1개의 랜덤 바이트 오류를 주입한 후 라운드 9에서 발생하는 차분 특성을 이용하여 12개의 선형식을 구성한다. 그리고, 라운드 10의 라운드 키를 추측한 후 구성된 선형식을 만족하는 후보 라운드 키를 계산한다. 이후 복구된 후보 라운드 키와 키스케줄 특성을 이용하여 128-비트 비밀키를 복구한다.

본 논문에서는 [8]에서 제안된 공격을 ARIA-128에 적용한다. 본 논문에서 제안하는 공격은 [7, 8]에서 제안된 DFA와 달리, 암·복호화 과정에 오류를 주입한다는 가정을 이용한다. 즉, 암·복호화 과정에서 라운드 10의 입력 레지스터에 2개의 바이트 오류를 각각 주입하여 라운드 12의 라운드 키를 각각 복구한다. 그리고 복구한 암·복호화 과정에서의 마지막 라운드 키와 키스케줄 특성을 이용하여 $O(2^{32})$의 계산 복잡도로 ARIA-128의 128-비트 비밀키를 복구한다.

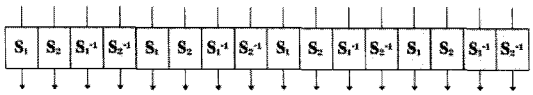
본 논문은 다음과 같이 구성되어 있다. 먼저, 2장에서는 ARIA-128을 간략히 소개한다. 3장에서는 암·복호화 과정의 마지막 라운드 키를 얻는 방법을 제안한 후, 4장에서 복구된 마지막 라운드 키로부터 비밀키를 복구하는 방법을 제안한다. 마지막으로 5장에서는 결론을 맺는다.

II. ARIA-128

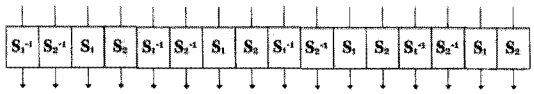
블록 암호 ARIA-128은 128-비트 비밀키를 사용하는 12라운드 SPN구조이며, 암·복호화가 동일한 involution 구조를 갖는다. ARIA-128의 라운드 함수는 〔그림 1〕과 같이 라운드 키 덧셈, 치환 계층, 확산 계층으로 구성된다.

라운드 키 덧셈 과정에서는 128-비트 라운드 키를 128-비트 라운드 입력값과 비트별 XOR한다. 치환 계층에서는 두 가지 유형의 8-비트 입·출력 S-box와 그들의 역변환으로 구성된다. 〔그림 2〕와 〔그림 3〕은 각각 홀수 라운드와 짝수 라운드의 치환계층을 나타낸다. 확산 계층은 다음과 같은 16×16 involution 이진 행렬을 사용한 바이트 간의 확산 함수로 구성되어 있다.

라운드 키 덧셈 과정에서는 128-비트 라운드 키를 128-비트 라운드 입력값과 비트별 XOR한다. 치환 계층에서는 두 가지 유형의 8-비트 입·출력 S-box와



〔그림 2〕 S-box layer type 1(홀수 라운드 치환계층)



〔그림 3〕 S-box layer type 2(짝수 라운드 치환계층)

그들의 역변환으로 구성된다. 〔그림 2〕와 〔그림 3〕은 각각 홀수 라운드와 짝수 라운드의 치환계층을 나타낸다. 확산 계층은 〔그림 4〕와 같이 16×16 involution 이진 행렬을 사용한 바이트 간의 확산 함수로 구성되어 있다.

ARIA-128의 키 스케줄은 초기화 과정과 라운드 키 생성 과정으로 나뉜다. 초기화 과정에서는 128-비트 비밀키 MK를 입력받아 KL에 저장하고 KR은 0으로 패딩하여 256-비트 값 $KL \| KR$를 구성한다.

$$KL \| KR = MK \| 00\cdots00.$$

$KL \| KR$을 이용하여 4개의 128-비트 값 W_0, W_1, W_2, W_3를 아래와 같이 생성한다. 여기서, F_o와 F_e는 각각 ARIA-128의 홀수, 짝수 라운드 함수를 의미한다.

- $W_0 = KL = MK.$
- $W_1 = F_0(W_0, CK_1),$
  $CK_1 = 0x517cc1b727220a94fe13abe8fa9a6ee0.$
- $W_2 = F_e(W_1, CK_2) \oplus W_0,$
  $CK_2 = 0x6db14acc9e21c820ff28b1d5ef5de2b0.$

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{15} \end{pmatrix} = \begin{pmatrix} 0&0&0&1&1&0&1&0&1&1&0&0&0&1&1&0 \\ 0&0&1&0&0&1&0&1&1&1&0&0&1&0&0&1 \\ 0&1&0&0&1&0&1&0&0&0&1&1&1&0&0&1 \\ 1&0&0&0&0&1&0&1&0&0&1&1&0&1&1&0 \\ 1&0&1&0&0&1&0&0&1&0&0&1&0&0&1&1 \\ 0&1&0&1&1&0&0&0&0&1&1&0&0&0&1&1 \\ 1&0&1&0&0&0&0&1&0&1&1&0&1&1&0&0 \\ 0&1&0&1&0&0&1&0&1&0&0&1&1&1&0&0 \\ 1&1&0&0&1&0&0&1&0&0&1&0&0&1&0&1 \\ 1&1&0&0&0&1&1&0&0&0&0&1&1&0&1&0 \\ 0&0&1&1&0&1&1&0&1&0&0&0&0&1&0&1 \\ 0&0&1&1&1&0&0&1&0&1&0&0&1&0&1&0 \\ 0&1&1&0&0&0&1&1&0&1&0&1&1&0&0&0 \\ 1&0&0&1&0&0&1&1&1&0&1&0&0&1&0&0 \\ 1&0&0&1&1&1&0&0&0&1&0&1&0&0&1&0 \\ 0&1&1&0&1&1&0&0&1&0&1&0&0&0&0&1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{pmatrix}$$

〔그림 4〕 확산계층

- $W_3 = F_o(W_2, CK_3) \oplus W_1,$
  $CK_3 = 0xdb92371d2126e9700324977504e8c90e.$

라운드 키 생성 과정에서는 위에서 생성한 W_0, W_1, W_2, W_3를 조합하여 다음과 같은 과정을 통해 $ek_1, ek_2, ek_3, \cdots, ek_{13}$을 생성한다.

