Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.6, Dec. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.6.1207

블록 암호 ARIA에 대한 Flush+Reload 캐시 부채널 공격*

배 대 현,[1†] 황 종 배,[1] 하 재 철[2‡]
[1,2]호서대학교 (학생, 교수)

Flush+Reload Cache Side-Channel Attack on Block Cipher ARIA*

Daehyeon Bae,[1†] Jongbae Hwang,[1] Jaecheol Ha[2‡]
[1,2]Hoseo University (Student, Professor)

요 약

하나의 서버 시스템에 여러 운영체제를 사용하거나 사용자간 메모리를 공유하는 클라우드 환경에서 공격자는 캐시 부채널 공격을 통해 비밀 정보를 유출할 수 있다. 본 논문에서는 국내 표준 블록 암호 알고리즘인 ARIA를 사전 연산 테이블 기반 최적화 기법을 이용해 구현할 경우, 캐시 부채널 공격의 일종인 Flush+Reload 공격이 적용되는 것을 확인하였다. ARIA-128을 대상으로 한 Ubuntu 환경에서의 실험 결과, Flush+Reload 공격을 통해 16바이트의 마지막 라운드 키를 찾을 수 있었으며 나아가 마지막 라운드 키와 첫 번째 라운드 키를 이용하면 마스터 키를 찾을 수 있음을 증명하였다.

ABSTRACT

Since the server system in the cloud environments can simultaneously operate multiple OS and commonly share the memory space between users, an adversary can recover some secret information using cache side-channel attacks. In this paper, the Flush+Reload attack, a kind of cache side-channel attacks, is applied to the optimized precomputation table implementation of Korea block cipher standard ARIA. As an experimental result of attack on ARIA-128 implemented in Ubuntu environment, we show that the adversary can extract the 16 bytes last round key through Flush+Reload attack. Furthermore, the master key of ARIA can be revealed from last and first round key used in an encryption processing.

Keywords: Microarchitectural attack, Cache side-channel attack, Flush+Reload Attack, ARIA

I. 서 론

여러 사용자 또는 OS가 하나의 서버 시스템에 존재하는 클라우드 컴퓨팅 환경에서도 보안은 필수적으로 고려해야 할 요소이다. 캐시 부채널 공격은 이러한 클라우드 환경에서 신뢰하지 않는 두 프로세스 사이의 기밀 정보 유출을 가능하게 하는 공격 중 하나이며, 메인 메모리의 접근 시간과 캐시 메모리의 접근 시간이 다르다는 점을 악용한다[1]. 또한, 최근 이러한 캐시 부채널 공격과 CPU의 비순차 실행, 추측 실행의 취약점이 접목된 멜트다운(meltwodn)[2], 스펙터(spectre)[3] 공격이 발견되어 전 세계적으로 큰 화제가 되었다. 이러한 공격들은 소프트웨어에서 CPU의 구조적 특성, 취약점을 악용하기에 소프트웨어 기반 마이크로아키텍처 공격(software based microarchitectural attack)이라고 불린다.

캐시 부채널 공격은 암호 알고리즘의 구현 단계에

Received(09. 24. 2020), Modified(11. 11. 2020) Accepted(11. 12. 2020)
* 이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2020R1F1A1074358)
† 주저자, noeyheadb@gmail.com
† 교신저자, jcha@hoseo.ac.kr(Corresponding author)

서 발생하는 취약점을 이용한 부채널 공격의 일종으로서 최근 Prime+Probe[4], Flush+Reload[5] 공격을 비롯한 많은 공격이 제안되었다. 또한, 이러한 캐시 부채널 공격을 이용해 CPU 캐시를 공유하는 다른 사용자, 다른 OS에서 동작하는 암호 알고리즘의 키를 복구하는 방법도 다수 연구되었다. 대표적으로 블록 암호 AES(Advanced Encryption Standard)를 T-Table을 이용하여 구현한 시스템[6]에서 라운드 키를 복구하는 사례가 있었다[7, 8]. 또한, RSA(Rivest, Shamir, and Adelman)와 같은 공개 키 암호 알고리즘을 대상으로 하는 공격[9, 10]은 개인 키 한 비트씩 복구하는 연구들이 있었다.

본 논문에서는 캐시 부채널 공격 기법 중 하나인 Flush+Reload 공격 기법을 이용하여 국내 표준 블록 암호 알고리즘인 ARIA-128의 마스터 키를 복구할 수 있음을 보이고자 한다. 실험 결과 ARIA-128의 암호화 과정과 복호화 과정을 각각 공격해 암호화 과정에서의 마지막 라운드 키, 복호화 과정에서의 첫 번째 라운드 키를 복구하고 이를 조합해 마스터 키를 계산할 수 있음을 실험을 통해 확인하였다.

II. 배경 지식

2.1 Flush+Reload 캐시 부채널 공격

Flush+Reload 공격(이하 FR공격)은 다른 캐시 부채널 공격과 마찬가지로 캐시 메모리에 접근하는 시간이 메인 메모리에 접근하는 시간보다 짧다는 점을 악용한 공격 방법이다. FR 공격은 캐시 셋 단위로 희생자의 접근을 모니터링하는 Prime+Probe 공격 등과 달리 캐시 라인 단위로 모니터링 할 수 있으므로 더욱 정밀한 공격을 수행할 수 있다. 이러한 FR 공격을 위해서는 공격자와 피해자 사이에 동일한 내용의 페이지를 공유하고 있어야 하며, 접근 여부를 관찰할 명령어 또는 테이블의 메모리 주소를 사전에 알 수 있어야 한다. 또한, 공격자는 공유 라이브러리 파일을 mmap 등의 함수를 통해 공격 프로세스 내부에서 직접 적재할 수 있는 권한을 가져야 한다.

