제 7 장 해쉬함수

7.1 개요

해쉬함수(hash function)는 임의의 길이에 이진 문자열을 고정된 길이의 이진 문자열로 매핑하여 주는 함수를 말하며, 해쉬함수의 결과를 해쉬값, 메시지 다이제스트, 메시지 지문이라 한다. 해쉬함수는 메시지 다이제스트, 메시지 지문과 같은 이름에서 알 수 있듯이 임의의 길이의 메시지를 대신 대표할 수 있는 고정된 길이의 값을 계산하여 주는 함수를 말한다. 따라서 해쉬함수는 크게 다음과 같은 요구사항을 충족해야 한다.

- **요구사항 1. 압축**: 임의의 길이의 이진 문자열을 일정한 크기의 이진 문자열로 변환해야 한다.
- **요구사항 2. 계산의 용이성**: x가 주어지면 $H(x)$는 계산하기 쉬워야 한다.
- **요구사항 3. 일방향성**(one-wayness): 입력을 모르는 해쉬값 y가 주어졌을 때, $H(x') = y$를 만족하는 x를 찾는 것은 계산적으로 어려워야 한다.
- **요구사항 4. 약한 충돌회피성**(weak collision-resistance): x가 주어졌을 때 $H(x') = H(x)$인 $x'(\neq x)$을 찾는 것은 계산적으로 어려워야 한다.
- **요구사항 5. 강한 충돌회피성**(strong collision-resistance): $H(x') = H(x)$인 서로 다른 임의의 두 입력 x와 x'을 찾는 것은 계산적으로 어려워야 한다.

일반적인 압축함수는 임의의 메시지를 현재 크기보다 작은 크기로 축소해줄 수 있어야 할 뿐만 아니라 나중에 다시 원래의 크기로 복원할 수 있어야 한다. 하지만 해쉬함수는 압축을 하지만 복원을 할 필요가 없고, 복원이 가능해서도 안 된다. 임의의 길이의 메시지를 고정된 길이의 메시지로 변환하므로 함수의 정의역 집합보다 치역 집합이 훨씬 크다. 따라서 서로 다른 메시지가 같은 해쉬값으로 매핑되는 것은 불가피하다. 이처럼 서로 다른 메시지가 같은 해쉬값으로 매핑되면 **충돌**(collision)이 발생하였다고 한다. 해쉬값은 입력 메시지를 대표하는 값이므로 서로 다른 메시지가 동일한 값에 의해 대표되면 그 역할을 하기 어렵다. 따라서 안전한 해쉬함수가 되기 위해서는 이와 같은 충돌을 찾는 것이 어려워야 한다. 충돌은 크게 약한 충돌과 강한 충돌로 구분되는데, 약한 충돌은 정해진 어떤 해쉬값으로 매핑되는 또 다른 메시지를 찾는 것이고, 강한 충돌은 같은 해쉬값으로 매핑되는 임의의 메시지 쌍을 찾는 것이다. 예를 들어 학생들이 모여 있는 집합에서 특정한 학생을 선택한 후에 이 학생과 같은 생일인 학생을 찾는 것이 약한 충돌이고, 모여 있는 집합에서 생일이 같은 임의의 두 학생을 찾는 것이 강한 충돌이다. 따라서 강한 충돌을 찾는 것이 더 쉽다. 그러므로 강한 충돌을 찾는 것이 어려운 해쉬함수가 가장 안전한 해쉬함수이다. 일방향성까지 충족되는 해쉬함수를 **일방향 해쉬함수**(OWHF, One-Way Hash Function)라 하며, 강한 충돌회피성까지 충족하는 해쉬함수를 **충돌회피 해쉬함수**(CRHF, Collision-Resistant Hash Function)라 한다.

해쉬함수는 크게 세 가지 용도로 사용된다. 첫째, 전자서명에서 사용된다. 전자서명 암호알고리즘은 보통 공개키 암호알고리즘을 많이 사용한다. 따라서 매우 큰 길이의 메시지 전체를 개인키로 암호화하여 전자서명하기에는 비용이 많이 소요된다. 따라서 전체 메시지 대

신에 그 메시지를 대표하는 짧은 길이의 해쉬값에 서명하는 경우가 많다. 원 메시지 대신에 메시지의 해쉬값을 서명하여 동일한 효과를 얻기 위해서는 기존 전자서명 요구사항이 여전히 충족되어야 한다. 이 때 강한 충돌회피성이 보장되지 않는 해쉬함수를 사용하면 기존 서명 블록을 다른 메시지의 서명 블록으로 활용할 수 있게 된다. 둘째, 메시지의 무결성을 제공하기 위해 사용된다. 셋째, 패스워드를 안전하게 유지하기 위해 사용된다.

