차례

6.1 개요

■ 접근제어(access control)

- 시스템 자원의 접근과 관련된 모든 보안 주제를 통칭하는 용어
- 인증과 인가
  - 인증(authentication) : 당신은 당신이라고 주장하는 사람이 맞는가?
  - 인가(authorization) : 당신은 그렇게 하도록 허락을 받았는가?

6.2 인증 방법

- 사람이 기계로부터 인증받을 수 있는 근거
  - 알고 있는 어떤 것
    - 패스워드
  - 가지고 있는 어떤 것
    - ATM 카드, 스마트폰
  - 본인 자체의 어떤 것
    - 생체인증(지문 생체인증, 안면 인식 등)
  - 추가적인 '어떤 것'이 가끔 필요하기도 함

6.3 패스워드

■ 이상적인 패스워드

- 사용자가 알고 있고 사용자가 알고 있는 것을 컴퓨터가 검증할 수 있어야 하며 무제한으로 컴퓨팅 자원에 접근할 수 있는 다른 누구라도 추측할 수 없는 것이어야 함

■ 패스워드처럼 행동하는 것들

- ATM 카드, PIN 번호 등

■ 왜 패스워드가 인기 있는가

- 첫째로는 비용이며 둘째로는 편리성
  - 패스워드는 무료이며 새로 생성하는 것이 훨씬 편리함

6.3 패스워드

■ 키 vs. 패스워드

- 트루디가 64비트 키를 사용하는 대칭 암호로 암호문 메시지를 해독한다고 가정
- 키는 무작위로 선택되었으며 지름길 공격도 없음
- 트루디는 정확한 키를 찾아내기 전에 평균적으로 2^{63}개의 키를 시도해 봐야만 함
- 트루디가 앨리스의 패스워드를 추측하고자 하는데 앨리스의 패스워드는 8자 길이이며 각 글자는 256개 중 선택 가능한 것으로 이루어져 있다고 가정

가능한 패스워드 $256^8=2^{64}$개

- 결과적으로 사용자는 기억하기 쉬운 것을 선택할 가능성이 훨씬 높음

Kf&Y3!aE보다

Frank012를 선택한다.

6.3 패스워드

■ 패스워드 선정

- 취약한 패스워드
  - Frank, Pikachu, incorrect, 10252010, AustinStamp

- 보안을 고려해 사용자들은 추측하기에 어려운 패스워드를 선택해야 함
  - jflej(43j-EmmL+y : 트루디가 알아내기 어렵겠지만 앨리스가 기억하기도 어려움
  - 09864376537263 : 대부분의 사용자가 기억하기 어려움
  - FS&7Yago : 앨리스가 기억하기에는 쉬우면서 트루디가 알아내기에는 어려움
  - Servenoterampartoriginal : 패스워드를 선정하는 최고의 방법

6.3 패스워드

■ 패스워드 실험(패스워드 선정과 관련하여 받은 충고)

- **그룹 A** 적어도 여섯 문자로 된 패스워드를 선정하라.
  - 패스워드의 약 30%가 해독하기 쉬웠음
- **그룹 B** 패스프레이즈에 근거해서 패스워드를 정하라.
  - 패스워드의 약 10%가 해독됨
- **그룹 C** 8자리의 문자를 무작위로 선택해 패스워드를 구성하라.
  - 패스워드의 약 10%가 해독됨

- 사용자들이 패스워드를 선택할 때 해줄 수 있는 최선의 조언은 단어에 기반한 긴 패스프레이즈를 선택하라는 것

6.3 패스워드

■ 패스워드를 통한 시스템 공격

- 트루디가 외부인이며, 특정 시스템에 대한 접근 권한이 없다고 가정
  - 트루디가 선택할 공격 경로 : 외부 → 일반 사용자 → 관리자

- 패스워드에 대한 공격이 감지되었을 때 어떻게 대응하는 것이 적절한가
  - 예: 일반적으로 틀린 패스워드를 3회 이상 반복해서 시도하면 시스템은 잠금 상태가 됨
  - 트루디는 모든 계정을 대상으로 각 계정에 대해 5분 내에 3회 패스워드 추측을 반복해서 시도함으로써 모든 계정을 영원히 잠금 상태에 있도록 만들 수 있음

