Journal of The Korea Society of Computer and Information
Vol. 23 No. 1, pp. 57-66, January 2018
www.ksci.re.kr
https://doi.org/10.9708/jksci.2018.23.01.057

Key Recovery Attacks on HMAC with Reduced-Round AES

Ga-Yeon Ryu*, Deukjo Hong**

Abstract

It is known that a single-key and a related-key attacks on AES-128 are possible for at most 7 and 8 rounds, respectively. The security of CMAC, a typical block-cipher-based MAC algorithm, has very high possibility of inheriting the security of the underlying block cipher. Since the attacks on the underlying block cipher can be applied directly to the first block of CMAC, the current security margin is not sufficient compared to what the designers of AES claimed. In this paper, we consider HMAC-DM-AES-128 as an alternative to CMAC-AES-128 and analyze its security for reduced rounds of AES-128. For 2-round AES-128, HMAC-DM-AES-128 requires the precomputation phase time complexity of 2^{97} AES, the online phase time complexity of $2^{98.68}$ AES and the data complexity of 2^{98} blocks. Our work is meaningful in the point that it is the first security analysis of MAC based on hash modes of AES.

▸Keyword: AES, HMAC, MAC

I. Introduction

메시지 인증 코드(Message Authentication Code, 이하 MAC)는 동일한 키를 공유하고 있는 송신자와 수신자가 전송하는 메시지의 인증성을 상대방에게 확인시키기 위해서 계산하는 값이다. MAC 알고리즘은 많은 현대의 암호시스템에서 안전한 보안 프로토콜을 위해 중요하게 사용되며, 보통 블록암호 또는 해시함수를 기반으로 설계된다. CMAC(Cipher-based Message Authentication Code)은 대표적인 블록암호 기반 MAC 알고리즘이며, 고정된 길이의 메시지에 대해서만 안전성을 보장하던 CBC-MAC의 단점을 보완하도록 설계되었다. 또한 미국표준기술연구소(NIST)의 권고 MAC 알고리즘으로서 잘 알려져 있다.

CMAC의 안전성은 기반 블록암호의 안전성을 물려받을 가능성이 매우 높다. 예를 들어, AES(Advanced Ecryption Standard)[1]는 전 세계에서 가장 널리 사용되고 있는 블록암호 알고리즘이다. 기반 블록암호 알고리즘에 AES-k (k는 키의 길이: k = 128, 192, 256)을 사용하는 경우, CMAC-AES-k로 표시한다. AES-128에 대한 단일키 공격(Single-key attack)은 7라운드, 연관키 공격(Related-key attack)은 8라운드까지 가능한 것으로 알려져 있다. 이 공격들은 CMAC의 첫 번째 블록에 그대로 적용될 수 있다. AES-128의 전체 라운드 수가 10이므로, AES-128 및 CMAC-AES-128의 안전성 마진(Security margin)은 2-라운드이하인 것으로 생각할 수 있다. AES가 처음 제안되었을 당시 고려되었던 안전성 기준과 비교하면 현재의 안전성 마진은 충분하다고 보기 어렵다.

이것에 대한 대안으로서, 대표적인 해시함수 기반 MAC 알고리즘인 HMAC(Hash-based Message Authentication Code)을 고려해볼 수 있다. HMAC은 ISO/IEC 9797-2 및 FIP 198에 포함된 국제 표준 알고리즘으로서 잘 알려져 있으며, 최근에는 주로 국제 표준 해시함수인 SHA-256을 기반으로 하여 사용되고 있다. 그러나 SHA-256과 AES는 입출력의 길이가 다르기 때문에 HMAC-SHA-256이 CMAC-AES를 대체하기 위해서는 메시지 블록 길이의 조정이나 출력 값 절단 등의 부수적인 작업이 필요하다.

본 논문에서는 CMAC-AES-128에 대한 또 다른 대안으로서, HMAC-DM-AES-128을 고려하고 그것의 안전성을 분석

• First Author: Ga-Yeon Ryu, Corresponding Author: Deukjo Hong
*Ga-Yeon Ryu (gryu720@naver.com), Dept. of Computer Engineering, Chonbuk National University
**Deukjo Hong (deukjo.hong@jbnu.ac.kr), Dept. of Information Technology Engineering , Chonbuk National University
• Received: 2017. 11. 22, Revised: 2017. 12. 04, Accepted: 2018. 01. 02.

한다. HMAC-DM-AES-128은 HMAC의 기반 해시함수에 AES-128의 DM(Davies-Meyer) 해시모드를 적용한 것이다. DM은 블록암호를 안전한 해시함수의 압축함수로 변경하는 PGV(Preneel, Govaerts, Vandewalle) 12가지 방법 중 하나이다[2]. CMAC-AES-128 보다 HMAC-DM-AES-128이 더 안전할 것으로 예상하는 이유는 중간상태값(internal state)에 대한 정보를 얻는 것은 물론, 그 정보를 공격에 이용하는 것도 훨씬 더 어렵기 때문이다. 본 논문에서는 AES-128이 1 라운드 또는 2 라운드로 축소된 경우에 대하여, 전수조사(복잡도 2^{128}) 보다 효율적인 HMAC-DM-AES-128의 키 복구 공격을 연구함으로써, 안전성을 상세하게 분석하였다. 제안된 공격들은 온전한 AES-128 알고리즘이 사용되었을 때의 HMAC-DM-AES-128 키 복구 공격에 적용될 수 있을 것으로 생각되지는 않는다. 하지만, CMAC-AES-128의 공격 가능 라운드 수가 7인 것에 비해 HMAC-DM-AES-128의 공격 가능 라운드 수가 2에 불과하다는 것은 후자가 앞으로 연구 및 개발될 새로운 공격 기법에 대한 저항성이 더 높다는 것을 보여준다. 본 논문의 결과는 HMAC-PGV-AES의 안전성을 평가하기 위한 연구의 시작점이 될 것으로 기대된다.

논문의 구성은 다음과 같다. 2 절에서는 MAC, HMAC, PGV, 해시함수의 안전성과 HMAC의 중간 상태 공격(Internal State Recovery Attack) 등의 연구를 위한 배경지식이 설명되고 3 절에서는 HMAC-DM-AES의 키 복구 공격 과정이 상세히 설명된다. 4 절에서는 결론이 제시된다.

II. Background and Related Works

1. MAC(Message Authentication Code)

메시지 인증 코드(Message Authentication Code, 이하 MAC)는 동일한 키를 공유하고 있는 송신자와 수신자가 전송하는 메시지의 인증성을 상대방에게 확인시키기 위해서 계산하는 값이며 MAC 값 혹은 MAC tag(τ)라고 한다. 암호키를 모르는 사용자는 MAC 값을 계산할 수 없고 암호 시스템의 MAC 알고리즘에 따라 생성한 태그(τ)를 보내는 메시지에 붙여 전송하는데 사용된다. 이는 메시지에 대한 출처를 인증하고 전송 도중에 변조되지 않았음을 확인할 수 있다.

