차례

5.1 개요

■ 이 장에서 다룰 주요 주제

- 암호화 해시 함수
  - 한 방향성, 충돌 저항성
  - 다양한 보안 환경에서 활용
- 표준 용도
  - 디지털 서명
  - 해시 기반 MAC(Message Authentication Code)
- 응용 프로그램
  - 온라인 입찰 무결성 검증
  - 블록체인 블록 연결 및 합의

5.1 개요

- 이 장에서 이야기할 몇 가지 암호화 관련 주제
  - 비밀 공유 (Secret Sharing)
    - 하나의 비밀을 여러 조각으로 나누어야 전체가 복원 가능
    - 시각적 암호화(Visual Cryptography)도 포함 → 그림을 조각내어 합치면 원본 복원
  - 암호화 난수 (Cryptographic Randomness)
    - 안전한 키 생성, 세션 토큰 발급에 필수
    - 의사난수 vs. 진정난수 (예: 하드웨어 기반 난수)
  - 정보 은닉 (Steganography & Digital Watermarking)
    - 스테가노그래피: 이미지, 오디오에 비밀 메시지 삽입
    - 디지털 워터마킹: 저작권 보호 및 위조 방지

5.2 암호화 해시 함수란?

- 암호화 해시 함수 $h(x)$는 다음 특성을 모두 만족해야 함
  - 압축
    - 입력 x의 크기와 상관없이 출력의 길이 $y=h(x)$는 고정 길이
      - 예: SHA-256 -> 항상 256비트 출력
  - 효율성 : 계산이 빠르다
    - 어떤 입력 x에 대해서든 $h(x)$를 계산하는 것이 쉬워야 함
  - 단방향
    - 어떤 값 y가 주어진다면 $h(x)=y$에서 x의 값을 찾는 계산이 어려워야 함
  - 약한 충돌 방지
    - 주어진 x와 $h(x)$에 대해 $h(y)=h(x)$를 만족하면서 $y \neq x$인 y를 찾는 것이 어려워야 함
  - 강한 충돌 방지
    - 해시 함수를 써서 똑같은 출력이 나오는 똑같은 두 개의 입력을 찾을 수 없어야 함

5.2 암호화 해시 함수란?

■ 해시 함수의 보안적 의미

- 충돌 가능성은 이론적으로 존재(입력공간 >> 출력 공간)
- 설계 목표: 충돌을 "찾기 매우 어렵게" 만드는 것
- 예시:
  - 128비트 해시 -> 가능한 출력은 2^{128}개
  - SHA-1 (160비트)은 충돌 공격 가능 → 현재는 안전하지 않음
  - SHA-2, SHA-3이 안전한 표준으로 사용

5.2 암호화 해시 함수란?

- 해시 함수는 보안에 굉장히 유용
- 해시 함수의 중요한 한 가지 용도는 디지털 서명을 계산하는 것
- 예) 앨리스가 암호화 해시 함수를 가지고 있음
  - $h(M)$은 파일 M에 대한 '지문' 역할을 할 수 있음
  - $h(M)$은 M보다 훨씬 작지만 M을 식별
  - M'이 M과 다르다면, 그것이 한 비트일지라도 해시는 달라짐
  - 충돌 방지 특성은 M을 $h(M)=h(M')$과 같이 다른 메시지 M'으로 대체하는 것이 불가능하다는 것을 의미
  - [그림 5-1]처럼 앨리스는 밥에게 M과 S를 보낼 수 있음
  - 밥은 M을 해시하고 앨리스의 공개키를 S에 적용한 값과 비교하여 디지털 서명을 확인

    밥은 $h(M)=\{S\}_{앨리스}$라고 확인

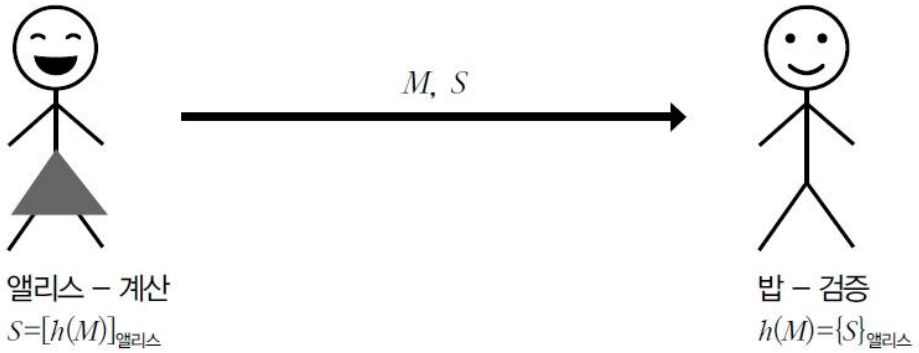

그림 5-1 디지털 서명을 하는 더 나은 방법

5.2 암호화 해시 함수란?

■ 디지털 서명의 안전성

- 안전성은 두 가지에 의존:
  - 공개키 암호 체계
  - 해시 함수의 충돌 방지성
- 만약 충돌 발견 시:
  - $M \neq M'$이지만 $h(M)=h(M')$ → 공격자가 M'을 M으로 위조 가능
  - 결론: 안전한 해시 함수 선택이 필수적

5.3 생일 문제

■ 생일 문제란?

- 정의: 임의로 선택된 N명의 사람 중, 두 사람 이상이 같은 생일을 가질 확률 계산 문제
- 직관과 달리, 작은 N에서도 높은 확률 발생
- 예: 23명만 모여도 두 사람 이상 생일이 같을 확률 ≈ 50%

■ 당신과 같은 생일을 가진 사람이 있을 확률

- $P = 1 - \left(\frac{364}{365}\right)^N$
- -> N = 253일 때 약 50%

■ 두 사람 이상 같은 생일 확률

- $P = 1 - \left(\frac{365}{365}\right) \cdot \left(\frac{364}{365}\right) \cdot \left(\frac{363}{365}\right) \cdot \left(\frac{362}{365}\right) \cdots \left(\frac{365 - N + 1}{365}\right)$
- N = 23 일 때 약 50%

5.3 생일 문제

■ 생일 문제와 암호학

- 해시 함수 h(x)가 n비트 출력 생성한다고 가정:
  - 가능한 출력 개수: 2^n
- 안전한 해시에서도 약 $2^{\frac{n}{2}}$ 시도 시 충돌 기대 가능
  - 예: n=128 → 약 2^{64} 시도 후 충돌 기대
- 따라서 해시 보안 수준은 출력 비트 길이의 절반 수준
  - 실제 적용SHA-1 (160비트) → 이론상 2^{80} 시도 후 충돌
  - SHA-256 → 2^{128} 시도 필요 → 현재 안전

5.3 생일 문제

■ 생일 문제 요약

- 작은 그룹에서도 충돌 발생 확률이 의외로 크다
- 해시 보안 수준을 직관적으로 설명하는 도구

■ 보안적 시사점

- 해시 비트 길이 = 보안 강도의 핵심
- 해시가 짧으면 "충돌 공격"에 매우 취약

■ 사례

- SHA-1 충돌 공격 (2017, Google SHAttered) → 실제로 두 개 다른 PDF에 같은 SHA-1 해시 발생

5.4 생일 공격

- 정의: 해시 함수의 충돌 가능성을 이용한 공격 기법
- 기반 아이디어:
  - 해시 출력 길이가 n비트일 때, 약 $2^{\frac{n}{2}}$ 시도로 충돌 가능
- 주요 목표:
  - 디지털 서명 체계를 속여 악의적 메시지에 유효한 서명을 얻게 함