6.3 패스워드

■ 패스워드 검증

- 컴퓨터는 어떤 형태로든 정확한 패스워드에 접근할 수 있어야만 함
  - 정보보안의 다른 분야와 마찬가지로 이러한 경우에 암호화가 좋은 해결 방법을 제공
  - 패스워드 파일을 대칭키로 암호화하는 방법을 먼저 생각해볼 수 있음
  - 하지만 패스워드를 검증하려면 복호화 키에도 접근이 가능해야 함(암호화는 가치가 X)
  - 해시된 패스워드로 저장하는 것이 훨씬 더 나음
  - 예: 앨리스의 패스워드가 servenoterampartoriginal일 때 파일에 다음과 같이 저장

$$y = h(\text{servenoterampartoriginal})$$

  - 패스워드를 보호하기 위해 암호화 해시 함수의 단방향(one-way) 특징에만 의존하고 있다는 것에 주목
  - 트루디가 다음과 같은 N개의 공통적인 패스워드를 담고 있는 사전을 가지고 있다고 가정

$$d_0,\ d_1,\ d_2,\ldots,\ d_{N-1}$$

  - 그러면, 트루디는 사전에서 각 패스워드의 해시를 미리 계산할 수 있음

$$y_0 = h(d_0),\ y_1 = h(d_1),\ldots,\ y_{N-1} = h(d_{N-1})$$

6.3 패스워드

- 트루디가 해시된 패스워드를 포함하는 패스워드 파일에 접근을 하게 된다면 패스워드 파일에 있는 엔트리와 미리 계산한 사전에 있는 엔트리를 비교하기만 하면 됨
- 미리 계산된 해시들을 담고 있는 사전은 재사용할 수 있으므로 트루디는 각 패스워드 파일별로 해시를 다시 계산하지 않아도 됨

- 공개키 암호화에 대한 순방향 탐색 공격을 방지하기 위해 메시지를 암호화하기에 앞서 메시지에 난수 비트를 추가한 것을 기억
- 솔트(salt)라고 불리는 비밀이 아닌 난수 값을 각 패스워드를 해시하기에 앞서 추가함으로써 패스워드에서도 이와 비슷한 효과를 볼 수 있을 것
- 패스워드 솔트는 CBC 모드 암호화에서의 초기화 벡터(IV)와 유사함

6.3 패스워드

■ 패스워드 크래킹의 수학

- 이 절에서는 모든 패스워드가 8개 문자로 되어 있고 각 문자는 128가지 선택이 가능해서 다음과 같은 수의 패스워드가 있다고 가정

$$128^8 = 2^{56}$$

- 패스워드가 패스워드 파일에 저장되어 있으며 2^{10}개의 해시된 패스워드를 포함하며, 트루디는 2^{20}개의 일반적인 패스워드를 수록한 사전을 가지고 있다고 가정
- 네 개의 다른 시나리오로 패스워드를 성공적으로 크래킹하는 가능성에 대해 알아볼 것
  - 첫째, 트루디가 앨리스의 패스워드를 알고 싶어 하며 트루디가 유사한 패스워드의 사전을 사용하지 않는다고 가정
  - 둘째, 트루디가 앨리스의 패스워드를 알고자 하며, 트루디가 유사어 사전을 사용하는 경우
  - 셋째, 트루디가 패스워드 파일에 있는 어떤 패스워드든 크랙하는 데 만족할 것이며 이 경우 트루디가 사전을 사용하지는 않음
  - 넷째, 트루디가 해시된 패스워드 파일에 있는 어떤 패스워드든 복구하고자 하며, 사전을 사용하는 경우

6.3 패스워드

■ 경우 I

- 유사어 패스워드 사전 없이 트루디는 모든 패스워드에 대해 확인을 시도

$$2^{56}/2=2^{55}$$

  - 패스워드가 솔트되거나 솔트되지 않거나 결과는 동일

■ 경우 II

- 트루디가 앨리스의 패스워드를 알아내려고 하는데 유사 패스워드 사용
  - 패스워드가 솔트되었다고 가정
  - 앨리스의 패스워드가 트루디의 사전에 있다고 가정

$$\frac{1}{4}\left(2^{19}\right)+\frac{3}{4}\left(2^{55}\right)\approx 2^{54.6}$$

  - 작업은 많아 봐야 2^{20}이며 성공 확률은 1/4일 것
  - 만약 패스워드가 솔트되지 않았다면, 공격 횟수가 많을수록 공격당 평균적인 작업량은 줄어듦