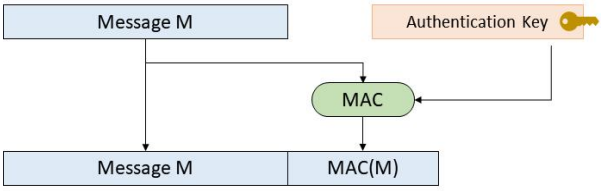

Fig. 1. MAC Algorithm Concept

2. Hash Function

해시함수(Hash Function)는 임의의 길이의 데이터를 고정된 길이의 데이터로 맵핑하는 함수이다. 해시 함수가 반환하는 값을 해시 값, 해시 코드, 해시라고 한다. 서로 다른 입력 데이터가 같은 해시 값으로 매핑 되었을 때 충돌(Collision)이 발생하였다고 한다. 암호학적 해시함수는 역상(Preimage), 제2역상(2nd Preimage), 충돌쌍(Collision)에 대한 3가지 안전성을 요구한다.

1) 역상 저항성(Preimage Resistance)

임의의 z가 주어질 때, H(x) = z인 x를 찾는 것이 어려워야 한다.

2) 제2역상 저항성(2nd-Preimage Resistance)

임의의 x가 주어질 때, H(y) = H(x)인 y를 찾는 것이 어려워야 한다. (y와 x는 같지 않다.)

3) 충돌 저항성(Collision Resistance)

H(x) = H(y)인 x, y를 찾는 것이 어려워야 한다. (x와 y는 같지 않다.)

해시 함수는 전송된 데이터의 무결성을 보장하는 데 사용되며 메시지 인증을 제공하는 HMAC(Hash-based MAC)의 기본 구성 요소이다.

3. Compression Function

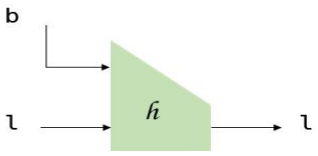

Fig. 2. Compression Function

암호학에서 단방향 압축함수는 두 개의 고정 길이 입력을 고정 길이 출력으로 변환하는 함수이다. 압축함수는 암호화 해시함수 내부를 Merkle-Damgård 구조로 구성할 때 사용된다.

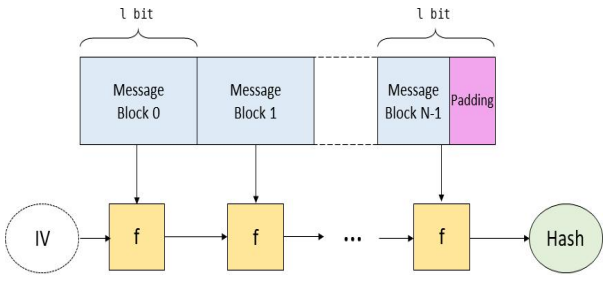

Fig. 3. Merkle-Damgård Hash Construction

어떠한 길이의 입력 값이라도 받아들여 고정된 길이의 값을 출력하는 함수를 직접 설계하는 것은 매우 어렵다. Merkle-Damgård 구조는 고정된 길이의 입력 값을 받아들여 고정된 길이로 출력하는 압축함수를 반복적으로 사용하여 긴 길이의 임의의 입력이라도 처리할 수 있는 해시 함수로 확대하

여 설계하는 방법이다. Merkle-Damgård 구조에서 압축함수의 출력 값을 연쇄 변수(Chaining Variable)이라고 하며 중간 상태 값(Internal State) 라고 한다. 압축함수의 입력이 되는 블록 크기가 l비트 일 때, 긴 길이의 메시지를 l비트 단위로 쪼개고 남은 메시지에 전체 메시지 길이 정보를 포함한 패딩을 붙인다. 이러한 패딩 방법을 MD Strengthening이라고 한다[3].

메시지 인증코드 생성 방법의 하나인 HMAC도 Merkle-Damgård 구조로 이루어져 있다.

4. HMAC(Hash-based Message Authentication Code)

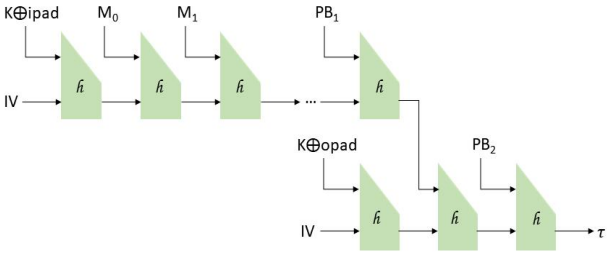

Fig. 4. The Structure of HMAC

HMAC은 ANSI, IETF, ISO, NIST에 의해 매우 표준화되었으며 MAC 알고리즘 중 가장 보편적으로 알려진 알고리즘 중 하나이다. 주로 은행 업무 처리나 인터넷 프로토콜(SSL, TLS, SSH)에서 사용된다. NMAC의 단일 키 버전으로, 상수 값(ipad, opad)을 사용하여 두 개의 키(K^{in}, K^{out})을 생성한다[4]. HMAC은 송수신자 사이에 비밀키를 공유하고 있는 상태에서 MAC을 메시지에 붙여 사용하며 신뢰성과 무결성을 보장하는데 사용된다. Hash를 HMAC의 기반 해시함수라 하자. ipad와 opad는 각각 상수이고, K는 HMAC의 비밀키, M은 HMAC에 입력되는 메시지라고 하자. (이 때, ipad, opad, K는 Hash의 압축함수의 메시지 블록과 길이가 같다.) 그러면 비밀키 K가 주어졌을 때, 메시지 M에 대한 MAC값 HMACK(M)은 다음과 같이 계산된다. :

$$HMACK(M)=Hash((K\oplus opad)||Hash((K\oplus ipad)||M)) \quad (1)$$

(1)에서 첫 번째 적용된 Hash를 내곽(Inner) 해시함수, 두 번째 적용된 Hash를 외곽(Outer) 해시함수로 구분한다.

5. PGV(Preneel, Govaerts, Vandewalle)

PGV는 블록 암호를 기반으로 안전한 해시 함수를 구성하는 방법이다. Preneel, Govaerts, Vandewalle[2]는 블록 길이와 키 길이가 같은 블록 암호로부터 압축함수를 구성하는 64가지 방법을 고려했고 그 중에서 안전한 12가지 PGV 압축함수를 제안하였다. 이 후, Black 등이 [5]에서 그 12가지 PGV 압축함수의 안전성을 수학적으로 증명했다. 본 논문에서는 12가지 PGV에서 가장 많이 쓰이는 PGV 중 하나인 DM(Davies-Meyer) 방식을 사용한다(Fig. 5).

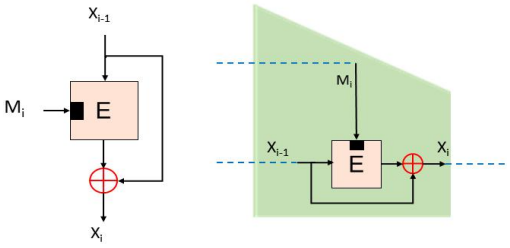

Fig. 5. The Structure of DM(Davies-Meyer)

PGV-DM 구조에서는 메시지나 패딩 블록을 블록 암호의 키 대신 입력하고, 이전 상태 값(Xi-1)을 블록 암호의 평문 값 대신 입력한 후 암호문과 XOR 연산하여 다음 상태 값(Xi)으로 출력한다. Fig. 5는 해시함수의 압축함수로 DM 방식의 블록암호를 넣었을 때의 모습을 나타낸다.