유닉스 시스템에서는 사용자의 계정명을 passwd라는 파일에 유지하며, 초기에는 이 파일에 각 계정의 패스워드도 평문 상태로 함께 보관하였다. 이와 같은 형태로 패스워드를 유지할 경우에는 이 파일에 접근 권한을 획득하면 모든 계정의 패스워드를 알 수 있게 된다. 이에 passwd 파일에서 패스워드 부분을 shadow라는 파일로 옮기고 평문 형태로 유지하던 패스워드 대신에 그것의 해쉬값을 유지하는 형태로 바꾸었다. 이 경우 사용자가 패스워드를 입력하면 그것의 해쉬값을 계산하여 유지하고 있는 해쉬값과 비교하여 올바른 패스워드가 입력되었는지 검사한다. 따라서 기존과 달리 이 파일에 대한 접근 권한을 획득하더라도 해쉬값의 일방향성 특성 때문에 계정의 패스워드를 알 수 없다. 하지만 패스워드의 해쉬값만 유지할 경우에는 크게 두 가지 문제점이 존재한다. 하나는 공격자들이 이 파일을 이용하여 사전공격(dictionary attack)을 할 수 있다는 것이고, 다른 하나는 해쉬값이 같은 두 사용자는 동일한 패스워드를 사용하고 있다는 것이 노출된다. 따라서 패스워드만을 이용하여 해쉬값을 계산하지 않고, salt라고 하는 랜덤값과 패스워드를 비트결합한 값을 입력으로 사용하여 해쉬값을 계산한다. 이 경우 올바른 패스워드가 입력되었는지 검사하기 위해서는 시스템에서 사용된 salt값을 알고 있어야 한다. 따라서 salt도 평문 그대로 해쉬값과 함께 저장된다. 이와 같은 방식을 사용하면 앞서 언급한 두 가지 문제를 모두 어느 정도 극복이 가능하다. 전자에 대해서는 salt를 사용하지 않으면 가능한 패스워드에 대한 해쉬값들만 계산하면 되지만 이제는 각 가능한 패스워드마다 사용된 salt의 개수만큼 계산해야 한다. 따라서 사전공격에 소요되는 비용이 증가하게 된다. 후자는 각 계정마다 다른 salt를 사용하므로 동일한 패스워드를 사용하는 사용자라 하더라도 결과 해쉬값은 서로 다르게 된다.

해쉬함수는 보통 비밀키를 사용하지 않는다. 따라서 메시지와 그것의 해쉬값을 함께 전송하거나 저장할 경우 그것의 무결성을 안전하게 보장하기가 어렵다. 그 이유는 고의적인 공격자는 메시지 뿐만 아니라 해쉬값도 함께 변경할 수 있기 때문이다. 따라서 보다 안전하게 무결성을 보장하기 위해 해쉬값을 생성할 때 비밀키를 사용하여 해당 비밀키를 모르는 경우에는 올바른 해쉬값을 계산할 수 없도록 하는 기법을 사용한다. 이 처럼 비밀키를 사용하여 계산되는 해쉬값을 **메시지 인증 코드**(MAC, Message Authentication Code)라 하고, 비밀키를 사용하지 않는 일반 해쉬값을 조작 탐지 코드(MDC, Modification/Manipulation Detection Code)라 한다.