6.3 패스워드

경우 III

- 트루디는 해시 처리된 패스워드 파일에 있는 1024개의 패스워드 중 어느 한 개를 찾으면 만족할 것임(트루디는 패스워드 사전에 대해 잊어버림)
  - $y_0, y_1, \ldots, y_{1023}$을 패스워드 해시라고 가정
  - 파일에 있는 2^{10}개의 모든 패스워드는 서로 다른 것으로 가정
  - $p_0, p_1, \ldots, p_{2^{56}-1}$을 2^{56}개의 가능한 패스워드의 목록이라고 가정
  - 트루디는 부합되는 것을 찾기까지 서로 다른 2^{55}번의 비교를 할 필요가 있음
  - 만약 패스워드가 솔트되지 않았다면 $2^{55}/2^{10}=2^{45}$
  - 패스워드가 솔트되었다고 가정하면 각 해시 계산은 오직 하나의 비교를 낳으며 그 결과 예상되는 작업량은 2^{55}인데 이는 위에서 언급한 경우 I과 동일

6.3 패스워드

경우 IV

- 트루디는 해시된 패스워드 파일에서 1024개의 패스워드 중에서 어떤 패스워드든 찾는 데 만족할 것이며 자신의 패스워드 사전을 사용

$$1 - \left(\frac{3}{4}\right)^{1024} \approx 1$$

  - 파일에서 적어도 하나의 패스워드가 트루디의 사전에 있다는 것을 안전하게 가정
  - 만약 패스워드가 솔트되지 않았다면, 트루디는 자신의 사전에 있는 모든 패스워드를 단순히 해시할 것이며 그 결과를 패스워드 파일에 있는 1024개의 해시와 비교할 것임
  - 다시, 패스워드 중 적어도 하나가 트루디의 사전에 있다고 가정

$$2^{19}/2^{10} = 2^{9}$$

  - 솔트되지 않은 경우, 트루디는 단지 파일에 있는 해시와 자신의 사전 패스워드의 해시를 비교하여 그 과정에서 사전에 있는 어떤 패스워드든 찾게 됨

6.3 패스워드

가장 현실적인 경우

- 트루디가 유사 패스워드 사전을 가지고 있고 패스워드 파일에서 어떤 패스워드든 찾는 데 만족할 것이며, 파일에 있는 패스워드는 솔트되어 있다고 가정
  - $y_0, y_1, \ldots, y_{1023}$가 패스워드 해시라고 하며 $s_0, s_1, \ldots, s_{1023}$를 이에 상응하는 솔트라고 하고, 또한, $d_0, d_1, \ldots, d_{2^{20}-1}$를 트루디의 패스워드 사전에 있는 단어라고 가정
  - 먼저 트루디는 $h(d_0, s_0)$를 계산하고 이를 y_0와 비교 → 트루디는 $h(d_1, s_0)$를 계산하여 그 값을 y_0와 비교 → $h(d_2, s_0)$를 계산한 후 y_0와 비교하는 방식으로 계속해서 진행함
  - 만약 y_0가 사전에 있다면, 트루디는 약 2^{19} 해시 후에 이를 찾을 것으로 예상
  - 만약 y_0가 사전에 없다면, 트루디는 2^{20} 해시를 계산할 것
  - $\frac{1}{4}\left(2^{19}\right) + \frac{3}{4} \cdot \frac{1}{4}\left(2^{20} + 2^{19}\right) + \left(\frac{3}{4}\right)^2 \frac{1}{4}\left(2 \cdot 2^{20} + 2^{19}\right) + \cdots + \left(\frac{3}{4}\right)^{1023} \frac{1}{4}\left(1023 \cdot 2^{20} + 2^{19}\right) < 2^{22}$
  - 합리적인 가정하에서 (솔트된) 패스워드를 크랙하는 데 필요한 작업은 대략 사전의 크기를 주어진 패스워드가 사전에 있을 확률로 나눈 것과 같음

$$\frac{2^{20}}{1/4} = 2^{22}$$

6.3 패스워드

■ 패스워드 크래킹의 핵심

- 계산된 해시의 수라는 점에서 작업량은 변하지 않지만, 패스워드를 크랙하는 데 필요한 시간은 해시 함수의 느림에 정비례해서 늘어남
- 패스워드를 크랙하고자 시도할 때 트루디는 전형적으로 많은 수의 해시를 계산해야 할 것이며, 이에 걸리는 시간은 합리적으로 잘 선택된 패스워드를 공격하기에는 불가능할 정도로 길어질 수 있음

