MD5(Message-Digest algorithm 5) 알고리즘

● MD5의 주요 특징

- 임의의 길이의 데이터를 입력받아 128비트(16바이트)의 고정된 길이의 해시 값(다이제스트)을 출력하는 암호학적 해시 함수

  * 단방향성 : 출력된 해시값으로부터 원래의 입력값을 복원하는 것은 이론적으로 불가능(단방향 암호화)
  * 일관성 : 같은 입력값에 대해서는 항상 같은 해시값이 출력
  * 충돌 회피 : 서로 다른 입력값에서 같은 해시값이 나올 확률(충돌)은 극히 낮도록 설계됨

● MD5의 용도

- MD5는 본래 보안 용도로 설계되었지만, 현재는 보안 취약점 때문에 그 용도가 제한적임

  * 데이터 무결성 확인 (Integrity Check)

    - 파일을 전송하거나 저장할 때, MD5 해시값을 함께 생성하여 보관
    - 나중에 이 파일을 다시 받거나 사용할 때 MD5 해시값을 다시 계산하여 기존 해시값과 비교함으로써, 파일이 전송 도중 또는 저장 중에 손상되거나 변조되지 않았는지 확인하는 용도로 사용 (체크섬 역할)

      * 체크섬(Checksum) - 데이터 전송이나 저장 과정에서 발생할 수 있는 오류(Error)나 변조(Alteration)를 검출하기 위해 데이터 블록을 기반으로 계산되는 작은 크기의 값

  * 과거의 암호 저장 (제한적)

    - 예전에는 사용자 비밀번호를 그대로 저장하지 않고 MD5로 해시하여 저장하는 데 많이 사용되었음
    - 사용자가 비밀번호를 입력하면 MD5로 변환하여 저장된 값과 비교하는 방식임
    - 현재는 충돌 취약점 때문에 보안에 중요한 비밀번호 암호화에는 SHA-256 등 더 강력하고 안전한 알고리즘을 사용하도록 권장됨

● MD5의 취약점과 현재 상태

- MD5는 설계된지 오래되었고, 기술 발전과 함께 심각한 보안 취약점이 발견되어 보안 관련 용도로 사용하는 것은 강력히 권장되지 않음

  * 충돌 공격(Collision Attack)
    - 2004년 이후, 서로 다른 두 입력값에서 동일한 MD5 해시값을 찾아내는 것이 가능하다는 사실이 밝혀졌음 ⇒ 해시 함수의 가장 중요한 보안 특성인 충돌 회피에 심각한 결함이 있다는 것을 의미

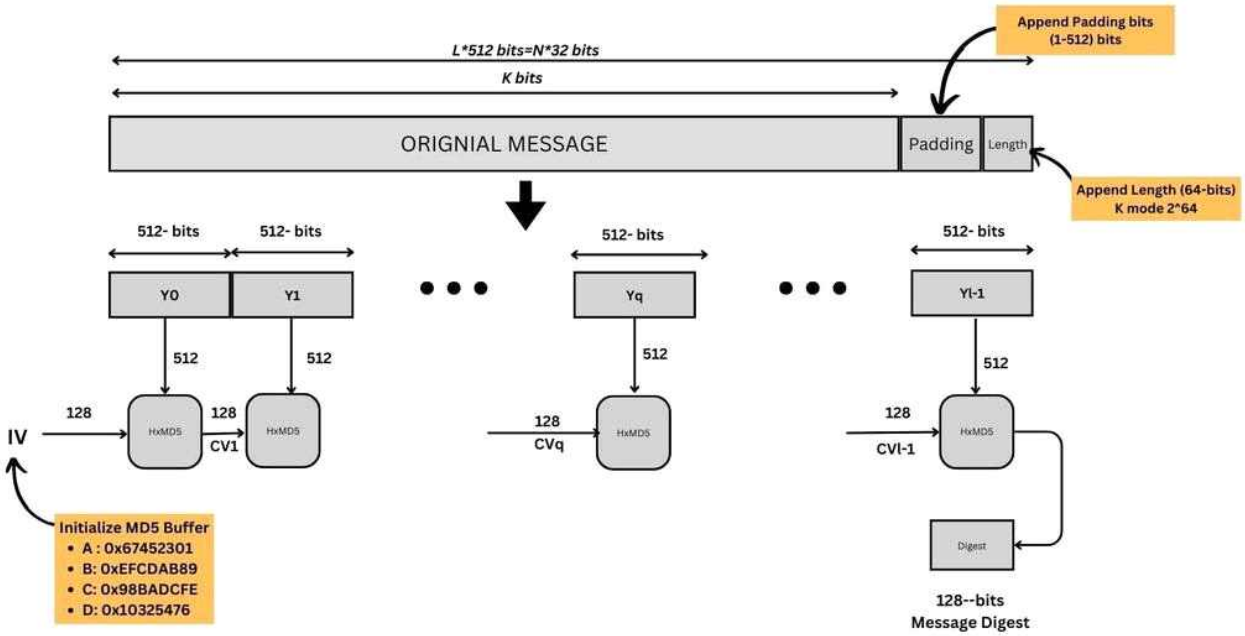
  * 빠른 연산 속도
    - MD5의 빠른 연산 속도는 무차별 대입 공격(Brute-force attack)이나 레인보우 테이블 공격(Rainbow table attack)에 취약하게 만드는 요인이 됨
      * 레인보우 테이블 공격(Rainbow Table Attack) - 미리 계산된 방대한 양의 해시 값 목록(레인보우 테이블)을 사용하며, 무차별 대입 공격보다 훨씬 효율적으로 비밀번호를 해독할 수 있음