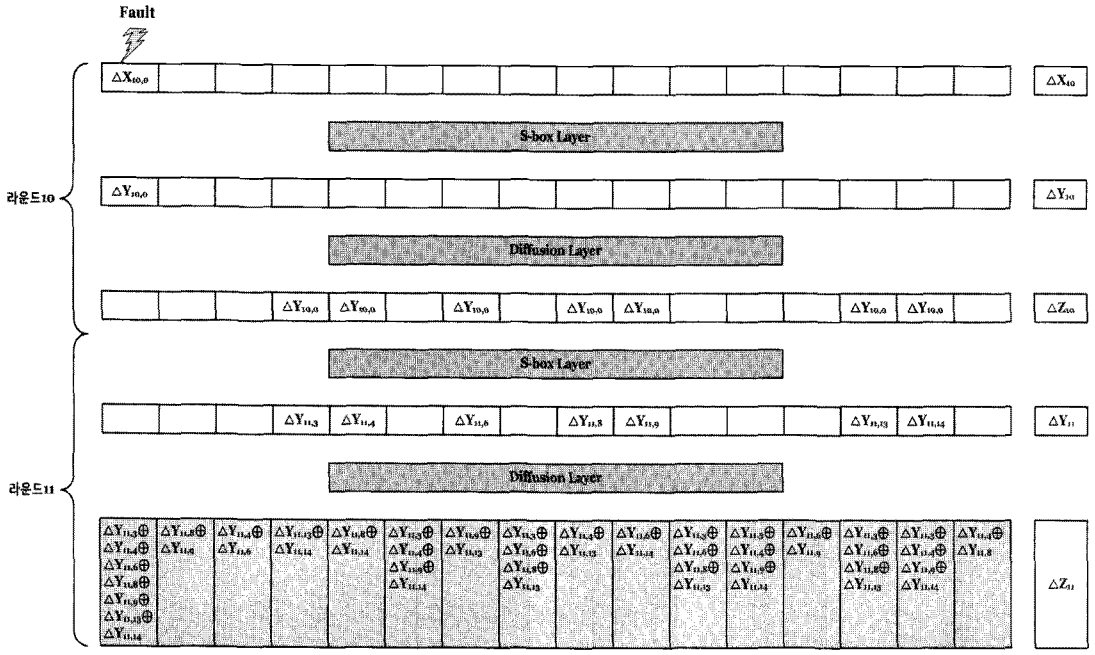
$$ek_1 = (W_0) \oplus (W_1^{\gg 19}), \quad ek_2 = (W_1) \oplus (W_2^{\gg 19}),$$
$$ek_3 = (W_2) \oplus (W_3^{\gg 19}), \quad ek_4 = (W_0^{\gg 19}) \oplus (W_3),$$
$$ek_5 = (W_0) \oplus (W_1^{\gg 31}), \quad ek_6 = (W_1) \oplus (W_2^{\gg 31}),$$
$$ek_7 = (W_2) \oplus (W_3^{\gg 31}), \quad ek_8 = (W_0^{\gg 31}) \oplus (W_3),$$
$$ek_9 = (W_0) \oplus (W_1^{\ll 61}), \quad ek_{10} = (W_1) \oplus (W_2^{\ll 61}),$$
$$ek_{11} = (W_2) \oplus (W_3^{\ll 61}), \quad ek_{12} = (W_0^{\ll 61}) \oplus (W_3),$$
$$ek_{13} = (W_0) \oplus (W_1^{\ll 31}).$$

III. ARIA-128에 대한 차분 오류 공격

본 절에서는 ARIA-128에 대한 차분 오류 공격을 소개한다. 본 공격은 암·복호화 과정에서 X_{10}(라운드 10의 입력 레지스터, 〔그림 1〕 참조)에 랜덤한 1-바이트 오류를 주입하여, 암·복호화 과정에서 라운드 12의 라운드 키 (ek_{13}, ek_1)을 각각 복구한다.

3.1 ARIA-128의 차분 특성 구성

ARIA-128은 암호화와 복호화가 동일한 involution 구조로 설계되어 있어 암호화 과정과 복호화 과정의 차분 확산 특성이 동일하다. 그러므로 본 소절에서는 암호화 과정의 X_{10}에 오류가 발생하였을 때, 발생하는 차분의 특성만 설명한다. X_{10}에 랜덤한 1-바이트 오류가 발생한다고 가정한다. 이 때, 공격자는 오류값과 오류 발생 위치를 알 수 없다. 오류가 발생하는 경우의 수는 16이므로, 오류가 발생하는 바이트 위치를 가정하여 16개의 차분 특성을 각각 구성한다. 본 소절에서는 $X_{10,0}$(라운드 10의 입력 레지스터의 0번째 바이트, Case Ⅰ)과 $X_{10,1}$(라운드 10의 입력 레지스터의 1번째 바이트, Case Ⅱ)에 오류가 발생하였다고 가정하였을 때, 발생하는 차분 특성을 통해 선형식을 구성하는 방법만을 각각 소개한다. $X_{10,2}, X_{10,3}, \cdots, X_{10,15}$에 오류가 발생한 경우도 유사한 방법으로 선형식을 구성할 수 있다.