6.3 패스워드

■ 다른 패스워드 문제들

- 꽤 많은 패스워드가 재사용됨
- 사회 공학 기법도 패스워드와 관련된 또 다른 주요 걱정거리
- 키 입력 로깅 소프트웨어나 이와 유사한 스파이웨어 또한 패스워드에 기반을 둔 보안에 심각한 위협이 됨

6.4 생체인증

■ 생체인증(biometrics)

- '본인 자체의 어떤 것'을 대표하는 인증 방법으로 슈나이어가 말한 것과 같이 '당신 자신이 키'
  - 지문, 손바닥 인증, 목소리 인증, 안면 인증, 보행 인증, 심지어 디지털 강아지 등

■ 이상적인 생체인증은 다음을 만족함

- 보편성
- 구별성
- 영구성
- 수집성
- 신뢰성
- 강건성
- 사용자 친화성

6.4 생체인증

- 생체인증은 다양한 식별 문제에 적용 가능
  - 식별 문제에서 '당신은 누구인가?'에 대한 질문의 답을 찾으려고 노력
  - 인증 문제에 대해서는 '당신은 당신이라고 주장하는 사람이 맞는가?'에 대한 답을 찾고자 노력
  - 식별 문제에서는 일대다의 비교, 인증에서는 일대일의 비교가 이루어짐

- 생체인증 시스템에는 두 단계가 존재
  - 첫 번째, 등록 단계
    - 대상자들이 그들의 생체인증 정보를 모으고 데이터베이스에 입력하는 단계
  - 두 번째, 인증 단계
    - 생체인증 시스템이 실제적으로 사용자를 인증할 것인가 인증하지 않을 것인가를 결정하는 시점에 이루어짐
    - 신속, 단순, 정확해야 함

6.4 생체인증

■ 오류의 종류

- 트루디가 앨리스인 척하고 시스템이 실수로 트루디를 앨리스로 인증했다고 가정
  - 이러한 잘못된 인증이 발생하는 비율을 기만율(fraud rate)이라고 함
- 앨리스가 자신임을 인증받으려고 하지만 시스템이 인증하는 데 실패했다고 가정
  - 이런 종류의 오류 비율을 모욕률(insult rate)이라고 함
- 그 어떤 생체인증에서든 기만율 혹은 모욕률을 감소시키면 반대편은 증가하게 됨
- 기만율과 모욕률이 같도록 시스템의 매개변수를 설정하면 동일한 오류율이 발생

6.4 생체인증

■ 생체인증 예시 - 지문

- 서명 형식으로 고대 중국에서 사용되었으며 역사상 다른 시대에서도 비슷한 목적으로 사용되어 옴(과학적 방법으로 사용한 것은 최근)
  - 1798년 마이어(J. C. Mayer)는 지문이 독특할 수도 있다는 것을 제안
  - 1823년 요하네스 에반젤리스트 푸르킨제(Johannes Evangelist Purkinje)는 9가지 지문 형태에 대해 논함(논문 수준에 머무름)
  - 1858년 인도에서 지문을 식별의 수단으로 사용(윌리엄 허셀 경이 손바닥과 지문을 사용해서 계약서에 서명함)
  - 1880년 헨리 폴즈(Henry Faulds) 박사가 지문을 식별 목적으로 사용하는 데 대해 논의한 기사를 네이처지에 발행
  - 1883년에 발행된 마크 트웨인(Mark Twain)의 『미시시피에서의 삶』에서 지문 인식을 통해 살인자를 잡음
  - 1892년, 프랜시스 골턴(Francis Galton) 경이 지문의 '융기선'에 기반을 둔 식별 시스템을 개발(지문이 널리 퍼짐) - 지문은 시간이 지나도 변하지 않는다는 사실을 검증함

6.4 생체인증

- 골턴의 식별 시스템에 있는 여러 종류의 지문 예시

(a) 고리 모양 (b) 소용돌이 모양 (c) 아치 모양

그림 6-1 골턴의 지문 융기선 무늬 예시

- 현대의 지문 생체인증은 먼저 지문의 이미지를 캡처하는 것에서 시작 → 이미지 처리 기술을 이용하여 이미지를 개선시킴 → 해당 이미지에서 다양한 점을 식별하여 추출