● 대체 알고리즘

- MD5의 보안 취약점 때문에 현재는 다음과 같은 더 안전한 해시 함수가 대체재로 사용되고 있음
  * SHA-256 (Secure Hash Algorithm 256-bit)
  * SHA-3 (Secure Hash Algorithm 3)
  * Bcrypt, Scrypt, Argon2 (비밀번호 저장 등 보안이 중요한 곳에 특히 권장)

MD5 (Message-Digest algorithm 5) 해시 과정

● 임의의 길이 메시지를 받아 128비트의 고정 길이 해시값을 생성



1. 패딩 비트 추가 (Append Padding Bits)

- 패딩의 목적
  * MD5(그리고 SHA 계열 등 대부분의 해시 함수)는 512비트(=64바이트) 단위로 데이터를 처리함 ⇒ 입력 메시지의 길이가 512의 배수가 아닐 경우, 마지막 블록을 딱 512비트로 맞추기 위해 패딩을 추가해야 함

- 입력 메시지의 길이가 512비트 블록의 정확한 배수보다 64비트가 모자라도록(즉, 메시지 길이를 512k + 448 비트가 되도록) 메시지 끝에 패딩 비트를 추가

  * 첫 비트 : 메시지 끝에 단일 비트 '1'을 추가
    - 왜 '1비트'를 먼저 붙일까?
      * 패딩을 시작할 때는 반드시 1(즉, 10000000...)이라는 1비트를 먼저 붙이고 그 다음엔 0비트들을 계속 채워 넣음 ⇒ "메시지의 끝이 어디인지 명확히 표시하기 위함"
  * 나머지 비트 : 그 뒤를 '0' 비트로 채움

☎ 1000비트의 메시지가 주어졌다고 가정하고 원본 메시지에 패딩 비트를 추가

- 여기서는 원본 메시지에 472개의 패딩 비트를 추가
- 패딩 비트를 추가한 후 첫 번째 단계의 원본 메시지/출력 크기는 1472가 됨
- 즉, 512의 정확한 배수보다 64비트가 적음(즉, 512*3 = 1536)
- 길이(원래 메시지 + 패딩 비트) = 512 * i - 64 여기서 i = 1,2,3 . . .

2. 메시지 길이 추가 (Append Length)

- 패딩 후 남은 64비트 공간에 패딩 전 원래 메시지의 길이를 64비트 정수 형태로 추가
  * 이 과정을 통해 전체 메시지 길이가 512비트의 정확한 배수가 되어,
    N개의 512비트 블록으로 나눌 수 있게 됨

☎ 길이 비트 추가

- 이 단계에서는 첫 번째 단계의 출력에 길이 비트를 추가하여 비트의 총 수가 512의 완벽한 배수가 되도록 함
- 간단히 말해서, 여기서는 첫 번째 단계의 출력에 64비트를 길이 비트로 추가
  * 첫 번째 단계의 출력 = 512 * n - 64
  * 길이 비트 = 64
  * 두 수를 더하면 512 * n이 나오는데, 이는 512의 정확한 배수임

3. 초기 버퍼 설정 (Initialize MD Buffer / Chaining Variables)

- MD5 알고리즘의 핵심인 128비트 상태(State)를 구성하는 네 개의 32비트 워드(A, B, C, D)를 미리 정의된 초기 상수값(Initial Value, IV)으로 초기화
    * 각 버퍼의 크기는 32비트
    * A, B, C, D는 각각 32비트 레지스터 역할을 하며, 이 네 워드를 합친 것이 최종 128비트 해시값의 중간 결과가 됨

4. 메시지 처리 (Process Message in 512-bit Blocks)