FR 공격 시나리오는 하나의 OS 내에 공격자와 희생자가 존재하는 경우, 그리고 공격자와 희생자가 하나의 하이퍼바이저(hypervisor)에서 동작하는 두 OS에 각각 존재하는 경우(cross-VM)로 크게 2종류가 있다. 하나의 OS 내에 공격자와 희생자가 존재하는 경우, 두 사용자는 공유 라이브러리 등을 통해, Cross-VM 환경의 경우 하이퍼바이저의 메모리 중복제거(memory deduplication) 기능에 의해 페이지 공유가 실현될 수 있다. 메모리 중복제거란, 하이퍼바이저가 주기적으로 메모리를 탐색하여 동일한 내용의 페이지를 공유시킴으로써 메모리 사용량을 줄이는 기능이다. 이는 리눅스의 KVM(Kernel-based Virtual Machine), Xen, VMWare ESXi 등에 구현되어 있다.

FR 공격은 Fig. 1과 같이 크게 3단계 절차로 이루어진다. 첫째로 공격자는 관찰하고자 하는 메모리 라인을 clflush 명령어 등으로 캐시에서 제거하고 대기한다. 이때, 본 논문의 공격은 인텔 x86 아키텍처의 clflush와 같은 특정 캐시 라인만을 제거할 수 있는 명령어가 존재하는 시스템에서 진행해야 한다. 둘째로 공격자 프로세스와 페이지를 공유하고 있는 희생자의 프로세스가 실행된다. 마지막으로 공격자는 앞서 캐시에서 제거한 공유 메모리에 다시 접근하여 소요되는 시간을 측정한다. 이때 희생자 프로세스가 해당 메모리 라인에 접근하였으면 이는 캐시에 적재되어 공격자가 짧은 시간 내에 다시 접근할 수 있다. 반면 희생자 프로세스가 접근하지 않았을 경우 공격자가 다시 접근할 때 상대적으로 긴 시간이 소요될 것이다. 공격자는 해당 정보를 이용해 희생자의 특정 메모리에 대한 접근 여부를 파악해 비밀 정보를 복구할 수 있다.

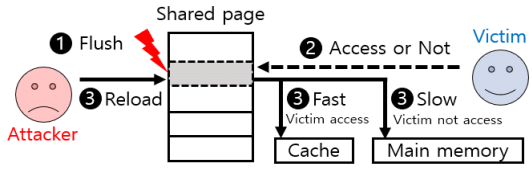


Fig. 1. Overview of Flush+Reload attack.

2.2 ARIA 암호 알고리즘

ARIA는 국가보안기술연구소와 학계, 정부 기관이 함께 개발한 Involutional SPN(Substitution Permutation Network) 구조의 블록 암호 알고리즘으로써 암호화 과정과 복호화 과정이 동일하다. 블록 크기는 128-비트이며 키와 라운드 수는 차례로 128, 192, 256-비트와 12, 14, 16라운드로 총 3

가지 조합을 지원한다. ARIA는 2004년에 국내 표준으로 지정되었으며 알고리즘의 세부 내용은 RFC5794[11] 문서에 기술되어 있다.

2.2.1 암호화 및 복호화 구조

ARIA의 암호화 및 복호화 과정은 Fig. 2와 같이 홀수 라운드는 F_o함수, 마지막 라운드를 제외한 짝수 라운드는 F_e함수, 마지막 라운드는 F_f함수로 구성된다. 마지막 라운드인 F_f를 제외한 F_o, F_e는 LT함수로 이루어진 치환 계층, 16×16 involutional 이진 행렬을 이용한 행렬 곱으로 정의되는 확산 계층으로 구성된다. LT는 4개의 SBox를 이용한 치환 연산으로 이루어진다.

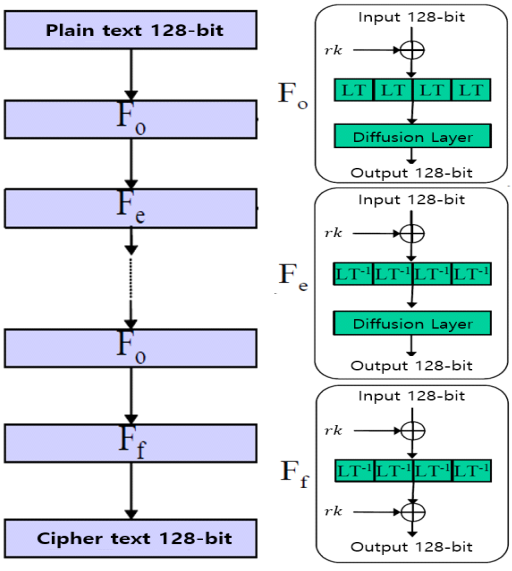


Fig. 2. Encryption process of block cipher ARIA.