7.2 생일 파라독스

해쉬함수의 안전성은 강한충돌을 찾는 것이 어느 정도 어려운지에 따라 결정된다. 해쉬함수의 강한충돌 측면의 안전성을 분석하기 위해서는 이것과 과련된 생일 파라독스에 대해 알아야 한다. 생일 파라독스란 N명 중 특정한 사람을 선택하여 이 사람과 생일이 같은 사람이 N명 중에 있을 확률이 50%가 넘기 위해서는 N이 183명이 넘어야 하지만 N명 중 생일이 같은 두 명이 있을 확률이 50%가 넘기 위해서는 N이 23명만 넘으면 된다는 것을 말한다. 이 파라독스와 해쉬함수를 매핑하면 다음과 같다. 먼저 해쉬함수의 입력은 사람이고, 출

력은 생일이라고 하면 생일이 같은 사람을 찾는 것은 해쉬함수에서 충돌을 찾는 것과 같다. 특히, 특정한 사람을 선택한 후에 이 사람과 생일이 같은 사람을 찾는 것은 약한 충돌이고, 생일이 같은 임의의 두 명을 찾는 것은 강한 충돌에 해당된다. 좀 더 수학적으로 생일 파라독스를 분석하여 보자. 출력의 크기가 365일 때 강한 충돌을 찾을 확률이 50%가 넘기 위해 검사해 보아야 하는 입력의 수를 생각하여 보자.

- 임의로 N명을 선택하였을 때, 이 중에 생일이 같은 사람이 있을 확률은?

  Prob = 1-(N명의 생일이 모두 다를 경우)

  - 이 확률을 계산할 때 사람들의 생일이 균일하게 분포되어 있다고 가정한다. 이 가정은 물론 현실적으로 잘못된 가정이지만 해쉬함수의 경우에는 해쉬함수의 출력이 균일하게 분포된다는 것을 가정하는 것은 무리가 없다.
- 위 확률을 일반화하여 생각하여 보자. M개의 통에 N개의 구술을 임의로 넣는 경우를 생각하여 보자. 여기서 M은 공역 범위의 크기이고, N은 사용된 입력의 개수이다. 이 경우 위 확률은 M개의 통 중에 2개 이상의 구술이 들어 있는 통이 있을 확률과 같다. 이 확률은 1-(N개의 구술이 모두 다른 통에 들어갈 확률)과 같으며, 다음과 같이 수식화할 수 있다.

$$\epsilon = 1 - 1 \times \left(\frac{M-1}{M}\right) \times \left(\frac{M-2}{M}\right) \times \cdots \times \left(\frac{M-N+1}{M}\right) = 1 - \prod_{i=1}^{N-1}\left(1-\frac{1}{M}\right)$$

  - 매클로린 급수 전개(Maclaurin series expansion)에 의하면 $e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots$ 이다. 이 때 x가 매우 작으면 $e^{-x} \approx 1 - x$로 근사화할 수 있다. 따라서 위 식은 다음과 같이 바꾸어 표현할 수 있다.

$$\epsilon = 1 - \prod_{i=1}^{N-1}\left(1-\frac{i}{M}\right) \approx 1 - \prod_{i=1}^{N-1} e^{-\frac{i}{M}} = 1 - e^{-\sum_{i=1}^{N-1}\frac{i}{M}} = 1 - e^{-\frac{1}{M}\sum_{i=1}^{N-1} i} = 1 - e^{-\frac{1}{M}\frac{N(N-1)}{2}}$$

- 우리가 찾고자 하는 확률을 $\epsilon \approx 1 - e^{-\frac{1}{M}\frac{N(N-1)}{2}}$이다. 따라서 $e^{-\frac{1}{M}\frac{N(N-1)}{2}} \approx 1 - \epsilon$이며, 양변에 로그를 취하면 다음과 같다.

$$-\frac{1}{M}\frac{N(N-1)}{2} \approx \ln(1-\epsilon)$$

- 식의 좌우를 다시 정리하면 다음과 같다.

$$N^2 - N \approx 2M \ln\frac{1}{1-\epsilon}$$

- $-N$항을 무시하면 다음과 같이 다시 근사화할 수 있다.

$$N \approx \sqrt{2M \ln\frac{1}{1-\epsilon}}$$

- $M = 365$, $\epsilon = 0.5$로 설정하면 $N \approx 23$이 된다. 즉, 23명이 넘으면 강한 충돌을 찾을 수 있는 확률이 50%가 넘는다.

이 수학적 분석을 해쉬함수에 적용하면 다음과 같다. 해쉬함수 출력의 범위가 2^{160}이면 강한 충돌을 찾기 위해 $\sqrt{2^{160}} = 2^{80}$개의 해쉬값을 계산하면 강한 충돌을 찾을 수 있는 확률이 50%가 넘는다.