6. AES(Advanced Ecryption Standard)

AES[1]는 고급 암호화 표준으로 2001년 미국 표준 기술 연구소(NIST)에 의해 제정된 암호화 방식이며 암호화와 복호화 과정에서 동일한 키(Key)를 사용하는 블록 암호 알고리즘이다.

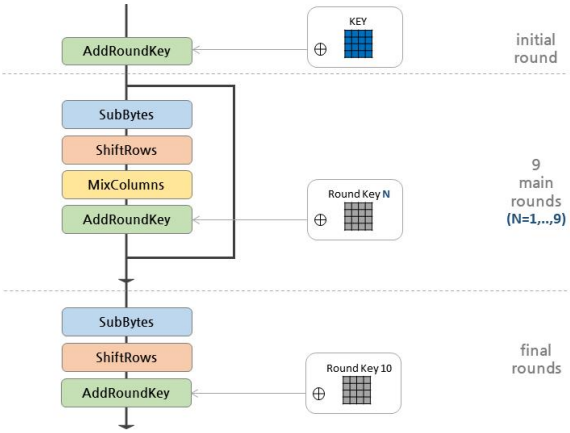

Fig. 6. AES-128 Encryption Process

AES는 총 10 라운드로 구성되어 있으며, 각 라운드는 SB(SubBytes), SR(ShiftRows), MC(MixColumns), ARK(Add RoundKey)로 구성된다. 단, 마지막 라운드에서는 MC를 제외한 3단계만 수행된다. SB 단계는 S-box를 이용한 바이트(Byte) 단위의 블록 교환이 이루어지고 SR 단계에서는 단순히 행과 행을 치환한다. MC에서는 열(Row)에 속한 모든 바이트를 순환 행렬을 통해 각 바이트로 대체한다. ARK는 Fig. 알고리즘을 통해 구해진 각 라운드 키를 현재 블록과 비트(bit)별로 XOR 연산하는 단계이다. 평문 블록과 첫 번째 라운드 입력 사이에 ARK가 한 번 적용되므로, AES-128에서는 11개의 라운드 키 RK_0, RK_1, ..., RK_{10}가 사용된다.

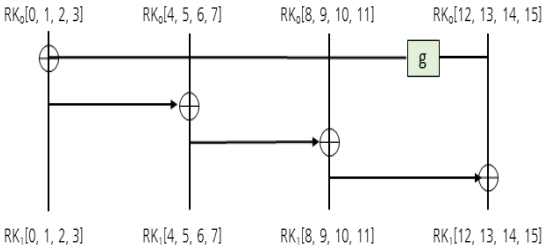

Fig. 7. AES-128 Key Schedule Algorithm

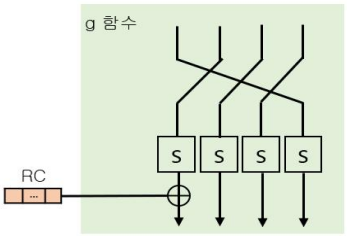

Fig. 8. g Function Structure

AES의 키 스케줄 알고리즘은 4바이트씩 차례로 묶어서 4개의 워드(word, 4byte)를 구성한 후 Fig. 7과 같이 XOR한다. 각 라운드 상수 RC(Round Constant)는 4바이트 값으로 가장 오른쪽의 3바이트는 모두 0인 값이다.

7. Internal State Recovery Attack on HMAC

Leurent의 연구[6]에 따르면 HMAC의 중간 상태 값(Internal State)를 공격 복잡도 2^{96}으로 복구 할 수 있다. 안전한 해시함수가 사용되었다는 가정에서 HMAC은 생일 한계(Birthday Bound)까지 안전한 것으로 증명되어 있으며[7], 기반 해시 함수가 l-비트 출력 길이를 가질 때 HMAC에 대해 $2^{l/2}$의 복잡도를 갖는 존재론적 위조 공격(Existential Forgery Attack)이 가능하다.

7.1 Functional Graph

n비트를 n비트로 매핑하는 랜덤 함수 f의 경우, 랜덤 함수의 크기 N = 2n이다. Flajolet과 Odlyzko의 두 가지 이론을 바탕으로 랜덤 함수 f의 반복으로 정의되는 함수의 그래프 구조에 대해 알 수 있으며 크기 N의 랜덤 매핑에서의 기대치는 다음과 같다. :

$$\text{Tail length}(\lambda) = \sqrt{\pi N/8}$$
$$\text{Cycle length}(\mu) = \sqrt{\pi N/8}$$
$$\text{Rho length}(\rho = \lambda + \mu) = \sqrt{\pi N/2}$$

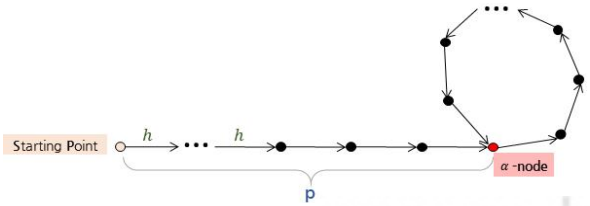

Fig. 9. Randomly Mapped Graph at the Starting Point

크기 N의 랜덤 매핑에서 임의의 시작점(Starting Point. SP)를 선택하고 함수 f를 반복함으로써, SP에서 시작하는 함수 그래프의 경로를 따라가면 높은 확률로 SP가 속한 컴포넌트 내의 사이클에 들어가게 되며 이 때, 컴포넌트는 임의의 시작점으로부터 함수를 반복하여 거치는 모든 노드의 그룹을 말한다. 이 경로의 모든 노드(혹은 포인트)의 개수를 Tail length라고 하며 Cycle을 구성하는 노드의 개수를 Cycle length라고 한다. Tail length와 Cycle length를 합한 전체 노드의 개수를 Rho(ρ) length라고 하며 경로의 Tail과 Cyclse을 연결하는 결합 지점(Join Point)의 노드를 α-node 라고 한다.

7.2 Internal State Recovery on HMAC

Leurent가 제안한 HMAC의 중간 상태 복구 공격[6]은 충돌 찾기, 충돌의 결합 지점 찾기, MAC 충돌이 발생하는 메시지 쌍 찾기, MAC 충돌 쌍의 결합 지점 찾기, 중간 상태 찾기의 5 단계로 구성된다.

1) 충돌(Collision) 찾기

해시 값이 l비트일 때, 어떤 임의의 포인트를 $2^{l/2}$번 매핑하면 생일 공격에 근거하여 매우 높은 확률로 Cycle에 들어가게 된다.

임의의 시작점 P를 2^t개 시도하여 메시지가 0으로 이루어진 블록([0]으로 표현)을 사용한 h 함수를 2^s번 반복한 결과 값을 비교하여 충돌을 찾는다. 이 때, 발견되는 충돌의 기대 개수는 $2^c (= 2^{2t-(l-s)})$이다.