〔그림 5〕 $X_{10,0}$에 오류 발생 시 확산 경로

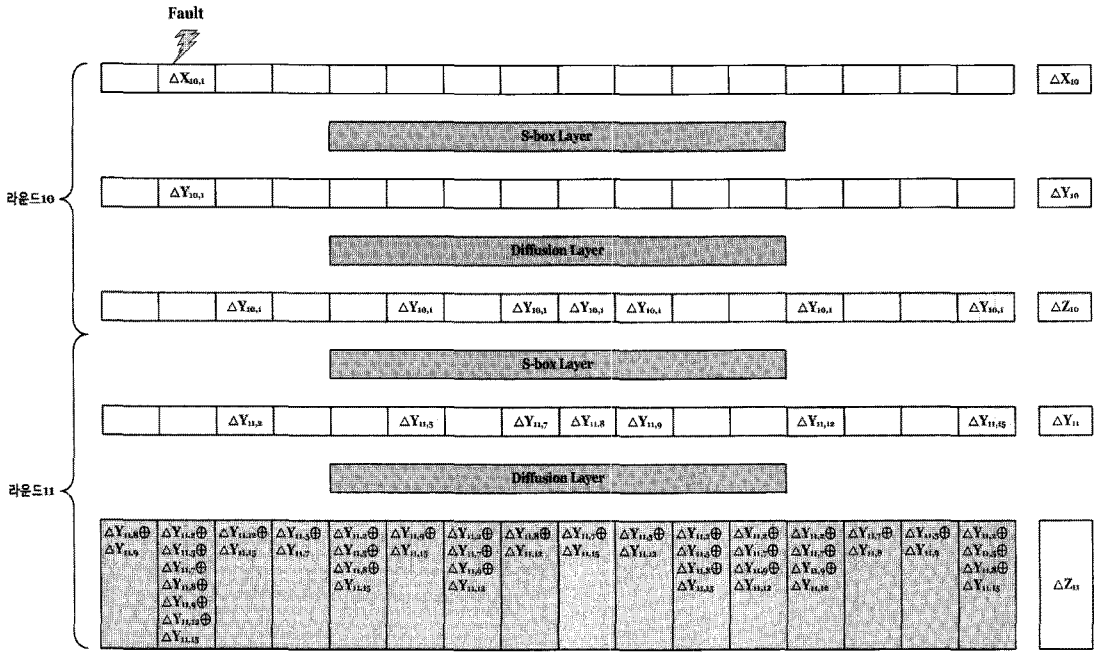
3.1.1 Case I : $X_{10,0}$에 오류가 발생하였을 때 ΔZ_{11}의 차분 특성

$X_{10,0}$에 오류가 발생하고 $X_{10,0}$의 차분값을 $\Delta X_{10,0}$이라고 가정하였을 때 차분 확산 경로는 〔그림 5〕와 같다. $\Delta X_{10,0}$이 S-box layer를 거친 후의 차분은 $\Delta Y_{10,0}$이 되며, $\Delta Y_{10,0}$은 Diffusion layer를 통해 $\Delta Z_{10,3}, \Delta Z_{10,4}, \Delta Z_{10,6}, \Delta Z_{10,8}, \Delta Z_{10,9}, \Delta Z_{10,13}, \Delta Z_{10,14}$로 확산된다. 이 때 이 차분값들은 $\Delta Y_{10,0}$과 동일하다. 이 차분들이 라운드 11의 S-box layer를 거친 후의 차분을 $\Delta Y_{11,3}, \Delta Y_{11,4}, \Delta Y_{11,6}, \Delta Y_{11,8}, \Delta Y_{11,9}, \Delta Y_{11,13}, \Delta Y_{11,14}$라고 하면, 이 차분들이 Diffusion layer를 지나면서 라운드 11의 출력값 차분 ΔZ_{11}은 〔그림 5〕와 같다. 여기서, ΔZ_{11}의 5번째 바이트 $\Delta Z_{11,5}$와 14번째 바이트 $\Delta Z_{11,14}$의 차분값은 알 수 없지만, 두 차분은 $\Delta Y_{11,3} \oplus \Delta Y_{11,4} \oplus \Delta Y_{11,9} \oplus \Delta Y_{11,14}$로 동일함을 알 수 있다. 즉, 식 (1)-⑨를 구성할 수 있다. 또한, $\Delta Z_{11,7}$과 $\Delta Z_{11,13}$이 동일하므로 식 (1)-⑥을 구성할 수 있다. 이러한 방법을 통해, 식 (1)과 같이 총 9개의 선형식을 구성할 수 있다. 이 때 구성된 9개의 선형식에서는 $\Delta Z_{11,0}$을 제외한 15개 바이트의 차분 정보를 얻을 수 있다.

$$
\begin{aligned}
&① \ \Delta Z_{11,1} \oplus \Delta Z_{11,8} \oplus \Delta Z_{11,9} \oplus \Delta Z_{11,13} \oplus \Delta Z_{11,14} = 0 \\
&② \ \Delta Z_{11,2} \oplus \Delta Z_{11,8} \oplus \Delta Z_{11,9} \oplus \Delta Z_{11,12} \oplus \Delta Z_{11,13} \oplus \Delta Z_{11,14} \oplus \Delta Z_{11,15} = 0 \\
&③ \ \Delta Z_{11,3} \oplus \Delta Z_{11,12} \oplus \Delta Z_{11,13} \oplus \Delta Z_{11,14} \oplus \Delta Z_{11,15} = 0 \\
&④ \ \Delta Z_{11,4} \oplus \Delta Z_{11,8} \oplus \Delta Z_{11,12} \oplus \Delta Z_{11,13} \oplus \Delta Z_{11,14} = 0 \\
&⑤ \ \Delta Z_{11,6} \oplus \Delta Z_{11,9} \oplus \Delta Z_{11,13} \oplus \Delta Z_{11,14} \oplus \Delta Z_{11,15} = 0 \\
&⑥ \ \Delta Z_{11,7} \oplus \Delta Z_{11,13} = 0 \\
&⑦ \ \Delta Z_{11,10} \oplus \Delta Z_{11,13} = 0 \\
&⑧ \ \Delta Z_{11,11} \oplus \Delta Z_{11,14} = 0 \\
&⑨ \ \Delta Z_{11,5} \oplus \Delta Z_{11,14} = 0
\end{aligned}
\tag{1}
$$