5.4 생일 공격

■ 예) 해시 함수 h가 n-비트 출력을 생성한다

- 원칙적으로 트루디는 다음과 같이 생일 공격을 시행함
  1. 트루디는 앨리스가 서명하지 않을 '악의적인' 메시지 E를 선택
  2. 그리고 트루디는 앨리스가 확실히 설명할 만한 무해한 메시지 I도 생성
  3. 그런 후에 트루디는 무해한 메시지를 약간 편집하여 $2^{n/2}$의 변수를 생성
  4. 이와 유사하게, 트루디는 악의적인 메시지 E_j의 $2^{n/2}$ 변수를 생성
  5. 트루디는 모든 악의적인 메시지 E_j와 모든 무해한 메시지 I_i를 해시

  충돌을 고려하여 트루디는 I_k를 앨리스에게 보내고 서명을 요청
  앨리스는 거기에 서명을 하고 I_k와 $[h(I_k)]_{\text{앨리스}}$를 트루디에게 보냄
  $h(E_j)=h(I_k)$이므로, $[h(E_j)]_{\text{앨리스}}=[h(I_k)]_{\text{앨리스}}$가 되고,
  결과적으로 트루디는 실질적으로 악의적 메시지 E_j에 대한 앨리스의 서명을 획득하게 됨

5.4 생일 공격의 영향

- 디지털 서명 위조
  - 공격자가 악의적 메시지를 합법적으로 서명된 것처럼 위조 가능
- 보안 강도 저하
  - 해시 함수 출력이 n비트여도 실제 보안 강도는 n/2 비트 수준
- 예시
  - SHA-1 (160비트) → 이론상 2^{80} 시도로 충돌 가능
  - 실제: 2017년 Google SHAttered 프로젝트에서 충돌 공격 입증

5.4 생일 공격의 방어

- 출력 비트 길이를 충분히 길게 설계
  - n비트 출력 → 실제 충돌 내성 ≈ $2^{\frac{n}{2}}$ 수준 고려 필요
- 안전한 해시 함수 선택
  - MD5, SHA-1: 더 이상 안전하지 않음
  - SHA-2, SHA-3 계열 권장
- 추가적 보안 대책
  - 디지털 서명 시 해시 이중 적용, 구조적 검증 강화
  - 최신 PKI 시스템에서는 취약 알고리즘 사용 금지

5.5 비암호화 해시

- 정의:
  - 데이터 요약 또는 오류 검출 목적으로 설계된 단순 해시 함수
- 특징
  - 빠르고 단순
  - 보안 속성(충돌 방지, 단방향성 등) 부족
- 용도
  - 주로 데이터 무결성 검사나 전송 오류 탐지
  - 암호화 응용에는 적합하지 않음

5.5 비암호화 해시

■ 단순 해시 함수 예시

- $h(X) = (X_0 + X_1 + \cdots + X_{n-1}) mod\ 256$
- 출력: 8비트 (매우 짧음)
- 문제점:
  - 출력 공간이 작아 충돌이 쉽게 발생
  - 단순한 연산이라 공격자가 의도적으로 충돌 만들 수 있음

- 예시
  - 두 개의 바이트 순서만 바꿔도 동일한 해시값 발생 → 충돌 유발

5.5 비암호화 해시

■ 많은 비암호화 해시는 암호화 응용 프로그램에 적합하지 않음

$X=(X_0,X_1,X_2,\ldots X^{n-1})$ 여기서 각각의 X_i는 바이트

$h(X)=(X_0+X_1+X_2+\cdots+X_{n-1})$ (mod 256)

- 두 개의 바이트를 교환하면 항상 충돌이 일어남

$h(10101010,00001111)=h(00001111,10101010)=10111001$

$h(X)=(nX_0+(n-1)X_1+(n-2)X_2+\cdots 2X_{n-2}+X_{n-1})$ (mod 256)

- 이 해시는 암호학적 측면에서 안전할까? 두 바이트를 서로 바꾼 결과

$h(10101010,00001111)\neq h(00001111,10101010)$

- 이 함수도 상대적으로 쉽게 충돌할 수 있음

$h(00000001,00001111)=h(00000000,00010001)=00010001$

5.5 비암호화 해시

■ 암호화 해시로 오해 받는 비암호화 해시

- 순환 중복 검사(CRC)
  - CRC 계산은 CRC '해시' 값으로 행동하는 나머지가 있는 긴 나눗셈
  - 일반적인 긴 나눗셈과 다르게 CRC에서는 빼기 대신에 XOR 연산을 사용

- WEP는 암호 무결성 검사가 필요한 곳에 실수로 CRC 체크섬을 사용
  - 프로토콜 공격자에게 문을 열어주는 것과 같음

5.5 비암호화 해시

■ 비암호화 해시의 보안적 한계

- 암호학적 요구사항 미충족
  - 단방향성 X
  - 충돌 방지 X
  - 약한 출력 길이
- 실제 위험 사례
  - WEP 보안 실패 (CRC 사용) → Wi-Fi 초창기 암호화 취약점
- 교훈
  - 해시 함수 = 무조건 안전하지 않음
  - 암호화에는 반드시 SHA-2, SHA-3 등 검증된 암호 해시 사용

5.6 SHA-3

■ SHA-3

- SHA-3 (Secure Hash Algorithm-3)
  - 2012년 공식 승인, SHA-1·SHA-2의 대체 표준
  - Keccak(케착) 알고리즘 기반
- 필요성
  - SHA-1, SHA-2 → MD5와 유사한 구조 (Merkle–Damgård)
  - MD5, SHA-1에서 취약점 발견 → 더 안전한 대안 필요
- 특징
  - 스폰지(Sponge) 구조 기반
  - 다양한 출력 길이 지원 (224, 256, 384, 512 비트)
- 케착 알고리즘은 메시지를 블록으로 해시('스폰지' 기술을 사용)
  - 스폰지의 '흡수' 과정 동안 각 블록은 차례로 현재의 내부 상태의 순열로 XOR 연산을 실시
  - 전체 메시지가 흡수되고 나면, '압축' 단계에서는 해시 값을 구성하는 내부 상태의 비트를 추출

5.6 SHA-3

- SHA-3 스폰지 기술
  - 스폰지 구조 특징
    - 입력 메시지를 블록 단위로 "흡수(absorb)"
    - 내부 상태에서 XOR·순열 연산 반복
    - 출력 단계에서 "짜내기(squeeze)"로 최종 해시 생성
  - 내부 상태 크기: 1600비트 (5×5×64 배열)
  - 라운드 함수: θ, ρ, π, χ, ι 단계 반복
  - 각 라운드 = 다섯 단계:
    - θ(Theta): 열 단위 패리티 → 확산
    - ρ(Rho): 비트 회전
    - π(Pi): 위치 치환
    - χ(Chi): 비선형 변환 (XOR & AND)
    - ι(Iota): 라운드 상수 추가
  - 24라운드 반복 (보안 강도 확보)

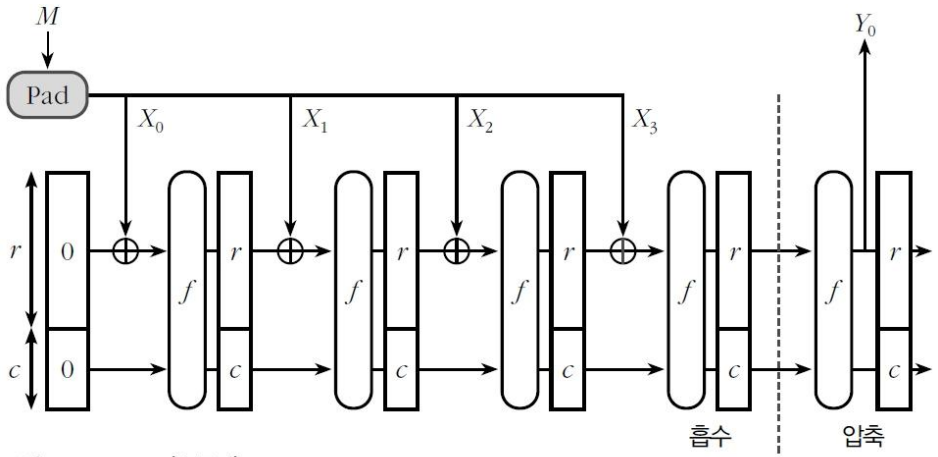

그림 5-2 SHA-3 '스폰지'