2) 충돌의 결합 지점 X찾기

1)에서 2^t개를 시도하였으므로 충돌을 여러 개 발견할 수 있다. 이 때, 각 충돌 지점은 Main Cycle 내부에 위치하고 있는 포인트이며 중간에 이미 한 번 충돌 한 값일 수 있으므로 Cycle에 들어가게 되는 결합 지점 즉, α-node를 찾는다. 충돌 노드의 시작점에서 다시 h[0]을 반복하여 충돌의 결합 지점 X를 찾으면 X에서 [0] 이외에 충돌을 일으키는 다른 블록 쌍(m_X, m_X')을 찾는다. 이 때, 2^l만큼의 압축함수 계산이 요구된다.

3) MAC 충돌이 발생하는 메시지 쌍 찾기

다음의 메시지(M)를 2^t개 구성하여 같은 HMAC값을 갖는 메시지 쌍을 찾는다.

$$M(i) = [i] \| [0]^{2^s} \quad (i = 1,2,...,2^t)$$

단계 1에서와 마찬가지로 2^c개의 충돌이 기대된다.

4) MAC 충돌 쌍의 결합 지점 찾기

각 충돌 (τ_i, τ_j)에 대해, MAC([i]||[0]^k) = MAC([j]||[0]^k)을 만족시키는 최소의 k를 계산하여 충돌 쌍의 결합 지점을 찾는다. 이 때, 이진 검색(Binary Search) 방법을 사용하며 과정은 다음과 같다. :

① $x_1 = 0$, $x_2 = 2^s$로 초기화한다.

② $x' = (x_1 + x_2)/2$

각 충돌 index(i, j)에 대해,

$MAC([i]||[0]^{x'}) = MAC([j]||[0]^{x'})$ 인지 즉, 충돌이 발생하는지 오라클에 질의한다.

발생하면 $x_2 = x'$, 발생하지 않으면 $x_1 = x'$을 수행한다.

③ $x_1 = x_2$가 될 때까지 ①, ②를 반복한다.

5) 중간 상태(Internal State) 찾기

$[i]||[0]^k = \mu_{ij}$라고 할 때, 단계 2에서 구한 각 (m_X, m_X')에 대해 $\mu_{ij}||m_X$와 $\mu_{ij}||m_X'$의 HMAC 값을 질의하여 MAC 충돌이 발생한다면 매우 높은 확률로 μ_{ij} 이후의 포인트는 중간 상태 X가 된다.

이 공격은 본 논문의 AES-128 기반 HMAC의 키 복구 공격을 위한 중간 상태 복구에 사용되며, 요구되는 공격 복잡도는 단계별로 Table 1과 같이 계산될 수 있다. 이 때 $c = l/2 - s$ 이고, $t = 3l/4 - s$ 이다. HMAC-DM-AES-128에서 $l = 128$이므로 $s = l/4$으로 놓으면, 중간 상태 복구에 오프라인 시간 복잡도 2^{97} 번의 압축함수 연산, 온라인 계산 복잡도 2^{96} 번의 압축함수 연산, 데이터 복잡도 2^{96} 블록, 메모리 복잡도 2^{64}가 소요된다.

Table 1. Attack Complexity of Internal State Recovery Attack on HMAC(Memory Complexity $2^{l/2}$)

Step	Attack Complexity	Complexity
1	$2^{t+s} = 2^{3l/4}$	Offline Phase
2	$2^{c+l/2} = 2^{l-s}$	Offline Phase
3	$2^{t+s} = 2^{3l/4}$	Data
4	$s \times 2^{c+s} = s \times 2^{l/2}$	Online Phase
5	$22^{c+s} = 2^{l-s}$	Online Phase

III. Key Recovery Attacks on HMAC-DM-AES-128

본 논문에서는 공격 대상인 HMAC-DM-AES에서 AES의 라운드를 축소시킨 1, 2-라운드에 대한 키 복구 공격을 시도한다. 앞서 설명했듯이, Leurent의 중간 상태 복구 공격[6]에 의해 내곽 해시함수의 결과 값을 알 수 있다. 이 값은 이후에 외곽 해시함수에서 키를 복구하는 데 사용된다.

1. Key Recovery Attacks on HMAC based with 1-Round AES-128

Fig. 4에서 내곽 해시함수의 값이 외곽 해시함수의 입력 값으로 사용된다. 키 복구는 외곽 해시함수의 가장 오른쪽 압축함수에서부터 거꾸로 중간 값을 복구하며 시도된다.

1.1 Y Recovery

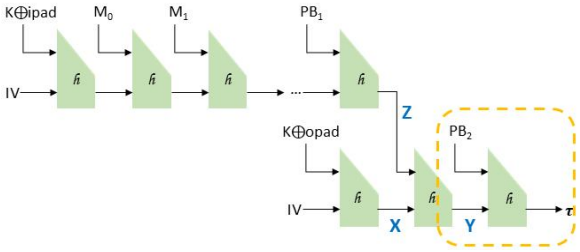

Fig. 10. Concepts to Recover Y on HMAC Outer Hash Function

먼저 외곽 해시함수에서 가장 오른쪽 압축함수의 Y값을 계산한다. 이 때 공격자는 PB_2의 값과 MAC값인 τ 값을 알고 있다. 압축함수가 PGV-DM 구조로 이루어져 있고 Fig. 8의 점선 박스를 구체화 한 것이 Fig. 11의 점선 박스이다.

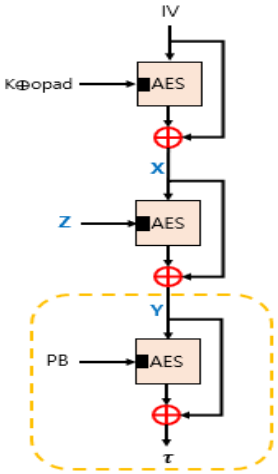

Fig. 11. DM Structure on Outer Hash Function

(1) 1-라운드 AES 구조

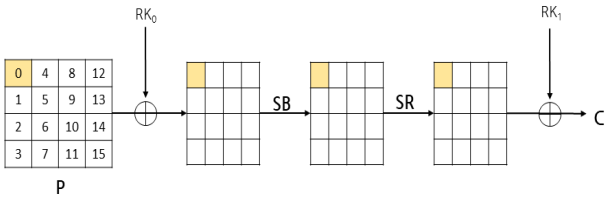

Fig. 12. The Structure of 1-Round AES

AES는 마지막 라운드에서 MC(MixColumns) 연산을 하지 않기 때문에 첫 번째 라운드가 마지막 라운드인 1-라운드 AES의 구조는 Fig. 12와 같다. 평문 P는 계산하고자 하는 Y값에 해당한다. 이 때 MAC 값은 PB_2를 AES의 Key로 사용하고 Y를 평문 입력으로 사용하여 나온 AES 암호화 값(Fig. 10의 C)에 Y를 다시 XOR한 값이다. 식으로 표현하면 다음과 같다. :