2.2.2 키 확장 구조

ARIA의 키 확장은 초기화 과정과 라운드 키 생성 과정의 두 부분으로 나뉜다. 초기화 과정에서는 마스터 키 MK로부터 4개의 128-비트 값인 W_0~W_3을 생성한다. Fig. 3은 초기화 과정을 나타내며 KL과 KR은 각각 128-비트로 구성된다. KL은 MK의 상위 128-비트를, KR은 MK의 129 번째 비트부터 사용하고, 사용할 비트가 없을 경우 0으로 채운다. 이렇게 구성된 KL, KR과 상수 CK_n 그리

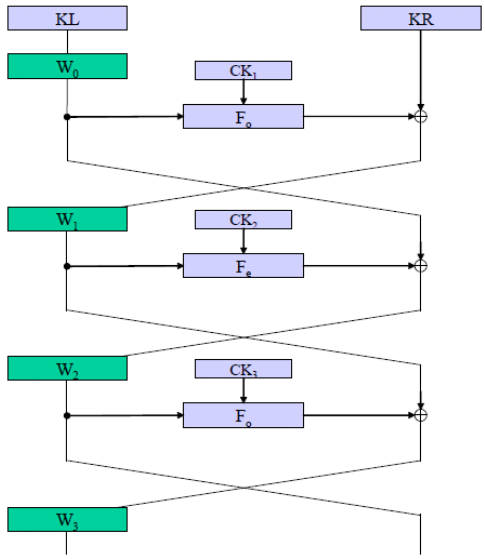


Fig. 3. The initialization phase of $W_0\sim W_3$ blocks for key extension.

고 함수 F_o, F_e를 Fig. 2의 페이스텔 구조의 알고리즘을 이용해 $W_0\sim W_3$을 생성한다.

위 과정을 통해 생성된 $W_0\sim W_3$을 이용해 다음 Table 1과 같이 라운드 키 $rk_1\sim rk_{17}$로 확장한다. ARIA는 마지막 라운드에 2개의 라운드 키가 사용되므로 N_r 라운드의 경우, N_r+1개의 라운드 키가 필요하다.

Table 1. Round key extension using W_0 ~ W_3 in ARIA

rk_n	Calculation	rk_n	Calculation
rk_1	$W_0\oplus(W_1^{\gg 19})$	rk_2	$W_1\oplus(W_2^{\gg 19})$
rk_3	$W_2\oplus(W_3^{\gg 19})$	rk_4	$(W_0^{\gg 19})\oplus W_3$
rk_5	$W_0\oplus(W_1^{\gg 31})$	rk_6	$W_1\oplus(W_2^{\gg 31})$
rk_7	$W_2\oplus(W_3^{\gg 31})$	rk_8	$(W_0^{\gg 31})\oplus W_3$
rk_9	$W_0\oplus(W_1^{\ll 61})$	rk_{10}	$W_1\oplus(W_2^{\ll 61})$
rk_{11}	$W_2\oplus(W_3^{\ll 61})$	rk_{12}	$(W_0^{\ll 61})\oplus W_3$
rk_{13}	$W_0\oplus(W_1^{\ll 31})$	rk_{14}	$W_1\oplus(W_2^{\ll 31})$
rk_{15}	$W_2\oplus(W_3^{\ll 31})$	rk_{16}	$(W_0^{\ll 31})\oplus W_3$
rk_{17}	$W_0\oplus(W_1^{\ll 19})$		

III. ARIA에 대한 Flush+Reload 공격

3.1 공격 대상 ARIA 구현체

ARIA의 소스코드는 현재 한국인터넷진흥원의 암호이용 활성화 사이트에서 공식적으로 배포하고 있다[12]. C언어로 작성된 2개 버전의 구현체와 Java 언어로 구현된 1개의 구현체가 공개되어 있으며, 2개의 C언어 구현체 중 하나는 구현 참조용이며 나머지 하나는 연산 최적화가 적용된 버전이다.

한국인터넷진흥원에서 배포하는 최적화된 ARIA 구현체는 AES의 T-Table 구현체와 비슷하게 사전 연산된 4바이트 원소 256개로 구성된 테이블 4개를 사용하도록 구현되어 있다. 이러한 특성과 더불어 최적화된 소스 코드의 마지막 라운드에 캐시 부채널 공격 취약점이 존재하기에 AES의 T-Table 구현체를 공격하는 방법을 일부 변형하여 ARIA의 최적화된 구현체를 공격할 수 있다. 한편 구현 참조용 ARIA는 해당 테이블을 사용하지 않고 구현되었으며 구조적 취약점이 존재하지 않아 본 논문의 FR 공격이 적용되지 않는다.