3.1.2 Case II : $X_{10,1}$에 오류가 발생하였을 때 ΔZ_{11}의 차분 특성

$X_{10,1}$에 오류가 발생하였을 때의 확산 경로는 〔그림 6〕과 같다. Case II도 Case I과 유사한 방법으로 식 (2)와 같은 9개의 선형식을 구성할 수 있다. 여기서도 $\Delta Z_{11,1}$을 제외한 15개의 바이트 차분 정보만 얻을 수 있다. 즉, Case I과 Case II의 결과에서 알 수 있듯이 $X_{10,i}(0 \le i \le 15)$에 오류가 발생했다면 $\Delta Z_{11,i}$를 제



〔그림 6〕 $X_{10,1}$에 오류 발생 시 확산 경로

외한 15개의 바이트 차분 정보를 얻을 수 있다. $X_{10,2}, X_{10,3}, \cdots, X_{10,15}$에 오류가 발생할 경우도 유사한 방법으로 각각 9개의 선형식을 구성할 수 있다.

$$
\begin{aligned}
&① \ \Delta Z_{11,0} \oplus \Delta Z_{11,8} \oplus \Delta Z_{11,9} \oplus \Delta Z_{11,12} \oplus \Delta Z_{11,15} = 0 \\
&② \ \Delta Z_{11,2} \oplus \Delta Z_{11,12} \oplus \Delta Z_{11,13} \oplus \Delta Z_{11,14} \oplus \Delta Z_{11,15} = 0 \\
&③ \ \Delta Z_{11,3} \oplus \Delta Z_{11,8} \oplus \Delta Z_{11,9} \oplus \Delta Z_{11,12} \oplus \Delta Z_{11,13} \oplus \Delta Z_{11,14} \oplus \Delta Z_{11,15} = 0 \\
&④ \ \Delta Z_{11,5} \oplus \Delta Z_{11,9} \oplus \Delta Z_{11,12} \oplus \Delta Z_{11,13} \oplus \Delta Z_{11,15} = 0 \\
&⑤ \ \Delta Z_{11,7} \oplus \Delta Z_{11,8} \oplus \Delta Z_{11,12} \oplus \Delta Z_{11,14} \oplus \Delta Z_{11,15} = 0 \\
&⑥ \ \Delta Z_{11,4} \oplus \Delta Z_{11,15} = 0 \\
&⑦ \ \Delta Z_{11,6} \oplus \Delta Z_{11,12} = 0 \\
&⑧ \ \Delta Z_{11,10} \oplus \Delta Z_{11,15} = 0 \\
&⑨ \ \Delta Z_{11,11} \oplus \Delta Z_{11,12} = 0
\end{aligned}
\tag{2}
$$

3.2 암·복호화 라운드 12의 라운드 키(ek_{13}, ek_1) 복구

본 소절에서는 두 개의 랜덤한 1-바이트 오류를 이용하여 (ek_{13}, ek_1)을 복구하는 방법을 소개한다. 공격 아이디어는 공격자가 오류가 발생한 차분의 바이트 위치를 추측한 후, 3.1절에서 구성한 선형식을 이용하여 틀린 후보 라운드 키들을 제거할 수 있다는 것이다. 오류가 발생한 바이트 위치를 틀리게 추측한 경우에는, 18개의 선형식(오류 1개당, 9개의 선형식을 구성)을 통해 높은 확률로 모든 후보 라운드 키가 필터링되며, 오류가 발생한 위치를 정확하게 추측한 경우에만 옳은 라운드 키를 복구할 수 있다.

ARIA는 암·복호화 과정이 동일한 involution 구조로 설계되어 있으며, 단지 복호화 과정에서는 암호화 과정의 라운드 키가 역순으로 사용되기 때문에 ek_1은 ek_{13}과 유사한 방법으로 복구된다. 따라서, 본 소절에서는 복호화 과정의 마지막 라운드 키인 ek_1에 대해서는 세부적인 공격 과정을 생략한다. (ek_{13}, ek_1)을 복구하는 과정은 다음과 같다.

(1) 공격자는 임의의 평문 P에 대해 암호문 C를 획득한다.

(2) 동일한 평문 P에 대해, X_{10}에 1-바이트 오류가 발생한 2개의 암호문 (C_1, C_2)를 얻는다.

(3) (C_1, C_2)의 오류가 발생할 수 있는 256(=16·16)가지 경우에 대해 각각 18개의 선형식을 구성한다.

(4) (C_1, C_2)의 오류 발생 위치를 선택(256가지 중 1가지)한 후, 다음과 같은 과정을 수행하여 후

〔표 1〕 Case III에 대한 공격 과정

추측할 ek_{13}	추측된 ek_{13}	사용하는 선형식	계산 복잡도	필터링 후 후보키 수
$ek_{13,5}, ek_{13,14}$	·	식 (1)-⑨	2^{16}	1
$ek_{13,11}$	$ek_{13,5}, ek_{13,14}$	식 (1)-⑧	2^{8}	2^{-8}
$ek_{13,7}, ek_{13,13}$	$ek_{13,5}, ek_{13,11}, ek_{13,14}$	식 (1)-⑥	2^{8}	2^{-8}
$ek_{13,10}$	$ek_{13,5}, ek_{13,7}, ek_{13,11}, ek_{13,13}, ek_{13,14}$	식 (1)-⑦	1	2^{-16}
$ek_{13,6}, ek_{13,9}, ek_{13,15}$	$ek_{13,5}, ek_{13,7}, ek_{13,10}, ek_{13,11}, ek_{13,13}, ek_{13,14}$	식 (1)-⑤	2^{8}	2^{-8}
$ek_{13,3}, ek_{13,12}$	$ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,9}, ek_{13,10}, ek_{13,11}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	식 (1)-③	2^{8}	2^{-8}
$ek_{13,4}, ek_{13,8}$	$ek_{13,3}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,9}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	식 (1)-④	2^{8}	2^{-8}
$ek_{13,1}$	$ek_{13,3}, ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,8}, ek_{13,9}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	식 (1)-①	1	2^{-16}
$ek_{13,2}$	$ek_{13,1}, ek_{13,3}, ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,8}, ek_{13,9}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	식 (1)-②	1	2^{-16}

보 ek_{13}을 계산한다.