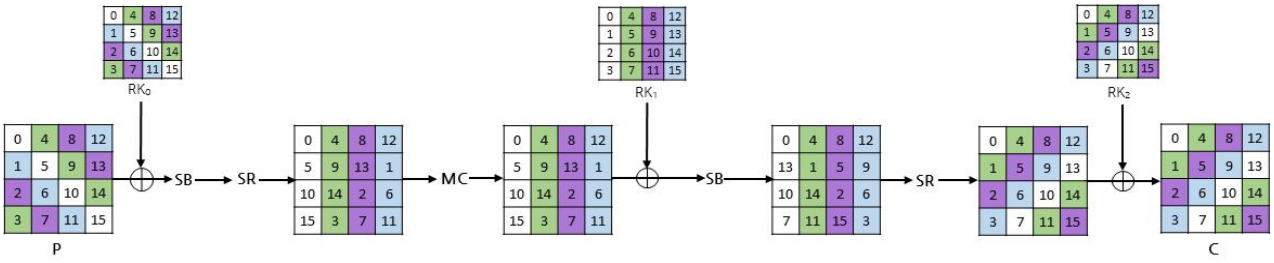

Fig. 15. The Structure of 2-Round AES

$$\tau = E_{PB_2}(Y) \oplus Y$$
$$\tau[i] = S(Y[i] \oplus PB[i]) \oplus RK_1[i] \oplus Y[i]$$
(단, $RK_0 = K$ 이고 $i = 0,1,...,15$ 이다.)

여기서 S는 S-box계산을 의미하고 RK_0은 원래의 키 값과 같은 값이다. 공격자가 모르는 값은 Y값뿐이므로 Y[i]에 대해 2^8번의 전수조사를 시행하고, 이를 16번 반복한다. Y값 전체를 계산하는 데 $2^8 \times 16 = 2^{12}$만큼의 계산이 수행된다.

1.2 X Recovery

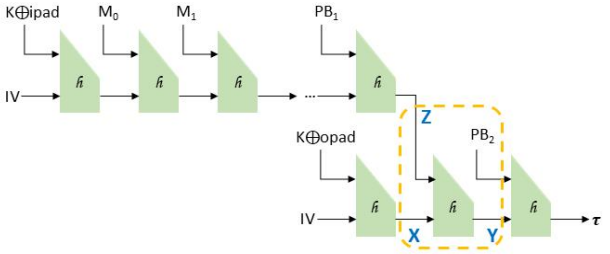

Fig. 13. Concepts to Recover X on HMAC Outer Hash Function

이전 단계에서 Y값을 2^{12}의 S-box 계산으로 복구 할 수 있었다. Fig. 13에서 Z값은 Leurent의 중간 상태 복구 공격[6]에 의해 복구 가능한 내곽 해시함수의 해시 값에 해당하며, 점선 박스에 해당하는 압축함수에서 공격자가 알지 못하는 값은 X값뿐이다. 이는 단계 (1)에서 Y값을 계산하던 방법과 동일한 방법으로 X값을 복구할 수 있다는 것을 뜻하며 X값 또한 2^{12}의 S-box계산으로 복구 할 수 있다.

1.3 Key K Recovery

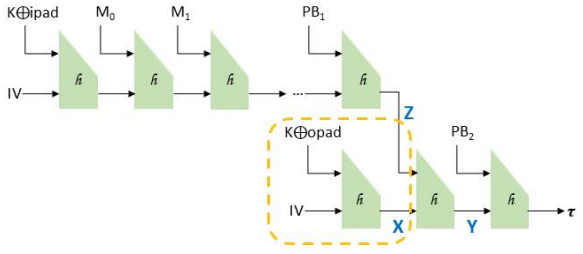

Fig. 14. Concepts to Recover Key K on HMAC Outer Hash Function

최종적으로 키 K값을 복구하기 위해 Fig. 14의 압축함수를 살펴본다. 공격자는 상수 값인 opad와 IV과 이전 단계에서 복구한 X값을 미리 알고 있다. RK_0과 RK_1에 대해 u, v로 나타내고 Fig. 11의 DM 구조를 참고하여 다음과 같은 식을 세울 수 있다. :

$$(RK_0[i], RK_1[i]) = (u, v)$$
$$X[i] = S(IV[i] \oplus u \oplus opad[i]) \oplus v \oplus IV[i]$$
$$v = X[i] \oplus IV[i] \oplus S(IV[i] \oplus u \oplus opad[i])$$

위 식에 따라 ($RK_0[0]$, $RK_1[0]$)의 후보 값을 28개 획득할 수 있다.

여기서 키 스케줄 알고리즘에 의해 필요한 계산을 줄일 수 있다. Fig. 7과 Fig. 8을 보면 $RK_0[0]$과 $RK_1[0]$로부터 $RK_0[13]$의 후보 값을 계산할 수 있다. 따라서 ($RK_0[0]$, $RK_1[0]$, $RK_1[13]$)에 대한 후보 값이 2^8개 존재한다. 마찬가지로 $RK_0[4]$의 추측 값에 대해, $RK_1[4]$를 계산할 수 있다. Fig. 7에 따라 $RK_0[4] \oplus RK_1[4] = RK_1[0]$를 만족시키는 $RK_1[0]$의 후보 값을 찾아 연결시키면 ($RK_0[0]$, $RK_0[13]$, $RK_0[4]$)의 후보 값을 2^8개 획득 할 수 있다. 같은 방법으로 $RK_0[8] \oplus RK_1[8] = RK_1[4]$식에 따라 $RK_1[4]$의 후보 값을 찾아 연결시키면 ($RK_0[0]$, $RK_0[13]$, $RK_0[4]$, $RK_0[8]$) 4개 바이트 값의 후보 값을 단 2^8개로 추릴 수 있다. 나머지 12바이트에 대해서도 같은 방법을 이용하면 후보 값 총 2^{32}개를 획득 할 수 있다. :

$RK_0[0]$, $RK_0[13]$, $RK_0[4]$, $RK_0[8]$ 후보 값 2^8개 수집
$RK_0[1]$, $RK_0[14]$, $RK_0[5]$, $RK_0[9]$ 후보 값 2^8개 수집
$RK_0[2]$, $RK_0[15]$, $RK_0[6]$, $RK_0[10]$ 후보 값 2^8개 수집
$RK_0[3]$, $RK_0[12]$, $RK_0[7]$, $RK_0[11]$ 후보 값 2^8개 수집

따라서 전체 키 후보의 수는 2^{128}에서 2^{32}로 줄어든다. 새로운 (M, τ)쌍에 대해, 남은 키 후보를 전수조사 하면 키 복구가 가능하다.

1.4 Attack Complexity

모든 공격 과정에서 Leurent의 중간 상태 복구 공격의 비용이 다른 부분들을 크게 상회하므로, 키 복구 공격의 총 복잡도는 2.6.2절에서 설명된 것과 유사하게 다음과 같이 계산된다. :

- 사전계산 복잡도 = 2^{97} 1-Round AES,
- 온라인 계산 복잡도 = 2^{96} 1-Round AES,
- 데이터 복잡도 = 2^{96} 블록
- 메모리 복잡도 = 2^{64}

2. Key Recovery Attacks on HMAC based with 2-Round AES

HMAC-DM의 기반 블록 암호로서 2-라운드 AES-128이 사용된다고 가정하면, 키 복구 과정은 Fig.10의 Y값 계산, X값 계산, 키 K 값 복구 순서로 이루어지며 1-라운드 AES이 사용될 때의 복구 과정과 계산 순서가 같다.