5.6 SHA-3

- 스폰지 기술의 보안성은(효율성뿐만 아니라) f의 선택에 달려 있음
- SHA-3에서 f함수는 24라운드로 구현되며 각 라운드는 5단계로 이루어져 있음
- SHA-3의 블록 크기는 1600비트
- 효율성을 위해 내부 상태는 64비트 단어의 5×5 그리드로 처리되며 각 단어는 '레인'으로 지칭
- 이 5×5 그리드의 레인 (i,j)에 있는 64비트 단어를 $A[i,j]$로 표시

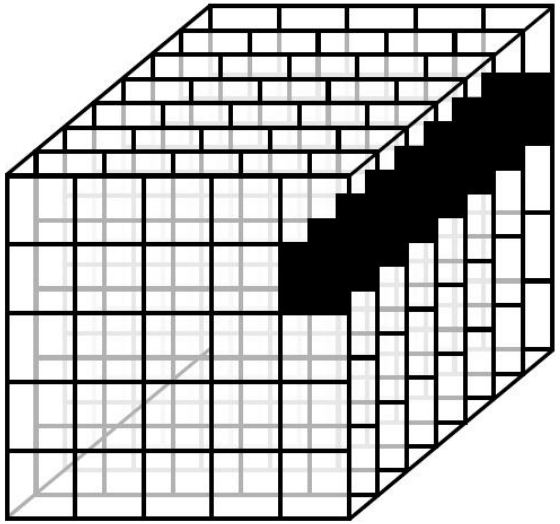

그림 5-3 SHA-3 상태 $A[x,y]$(64비트 단어의 5 x 5 배열)

5.6 SHA-3

- SHA-3의 i 라운드에서의 단계

표 5-1 SHA-3의 i라운드에서의 단계

```
// θ 단계
for x = 0 to 4
  C[x] = A[x,0]⊕A[x,1]⊕A[x,2]⊕A[x,3]⊕A[x,4]
next x
for x = 0 to 4
  D[x] = C[x−1]⊕(C[x+1] ⋘ 1)
next x
for x = 0 to 4
  for y = 0 to 4
    A[x,y] = A[x,y]⊕D[x]
  next y
next x
// ρ, π 단계
for x = 0 to 4
  for y = 0 to 4
    B[y,2x+3y] = (A[x,y] ⋘ r[x,y])
  next y
next x
// χ 단계
for x = 0 to 4
  for y = 0 to 4
    A[x,y] = B[x,y]⊕((~B[x+1,y]) & B[x+2,y])
  next y
next x
// ι 단계
A[0,0]=A[0,0]⊕R_i
```

5.6 SHA-3

- 보안성
  - Merkle–Damgård 구조의 취약점(길이 확장 공격) 없음
  - 현재까지 알려진 충돌 공격 없음
- 유연성
  - 다양한 출력 길이 지원
  - SHAKE(확장 출력 함수, XOF) 제공 → 출력 길이를 원하는 만큼 생성
  - 실무 활용블록체인, 디지털 서명, 메시지 무결성 검증 등

5.6 SHA-3

- 보안성
  - Merkle–Damgård 구조의 취약점(길이 확장 공격) 없음
  - 현재까지 알려진 충돌 공격 없음
- 유연성
  - 다양한 출력 길이 지원
  - SHAKE(확장 출력 함수, XOF) 제공 → 출력 길이를 원하는 만큼 생성
- 실무 활용
  - 블록체인, 디지털 서명, 메시지 무결성 검증 등

5.7 HMAC

- 메시지 무결성을 위해 메시지 인증 코드인 MAC를 계산할 수 있음
- 메시지 무결성을 검증하기 위해서는 해시를 사용할 수 있어야 함

- 앨리스가 단순히 $h(M)$을 계산하거나 또는 M과 $h(M)$을 밥에게 보냄으로써 M의 무결성을 보호할 수 있는가?
  - 가장 확실한 방법은 앨리스가 해시 값을 대칭 암호 $E(h(M),K)$로 암호화해서 이것을 밥에게 보내는 것

- 실제로는 해시된 MAC인 HMAC를 계산하는 데 조금 다른 방법을 사용
  - 해시를 암호화하는 대신에 해시를 계산할 때 키를 M에 직접 혼합

    키를 메시지 시작 부분에 더하기: $h(K,M)$

    키를 메시지 뒤에 첨부하기: $h(M,K)$

5.7 HMAC

- 메시지 무결성을 위해 메시지 인증 코드인 MAC를 계산할 수 있음
- 메시지 무결성을 검증하기 위해서는 해시를 사용할 수 있어야 함

- MAC (Message Authentication Code)
  - 메시지 무결성과 인증 제공
  - 블록 암호 기반(CBC-MAC) 또는 해시 기반(HMAC)으로 구현
- HMAC (Hash-based MAC)
  - 해시 함수 + 비밀 키 조합
  - 메시지 무결성 + 송신자 인증 보장

- 앨리스가 단순히 $h(M)$을 계산하거나 또는 M과 $h(M)$을 밥에게 보냄으로써 M의 무결성을 보호할 수 있는가?
  - 가장 확실한 방법은 앨리스가 해시 값을 대칭 암호 $E(h(M),K)$로 암호화해서 이것을 밥에게 보내는 것

- 실제로는 해시된 MAC인 HMAC를 계산하는 데 조금 다른 방법을 사용
  - 해시를 암호화하는 대신에 해시를 계산할 때 키를 M에 직접 혼합

    키를 메시지 시작 부분에 더하기: $h(K,M)$

    키를 메시지 뒤에 첨부하기: $h(M,K)$

5.7 HMAC

- 메시지 무결성을 위해 메시지 인증 코드인 MAC를 계산할 수 있음
- 메시지 무결성을 검증하기 위해서는 해시를 사용할 수 있어야 함

- 앨리스가 단순히 $h(M)$을 계산하거나 또는 M과 $h(M)$을 밥에게 보냄으로써 M의 무결성을 보호할 수 있는가?
  - 가장 확실한 방법은 앨리스가 해시 값을 대칭 암호 $E(h(M),K)$로 암호화해서 이것을 밥에게 보내는 것

- 실제로는 해시된 MAC인 HMAC를 계산하는 데 조금 다른 방법을 사용
  - 해시를 암호화하는 대신에 해시를 계산할 때 키를 M에 직접 혼합

    키를 메시지 시작 부분에 더하기: $h(K,M)$

    키를 메시지 뒤에 첨부하기: $h(M,K)$

5.7 HMAC

- HMAC를 $h(K,M)$으로 계산하는 것을 선택했다고 가정
  - 대부분의 암호화 해시는 메시지를 블록으로 해시
  - MD5와 SHA-1 블록 크기는 512비트
  - 만약 $M=(B_1,B_2)$라면 여기서 B_i는 512비트
  - 그러면 함수 F에 대해 다음이 성립(A는 고정된 초기 상수)

$$h(M)=F(F(A,B_1),B_2)=F(h(B_1),B_2) \qquad (5.1)$$

  - 만약 트루디가 $M'=(M,X)$가 되도록 M'을 선택한다면 식 (5.1)을 사용해 $h(K,M)$에서 $h(K,M')$을 찾을 수 있음. (5.2)가 성립하기 때문

$$h(K,M')=h(K,M,X)=F(h(K,M),X) \qquad (5.2)$$

- 이번에는 $h(M,K)$를 선택했다고 가정
  - 해시 함수 h에 대해 알려진 충돌이 있어서 M'이 $h(M')=h(M)$으로 존재한다면 식 (5.1)에 의해 다음과 같이 됨

$$h(M,K)=F(h(M),K)=F(h(M')K)=h(M'K) \qquad (5.3)$$

  - 여기서 M과 M'은 각각 블록 크기의 배수. 만약에 충돌이 일어나면 해시 함수는 안전하지 않은 것으로 봄

5.7 HMAC

- HMAC를 계산하는 승인된 방법
  - B를 해시의 바이트 단위 블록 길이라고 가정
  - 현재 많이 사용하는 해시 함수에서는 B=64
    - ipad=0x36을 B번 반복
    - opad=0x5C를 B번 반복
  - M의 HMAC는 다음과 같이 계산
    - $\mathrm{HMAC}(M,K)=h(K \oplus \mathrm{opad}, h(K \oplus \mathrm{ipad}, M))$