- 메시지를 512비트 블록 단위로 쪼개어 차례로 처리하는 메인 알고리즘이 반복
- 4라운드에서 총 64개의 연산이 수행 1라운드에서는 16개의 연산, 2라운드에서는 16개의 연산, 3라운드에서는 16개의 연산, 4라운드에서는 16개의 연산이 수행됨
    * 각 라운드마다 다른 함수를 적용함
        - 1라운드에는 F 함수, 2라운드에는 G 함수, 3라운드에는 H 함수, 4라운드에는 I 함수를 적용함
    * 각 512비트 블록은 16개의 32비트 워드 M_0, M_1, ..., M_15로 나뉨
    * 이 블록은 초기화된(또는 이전 블록 처리에서 업데이트된) 128비트 상태 (A, B, C, D)와 함께 압축 함수(Compression Function)에 입력됨
        - 압축 함수 (라운드 처리)
            * 압축 함수는 하나의 512비트 블록을 처리하여 A, B, C, D 레지스터의 값을 업데이트
            * 4개의 라운드 : 압축 함수는 4개의 연속적인 라운드(Round)로 구성되며 각 라운드는 16개의 기본적인 연산(Operation)을 수행
            * 따라서 총 4 * 16 = 64번의 연산이 이루어짐
    * 기본 연산 : 각 연산은 다음과 같은 요소를 사용
        - 비선형 함수 (F, G, H, I) : 각 라운드마다 다른 비선형 논리 함수(XOR, AND, OR, NOT 등의 조합)가 사용되어 비트의 혼란을 야기함
        - 모듈러 덧셈 (+_{2^{32}}) : 32비트 워드 간의 모듈로 2^{32} 덧셈을 수행
        - 왼쪽 비트 회전 (Left Rotation) : 결과를 특정 비트 수만큼 왼쪽으로 순환 이동시킴
        - 상수 T : 각 연산마다 서로 다른 미리 정의된 상수
    * 각 512비트 블록을 처리하고 나면, 새로운 A, B, C, D 값이 다음 블록 처리의 시작 상태(또는 최종 해시값)가 됨

5. 최종 해시값 출력

- 모든 512비트 메시지 블록에 대한 처리가 완료되면, 마지막으로 업데이트된 32비트 워드 (A, B, C, D)를 연결하여 최종 128비트 해시값을 생성
    ⇒ “MD5 다이제스트 또는 체크섬”

■ MD5에서 입력 메시지가 각 라운드에 어떻게 적용되는지를 라운드 1의 동작을 시뮬레이션

▶ 예시 : "abc" 메시지를 MD5 Round 1에 적용

♣ 입력 메시지 : "abc"

- ASCII 값 - 'a' = 0x61, 'b' = 0x62, 'c' = 0x63
- 메시지 길이 : 3 bytes = 24 bits

♣ 패딩 및 길이 추가

- MD5는 512비트 블록으로 처리하므로
  * 메시지 뒤에 1 비트 + 0 비트들 추가
  * 마지막 64비트에 메시지 길이(24)를 추가
- 결과적으로 "abc"는 512비트 블록으로 변환(16개의 32비트 워드 = M[0] ~ M[15])

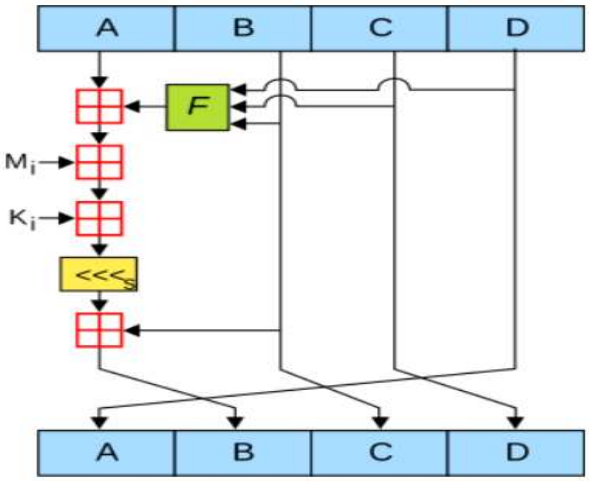
♣ 초기 상태 값 설정

- MD5는 다음의 초기 상태 값을 사용
  * A = 0x67452301, B = 0xefcdab89, C = 0x98badcfe, D = 0x10325476

♣ Round 1 적용 (16번 반복)

- 각 반복에서 다음을 수행

* 함수 : F(B, C, D) = (B & C) | (~B & D)



$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

- 입력 : A, B, C, D, M[i], T[i], s (시프트 값)
  - 연산 : A = B + ((A + F(B, C, D) + M[i] + T[i]) <<< s)

* 예시 : 첫 번째 반복 (i = 0)
  - M[0] = 메시지 블록의 첫 32비트
  - T[0] = 0xd76aa478 (고정 상수)
    * 라운드 상수(Kj)는 각 단계의 계산에 무작위성을 더하고 공격을 어렵게 하기 위해 사용되는 미리 정해진 64개의 32비트 상수
  - s = 7 (시프트 값)

  - 계산
    1. F(B, C, D) 계산
    2. A + F + M[0] + T[0]
    3. 왼쪽으로 7비트 회전
    4. 결과를 B에 더함 → 새로운 A 값