다음 Fig. 4는 C언어 기반 최적화된 ARIA의 마지막 라운드 연산 소스 코드이다. 해당 소스 코드는 암호화 과정 및 복호화 과정에서 동일하므로 o는 암호문 또는 복호문 모두 해당될 수 있다. 또한, rk는 라운드 키, $t_0 \sim t_3$은 이전 라운드의 결과값, X_1, X_2, S_1, S_2는 연산 최적화를 위해 사전에 계산된 테이블이다. 사전 연산 테이블 X_1, X_2, S_1, S_2는 각각 4-바이트 원소 256개로 구성된다.

```
o[ 0] = (Byte)(X1[BRF(t0,24)]    ) ^ rk[ 3];
o[ 1] = (Byte)(X2[BRF(t0,16)]>>8) ^ rk[ 2];
o[ 2] = (Byte)(S1[BRF(t0, 8)]    ) ^ rk[ 1];
o[ 3] = (Byte)(S2[BRF(t0, 0)]    ) ^ rk[ 0];
o[ 4] = (Byte)(X1[BRF(t1,24)]    ) ^ rk[ 7];
o[ 5] = (Byte)(X2[BRF(t1,16)]>>8) ^ rk[ 6];
o[ 6] = (Byte)(S1[BRF(t1, 8)]    ) ^ rk[ 5];
o[ 7] = (Byte)(S2[BRF(t1, 0)]    ) ^ rk[ 4];
o[ 8] = (Byte)(X1[BRF(t2,24)]    ) ^ rk[11];
o[ 9] = (Byte)(X2[BRF(t2,16)]>>8) ^ rk[10];
o[10] = (Byte)(S1[BRF(t2, 8)]    ) ^ rk[ 9];
o[11] = (Byte)(S2[BRF(t2, 0)]    ) ^ rk[ 8];
o[12] = (Byte)(X1[BRF(t3,24)]    ) ^ rk[15];
o[13] = (Byte)(X2[BRF(t3,16)]>>8) ^ rk[14];
o[14] = (Byte)(S1[BRF(t3, 8)]    ) ^ rk[13];
o[15] = (Byte)(S2[BRF(t3, 0)]    ) ^ rk[12];

o : cipher text | X1, X2, S1, S2 : pre-computed table
BRF(t, n) : right bit rotate (t >>> n) | rk : round key
t0, t1, t2, t3 : output value of previous round
```

Fig. 4. The source code for the last round computation of optimized ARIA.

Fig. 4에 나타낸 바와 같이 테이블의 결과값이 마지막 라운드 키와 XOR되어 암호문이 결정된다. 따라서 공격자는 희생자의 테이블 X_1, X_2, S_1, S_2에 대한 접근 여부를 통해 테이블 출력값 후보를 추측할 수 있고, 이와 따로 계산된 암호문 후보군을 XOR하여 키 후보군을 추측할 수 있다.

이때, 암호문 후보군은 테이블 맨 앞 16개 항목과 라운드 키가 XOR 되었을 때의 값으로 공격자가 통계적 분석을 통해 추측할 수 있다. 이로 인해 공격자는 라운드 키 후보군에서도 통계적 분석을 통해 가장 높은 빈도로 등장한 바이트를 마지막 라운드 키로 추측할 수 있다.

3.2 실험 환경

최적화된 ARIA에 대한 FR 공격 취약성을 증명하기 위해 본 논문에서는 cross-VM 환경이 아닌 공격자와 희생자가 같은 OS에 있는 상황을 가정해 실험을 진행한다. 실험을 위해 한국인터넷진흥원에서 배포하는 최적화된 ARIA의 C언어 코드를 리눅스 공유 라이브러리로 제작해 사용하여 페이지 공유가 이루어지도록 한다. 또한, 실험의 편의를 위해 공격자와 희생자를 따로 두지 않고 공격자 프로세스 내에서 공유 라이브러리를 통해 암호화 또는 복호화 함수를 호출해 희생자의 역할을 대신한다. 한편 공격자와 희생자를 서로 다른 프로세스로의 분리 여부와 상관없이, 공격자와 희생자가 페이지를 공유하며 이에 접근할 수 있다면 공격에 성공할 수 있다. 마지막으로 본 실험에서 사용한 장비 및 환경은 Table 2와 같다.

Table 2. Experimental equipment used in Flush+Reload attack.

Category	Specification
CPU	Intel i5-2500
OS	Ubuntu 16.04 LTS
Size of L3 cache	6,144 KByte
Size of L3 cache line	64 Byte

3.3 실험 과정 및 공격 알고리즘

공격에 앞서 공격자는 4개의 사전 연산 테이블 X_1, X_2, S_1, S_2의 시작 메모리 주소를 알 수 있어야 한다. 이는 공격자가 공유 라이브러리 파일인 .so 파일(shared object)에 접근해 찾아낼 수 있다. 본 실험에서는 리눅스의 readelf 명령어를 이용해 테이블의 주소를 찾아낸다.

실험에서 사용한 CPU의 각 캐시 라인 크기는 64-바이트이며 사전 연산 테이블의 각 항목의 크기는 4-바이트이다. 즉, 테이블 시작 주소와 연관된 캐시 라인에는 테이블의 맨 앞 16개 항목이 저장된다. 본 논문에서는 암호화 과정 전체 동안 각 테이블의 해당 16개 항목에 대한 접근 여부와 암호문 바이트를 조합해 마지막 라운드 키를 복구할 수 있다. 그러나 ARIA는 키 확장 과정의 특성으로 인해 마지막 라운드 키만으로 마스터 키를 복구할 수 없다[13]. 따라서 암호화 과정과 복호화 과정을 각각 공격해 마지막 라운드 키와 1라운드 키를 복구한다. 다음으로 이 두 라운드 키를 이용해 ARIA-128의 마스터 키를 복구한다.

ARIA는 Involutional SPN 구조이기에 암호화 과정과 복호화 과정이 동일하다. 따라서 라운드 키만 역순으로 사용하는 복호화 과정에서도 본 논문의 공격을 그대로 적용할 수 있다. 복호화 과정에서의 마지막 라운드 키는 암호화 과정에서 첫 번째 라운드 키이기 때문에 이 둘을 이용해 마스터 키를 복구할 수 있게 된다. 본 논문에서는 용어의 통일을 위해 암호화 과정을 대상으로 공격 과정 및 알고리즘을 서술한다.