5.7 HMAC

- 실제 활용 사례
  - TLS/SSL → HTTPS 통신 무결성 보장
  - IPSec → 네트워크 계층 보안
  - JWT (JSON Web Token) → 토큰 위변조 방지
  - API 인증 → 클라우드 서비스, 결제 시스템

- 정리
  - HMAC = 해시 함수 + 비밀 키 조합으로 메시지 무결성 & 인증 보장
  - 두 번 해시 구조 (ipad, opad) → 충돌·위변조 공격 방어
  - 보안 프로토콜 전반에 필수적으로 사용되는 핵심 기술

5.8 암호 해시 응용 프로그램

■ 온라인 입찰

- 앨리스, 밥, 척이 입찰하고자 하는 경매 물건이 온라인에 나왔다고 가정
  - 앨리스는 10달러에 입찰을 하고 밥은 12달러에 입찰을 한다고 가정 → 만약 척이 두 사람의 입찰 금액을 입찰 마감 전에 미리 안다면 12달러 1센트를 적어내서 입찰에 성공

    ↳ 나중에 입찰하는 것이 유리할 수 있기 때문에 어느 누구도 입찰 금액을 먼저 적어 내려고 하지 않음
  - 이러한 우려를 없애기 위해 각 입찰자는 자신의 입찰 금액을 결정(비밀)
  - 앨리스는 $h(A)$를 제출하고 밥은 $h(B)$를, 척은 $h(C)$를 제출
  - 세 사람이 세 개의 해시 입찰을 제출하면 해시 값을 온라인에 게시해 모두가 볼 수 있음

- 이 방법이 입찰을 직접 제출하는 것보다 나은 이유는 무엇일까?
  - 해시 값은 입찰 자체에 대한 정보는 노출시키지 않고 입찰자를 그들의 원래 입찰과 함께 묶어버림
  - 해시 값을 한 번 제출하고 나면 입찰을 바꾸는 것이 불가능하다면 입찰을 직접 제출했을 때 발생하는 문제를 예방

5.8 암호 해시 응용 프로그램

■ 블록 체인(blockchain)

- 비트코인과 같이 암호화폐 영역에서 유명해짐

■ 디지캐시(DigiCash)

- 가장 초기 형태이자 가장 잘 알려진 전자현금
- 디지캐시를 이루는 근본 아이디어는 차움의 블라인드 서명을 사용한다는 것

- 완전히 탈중앙화된 전자화폐 개념은 훨씬 최근에 발명됨
- 탈중앙화된 전자화폐 보안에 블록체인을 사용하는 것은 몇 가지 뛰어난 암호학을 기반으로 함

5.8 암호 해시 응용 프로그램

■ 원장

- 금융거래의 장부로 정의
- 미국 달러를 포함하는 모든 거래가 원장에 기록된다고 가정

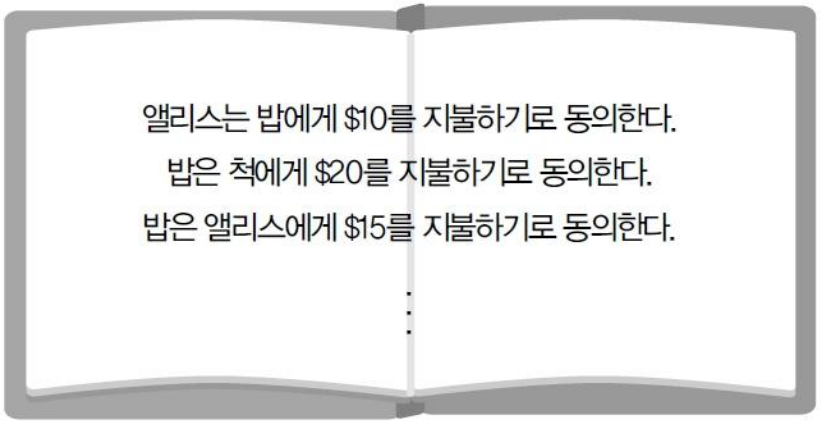

그림 5-4 원장 예시

- 블록체인을 사용하여 분산원장을 생성하고 유지할 수 있음
- 탈중앙화된 전자화폐는 암호 기술에 의지하기 때문에 이를 암호화폐라고 부를 것임

5.8 암호 해시 응용 프로그램

■ 암호화폐의 장점과 단점

- 장점
  - 완전히 탈중앙화된 암호화폐는 그 어떤 중앙 권한 즉, 은행이나 정부도 필요하지 않으므로 정치적 영향력이나 실물 화폐에 영향을 주는 다양한 조작으로부터 자유로움
- 단점
  - 범죄자들이 전자화폐 거래를 선호할 수 있어 범죄 수단이 될 가능
  - 정부는 통화를 조작할 수 있는 능력을 잃을 수 있어서 정책을 수립하고 이행하는 데 어려움을 겪을 수 있음

- 탈중앙화된 전자화폐를 구성하기 위해서는 작업 단위가 필요
  - 암호 해시를 사용

- 암호 해시 함수 $h(x)$가 N비트 출력을 생성해낸다고 가정
  - 어떤 입력 R에 대해 $0 \le h(R) < 2^N$이며, 입력 $R_1, R_2, R_3, \ldots$에 대해 그 결과로 나오는 해시 값이 $h(R_1), h(R_2,), h(R_3)\ldots$로 균등하게 배분된다는 점에서 이 범주 안의 모든 해시 값은 동등

5.8 암호 해시 응용 프로그램

■ 분산원장

- 중앙 권한이 없는 분산원장의 경우
  - '앨리스가 밥에게 10달러를 주기로 동의했다'와 같은 항목이 유효한지 어떻게 알 수 있는가?
  - 원장 항목에 디지털 서명을 요구하면 쉽게 해결됨

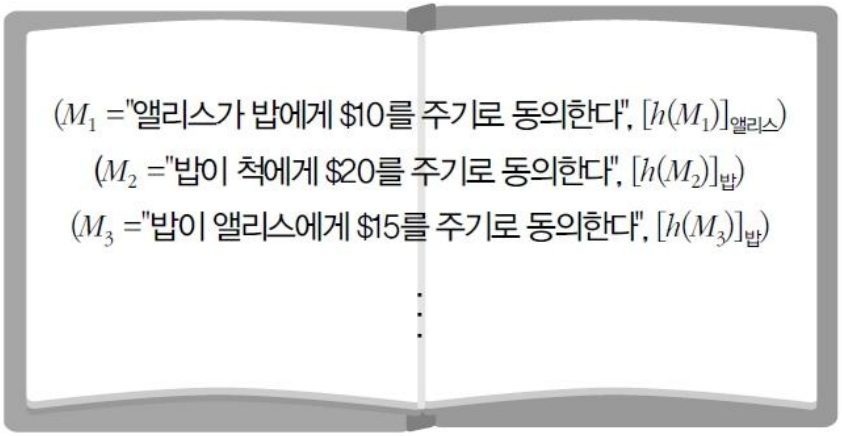

그림 5-5 서명된 원장 항목

  - 서명된 전자원장에는 몇 가지 명확한 보안 문제가 남아 있음
    ↳ 모든 항목은 몇 번을 반복해도 여전히 유효하다는 것

5.8 암호 해시 응용 프로그램

- 다음과 같은 내용이 다섯 번 반복된다고 가정

$$(M_1 = \text{"앨리스가 밥에게 \$10를 주기로 동의한다"}, [h(M_1)]_{\text{앨리스}})$$

  - 실제로는 앨리스가 이 거래 중 하나에만 서명을 했을지라도 앨리스가 밥에게 50달러를 주기로 동의했다는 것처럼 보일 수 있음
  - 이 문제를 해결하기 위해, 각 항목에 거래 번호를 추가할 수 있음