① 1~3개 바이트 단위로 추측할 $ek_{13,i}(0 \le i \le 15)$를 선택한다. 여기서, 추측할 $ek_{13,i}$의 순서는 사용되는 선형식에 따라 정해진다(〔표 1〕, 〔표 2〕 참조).

② 추측한 $ek_{13,i}$로 C와 (C_1, C_2)에서 복구 가능한 바이트를 복호화하여 (C, C_1)과 (C, C_2)에 대한 $\Delta Z_{11,i}$를 각각 계산한다.

③ 단계 ②에서 얻은 $\Delta Z_{11,i}$가 (3)에서 구성된 선형식을 만족하면 단계 (4)-①로 간다. 그렇지 않으면 단계(4)로 가서 (C_1, C_2)의 256가지 중 남은 오류 발생 위치를 선택한다.

(5) 복호화 과정에서 암호문 C에 대해 X_{10}에 1-바이트 오류가 발생한 2개의 평문 (P_1, P_2)를 얻는다.

(6) 단계 (4)와 유사한 방법으로 후보 ek_1을 계산한다.

위의 공격은 오류가 발생한 위치에 따라, 동일한 위치에서 오류가 발생한 경우(Case III)와 서로 다른 위치에서 오류가 발생한 경우(Case IV)로 구분된다. Case III의 경우, 두 오류가 동일한 바이트($X_{10,i}$)에 발생되기 때문에 3.1절에서 언급한 것처럼, $\Delta Z_{11,i}$에 대한 정보를 얻을 수 없다. 따라서 $\Delta Z_{11,i}$에 대응하는 $ek_{13,i}$를 복구할 수가 없기 때문에 120-비트 라운드 키만 복구할 수 있다. 하지만, Case IV의 경우는 두 오류가 서로 다른 바이트에서 발생하였기 때문에 128-비트 라운드 키를 모두 복구할 수 있다. 본 소절에서는 $X_{10,0}$에 오류가 발생한 암호문 C_1, C_2를 얻은 경우(Case III)와, $X_{10,0}$에 오류가 발생한 암호문 C_1과 $X_{10,1}$에 오류가 발생한 암호문 C_2를 얻은 경우(Case IV)에 대해, ek_{13}을 구하는 방법만을 설명한다. 나머지 254가지의 경우도 유사한 방법으로 후보 라운드 키를 계산할 수 있다.

3.2.1 Case III에 대한 공격 과정

$X_{10,0}$에 오류가 발생한 암호문 (C_1, C_2)를 얻었다고 가정한다. ek_{13}을 추측하여 C와 (C_1, C_2)를 각각 복호화한 후, (C, C_1)과 (C, C_2)에 대한 ΔZ_{11}을 계산한다. 그리고 식 (1)과 식 (2)의 선형식을 이용하여 틀린 후보 라운드 키를 필터링한다. 계산 복잡도를 줄이기 위해서 〔표 1〕과 같은 순서로 ek_{13}을 나누어서 추측한다. 첫 번째로 $(ek_{13,5}, ek_{13,14})$를 추측하고, 식 (1)-⑨를 만족하는 $(ek_{13,5}, ek_{13,14})$만을 저장한다. 여기서, 가능한 $(ek_{13,5}, ek_{13,14})$의 수는 2^{16}개이고, 추측된 $(ek_{13,5}, ek_{13,14})$가 식 (1)-⑨를 만족할 확률은 2^{-8}이므로 (C, C_1)과 (C, C_2)에 대해 각각 식 (1)-⑨를 적용할 경우 1개$(=2^{16} \cdot 2^{-8} \cdot 2^{-8})$의 후보$(ek_{13,5}, ek_{13,14})$가 남을 것으로 기대할 수 있다. 앞 단계를 통과한 후보 $(ek_{13,5}, ek_{13,14})$에 대해, $ek_{13,11}$을 추측하여 식 (1)-⑧을 만족하는 후보$(ek_{13,5}, ek_{13,11}, ek_{13,14})$를 저장한다. 이 때 가능한 후보$(ek_{13,5}, ek_{13,11}, ek_{13,14})$의 개수의 기댓값은 $2^{8}(=1 \cdot 2^{8})$이고, 위와 동일하게 두