3.3.1 암호문과 희생자의 캐시 접근 정보 수집

마지막 라운드 키 복구를 위해 필요한 정보의 수집 절차는 Fig. 5에 나타낸 바와 같다. 입력의 Address(T)는 해당 테이블의 시작 메모리 주소를, N_e는 암호화 반복 횟수, TH는 캐시 접근 여부를 판단하기 위한 임계 값이다. 출력의 O는 16-바이트의 암호문 N_e개로 이루어진 2차원 배열이며, $\overline{X_1}$, $\overline{X_2}$, $\overline{S_1}$, $\overline{S_2}$는 입력으로 넘겨준 메모리 주소에 해당하는 각 테이블에 대한 희생자의 접근 여부를 저장하는 길이 N_e의 1차원 배열이다. 즉, 각 테이블의 맨 앞 16개 항목에 대한 희생자의 접근 여부를 나타낸다.

Algorithm 1. Collect cipher text of ARIA and vector of cache access status

INPUT Address(X_1), Address(X_2), Address(S_1), Address(S_2), N_e, TH

OUTPUT O, $\overline{X_1}$, $\overline{X_2}$, $\overline{S_1}$, $\overline{S_2}$

```
1.  K ← fixed_key
2.  rk ← ARIA_Key_expansion(K)
3.  for i=0 to N_e−1 do
4.      clflush(X_1, X_2, S_1, S_2)
5.      O[i]←ARIA(random_16byte, rk)
6.      for t ∈ [X_1, X_2, S_1, S_2] do
7.          if access_cycle(t) 〈 TH then
8.              t̄[i] ← 1
9.          else
10.             t̄[i] ← 0
11.         end if
12.     end for
13. end for
14. return O, X̄_1, X̄_2, S̄_1, S̄_2
```

Fig. 5. Algorithm for collecting cipher text of ARIA and vector of cache access status.

먼저 암호화가 이루어지기 전에 마스터 키로 키 확장을 진행해 라운드 키를 저장해 둔다. 그리고 4개 테이블의 각 첫 번째 16개 항목을 캐시에서 제거한다. 다음으로 ARIA 암호 알고리즘과 고정된 라운드 키 rk를 이용해 암호화를 진행한 후 암호문을 $O[i]$에 저장한다. 마지막으로 공격자는 4개의 테이블에 각각 다시 접근하여, 이때 걸리는 시간을 측정한다. 테이블에 접근하는 시간이 임계 값인 TH보다 작을 경우, 해당 테이블 항목들은 희생자가 암호화 과정동안 접근해서 캐시에 적재되었다고 판단하여 $\overline{T}[i]$에 1을 저장하고 아닐 경우 0을 저장한다.

이 과정을 N_e번 반복하여 N_e개의 암호문 O, 각 N_e개의 항목으로 구성된 배열 $\overline{X_1}$, $\overline{X_2}$, $\overline{S_1}$, $\overline{S_2}$를 수집한다. 이후 과정에서 공격자는 O, $\overline{X_1}$, $\overline{X_2}$, $\overline{S_1}$, $\overline{S_2}$만을 이용하여 마지막 라운드 키를 복구할 수 있다.

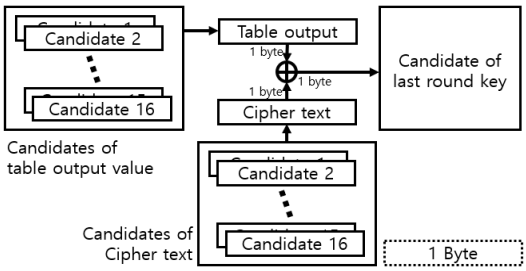


Fig. 6. Attack concept to find last round key candidates using cipher text and table result value.

3.3.2 캐시 접근 정보와 암호문을 통한 라운드키 복구

앞서 3.1에서 언급한 바와 같이 최적화된 ARIA는 마지막 라운드에서 사전 연산 테이블의 결과값이 라운드 키와 XOR 되어 최종 암호문이 생성된다. 즉, Fig. 6에 나타낸 바와 같이 공격자는 마지막 라운드의 사전 연산 테이블 결과 후보군과 암호문 후보군을 추측할 수 있으면, 이 두 후보군을 XOR하여 마지막 라운드 키 후보군을 구할 수 있다. 그리고 마지막 라운드 키 후보군 중에서 가장 많이 등장한 바이트를 라운드 키 1바이트로 결정할 수 있다.

다음 Fig. 7은 마지막 라운드 키 전체 바이트를 복구하는 알고리즘이며 3.3.1에서 수집된 O, $\overline{X_1}$, $\overline{X_2}$, $\overline{S_1}$, $\overline{S_2}$을 이용해 마지막 라운드 키 전체 바이트인 $\overline{rk}$를 찾을 수 있다.

먼저 공격자는 마지막 라운드 키 바이트와 연관된 암호문의 바이트 순서를 찾고, 각 바이트 순서별로 암호문의 빈도를 카운트한다. 라운드 키와 암호문의 바이트 연관성은 앞서 Fig. 4에서 확인할 수 있다. 예를 들어 라운드 키 첫 번째 바이트는 암호문 네 번째 바이트와 연관되어 있다. 공격자는 암호문의 빈도를 카운트하는 도중 암호문 바이트와 연관된 테이블의 접근 여부 $\overline{T}[i](\overline{T} \in [\overline{X_1}, \overline{X_2}, \overline{S_1}, \overline{S_2}])$를 확인한다.