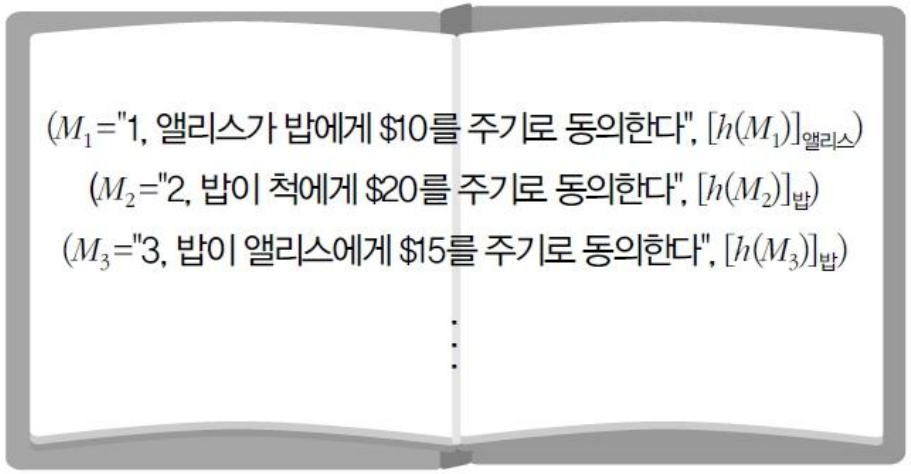

그림 5-6 숫자가 매겨지고 서명된 원장 항목

5.8 암호 해시 응용 프로그램

- 원장이 가지는 또 다른 문제
  - 어떤 사람이 실제로 낼 수 없는 돈을 지불하겠다고 할 수 있다는 것
  - 그런 과소비를 막기 위해, 모든 사람들이 장부에 초기 금액을 지불하도록 요구할 수 있고, 그다음 그들이 가진 것보다 더 많이 쓰지 않는지 확인할 수 있음

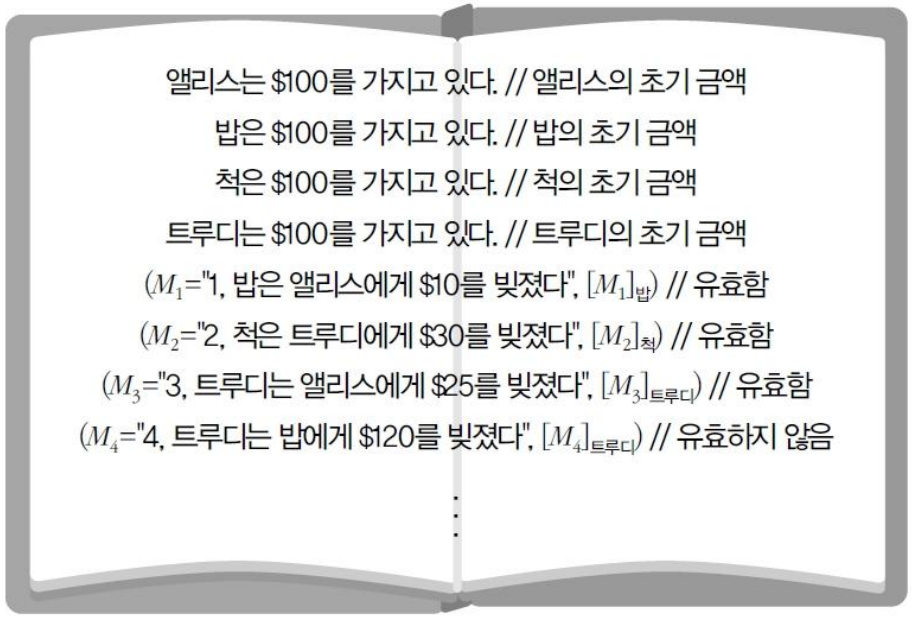

그림 5-7 유효하지 않은 정보가 추가된 원장

5.8 암호 해시 응용 프로그램

- 분산형 원장이 갖는 문제를 해결하기 위한 규칙
  - 거래는 서명되어야만 함
  - 어느 누구도 초과 인출할 수 없음
  - 거래는 모든 사람에게 공개되어야 함

    ↳ 이 규칙을 적용하더라도 여전히 여러 개의 원장이 언제든 존재 가능

- 스탬프코인(stampcoin)
  - 모든 화폐는 그것이 암호화폐든 아니든 그 자체를 나타내는 상징이 필요
  - 여기서 논의되는 암호화폐의 상징으로 우리는 $를 사용

5.8 암호 해시 응용 프로그램

■ 블록체인은 블록이랑 체인이 필요

- 개인 거래를 묶어서 블록(여기서는 B)으로 그룹 짓겠음

- 암호 해시 함수 h는 N-비트 출력을 생산
- 특정한 m에 대해 $h(B,R)<2^m$이 되는 숫자 R을 찾아보기
  - 이것은 $h(B,R)$이 적어도 0s를 이끄는 $N-m$개를 가지는 것과 같음
  - R을 찾도록 예상하기 전에 2^{N-m}에 대해 계산해야 함
  - 주어진 B에 대해 만약 $h(B,R)<2^m$이 되도록 R을 생산해낼 수 있다면, 이것은 평균적으로 2^{N-m} 작업의 단위를 수행했음을 확인하는 데 도움이 될 것임
  - 이러한 해시 값은 블록을 검증하는 역할을 함
  - 물론, R이 주어지면, 단일 해시를 계산하여 그렇게 주장된 유효성을 검증 가능

5.8 암호 해시 응용 프로그램

- 유효한 *R*을 찾는 데 필요한 해시를 계산하는 사람을 위한 인센티브는 무엇인가? 정답은 '공짜' 돈
  - 누구든지 제일 먼저 찾는 사람은 1달러인 1스탬프코인을 받게 됨
  - 누구든지 새로운 블록 *B*를 만들어낼 수 있지만 유효한 해시를 생산하는 *R*을 찾는 데는 많은 노력이 듦

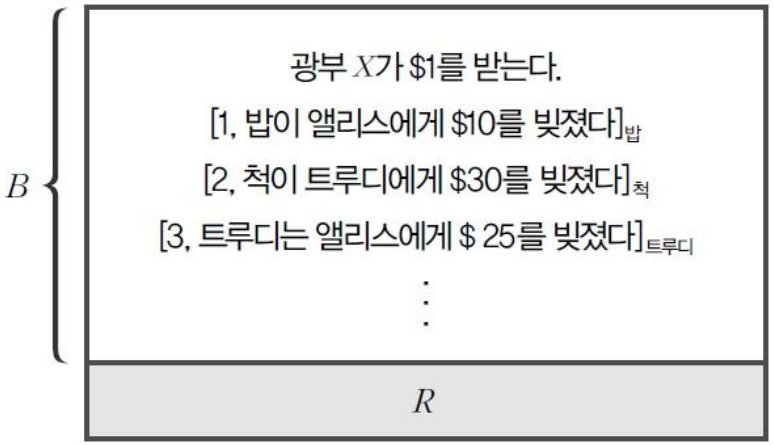

그림 5-8 $h(B,R)<2^m$인 블록 B와 R

5.8 암호 해시 응용 프로그램

- 체인은 효율성을 고려한 것
  - 단계마다 각각의 블록을 재검증하고 싶지 않기 때문에 블록을 체인으로 묶게 되는 것
  - 체인을 생성하기 위해 이전 블록의 해시 값을 현재 블록의 헤더에 둠
  - 만약 Y가 이전 블록의 해시라면 블록 B를 위한 검증 과정은 $h(Y,B,R)<2^m$이 되도록 R을 찾고자 함