(a) 원본 지문 (b) 일부 융기선

그림 6-2 추출한 융기선

6.4 생체인증

■ 생체인증 예시 - 손의 기하학적 형태

- 손의 모양, 즉 장문을 세밀하게 측정 : 보안 시설 출입에 사용

- 장점
  - 손의 기하학적 형태 시스템의 한 가지 장점은 빠르다는 것
  - 인간의 손은 대칭이어서 등록된 손에 붕대를 감고 있다면 다른 손의 손바닥을 위로 가게 하여 그 손을 대신 이용할 수도 있음

- 단점
  - 아주 어리거나 아주 나이가 많은 경우 사용하기 어려움(오류율이 상당히 높음)

6.4 생체인증

■ 생체인증 예시 - 홍채 인식

- 이론상으로 최고의 생체인증 방법 중의 하나
- 홍채의 발달(눈의 색깔 부분)은 복잡하여 사소한 변화도 큰 차이를 만듦
- 일란성 쌍둥이라도 다르며, 한 사람의 두 눈동자도 서로 다름
- 패턴이 사람의 전 생애 동안 안정적임

  - 1936년에 프랭크 버치 Frank Burch에 의해 제안
  - 1980년대에 제임스 본드 영화에서 홍채 인식 아이디어가 다시 살아남
  - 1986년이 되어서야 특허를 내면서 사람들이 이 기술의 잠재력을 보기 시작함
  - 1994년에 캠브리지 대학의 연구자 존 더그맨(John Daugman)이 현재 사용하고 있는 최고의 홍채 인식 접근 방법에 대한 특허를 받음

6.4 생체인증

- 홍채 시스템에는 상당히 정교한 장비와 소프트웨어가 필요
  - 자동화된 홍채 스캐너가 홍채의 위치를 잡고 눈의 이미지를 캡처
  - 그 결과 나타나는 이미지는 이차원적 웨이블릿 변환을 이용해서 처리됨

  → 256바이트(즉, 2048비트)의 홍채 코드로 나타나게 됨

  - 두 개의 홍채 코드는 코드 간의 해밍 거리를 기반으로 비교
  - 앨리스가 홍채 스캔을 이용해서 인증을 한다고 가정

$$d(x,y) = \text{불일치 홍채의 수/비교된 홍채의 수} \qquad (6.1)$$

  - 예를 들어, $d(0010,0101)=3/4$과 $d(101111, 101001)=1/3$이다.

6.4 생체인증

- 히스토그램은 일치된 경우에서 수집된 데이터를 나타내고, 오른쪽의 히스토그램은 불일치의 경우에 수집된 데이터를 나타냄
  - 이 그래프에서 일치 데이터의 평균은 0.11이고 불일치 데이터의 평균은 0.46

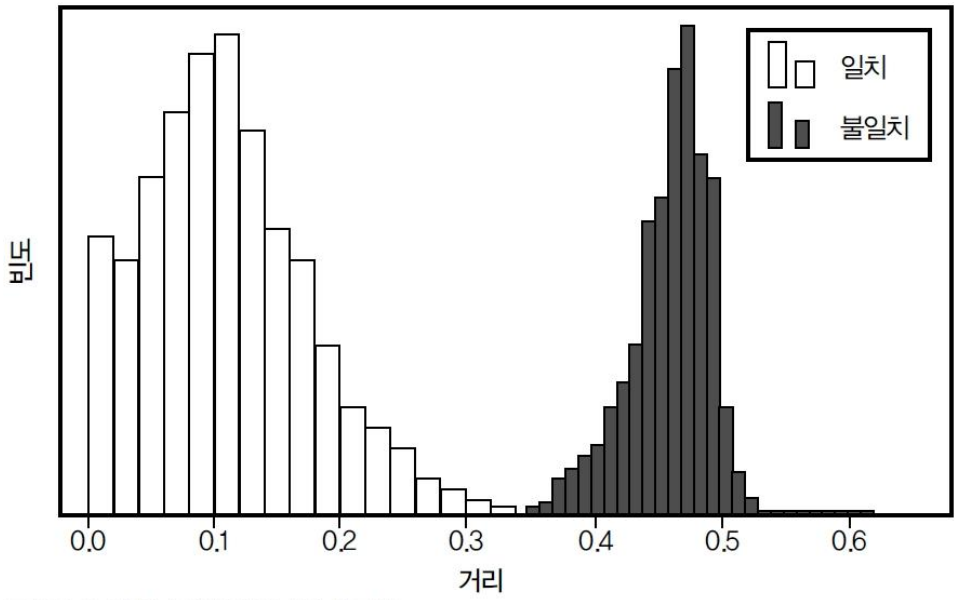


그림 6-3 홍채 스캔 결과의 히스토그램

- 230만 개의 비교에 근거를 둔 [그림 6-3]의 히스토그램을 보면 '같은(일치)'과 '다른(불일치)' 경우 사이에 겹치는 부분이 사실상 존재하지 않기 때문에, 홍채 스캐닝은 인증을 위한 최상의 생체인증 방법으로 일컬어짐