Algorithm 2. Guess last round key using cipher text and vector of cache access status

INPUT O, $\overline{X_1}$, $\overline{X_2}$, $\overline{S_1}$, $\overline{S_2}$, N_e

OUTPUT last round key 16-byte $\overline{rk}$

```
K ← fixed_key
map_rk ← [3,2,1,0,7,6,5,4,11,10,9,8,
          15,14,13,12]
for i=0 to N_e−1 do
    for j=0 to 15 do
        C_all[j][O[i][map_rk[j]]]++
    end for
    if X̄_1[i] = 1 then
        for j=0 to [3,7,11,15] do
            C_hit[j][O[i][map_rk[j]]]++
        end for
    end if
    if X̄_2[i] = 1 then
        for j ∈ [2,6,10,14] do
            C_hit[j][O[i][map_rk[j]]]++
        end for
    end if
    if S̄_1[i] = 1 then
        for j ∈ [1,5,9,13] do
            C_hit[j][O[i][map_rk[j]]]++
        end for
    end if
    if S̄_2[i] = 1 then
        for j ∈ [0,4,8,12] do
            C_hit[j][O[i][map_rk[j]]]++
        end for
    end if
end for
hit_ratio ← C_hit / C_all
for i=0 to 15 do
    I_hit ← argsort_dec(hit_ratio[i])[0...15]
    T_out ← calc_table_output(i)
    for j ∈ I_hit do
        for t ∈ T_out do
            C_key[i][j⊕t]++
        end for
    end for
end for
for i=0 to 15 do
    r̄k[map_rk[i]] ← argmax(C_key[i])
end for
return r̄k
```

Fig. 7. Algorithm for recovering the last round key using cipher text and table access information.

이때, $\overline{T}[i]$가 1이라면, 희생자가 해당 암호문을 최종적으로 생성할 때 테이블의 맨 앞 16개 항목 중 하나를 참조한 것으로 간주하고 해당 암호문의 빈도를 카운트한다. 총 N_e번 위 작업을 거친 후 공격자는 $\overline{T}[i]$가 1일 때 카운트된 암호문 빈도를 전체 암호문의 빈도로 나누어 암호문 바이트별 테이블의 적중률(hit ratio)을 구한다. 이때 테이블 적중률이 높은 암호문들이 암호문 후보군이 된다.

Fig. 7의 5~7-라인에서 모든 암호문의 빈도를 카운트 하고, 8~27-라인에서 희생자가 각 테이블에 접근했을 경우 해당 암호문을 따로 카운트 한다. 그리고 앞서 카운트 한 배열을 바탕으로 29-라인에서 각 암호문 별 테이블 적중률을 계산한다. 그리고 31-라인에서 테이블 적중률이 높은 순서대로 암호문 후보군을 선별한다.

이와 같이 암호문별로 테이블 적중률을 계산하였을 때 적중률이 높다는 것은, 최종 암호문 생성 과정에서 테이블의 맨 앞 16개 항목과 라운드 키가 XOR 되었을 때의 암호문일 확률이 높다는 것을 의미한다. 본 논문에서는 위 과정들을 통해 암호문 후보군을 각 대상 바이트마다 16개씩 선정하였다.

한편 올바른 암호문 후보군이 계산되려면 $\overline{T}[i]$가 희생자의 최종 암호문 생성을 위한 테이블 참조를 의미해야 한다. 예를 들어, $rk[12]$를 찾으려고 할 때, $\overline{T}[i]$는 ARIA의 암호화 과정 전체동안 발생하는 48번의 테이블 참조 중 48번째 테이블 참조 여부와 동일해야 올바른 암호문 후보군이 생성된다. 따라서 통계적으로 각 암호문 별 테이블 적중률이 높으면 올바르게 계산된 암호문 후보군일 확률이 높다.

다음으로 공격자는 테이블 결과값 후보군을 선별해야 한다. 테이블 결과값 후보군은 추가 연산이 필요하지 않으며 각 테이블의 맨 앞 16개 항목이다. 앞서 Fig. 4에 나타낸 바와 같이 테이블 X_1, S_1, S_2는 결과값의 최하위 1바이트를, X_2는 하위 2번째 바이트만을 사용한다. 따라서 공격자는 찾고자 하는 라운드 키 바이트와 연관된 테이블의 결과값 후보군을 쉽게 계산할 수 있다. 테이블 결과값 후보군은 Fig. 8과 같이 테이블마다 16개씩 획득할 수 있다.

이를 바탕으로 공격자는 테이블 결과 값 후보군과 암호문 후보군 모두를 각각 XOR하여 라운드 키 후보군을 생성할 수 있다. 이때, 테이블 결과값 후보군의 개수는 실험 PC의 캐시 라인 크기에 따라 결정되기에 본 실험에서는 16개지만, 암호문 후보군 개수는 공격자가 임의로 설정할 수 있다.

따라서 Fig. 6과 같이 테이블 결과값 후보군 16개와 암호문 후보군 16개를 모두 XOR 하여 256개의 라운드 키 후보군을 생성한다. 이때, 가장 높은 빈도로 등장하는 바이트를 찾고자 하는 마지막 라운드 키의 특정 바이트로 결정할 수 있다. 키 복구 알고리즘을 나타낸 Fig. 7의 30~41-라인이 최종적인 라운드 키를 계산하는 과정이다.