Fig. 15는 2-라운드 AES의 내부 구조이다. P로부터 정방향 계산과 C로부터의 역방향 계산을 통해 중간 값을 비교하여 값을 찾아내는 방법을 사용했다. 2-라운드부터는 MixColumns 연산이 포함되므로 4바이트씩 묶어서 계산하는 것이 효과적이라고 판단하였다.

2.1 Y Recovery

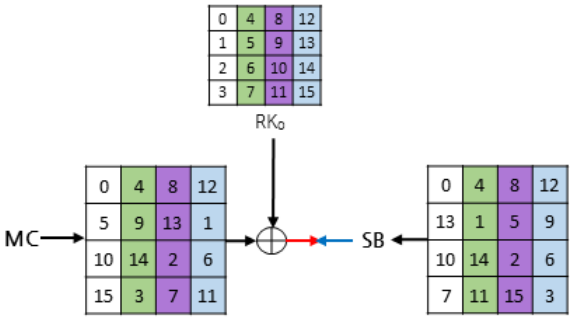

Fig. 16. The Structure of 2-Round AES to Recover Y

정방향 계산과 역방향 계산이 만나는 지점을 비교하여 식을 세울 수 있다.

(1) Y값 1열 계산

4개의 열중에 첫 번째 열로 세운 식은 다음과 같다. :

$$\mathrm{M}\begin{pmatrix} S(Y[0]\oplus RK_0[0]) \\ S(Y[5]\oplus RK_0[5]) \\ S(Y[10]\oplus RK_0[10]) \\ S(Y[15]\oplus RK_0[15]) \end{pmatrix} \oplus \begin{pmatrix} RK_1[0]) \\ RK_1[1]) \\ RK_1[2]) \\ RK_1[3]) \end{pmatrix}$$

$$= \begin{pmatrix} S^{-1}(Y[0]\oplus\tau[0]\oplus RK_2[0]) \\ S^{-1}(Y[13]\oplus\tau[13]\oplus RK_2[13]) \\ S^{-1}(Y[10]\oplus\tau[10]\oplus RK_2[10]) \\ S^{-1}(Y[7]\oplus\tau[7]\oplus RK_2[7]) \end{pmatrix}$$

이 때, M은 MC과정에서 사용되는 행렬을 뜻하며 좌변은 괄호에 쌓인 행렬과의 곱셈을 표현한 것이다.

먼저 (Y[0], Y[5], Y[10], Y[15])의 모든 가능한 값을 추측한다. 우변과 같은 좌변을 만들어내는 (Y[5], Y[10], Y[15])은 2^{16}개 존재하고 Y[0]은 2^8개이므로 $2^8 \times 2^{16} = 2^{24}$개를 테이블에 저장한다. 좌변과 우변에 중복되는 값 Y[0]과 Y[10]을 제외한 (Y[7], Y[13])의 모든 가능한 값을 추측하여 우변 식을 계산하고 테이블 값과 비교하여 일치하는 (Y[0], Y[5], Y[7], Y[10], Y[13], Y[15])을 올바른 값의 후보 값으로 남겨둔다. 예상 개수는 $2^{16} \times 2^{16} \times 2^{-16} = 2^{16}$개 이다. RK1 XOR 이후 부분에서, 정방향 계산과 역방향 계산의 왼쪽 첫 번째 열을 비교하였을 때, 결과가 다음과 같다. (T1은 후보 값을 저장한 테이블1) :

2^{16} 개 (Y[0], Y[5], Y[7], Y[10], Y[13], Y[15]) $\in T_1$

(2) Y값 2열 계산

1열 계산과 동일하며 RK1 XOR 이후 정방향 계산과 역방향 계산의 왼쪽 두 번째 열을 비교하였을 때, 결과가 다음과 같다. (T2은 후보 값을 저장한 테이블2) :

2^{16} 개 (Y[1], Y[3], Y[4], Y[9], Y[11], Y[14]) $\in T_2$

(3) Y값 3열 계산

Fig. 16의 세 번째 열에 대해 (1)단계와 같이 식을 세울 수 있다. Y[2], Y[8]의 모든 가능한 값을 추측하고 T_1의 2^{16}개 값들과 함께 조사하여, 식을 만족하는 (Y[0], Y[2], Y[5], Y[7], Y[8], Y[10], Y[13], Y[15])를 찾는다. 즉, RK_1 XOR 이후 정방향 계산과 역방향 계산의 왼쪽 세 번째 열을 비교하였을 때, 결과가 다음과 같다. :

1개의 (Y[0], Y[2], Y[5], Y[7], Y[8], Y[10], Y[13], Y[15]) 값 획득

(3) Y값 4열 계산

Fig. 15의 세 번째 열에 대해 (1)단계와 같이 식을 세울 수 있다. Y[6], Y[12]의 모든 가능한 값을 추측하고 T_2의 2^{16}개 값들과 함께 조사하여, 식을 만족하는 (Y[1], Y[3], Y[4], Y[6], Y[9], Y[11], Y[12], Y[14])를 찾는다. 즉, RK1 XOR 이후 정방향 계산과 역방향 계산의 왼쪽 네 번째 열을 비교하였을 때, 결과가 다음과 같다. :

1개의 (Y[1], Y[3], Y[4], Y[6], Y[9], Y[11], Y[12], Y[14]) 값 획득

2.2 X Recovery

이전 단계에서 Y값을 계산하던 방법과 동일한 방법으로 복구 가능하며 X값 또한 두 번의 2^{24} S-box 계산과 두 번의 2^{16}의 테이블 계산이 요구된다.

2.3 Key K Recovery

키 값을 복구하기 위해서, 앞의 Y, X값의 계산과 달리 RK_1

에 대하여 정리한다. 1-라운드와 마찬가지로 키 스케줄 알고리즘의 구조에 따라 후보 값의 개수를 줄이는 방법을 사용한다. Fig. 10에 따라 키 복구를 위한 식은 $E_{K\oplus opad}(IV)\oplus IV = X$ 이며 이 때, 공격자는 상수 값 opad와 IV, 앞서 2.2.에서 복구한 X 값을 알고 있다.