7.3 SHA-1

SHA(Secure Hash Algorithm)은 NIST(National Institute of Standards and Technology)에서 개발한 미국 표준 해쉬함수이다. SHA는 1993년에 표준으로 제정되었으며, 표준 문서 번호는 FIPS 180이다. Rivest는 1990년에 MD4라는 해쉬함수를 개발하였으며, 1992년에 이를 개선한 MD5를 제안하였다. 1993년에 표준으로 제정된 SHA 함수는 MD5와 매우 유사한 해쉬함수이다. 1995년에는 기존 SHA를 개선한 새 표준을 제정하였으며, 기존과 구분하기 위해 새 표준을 SHA-1이라고 명명하였으며, 이 표준의 문서 번호는 FIPS 180-1이다. SHA-1의 해쉬값의 길이는 160비트이다. 따라서 생일 파라독스에 의하면 그것의 안전성은 $O(2^{80})$이다. 하지만 2005년 X. Wang, Y.L. Yin, H. Yu는 SHA-1에 대한 새 공격을 발표하였으며, 이 공격에 의하면 $O(2^{69})$의 비용으로 충돌을 찾을 수 있다.

앞서 언급한 바와 같이 SHA-1의 출력값의 길이는 160비트이며, 입력은 2^{64} 비트보다 작아야 한다. 내부적으로 512비트 단위로 처리하며, 전체 입력의 크기가 512비트의 배수가 아니면 일반적인 채우기 방법을 사용한다. 즉, 일정한 값으로 채우고 마지막 64비트에 실제 입력의 크기를 기록한다.

첫 512비트 입력을 처리하는 방법은 다음과 같다.

- 512비트 입력을 16개의 워드 M_i라 가정하였을 때, 이 입력은 총 80개의 32비트 워드 W_i로 확장된다. 이 때 첫 16개 워드는 입력과 동일하며, 나머지 64개 워드는 기존에 생성한 워드들을 이용하여 다음과 같이 계산된다.

$$W_i = \begin{cases} M_i & i = 0,..,15 \\ W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} & i = 16,...,79 \end{cases}$$

- 다음과 같은 초기 5개의 워드는 총 80 단계를 통해 변형되어 결국 최종 해쉬값이 된다. 이 때 80개의 워드로 확장된 워드들은 각 단계마다 하나씩 사용된다.

A = 0x67452301
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476
E = 0xC3D2E1F0

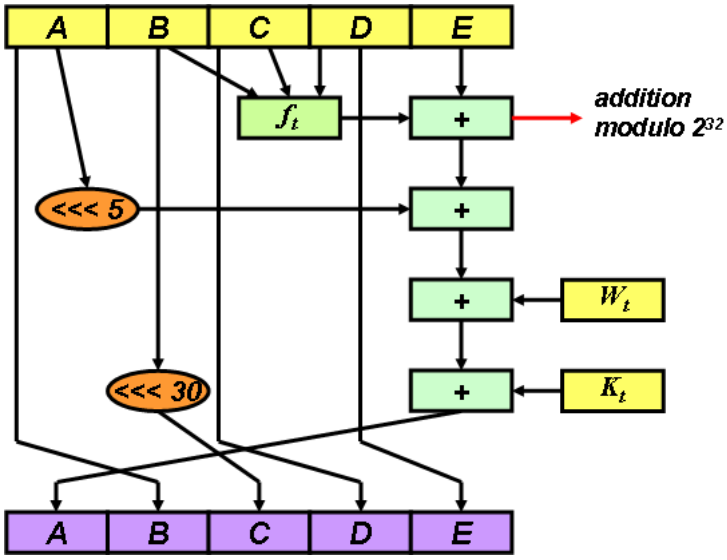
  - 80 단계는 크게 4개의 라운드로 구분되며, 각 라운드는 20 단계로 구성된다. 각 단계는 그림 7.1과 같다. 이 때 K_t는 다음과 같으며,

K_t = 0x5A827999, $t = 0, ..., 19$
K_t = 0x6ED9EBA1, $t = 20, ..., 39$
K_t = 0x8F1BBCDC, $t = 40, ..., 59$
K_t = 0xCA62C1D6, $t = 60, ..., 79$

$f_t(X,Y,Z)$는 다음과 같다.