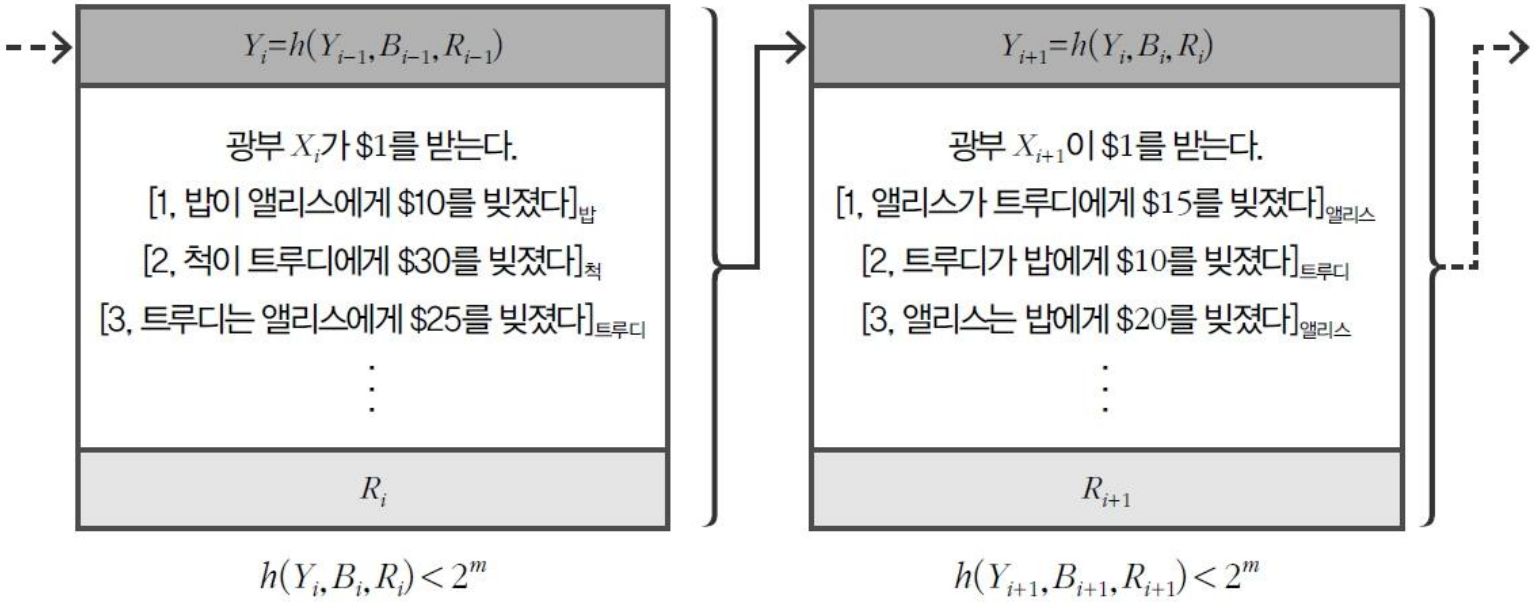

그림 5-9 블록체인의 일부

5.8 암호 해시 응용 프로그램

■ 블록체인 기반의 암호화폐의 주요 요점

- 새로운 거래는 방송된다.
- 광부들은 거래를 모아서 블록으로 만든다.
- 광부들은 유효한 블록 해시를 계산하기 위해 경쟁하는데 이를 통해 새로운 블록이 블록체인에 통합될 수 있다.
- 유효한 해시를 찾은 광부는 이것을 공개한다.
- 새롭게 채굴된 블록은 모든 거래가 서명되고, 초과 인출이 없고 새로운 블록 해시가 유효하면 받아들여지게 된다.
- 새롭게 받아들여진 블록은 블록체인으로 확장되고 광부들은 새롭게 확장된 블록체인을 계속되는 채굴 계산에 사용하게 된다.

5.8 암호 해시 응용 프로그램

■ 공격 시나리오: 트루디가 100달러를 가지고 있음

- T = [1,트루디가 앨리스에게 $100를 지불한다]$_{\text{트루디}}$
  - 트루디는 이것을 오직 앨리스에게만 보냄(트루디의 사적 거래 T)
  - 앨리스가 이 사적 거래를 받아들이면 트루디는 자신의 100달러를 또 마음껏 쓸 수 있음

- 트루디가 사용하는 이러한 이중 소비 공격을 무엇으로 막을 수 있을까?
- 트루디의 공격이 성공할 가능성은 트루디의 통제하에 있는 네트워크의 전체 컴퓨팅 파워(computing power)에 정비례함
  - 블록체인의 가장 최근 블록은 신뢰도가 가장 낮다. 아주 적은 컴퓨팅 파워로도 트루디는 블록을 채굴할 수 있으며 다른 누구보다 먼저 사적 블록체인을 만들 수 있음

- 암호화폐 접근 방식에는 몇 가지 가능한 개선 사항이 존재
  - 네트워크의 컴퓨팅 파워가 증가함에 따라 매개변수 m에 기반을 둔 해시의 예상되는 수를 조정 가능
  - 머클 트리(merkle trees) 사용을 포함하는 것

5.9 기타 암호 관련 주제

■ 비밀 공유

- 앨리스와 밥은 비밀 s를 다음과 같은 의미로 공유하고자 함
  - 앨리스와 밥 모두 단독으로는 비밀 s를 단지 막연히 추측하는 것 이상으로 더 정확하게 알아낼 수는 없음
  - 앨리스와 밥은 함께 비밀 s를 구할 수 있음
  - 비밀 s를 획득하기 위해 두 참가자가 모두 협동해야 하기 때문에 이것을 비밀 공유 체계라고 부름

5.9 기타 암호 관련 주제

- 비밀 *s*가 실제 숫자라고 가정

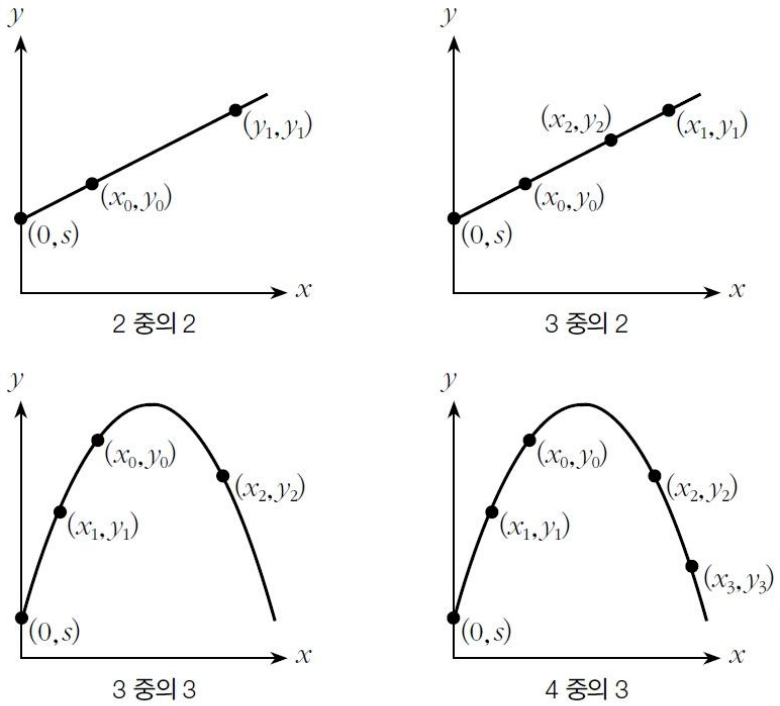

그림 5-10 비밀 공유 체계

5.9 기타 암호 관련 주제

■ 조건부 키

- 비밀 공유가 유용하게 사용될 특정 응용 분야 중 하나는 조건부 키 문제
- 사용자의 키를 공식적인 조건부 기관에 맡기도록 요청했다고 가정
  - 미국 정부는 조건부 키를 촉진하고 심지어 조건부 키를 포함하는 체계 (clipper and capstone)를 개발하고자 함