3.4 실험 결과

본 논문에서는 ARIA-128을 대상으로 실험을 진행하며 ARIA 문서의 테스트 벡터를 참고해 "00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff"를 마스터 키로 설정하였다. 또한, 3.3.1의 테이블 접근 정보와 암호문 쌍을 20,000개 수집하여 실험하였다.

실험 결과 ARIA-128의 마지막 라운드 키 각 바

```
Table X1 (first 16 items)
0x52520052, 0x09090009, 0x6A6A006A, 0xD5D500D5, 0x30300030, 0x36360036, 0xA5A500A5, 0x38380038,
0xBFBF00BF, 0x40400040, 0xA3A300A3, 0x9E9E009E, 0x81810081, 0xF3F300F3, 0xD7D700D7, 0xFBFB00FB, …

Table X2 (first 16 items)
0x30300000, 0x68680000, 0x99990000, 0x1B1B0000, 0x87870000, 0xB9B90000, 0x21210000, 0x78780000,
0x50500000, 0x39390000, 0xDBDB0000, 0xE1E10000, 0x72720000, 0x09090000, 0x62620000, 0x3C3C0000, …

Table S1 (first 16 items)
0x00636363, 0x007C7C7C, 0x00777777, 0x007B7B7B, 0x00F2F2F2, 0x006B6B6B, 0x006F6F6F, 0x00C5C5C5,
0x00303030, 0x00010101, 0x00676767, 0x002B2B2B, 0x00FEFEFE, 0x00D7D7D7, 0x00ABABAB, 0x00767676, …

Table S2 (first 16 items)
0xE200E2E2, 0x4E004E4E, 0x54005454, 0xFC00FCFC, 0x94009494, 0xC200C2C2, 0x4A004A4A, 0xCC00CCCC,
0x62006262, 0x0D000D0D, 0x6A006A6A, 0x46004646, 0x3C003C3C, 0x4D004D4D, 0x8B008B8B, 0xD100D1D1, …
```

Fig. 8. Candidate table value of the X_1, X_2, S_1, and S_2.

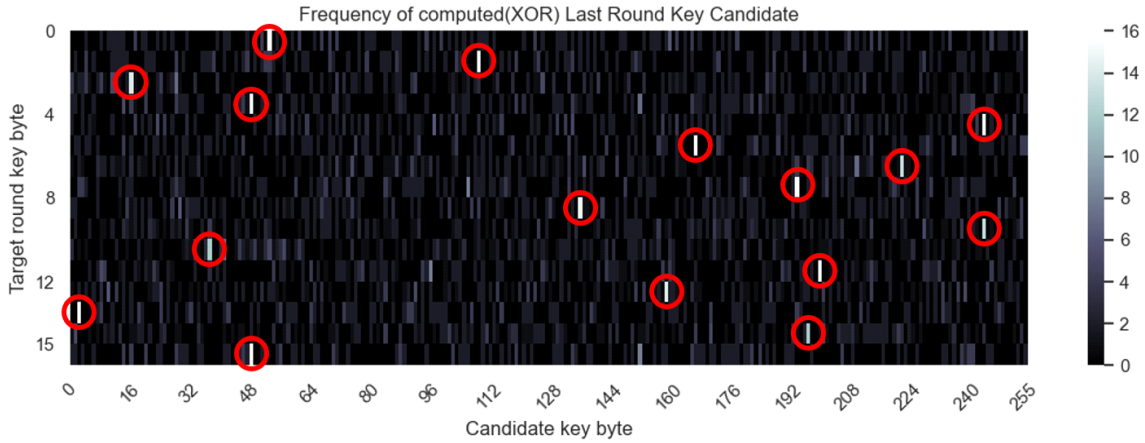


Fig. 9. Frequency of XOR results for the cipher text candidates and table result value candidates for all target round key bytes.

이트별로 암호문 후보군 16개, 테이블 결과값 후보군 16개를 모두 XOR 해준 256개 후보 키의 등장 빈도를 시각화한 것이 Fig. 9이다. 따라서 Fig. 9의 각 행의 총 합은 256이 되며, 각 행에서 뚜렷하게 높은 빈도로 등장하는 값을 해당 라운드 키의 바이트로 추측할 수 있다. Fig. 9를 통해 추측한 16바이트 값이 실제 마지막 라운드 키인 "35 6d 10 30 f4 a7 de c2 88 f4 25 c8 9f 02 c5 30"와 같음을 확인할 수 있었다.

또한, 복호화 과정도 동일한 방법으로 공격해 암호화 과정에서의 첫 번째 라운드 키를 찾을 수 있다. 본 논문에서는 Seo 등[13]의 방법을 이용해 두 개의 라운드 키를 이용해 ARIA-128의 마스터 키인 "00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff"를 복구할 수 있음을 확인하였다.

3.5 수집 샘플 수에 따른 성능 분석

본 논문에서는 수집 샘플의 개수에 따른 바이트 복구 성능을 분석하기 위해, 측정 샘플 수를 1,000개부터 20,000개까지 1,000개 단위로 나누어 실험을 진행하였다. 각 샘플 수에 따라 총 30번 공격을 진행한 후 바이트 복구 정확도의 평균을 계산한 결과 Fig. 10과 같다. 즉, 본 실험 환경에서는 15,000개 이상의 샘플을 수집할 수 있으면 평균 99% 정확도로 마지막 라운드 키 바이트를 복구할 수 있음을 확인하였다.