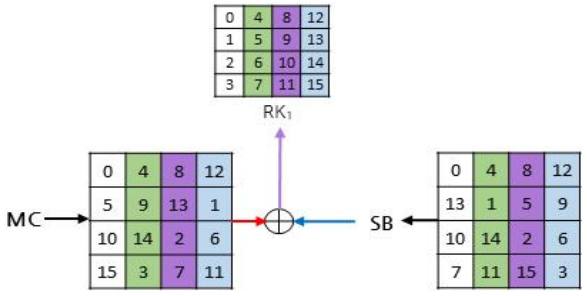

Fig. 17. The Structure of 2-Round AES to Recover Key K

Fig. 17을 참고하여 정방향, 역방향 값을 XOR계산 한 결과를 RK1과 비교하여 다음과 같은 식을 세울 수 있다. :

$$M\begin{pmatrix} S(IV[0]\oplus RK_0[0]) \\ S(IV[5]\oplus RK_0[5]) \\ S(IV[10]\oplus RK_0[10]) \\ S(IV[15]\oplus RK_0[15]) \end{pmatrix} \oplus \begin{pmatrix} RK_1[0]) \\ RK_1[1]) \\ RK_1[2]) \\ RK_1[3]) \end{pmatrix}$$

$$= \begin{pmatrix} S^{-1}(IV[0]\oplus X[0]\oplus RK_2[0]) \\ S^{-1}(IV[13]\oplus X[13]\oplus RK_2[13]) \\ S^{-1}(IV[10]\oplus X[10]\oplus RK_2[10]) \\ S^{-1}(IV[7]\oplus X[7]\oplus RK_2[7]) \end{pmatrix}$$

(1) 키 K 복구 1단계

Fig. 17에 따라 RK1에 대하여 정방향 계산과 역방향 계산의 왼쪽 첫 번째 열을 XOR 하였을 때, 결과가 다음과 같다. :

2^{64}개의 (RK0[0, 5, 10, 15], RK_1[0, 1, 2, 3], RK_2[0, 7, 10, 13]) 후보 값 $\in T_{단계1}$

이 때, 키 스케줄 구조를 통해 다음 두 개의 식이 정의 된다.(Fig. 7) :

$$RK_0[0]\oplus RK_1[0] = RK_0[13]$$
$$RK_1[0]\oplus RK_2[0] = RK_1[13]$$

(2) 키 K 복구 2단계

Fig. 17에 따라 RK_1에 대하여 정방향 계산과 역방향 계산의 왼쪽 두 번째 열을 XOR 하였을 때, 결과가 다음과 같다. :

2^{64}개의 (RK_0[3, 4, 9, 14], RK_1[4, 5, 6, 7], RK_2[1, 4, 11, 14]) 후보 값 $\in T_{단계2}$

이 때, 키 스케줄 구조를 통해 다음 식이 정의 되며 이 식 전체를 <수식 ①>이라고 한다.(Fig. 7) :

$$RK_0[5]\oplus RK_1[1] = RK_1[5]$$
$$RK_1[0] = RK_1[4]\oplus RK_0[4]$$
$$RK_2[0] = RK_1[4]\oplus RK_2[4]$$

효율적인 탐색을 위해, 위 <수식 ①>의 좌변 값을 정렬한 2^{24} 크기의 보조 테이블 $T_{단계1}{}^0$를 고려한다. 그리고 $T_{단계1}$은 $T_{단계1}{}^0$의 각 값에 따라 Fig. 18과 같이 정렬된다고 하자. 그러면, 평균적으로 각 $T_{단계1}{}^0$ 값에 2^{40}개의 $T_{단계1}$ 값들이 대응된다. :

$T_{단계1}{}^0[g] = RK_0[5]\oplus RK_1[1], RK_1[0], RK_2[0],$
$T_{단계1}[i][j] = RK_0[...]RK_1[...]RK_2[...]$, i = $T_{단계1}{}^0[g]$

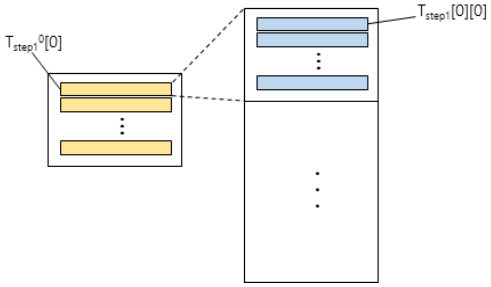

Fig. 18. The Structure of $T_{step1}{}^0$ and T_{step1}

$T_{단계2}$도 $T_{단계1}$과 같은 방법으로 보조 테이블 $T_{단계2}{}^0$를 이용하여 정렬할 수 있다. $T_{단계2}{}^0$에는 <수식 ①>의 우변이 테이블 값으로 저장된다. 2단계 최종 키 후보 값은 다음 알고리즘을 통해 획득한다.

• 2단계 키 후보 값을 찾는 알고리즘

```
1  for(<수식 ①>의 모든 우변 값) ∈ T단계2⁰ :
2    flag ← 0
3    if(<수식 ①>의 우변 값 ∈ T단계1), flag=1
4    if (flag=1)
5      for kb∈T단계2[<수식 ①>의 모든 우변 값] :
6        for ka∈T단계1[<수식 ①>의 모든 좌변 값] :
7              T단계2최종에 kc=ka∪kb 저장
```

위 알고리즘의 7행에서, $k_c = k_a \cup k_b$은 k_a와 k_b 요소들의 합집합으로 구성된 값을 의미한다. 즉, 테이블 $T_{단계2최종}$에는 다음과 같은 키 후보가 저장된다. :

(RK_0[0, 3, 4, 5, 9, 10, 14, 15], RK_1[0, 1, 2, 3, 4, 5, 6, 7, 13], RK_2[0, 1, 4, 7, 10, 11, 13, 14]) $\in T_{단계2최종}$

위 알고리즘의 실행에

$2^{24}\times(\log 2^{24} + 2^{40}\times 2^{40}) = 2^{24}\times(24+2^{80}) \approx 2^{104}$ 번의 테이블 참조가 요구되며, $T_{단계2최종}$에는 2^{104} 개의 후보 값이 저장된다.

(3) 키 K 복구 3단계

Fig. 17에 따라 RK1에 대하여 정방향 계산과 역방향 계산의 왼쪽 세 번째 열을 XOR 하였을 때, 결과가 다음과 같다. :

2^{64}개의 (RK_0[2, 7, 8, 13], RK_1[8, 9, 10, 11], RK_2[2, 5, 8, 15]) 획득 $\in T_{단계3}$

키 K 복구 2단계에 의해, $T_{단계2최종}$ 에는 2^{104}개의
(RK_0[0, 3, 4, 5, 9, 10, 14, 15],
RK_1[0, 1, 2, 3, 4, 5, 6, 7, 13],
RK_2[0, 1, 4, 7, 10, 11, 13, 14]) 후보 값이 저장되어 있다. 이 때, 키 스케줄 구조를 통해 다음 식이 정의되며 이 식 전체를 <수식 ②>이라고 한다. :

$$RK_1[4] \oplus RK_0[5] = RK_1[8]$$
$$RK_2[4] = RK_1[8] \oplus RK_2[8]$$
$$RK_1[5] \oplus RK_2[1] = RK_2[5]$$
$$RK_1[3] \oplus RK_1[7] = RK_0[7]$$
$$RK_0[9] \oplus RK_1[5] = RK_1[9]$$
$$RK_0[10] \oplus RK_1[6] = RK_1[10]$$
$$RK_2[11] \oplus RK_2[7] = RK_1[11]$$
$$RK_2[13] \oplus RK_1[13] = RK_1[9] \oplus RK_2[5]$$
$$RK_2[14] \oplus RK_0[14] \oplus RK_2[10] = RK_1[10]$$
$$RK_0[15] \oplus RK_2[11] = RK_1[11] \oplus RK_2[15]$$

Fig. 18과 마찬가지로 $T_{단계2최종}$에 위 <수식 ②>식의 좌변 값을 정렬한 2^{80} 크기의 보조 테이블 $T_{단계2최종}{}^0$을 고려한다. :

$T_{단계2최종}{}^0$[g] = <수식 ②>식의 좌변 값
$T_{단계2최종}$[i][j] = $RK_0[...]RK_1[...]RK_2[...]$

3단계 최종 키 후보 값은 다음 알고리즘을 통해 획득한다.