    ↳ 조건부 키 아이디어는 많은 비난을 받고 결국 취소됨

  - 조건부 키와 관련된 한 가지 문제는 조건부 기관을 신뢰하지 못할 수도 있다는 것

    ↳ 여러 개의 조건부 기관을 둠으로써 그리고 사용자가 키를 *n*개로 나눠서 *n*개 중 *m*개를 함께 이용해야만 키를 복구할 수 있게 함으로써 어느 정도 개선 가능

  - 샤미르의 비밀 공유 체계는 이런 조건부 키 체계를 실행하는 데 사용될 수 있음

5.9 기타 암호 관련 주제

■ 시각적 암호

- 나오와 샤미르[89]는 흥미로운 시각적 비밀 공유 체계를 제안
- 예) 흑백 이미지로 두 개의 투명도를 만드는데 하나는 앨리스를 위한 것이고 또 다른 하나는 밥을 위한 것
  - 여기서 몫 1은 앨리스의 투명도, 몫 2는 밥의 투명도로 이해할 수 있음

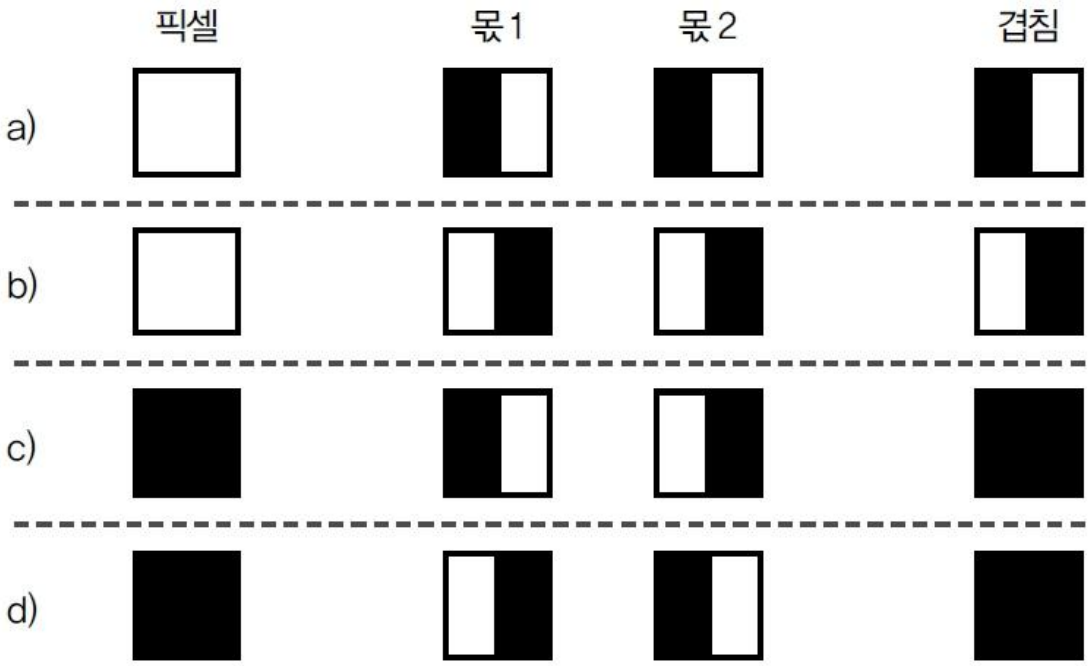

그림 5-11 픽셀을 몫 1과 몫 2로 공유

5.9 기타 암호 관련 주제

- 원본 픽셀이 검은색이면 겹치는 몫은 항상 검은색 픽셀로 나타남
- 원본 픽셀이 흰색이면 겹치는 몫은 반은 흰색/반은 검은색 픽셀로 나타나며 이것은 회색으로 인식됨
- 이것은 대비의 손실(검은색과 회색 대 검은색과 흰색)로 나타나겠지만 원본 이미지는 여전히 명확하게 구별
- 예) [그림 5-12]에서 앨리스와 밥에 대한 각각의 몫과 함께 앨리스와 밥의 몫이 겹쳐져서 생기는 이미지도 보여줌

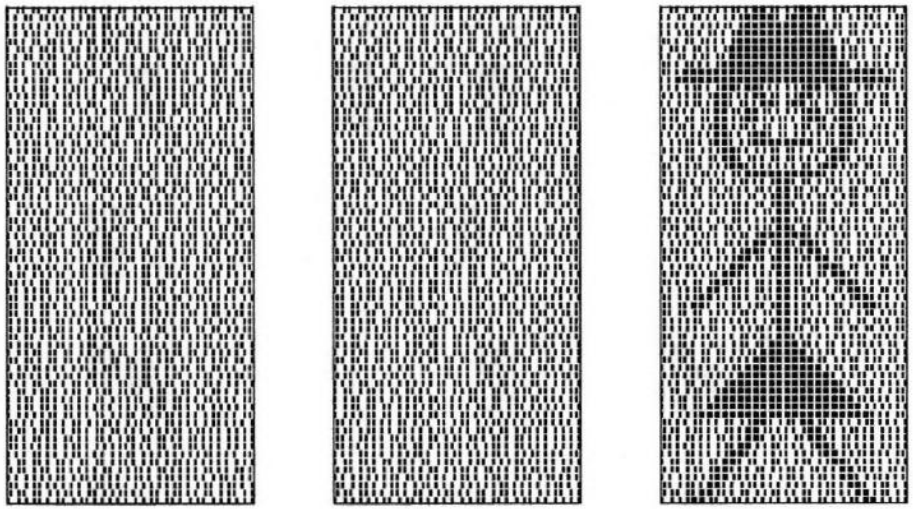

그림 5-12 앨리스의 몫, 밥의 몫, 그리고 겹치는 이미지

5.9 기타 암호 관련 주제

■ 난수

- 암호학에서 난수는 대칭키와 RSA 키 쌍, 즉, 무작위로 선택되는 큰 소수들 그리고 디피-헬먼 비밀 지수 등을 생성하는 데 사용되어 옴
- 난수는 시뮬레이션과 다양한 통계 응용 프로그램과 같은 비보안 응용 프로그램에서 사용
  - 난수는 통계적으로 무작위인 특정 분산 요구를 만족해야 함

- 서버가 사용자들을 위한 일련의 대칭키를 생성한다고 가정
  - K_A는 앨리스의 키, K_B는 밥의 키, K_C는 척의 키, K_D는 데이브의 키
  - 만약 K_D를 K_A, K_B, K_C를 통해 알아낼 수 있다면 그 체계의 보안은 무너지게 됨
  - 흔히 사용되는 의사 난수 발생기는 예측 가능