$f_t(X, Y, Z) = (X \wedge Y) \vee ((\neg X) \vee Z)$, $t = 0, ..., 19$
$f_t(X, Y, Z) = (X \oplus Y \oplus Z)$, $t = 20, ..., 39, 60, ..., 79$

$$f_t(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), \quad t = 40, \ldots, 59$$



<그림 7.1> SHA-1의 내부 단계

7.4 메시지 인증 코드

7.4.1 메시지 인증 코드 개요

MAC은 일반 해쉬함수와 달리 MAC 값을 계산하기 위해서는 입력 메시지뿐만 아니라 암호키가 필요하다. 따라서 암호키를 모르는 사용자는 MAC 값을 계산할 수 없다. 그러므로 MAC은 해쉬함수와 동일한 안전성(일방향성, 충돌회피성)을 가져야 하며, 추가적으로 사용되는 암호키와 관련하여 다음과 같은 요구사항들을 충족해야 한다.

- **요구사항 1.** M과 $MAC_K(M)$을 공격자가 가지고 있더라도 K를 알아내는 것은 계산적으로 어려워야 한다.
- **요구사항 2.** M과 $MAC_K(M)$을 공격자가 가지고 있더라도 $MAC_K(M) = MAC_K(M')$인 메시지 M'을 찾는 것이 계산적으로 어려워야 한다.
- **요구사항 3.** $MAC_K(M)$은 균일하게 분포되어야 한다. 즉, 임의로 선택한 두 개의 메시지 M과 M'에 대해 $\Pr[MAC_K(M') = MAC_K(M)] = 2^{-n}$이어야 한다. 여기서 n은 MAC의 길이이다.

요구사항 1에 대해 좀 더 자세히 살펴보면 다음과 같다. 공격자가 M과 $MAC_K(M)$을 가지고 있는 경우 전사공격으로 K를 알아내기 위한 비용을 알아보면 다음과 같다.

- $|K| > |MAC|$: 가능한 모든 키를 이용하여 $MAC_K(M)$을 계산하면 총 $2^{|K|}$개의 MAC이 생성된다. 그러나 $2^{|MAC|}(< 2^{|K|})$개의 MAC만 존재하므로 대략 $2^{|K|-|MAC|}$개의 키 중 어떤

키가 올바른 키인지 알 수 없다. 동일한 키로 계산한 MAC 값을 여러 개 가지고 있을 경우에는 이 과정을 반복하면 정확한 키를 찾을 수 있다. 그러나 전체 비용은 같은 길이의 키를 사용하는 암호문에 대한 전사공격을 하는 것보다 어렵다.

- 예7.1) $|MAC|$ = 32 비트이고, $|K|$ = 80 비트이면 첫 번째 시도에는 2^{80}개의 키를 검사하면 주어진 MAC을 생성하는 키는 대략 2^{48}개를 발견하게 된다. 발견한 이 키들을 이용하여 두 번째 쌍에 대해 검사하면 두 쌍에 대해 모두 일치하는 키는 대략 2^{16}개가 존재한다.

- $|K| < |MAC|$: 가능한 모든 키를 이용하여 $MAC_K(M)$을 계산하면 총 $2^{|K|}$개의 MAC이 생성된다. 앞의 예와 달리 이 경우에는 전사공격의 비용이 대략 $O(2^{|K|})$가 된다.

7.4.2 대칭 암호알고리즘을 이용한 MAC 구현

이미 배운 바와 같이 블록방식의 대칭 암호알고리즘을 CBC나 CFB 방식으로 메시지를 암호화하면 마지막 블록 값을 그 메시지의 MAC 값으로 활용할 수 있다. 이 때 마지막 블록 전체를 MAC 값으로 사용할 수 있고, 왼쪽 m 비트만 MAC 값으로 사용할 수 있다. 마지막 블록 전체 대신에 이보다 작은 왼쪽 m 비트만 사용하는 것은 전사공격을 더욱 어렵게 만들기 위함이다. 물론 비트의 수가 적으면 임의로 추측할 확률(2^{-m})은 증가한다. 하지만 보유하고 있는 쌍의 개수가 많으면 이와 같은 방법을 통해 안전성을 높이는 효과는 적다.

국제 표준 ISO 8731-1에 DES 암호알고리즘을 이용한 MAC이 정의되어 있다. 이 MAC은 다음과 같은 단계를 통해 MAC 값을 계산한다.