• 3단계 키 후보 값을 찾는 알고리즘

```
1  for k^a ∈ T단계3 :
2    flag ← 0
3    if(<수식 ②>의 좌변 값 ∈ T단계최종2), flag=1
4    if (flag=1)
5      for k_b∈T단계2최종[<수식 ②>의 모든 우변 값] :
6            T단계3최종에 k_c=k_a∪k_b 저장
```

위 알고리즘의 실행에
$2^{64} \times (\log 2^{80} + 2^{24}) = 2^{64} \times (80 + 2^{24}) \approx 2^{88}$ 번의 테이블 참조가 요구되며, $T_{단계3최종}$에는 2^{88} 개의 후보 값이 저장된다.

(4) 키 K 복구 4단계
Fig. 17에 따라 RK1에 대하여 정방향 계산과 역방향 계산의 왼쪽 네 번째 열을 XOR 하였을 때, 결과가 다음과 같다. :

2^{64}개의 (RK_0[1, 6, 11, 12], RK_1[12, 13, 14, 15], RK_2[3, 6, 9, 12]) 획득 $\in T_{단계4}$

이전 단계에서 사용했던 방법과 마찬가지로 $T_{단계3최종}$에 대한 두 개의 테이블을 생성한다.
이 때, 키 스케줄 구조를 통해 다음 식이 정의 된다. :

$$RK_i[n] = RK_{i-1}[n] \oplus RK_i[n-4]$$

$4 \le n \le 15$ 이고 $i = 1,2$ 일 때, 만들 수 있는 식의 개수는 전체 식에서 2, 3단계에서 사용한 식을 제외한 11개이며 위 알고리즘의 실행에 $2^{64} \times \log 2^{88} \approx 2^{70.46}$ 번의 테이블 참조가 요구된다. 그리고 $T_{단계4최종}$ 에는 2^{64} 개의 후보 값이 저장된다. 새로운 (M, τ)쌍에 대해, 남은 키 후보를 전수조사 하면 키 복구가 가능하다.

2.4 Attack Complexity

2라운드 AES가 적용된 HMAC에서 키 K를 복구 할 때,
$2^{64} + 2^{104} + 2^{88} + 2^{64} \approx 2^{104}$번의 테이블을 참조한다. 2-라운드 AES의 1회 연산에 2*(16+ 4) = 40번의 S-box 연산(= 테이블 참조)이 필요하므로 Leurent의 중간 상태 복구 공격과 본 논문에서 제안된 키 복구 과정을 고려한 공격의 총 복잡도는 다음과 같이 계산 된다. :

- 사전계산 복잡도 = 2^{97} 2-Round AES
- 온라인 계산 복잡도 = $2^{98.68}$ 2-Round AES
- 데이터 복잡도 = 2^{96} 블록
- 메모리 복잡도 = 2^{104}

IV. Conclusions

본 논문에서는 AES의 DM(Davies-Meyer) 해시 모드를 기반으로 구성되는 HMAC인, HMAC-DM-AES의 키 복구 공격에 대한 안전성을 분석하였고, 1-라운드 및 2-라운드로 축소된 버전의 AES를 가정할 경우 전수조사 공격 보다 효율적으로 키 복구가 가능함을 보였다. 결과는 Table 2와 같다.

Table 2. Attack Complexity

Complexity	1 Round	2 Round
Precomputation	2^{97} 1-Round AES	2^{97} 2-Round AES
Online	2^{97} 1-Round AES	$2^{98.68}$ 2-Round AES
Data	2^{96} Blocks	2^{96} Blocks
Memory	2^{64}	2^{104}

이 공격들이 온전한 AES-128 알고리즘이 사용되었을 때의

HMAC-DM-AES-128에 대한 키 복구 공격에 적용될 수 있을 것으로 생각되지는 않는다. 하지만, CMAC-AES-128의 공격 가능 라운드 수가 7인 것에 비해 HMAC-DM-AES-128의 공격 가능 라운드 수가 2에 불과하다는 것은 후자가 앞으로 연구 및 개발될 새로운 공격 기법에 대한 저항성이 더 높다는 것을 보여준다. 가장 많이 사용되고 있는 AES 기반 MAC인 CMAC에 비해 HMAC-DM-AES는 기존의 블록암호 분석 공격들의 적용이 어렵다는 장점이 있기 때문에, 현재까지는 HMAC-DM- AES를 CMAC의 더 안전한 대안 알고리즘으로서 고려될 수 있다. 게다가, 향후 추가 연구를 통하여 HMAC-PGV-AES의 안전성에 대한 더욱 정밀한 분석 결과를 얻을 수 있을 것으로 기대된다.

Acknowledgements

이 논문은 2017년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.B0722-16-0006, (창조씨앗-2단계)암호와 물리계층보안을 결합한 IoT 네트워크 보안 기술 개발)

REFERENCES

[1] FIPS-197, Announcing the ADVANCED ENCRYPTION STANDARD (AES), 2009.
[2] Preneel, Govaerts and Vandewalle, “Hash functions based on block ciphers: a synthetic approach”, In: Stinson D.R. (eds) CRYPTO 1993. LNCS, vol 773. Springer, Berlin, Heidelberg, 1993.
[3] Merkle and Damgård, “A Design Principle for Hash Functions”, CRYPTO '89 Proceedings, LNCS, vol.435, Brassard, ed, Springer, 1989, pp. 416-427.
[4] Bellare, “New Proofs for NMAC and HMAC: Security without Collision-Resistance”, In Dwork, C., ed.: CRYPTO. Vol.4117 of Lecture Notes in Computer Science., Springer (2006) 602-619
[5] Black, Rogaway and Shrimpton, “Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV”, In: Yung M. (eds) CRYPTO 2002. LNCS, vol 2442. Springer, Berlin, Heidelberg.
[6] Leurent, Peyrin and Wang, “New Generic Attacks against Hash-Based MACs” In:Sako K., Sarkar P.(eds) ASIACRYPT 2013. LNCS, vol.8270, Springer, Berlin, Heidelberg, 2013.
[7] Menezes, Oorschot and Vanstone, Handbook of Applied Cryptography. CRC Press, 1996.
[8] Derbez, Fouque and Jean, 2013. “Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting”, EUROCRYPT 2013, LNCS 7881, pp.371-387, 2013.
[9] Bellare and Kohno “Hash function balance and its impact on birthday attacks”, IACR, 2003

Authors

Ga-Yeon Ryu received the B.S. in department of Information Technology and Engineering from Chonbuk National University, Korea, in 2016. She is currently a M.S. Student in department of Computer Engineering at Chonbuk National University, Korea. She is interested in cryptography and digital forensics.

Deukjo Hong received B.S. and M.S. degrees in mathematics from Korea University in 1999 and 2002, respectively, and Ph.D. degree in information security from Korea University in 2006. From 2007 to 2015, he has worked at ETRI. He is now an assistant professor of department of information technology & engineering at Chonbuk National University. He is interested in cryptography and network security.