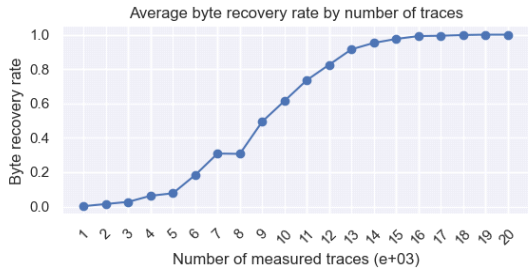


Fig. 10. Average byte recovery rate according to the number of samples.

IV. 결 론

본 논문에서는 국내 표준 블록 암호 알고리즘인 ARIA를 대상으로 캐시 부채널 공격의 일종인 Flush+Reload 공격을 시도하였다. 실험 결과, 공격자에 의해 마지막 라운드 키와 첫 번째 라운드 키가 복구될 수 있고, 이를 통해 ARIA-128의 마스터 키를 복구할 수 있음을 실험을 통해 확인하였다.

한국인터넷진흥원에서 배포하는 구현 참조용 구현체에는 본 논문의 부채널 공격이 적용되지 않지만 사전 연산 테이블을 이용한 최적화된 구현체에는 공격이 적용된다. 따라서, 여러 사용자가 시스템을 공유하는 환경에서 동작하는 암호 알고리즘에 대해 최적화 기법을 적용할 경우 캐시 부채널 공격 등에 대한 안전성을 충분히 검토해야 할 것이다.

References

[1] D. Bernstein, "Cache-Timing Attacks on AES," Available at http://cr.yp.to/antiforgery/cachetiming-20050414.pdf, Apr. 2005.

[2] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," Proceedings of the 27th USENIX Security Symposium, pp. 973-990, Aug. 2018.

[3] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," IEEE Symposium on Security and Privacy, pp. 1-19, May. 2019.

[4] D. Osvik, A. Shamir and E. Tromer, "Cache Attacks and Countermeasure: The Case of AES," CT-RSA'06, LNCS 3860, pp. 1-20, Feb. 2006.

[5] Y. Yarom and K. Falkner, "FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack," Proceedings of the 23rd USENIX conference on Security Symposium, pp. 719-732, Aug. 2014.

[6] J. Daemen and V. Rijmen, The Design of Rijndael: AES - The Advanced Encryption Standard, Springer-Verlag, Berlin, Heidelberg, Jan. 2002.

[7] G. Irazoqui, M. Inci, T. Eisenbarth and B. Sunar, "Wait a minute! A fast, Cross-VM attack on AES," RAID'14, LNCS 8688, pp. 299-319, Sep. 2014.

[8] B. Gülmezoğlu, M. İnci, G. Irazoqui and T. Eisenbarth, "A Faster and More Realistic Flush+Reload Attack on AES," COSADE'15, LNCS 9064, pp. 111-126, Apr. 2015.

[9] M. Inci, B. Gülmezoglu, G. Irazoqui, T. Eisenbarth and B. Sunar, "Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud," IACR Cryptology ePrint Archive, Avaliable at https://eprint.iacr.org/2015/898, 2015.

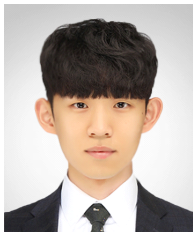
[10] Y. Yarom and N. Benger, "Recovering OpenSSL ECDSA Nonces Using the FLUSH+RELOAD Cache Side-channel Attack," IACR Cryptology ePrint Archive, Avaliable at https://eprint.iacr.org/2014/140, 2014.

[11] J. Lee, J. Kim, D. Kwon and C. Kim, "Description of the ARIA Encryption Algorithm," RFC5794, Mar. 2010.

[12] Korea Information Security Agency, "Block Cipher ARIA," Avaliable at https://seed.kisa.or.kr/kisa/Board/19/detailView.do, Jan. 2019.

[13] J. Seo, C. Kim, J. Ha, S. Moon and I. Park, "Differential Power Analysis Attack of a Block Cipher ARIA," Journal of the Korea Institute of Information Security & Cryptology, Vol. 15, No. 1, pp. 99-107, Feb. 2005.

〈저 자 소 개〉



배 대 현 (Daehyeon Bae) 학생회원
2017년 3월~현재: 호서대학교 컴퓨터정보공학부 학부과정
2020년 3월~현재: 호서대학교 학·석사연계과정
〈관심분야〉 부채널 공격, 암호학, 머신러닝

황 종 배 (Jongbae Hwang) 학생회원
2017년 3월~현재: 호서대학교 컴퓨터정보공학부 학부과정
2020년 3월~현재: 호서대학교 학·석사연계과정
〈관심분야〉 암호학, 부채널 공격, 인공지능 보안

하 재 철 (Jaecheol Ha) 종신회원
1989년 2월: 경북대학교 전자공학과 학사
1993년 8월: 경북대학교 전자공학과 석사
1998년 2월: 경북대학교 전자공학과 박사
1998년 3월~2007년 2월: 나사렛대학교 정보통신학과 교수
2007년 3월~현재: 호서대학교 컴퓨터정보공학부 교수
2013년 1월~현재: 한국정보보호학회 상임부회장
2009년 1월~현재: 한국산학기술학회 이사
〈관심분야〉 정보보호, 네트워크 보안, 부채널 공격