- **단계 1.** 초기벡터 IV를 모두 0으로 설정한다. 메시지가 블록의 배수가 아니면 정해진 채우기를 한다.
- **단계 2.** 전체 메시지를 CBC 모드로 암호화한다.
- **단계 3.** 마지막 블록을 단계 2에서 사용하였던 키와 다른 키로 복호화하고, 이것을 다시 단계 2에서 사용하였던 키로 암호화한다. 결과 블록 전체를 MAC 값으로 사용하거나 왼쪽 m 비트만 MAC 값으로 사용할 수 있다.

단계 3과 같은 안전성 강화 기술을 사용하지 않으면 다음과 같은 위조가 가능하다.

- x_1이 64 비트 평문 블록이고, 이것에 대한 DES-CBC-MAC 값은 다음과 같다.
$$MAC_K(x_1) = E_K(x_1 \oplus IV) = E_K(x_1)$$
- $x_2 = MAC_K(x_1)$이면 이것에 대한 DES-CBC-MAC 값은 다음과 같다.
$$MAC_K(x_2) = E_K(x_2 \oplus IV) = E_K(x_2) = E_K(MAC_K(x_1))$$
- $\perp b$가 b에 대한 64 비트 블록 표현일 때 $x_1 \| \perp 0$에 대한 DES-CBC-MAC 값은 다음과 같으며, 이 값은 $MAC_K(x_2)$와 같다.
$$MAC_K(x_1 \| \perp 0) = E_K(\perp 0 \oplus E_K(x_1 \oplus IV)) = E_K(E_K(x_1)) = E_K(MAC_K(x_1))$$

단계 2까지만 사용하는 방식은 미국 표준 FIPS PUB 113에 정의되어 있다. 단계 3에 제시된 강화 기술은 Black과 Rogaway가 제안한 방식이며, Iwata와 Kurosawa는 다음과 같은

강화 기술을 제안하였다. 여기서 M_n이 마지막 평문 블록이면 마지막 암호문 블록은 기존과 달리 또 다른 키를 하나 더 활용하여 $C_n = E_{K_1}(M_n \oplus C_{n-1} \oplus K_2)$과 같이 계산한다.

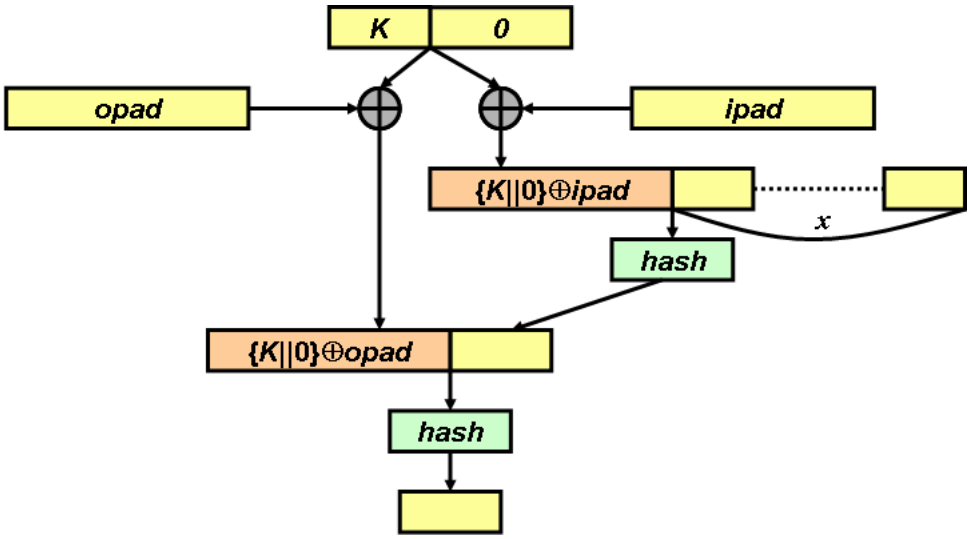
7.4.3 일반 해쉬함수를 이용한 MAC 구현

해쉬값을 계산할 때 메시지뿐만 아니라 암호키를 포함시켜 계산하여 그 값을 MAC 값으로 사용하는 것을 생각해 볼 수 있다. 특히, 해쉬함수의 일방향 특성 때문에 이 방법을 사용하면 MAC 값으로부터 암호키를 공격자가 계산하는 것은 계산적으로 가능하지 않다. 하지만 이것만으로는 MAC의 안전성을 보장할 수 없다.