5.9 기타 암호 관련 주제

■ 텍사스 홀뎀 포커

- 난수를 생성하는 잘못된 방법을 잘 표현하는 실제 세계 예

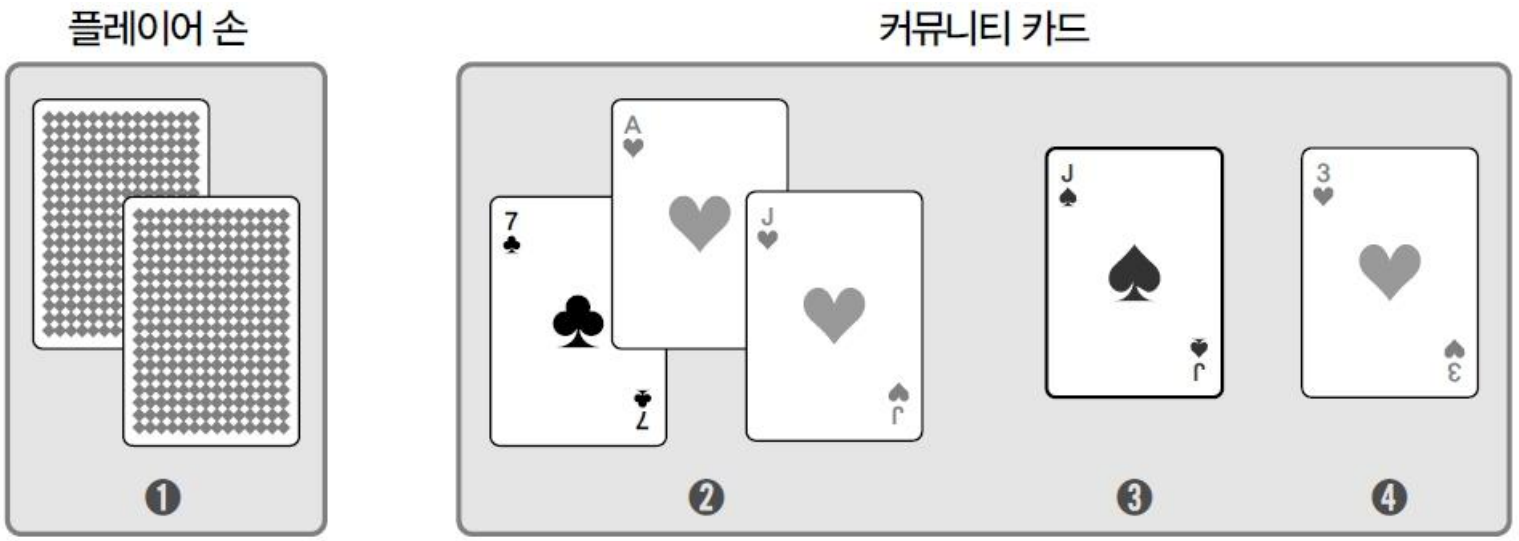

그림 5-13 텍사스 홀뎀 포커

5.9 기타 암호 관련 주제

■ 무작위 비트 생성

- 진정한 무작위는 찾기 힘들 뿐만 아니라 정의하기도 힘듦
  - 아마도 최선의 방법은 클라우드 섀넌이 개발한 엔트로피 개념일 것

- 진정한 무작위의 원천은 확실히 존재
  - 예를 들어, 방사능 물질의 붕괴가 그 무작위가 될 수 있음

5.9 기타 암호 관련 주제

- 정보 은닉 – 스테가노그래피(steganography)
  - '감추어진 기록'이란 의미로 정보가 전송된다는 사실을 숨기려는 시도
  - 전장에서 긴 역사를 가지고 있음
  - 현대 버전의 스테가노그래피는 사진 파일, 음성 데이터 또는 소프트웨어와 같은 미디어 속에 정보를 은닉하는 것이 포함

- 워터마크는 투명성과 가시성으로 분류
  - **투명성** 사람들 눈에 보이지 않는 워터마크
  - **가시성** 일급 기밀 도장처럼 눈에 보이는 워터마크

- 워터마크는 강한 워터마크와 약한 워터마크로 분류
  - **강한 워터마크** 공격을 받아도 읽을 수 있는 워터마크
  - **약한 워터마크** 공격을 받으면 파괴되거나 손상되는 워터마크

5.9 기타 암호 관련 주제

■ 스테가노그래피의 형태로 숨어 있는 정보에 접근하는 예①

- 디지털 이미지에 적용 가능
  - [그림 5-14]에서 두 개의 앨리스 이미지를 살펴보기
  - 왼쪽의 앨리스는 어떤 숨겨진 정보도 없지만 오른쪽의 앨리스는 낮은 자리 비트의 RGB에 『이상한 나라의 앨리스』 책이 있음
  - 보통 사람의 눈에는 어떤 해상도에서든 두 이미지가 동일하게 보임
  - 두 이미지의 비트를 비교하면 차이가 있음

(a) 평문 앨리스　　(b) 그녀의 책을 숨긴 앨리스

그림 5-14 두 앨리스의 이야기

5.9 기타 암호 관련 주제

■ 스테가노그래피의 형태로 숨어 있는 정보에 접근하는 예②

- 「월러스와 목수」[15]의 다음과 같은 문구를 담고 있는 HTML

"때가 되었어." 월러스가 말했다.
"많은 것들에 대해 이야기 할:
신발과 배와 실링 왁스에 대해
양배추와 왕에 대해
그리고 왜 바다가 뜨겁게 끓고 있는지
그리고 돼지가 날개를 가지고 있는지 아닌지에 대해."

- HTML에서 RGB 폰트 컬러는 다음과 같은 형태의 태그로 특정됨

 ...

(여기서 rr은 16진법에서 R의 값을, gg는 G를, bb는 B의 값을 나타냄)

- R, G, B의 낮은 자리 비트가 인식된 색에 영향을 주지 않기 때문에 [표 5-2]의 HTML의 일부분에서 나타나듯이, 이 비트에 정보를 숨길 수 있음

5.9 기타 암호 관련 주제

- 각 RGB 색의 낮은 자리 비트를 읽으면 '숨겨진' 정보 101 110 101 010 110 101이 나타남

표 5-2 간단한 스테가노그래피 예시

```
〈font color="#010001"〉"때가 되었어,"
                        월러스가 말했다.〈/font〉〈br〉
〈font color="#010100"〉"많은 것들에 대해 이야기 할:〈/font〉〈br〉
〈font color="#010001"〉신발과 배와 실링 왁스에 대해〈/font〉〈br〉
〈font color="#000100"〉양배추와 왕들에 대해〈/font〉〈br〉
〈font color="#010100"〉그리고 왜 바다가 뜨겁게 끓고 있는지〈/font〉〈br〉
〈font color="#010001"〉그리고 돼지가 날개를 가지고 있는지 아닌지에 대해〈/font〉〈br〉
```

5.10 요약

- 암호화 해시 함수
- SHA-3
- MAC(HMAC)를 계산하는 올바른 방법
- 암호 해시 함수의 응용 프로그램
  - 블록체인 기술, 암호화폐 응용 프로그램
- 샤미르의 비밀 공유 체계
- 나오와 샤미르의 시각적 암호화
- 난수 생성
- 정보 은닉
  - 디지털 스테가노그래피, 디지털 워터마킹

5.11 실습

■ 문제 1: 생일 공격 (Birthday Attack) 시뮬레이션

- "생일 문제"는 사람이 23명만 모여도 생일이 겹칠 확률이 50%가 넘는다는 걸 보여주는 유명한 확률 문제
- 이 원리는 해시 함수에도 똑같이 적용
- 즉, 해시값이 n비트라면, 약 $2^{n/2}$번만 시도해도 충돌을 발견할 수 있다는 뜻

■ 실습 요구사항

- SHA-256 해시를 구함
  - Hashlib.sha256(msg).digest() 활용
- 해시 결과 중 앞 16비트(=2바이트) 만 비교해서 충돌이 언제 생기는지 시뮬레이션
- 50의 중복 발생 관찰 시도 중 중복이 생길 때까지 진행하여 평균 몇 번 째에 중복이 발생하는지 출력

■ 문제 2: SHA-3 (Keccak) 해시 계산

- SHA-3는 최신 해시 함수
- 직접 내부 구조를 구현하기는 복잡하지만, 파이썬 표준 라이브러리를 사용해서 쉽게 해시를 계산할 수 있음

■ 실습 요구사항

- 문자열 "hello"와 "hello!"의 SHA3-256 해시를 생성
- 해시 결과를 16진수 문자열로 출력
- 두 값이 얼마나 다른지 비교해보세요. (입력 차이는 한 글자지만, 출력은 완전히 달라집니다.)