〔표 2〕. Case Ⅳ에 대한 공격 과정

추측할 ek_{13}	추측된 ek_{13}	사용하는 선형식		계산 복잡도	필터링 후 후보키 수
		(C, C_1)	(C, C_2)		
$ek_{13,5}, ek_{13,14}$	·	식 (1)-⑨	·	2^{16}	2^8
$ek_{13,11}$	$ek_{13,5}, ek_{13,14}$	식 (1)-⑧	·	2^{16}	2^8
$ek_{13,12}$	$ek_{13,5}, ek_{13,11}, ek_{13,14}$	·	식 (2)-⑨	2^{16}	2^8
$ek_{13,6}$	$ek_{13,5}, ek_{13,11}, ek_{13,12}, ek_{13,14}$	·	식 (2)-⑦	2^{16}	2^8
$ek_{13,7}, ek_{13,13}$	$ek_{13,5}, ek_{13,6}, ek_{13,11}, ek_{13,12}, ek_{13,14}$	식 (1)-⑥	·	2^{24}	2^{16}
$ek_{13,10}$	$ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}$	식 (1)-⑦	·	2^{24}	2^{16}
$ek_{13,15}$	$ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}$	·	식 (2)-⑧	2^{24}	2^{16}
$ek_{13,4}$	$ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	·	식 (2)-⑥	2^{24}	2^{16}
$ek_{13,8}$	$ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	·	식 (2)-⑤	2^{24}	2^{16}
·	$ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,8}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	식 (1)-④	·	2^{16}	2^8
$ek_{13,9}$	$ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,8}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	식 (1)-⑤	·	2^{16}	2^8
·	$ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,8}, ek_{13,9}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	·	식 (2)-④	2^8	1
$ek_{13,3}$	$ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,8}, ek_{13,9}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	식 (1)-③	·	2^8	1
·	$ek_{13,3}, ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,8}, ek_{13,9}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	·	식 (2)-③	1	2^{-8}
$ek_{13,2}$	$ek_{13,3}, ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,8}, ek_{13,9}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	식 (1)-②	·	1	2^{-8}
·	$ek_{13,2}, ek_{13,3}, ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,8}, ek_{13,9}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	·	식 (2)-②	2^{-8}	2^{-16}
$ek_{13,1}$	$ek_{13,2}, ek_{13,3}, ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,8}, ek_{13,9}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	식 (1)-①	·	2^{-8}	2^{-16}
$ek_{13,0}$	$ek_{13,1}, ek_{13,2}, ek_{13,3}, ek_{13,4}, ek_{13,5}, ek_{13,6}, ek_{13,7}, ek_{13,8}, ek_{13,9}, ek_{13,10}, ek_{13,11}, ek_{13,12}, ek_{13,13}, ek_{13,14}, ek_{13,15}$	·	식 (2)-①	2^{-8}	2^{-16}

개의 선형식을 사용하므로 이 과정을 통과하는 후보($ek_{13,5}, ek_{13,11}, ek_{13,14}$)의 수의 기댓값은 2^{-8}개 ($=2^8 \cdot 2^{-8} \cdot 2^{-8}$)이다. 모든 과정을 거치면 총 18개의 선형식이 사용되며 $ek_{13,0}$을 제외한 120-비트 라운드 키가 복구된다. 그러나, 가정과 다르게 C_1과 C_2가 $X_{10,0}$에 오류가 주입된 암호문이 아니라면 2^{-16} 확률로 모든 후보 ek_{13}이 필터링 된다. 실제 오류가 발생한 바이트 위치와 오류가 발생했다고 가정한 바이트 위치가 일치할 경우에만 옳은 키가 남을 것이다. 따라서, $ek_{13,0}$을 추측하여 2^8개의 라운드 키 후보가 남는다. 즉, Case Ⅲ과 같이 동일한 바이트 $i(0 \le i \le 15)$에 오류가 발생한 경우에는 $ek_{13,i}$를 제외한 120-비트 라운드 키를 복구할 수 있으며, 결과적으로 2^8개의 라운드 키 후보가 남게 된다.

3.2.2 Case Ⅳ에 대한 공격 과정

$X_{10,0}$에 오류가 발생한 암호문 C_1과 $X_{10,1}$에 오류가 발생한 암호문 C_2를 얻었다고 가정하면, Case Ⅲ과 유사한 방법을 통해 ek_{13}의 후보키를 구할 수 있다. 가정으로부터, (C, C_1)에 대한 $\triangle Z_{11}$에 대해서는 식 (1)의 선형식을 사용하고, (C, C_2)에 대한 $\triangle Z_{11}$에 대해서는 식 (2)의 선형식을 사용한다. 추측하는 ek_{13} 순서는 〔표 2〕와 같다. Case Ⅳ에서는 Case Ⅲ과 달리 128

-비트 ek_{13}을 추측한다. 18개의 선형식을 사용하기 때문에 실제 오류가 발생한 바이트 위치와 오류가 발생했다고 가정한 위치가 일치할 경우에만 옳은 라운드 키가 남는다. 즉, Case Ⅳ과 같이 서로 다른 바이트에 오류가 발생한 경우에는 128-비트 라운드 키를 유일하게 찾을 수 있다.

나머지 254가지 경우에도 실제 오류가 발생한 위치와 오류가 발생했다고 가정한 위치가 동일한 경우에만, 후보 라운드 키가 남는다. 그러므로 가능한 256가지의 오류 발생 위치를 추측하여 $1 \sim 2^8$개의 후보 ek_{13}을 구할 수 있다.

3.3 계산 복잡도 분석

본 공격의 계산 복잡도는 각 선형식을 만족하는 후보 라운드 키의 수와 그 다음 단계에서 추측되는 키 바이트의 수에 의존한다. 즉, [표 1]의 첫 번째 단계에서 2개의 후보 라운드 키 바이트를 추측하므로 2^{16}의 계산 복잡도가 필요하다. 두 번째 단계에서는 첫 번째 단계에서 필터링 후 남은 1개의 후보 라운드 키와 1개의 라운드 키 바이트를 추측해야 하므로 이 단계의 계산 복잡도는 2^8이 된다. [표 1]과 [표 2]에서는 최대 계산 복잡도가 2^{24}이지만 256가지의 모든 경우에 대해 최대 계산 복잡도를 계산하면 2^{32}이다. 따라서, 마지막 라운드 키를 복구하는데 필요한 계산 복잡도는 $O(2^{32})$이다. 본 공격에서 동일한 바이트에 동일한 오류가 주입되는 경우에는 1개의 오류 주입 암호문을 얻은 것과 동일하게 되어 라운드 키를 복구할 수 없으므로 본 공격의 실패 확률은 $2^{-12}(=2^{-4} \cdot 2^{-8})$이다.

3.4 구현결과

본 논문에서 제안하는 공격을 Intel(R) Core (TM)2 CPU 6400 @2.13GHz, RAM 2GB, Visual Studio 2008 환경에서 시뮬레이션 하였다. 구현 결과는 [표 3]과 같다.