먼저 일반 해쉬함수 H를 이용하여 메시지 x에 대한 MAC 값을 $H(K\|x)$와 같이 계산한다고 가정하자. 또 사용한 해쉬함수 H가 SHA-1과 같이 어떤 일정 블록 단위로 동일한 연산을 반복하여 계산하는 해쉬함수라 하고, 이 때 $K\|x$는 해쉬함수에 적용할 때 채우기가 필요 없다고 가정하자. 그러면 공격자는 키를 모르는 상태에서 $x\|x'$에 대한 MAC 값을 계산할 수 있다. 해쉬함수 H가 블록 단위로 나누어 반복적으로 동일한 연산을 적용하는 해쉬함수이므로 해쉬함수에서 사용하는 초기값 대신에 $H(K\|x)$를 초기값으로 사용하면 $H(K\|x\|x')$을 계산할 수 있다.

반대로 $H(K\|x)$ 대신에 $H(x\|K)$ 형태를 사용하는 것을 고려해 볼 수 있다. 이 경우에는 해쉬함수의 반복적 계산 특성을 이용하여 $H(x\|x'\|K)$를 계산하는 것이 어렵지만 임의의 메시지를 앞에 추가하는 $H(x'\|x\|K)$를 계산할 수 있는 가능성이 있다. 또한 생일 파라독스를 이용하여 $H(x) = H(x')$인 x'을 찾아내면 키를 모르는 상태에서는 x'에 대한 MAC 값을 계산할 수 있는 가능성이 있다.

따라서 $H(K\|p\|x\|K)$ 형태로 계산하는 것이 안전하다. 여기서 p는 K를 해쉬함수에 한 블록으로 만들기 위한 채우기 값이다. 이것은 최소 두 번의 내부 연산이 수행되도록 하기 위함이다. 다른 대안으로 $H(K_1\|p\|x\|K_2)$, $H(K_1\|p\|H(K_2\|p\|x))$ 형태도 제안되어 사용되고 있다.



<그림 7.2> HMAC 표준

IETF에서 국제 표준으로 제정된 MAC은 $H(K_1||p||H(K_2||p||x))$ 형태이다. 이 표준은 SHA-1 해쉬함수를 사용하며, MAC을 계산하는 방법은 다음과 같으며 그림 7.x에 기술되어 있다.

- **단계 1.** 키 K가 64 바이트보다 작으면 0으로 채우기하여 SHA-1의 내부 블록과 같은 크기로 만든다. 만약 키가 64 바이트보다 크면 K를 해쉬한 후에 0으로 채우기하여 내부 블록과 같은 크기로 만든다.
- **단계 2.** 단계 1에서 만든 블록과 64 바이트가 모두 0x36인 또 다른 블록 ipad와 XOR 연산을 한 후에 입력 x와 비트 결합하여 그것을 입력으로 해쉬함수 값을 구한다.
- **단계 3.** 단계 1에서 만든 블록과 64 바이트가 모두 0x5C인 또 다른 블록 opad와 XOR 연산을 한 후에 단계 2에서 구한 해쉬값과 비트 결합하여 그것을 입력으로 해쉬함수 값을 구한다.

즉, 메시지 x에 대한 MAC 값은 다음과 같다.

$$H(\{K||0\} \oplus opad||H(\{K||0\} \oplus ipad||x))$$

참고문헌

[1] Secure Hash Standard, FIPS Pub 180-1, Apr. 1995.

[2] Computer Data Authentication, FIPS Pub 113, May 1985.

[3] John Black and Phillip Rogaway, "CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions," Advances in Cryptology, Crypto 2000, LNCS 1880, Springer, pp. 197–215, 2000.

[4] Tetsu Iwata and Kaoru Kurosawa, "OMAC: One-Key CBC-MAC," Proc. of the Fast Software Encryption 2003, LNCS 2887, Springer, pp. 129–153, 2003.

[5] H. krwaczyk, M. Bellare, R. Canetti, HMAC: Keyed-Hashing for Message Authentication, RFC 2104, Feb. 1997.