라운드 키 (ek_{13}, ek_1) 복구 공격을 각각 100회 실시하였다. Case Ⅲ과 같이 암·복호화 과정에서 X_{10}의 동일한 바이트에 오류가 주입된 경우는 ek_{13}이 6회, ek_1이 10회였다. 16회 모두 2^8개의 ek_{13} 또는 ek_1이 복구되었다. Case Ⅳ처럼 암·복호화 과정에서 X_{10}의 서로 다른 바이트에 오류가 주입된 경우는 ek_{13}이 94회, ek_1이 90회였다. 이 때 복구된 (ek_{13}, ek_1)은 4, 8, 16개였다. 동일한 바이트에 오류가 주입된 경우에는 기대한 것처럼 2^8개의 후보 라운드 키가 남았지만, 서로 다른 바이트에 오류가 주입된 경우에는 기댓값보다 많은 4, 8 또는 16개의 후보 라운드 키가 남았다. 이는 ARIA *S-box*의 차분 분포 특성으로 인해 발생한다. ARIA *S-box*의 차분 분포표의 값은 0, 2 또는 4이다. 즉, 임의의 출력 차분값에 대해 2 또는 4개의 가능한 입력 차분값이 존재한다. 따라서, 18개의 선형식에서 한 번만 필터링 되는 $\Delta Z_{11,i}(0 \le i \le 15)$의 경우 옳은 라운드 키 외에 1개나 3개의 틀린 라운드 키가 함께 필터링 과정을 통과한다. 예를 들어, Case Ⅳ에서 사용된 18개의 선형식에서 $\Delta Z_{11,0}, \Delta Z_{11,1}$은 한 번만 필터링 된다. 즉, $\Delta Z_{11,0}, \Delta Z_{11,1}$의 필터링 과정을 통해 얻게 되는 $(ek_{13,0}, ek_{13,1})$은 옳은 키 이외에 1개나 3개의 틀린 키가 함께 필터링 과정을 통과하게 된다. 따라서, 나머지 14개 바이트$(ek_{13,2}, ek_{13,3}, \cdots, ek_{13,15})$가 유일하게 결정되더라도 4, 8 또는 16개의 후보 라운드 키가 남게 된다.

[표 3] 실험 결과

ek_{13} 복구			
오류 주입 위치	시도횟수	후보 라운드 키 수	성공횟수
동일한 바이트 (Case Ⅲ 유형)	6	2^8	6
서로 다른 바이트 (Case Ⅳ 유형)	94	4	42
		8	33
		16	19
ek_1 복구			
오류 주입 위치	시도횟수	후보 라운드 키 수	성공횟수
동일한 바이트 (Case Ⅲ 유형)	10	2^8	10
서로 다른 바이트 (Case Ⅳ 유형)	90	4	49
		8	31
		16	10

Ⅳ. 라운드 키로부터 비밀키 복구 방법

본 절에서는 3절에서 제안한 방법을 통하여 ek_1과 ek_{13}을 복구하였을 때, 이를 이용하여 ARIA-128의 128-비트 비밀키를 복구하는 방법을 소개한다. ARIA-128의 키스케줄에서 ek_1과 ek_{13}은 다음과 같이 계산된다.

$$ek_1 = (W_0) \oplus (W_1^{\gg 19}). \quad (3)$$

$$ek_{13} = (W_0) \oplus (W_1^{\ll 31}). \quad (4)$$

식 (3)과 식 (4)를 XOR하여 식 (5)를 구할 수 있다.

$$ek_1 \oplus ek_{13} = (W_1^{\gg 19}) \oplus (W_1^{\ll 31}) \quad (5)$$

먼저 예시를 통해 식 (5)에서 W_1을 구하는 방법을 살펴보자.

예시 1. a를 임의의 4-비트 값, M을 4×4 행렬로 정의하였을 때 $(a \ll 1 \oplus a \gg 2) = M(a)$를 만족하는 a를 구해보자.

a는 4개의 변수 $a = a_0 a_1 a_2 a_3, (a_i \in GF(2), 0 \le i \le 3)$로 표현할 수 있다. 이 때, M_1, M_2를 4×4 행렬로 정의하면 $(a \ll 1) = M_1 \cdot (a)$와 $(a \gg 2) = M_2 \cdot (a)$를 만족하는 M_1, M_2는 식 (6)과 같이 구할 수 있다.

$$M_1 = \begin{pmatrix} 0100 \\ 0010 \\ 0001 \\ 1000 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 0010 \\ 0001 \\ 1000 \\ 0100 \end{pmatrix} \quad (6)$$

$(a \ll 1 \oplus a \gg 2) = (M_1 \oplus M_2)(a) = M(a)$이므로 식 (6)을 통해 식 (7)을 구할 수 있다.

$$M = M_1 \oplus M_2 = \begin{pmatrix} 0110 \\ 0011 \\ 1001 \\ 1100 \end{pmatrix} \quad (7)$$

$(a \ll 1 \oplus a \gg 2) = (M_1 \oplus M_2)(a) = M(a)$가 되므로 이 식을 만족하는 a값을 구하기 위하여 M을 가우스 소거법으로 변형하면 $\begin{pmatrix} 1001 \\ 0101 \\ 0011 \\ 0000 \end{pmatrix}$이 된다. 즉, M의 *rank*는 3이고 *nullity*는 1이므로 가능한 a의 개수는 $2^1 = (2^{4-3})$개가 된다.

예시 1과 같은 방법으로 128-비트 W_1을 128개의 변수 $w_1^0, w_1^1, w_1^2 \cdots, w_1^{127} (w_1^j \in GF(2), 0 \le j \le 127)$로 표현하면, 식 (5)의 우변은 식 (8)과 같다. 여기서 M은 128×128 행렬이다.

$$(W_1^{\gg 19}) \oplus (W_1^{\ll 31}) = M \cdot W_1 \quad (8)$$

계산 결과, M의 *rank*는 126, *nullity*는 2임을 확인하였다. 즉, ek_1과 ek_{13}이 주어지면 $2^2 (= 2^{128-126})$개의 가능한 W_1을 계산할 수 있다. 식 (3)이나 식 (4)에 W_1을 대입하여 쉽게 W_0을 구할 수 있으므로 ek_1의 후보키가 i개, ek_{13}의 후보키가 j개가 주어지면 $2^2 \times i \times j$개의 후보 (W_0, W_1)을 구할 수 있다. 또한, W_0, W_1은 ARIA-128의 키스케줄에 따라 식 (9)를 만족해야 한다.

$$W_1 = F_o(W_0, CK_1). \quad (9)$$

임의의 W_0, W_1이 식 (9)를 만족할 확률은 2^{-128}이다. 3.2절에서 언급하였듯이, ek_1와 ek_{13}의 후보수는 각각 최대 2^8개가 남으므로 식 (9)를 통과하는 후보 (W_0, W_1)의 개수는 $2^2 \cdot 2^8 \cdot 2^8 \cdot 2^{-128} = 2^{-110}$이다. 이는 틀린 비밀키가 본 논문에서 제안하는 공격을 통과할 확률이 매우 낮음을 의미한다. 이 과정의 계산 복잡도는 3.2절의 라운드 키 복구 계산 복잡도에 비해 미미하므로 전체 계산 복잡도는 $O(2^{32})$이다.

V. 결 론

2008년 Wei 등은 평균 45개의 오류를 주입하여 ARIA-128에 대한 DFA를 제안하였다. 하지만, 이 공격은 마지막 4개를 라운드 키를 유일하게 찾을 때까지 오류를 주입하여야 하므로 현실적으로 많은 제약점이 있다. 따라서, 본 논문에서는 오류 주입 수를 줄여 좀 더 현실적인 DFA를 제안했다. [표 4]에 제시된 것처럼 본 논문에서는 단지 4개의 오류를 주입하여 128-비트 비밀키를 복구한다.

암·복호화 과정의 라운드 10에 각각 2개의 오류를 주입한 후 라운드 11에서 발생하는 차분 특성을 이용하여 선형식을 구성하고, 이 선형식을 이용하여 암·복호화 마지막 라운드 키 (ek_{13}, ek_1)을 복구하였다. 마지막으로 복구된 라운드 키와 키 스케줄의 특성을 이

[표 4] 공격 결과 비교

참고 문헌	오류 가정	오류 수	계산복잡도
[7]	1 byte random fault	평균 45개	1
본 논문	1 byte random fault	총 4개	2^{32}

용하여 비밀키를 복원하였다. 즉, 4개의 오류를 주입하여 $O(2^{32})$의 계산 복잡도로 ARIA-128에 대한 비밀키를 복구하였다. 이 공격은 기제안된 ARIA-128에 대한 DFA 공격보다 더 적은 오류 주입을 통해 128-비트 비밀키를 복구한다.

참고문헌

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

[2] 정기태, 성재철, 홍석희, "블록 암호 SEED에 대한 차분 오류 공격," 정보보호학회논문지, *20*(4), pp. 17-24, 2010년 8월.

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

[4] P. Dusart, G. Letourneux, and O. Vivolo, "Differntial Fault Attack on AES," *ACNS'03*, LNCS 2846, pp. 293-306, 2003.

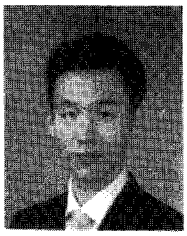
[5] L. Hemme, "A Differential Faut Aanalysis against Early Rounds of (Triple)-DES," *CHES'04*, LNCS 3156, pp. 254-267, 2004.

[6] Daesung Kwon, Jaesung Kim, Sangwoo Park, Soo Hak Sung, Yaekwon Sohn, Jung Hwan Song, Yongjin Yeom, E-Joong Yoon, Sangjin Lee, Jaewon Lee, Seongtaek Chee, Daewan Han, and Jin Hong, "New block cipher ARIA," *ICISC '03, LNCS* 2971, pp. 432 - 445, 2003.

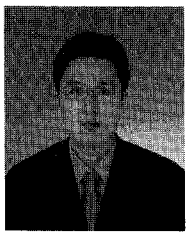
[7] W. Li, D. Gu and J. Li, "Differential fault analysis on the ARIA algorithm," *Information Sciences*, Vol. 178, no. 19, pp. 3727-3737, Oct. 2008.

[8] M. Tunstall1 and D. Mukhopadhyay, "Diff erential Fault Analysis of the Advanced Encryption Standard using a Single Fault," IACR ePrint 2009-575, Nov. 2009.

〈著者紹介〉

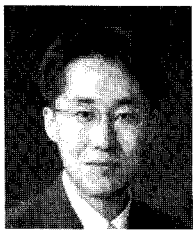


박 세 현 (Sehyun Park) 학생회원
2000년 2월: 육군사관학교 토목공학과 학사
2010년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 대칭키 암호의 분석 및 설계

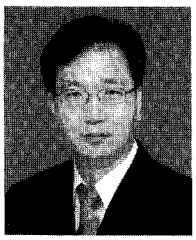


정 기 태 (Kitae Jeong) 학생회원
2004년 2월: 고려대학교 수학과 학사
2006년 2월: 고려대학교 정보보호대학원 석사
2006년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 대칭키 암호의 분석 및 설계

이 유 섭 (Yuseop Lee) 학생회원
2007년 2월: 서울시립대학교 수학과 학사
2007년 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~2008년 8월: 고려대학교 정보보호대학원 조교수
2008년 9월~현재: 고려대학교 정보보호대학원 부교수
〈관심분야〉 대칭키·공개키 암호 분석 및 설계, 컴퓨터 포렌식