6.4 생체인증

- 홍채 인식 거리와 기만율

표 6-1 홍채 인식 거리와 기만율

거리	기만율
0.29	$1/1.3\times10^{10}$
0.30	$1/1.5\times10^{9}$
0.31	$1/1.8\times10^{8}$
0.32	$1/2.6\times10^{7}$
0.33	$1/4.0\times10^{6}$
0.34	$1/6.9\times10^{5}$
0.35	$1/1.3\times10^{5}$

6.4 생체인증

■ 생체인증 오류율

- 동일 오류율(기만율과 모욕률이 동등한 지점)은 서로 다른 생체인증 시스템을 비교하는 데 사용할 수 있는 측정 수단

표 6-2 생체인증 동일 오류율 [121]

생체인증	동일 오류율
지문	2.0×10^{-3}
손의 기하학적 형태	2.0×10^{-3}
음성 인식	2.0×10^{-2}
홍채 스캔	7.6×10^{-6}
망막 스캔	1.0×10^{-7}
서명 인식	2.0×10^{-2}

6.4 생체인증

■ 생체인증 결론

- 생체인증은 불가능하지는 않지만 위조하기 어려움
- 인증에 대해서는 소프트웨어를 대상으로 하는 공격의 가능성도 많음
- 해독된 암호화 키나 잊어버린 패스워드는 폐지되고 대체될 수 있는 반면에, 해독된 생체인증 정보를 어떻게 폐지할지에 대해서는 불분명

6.5 당신이 가지고 있는 어떤 것

■ '가지고 있는 어떤 것'

- 스마트폰, 스마트카드, 다른 하드웨어 토큰, 노트북 컴퓨터(맥 주소에 기반을 둔), ATM 카드 등

- 앨리스가 스마트폰을 사용해서 밥에게 자신이 앨리스라는 것을 인증한다고 가정

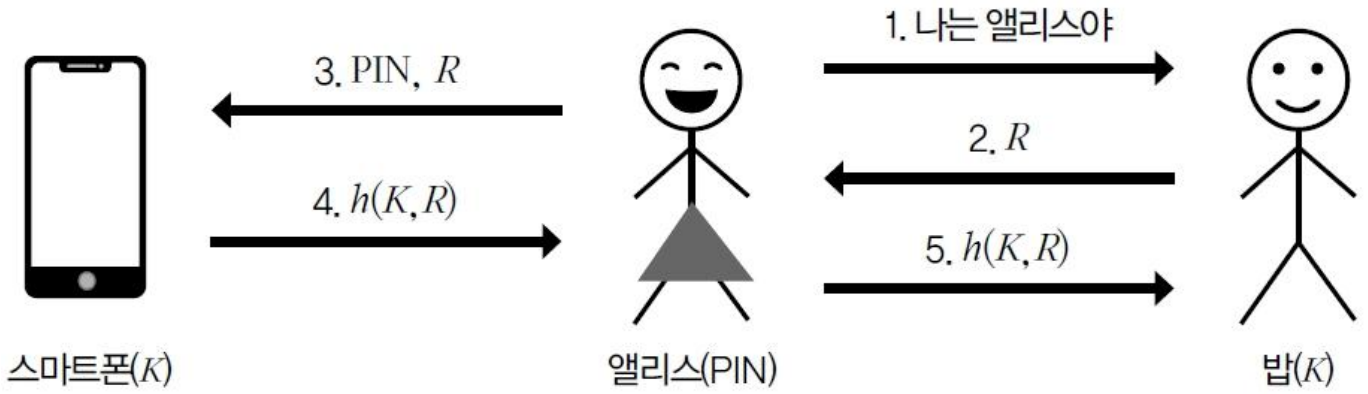


그림 6-4 인증을 위한 스마트폰

6.6 이중 인증

- 스마트폰에 기반을 둔 인증 계획은 '가지고 있는 어떤 것(스마트폰)'과 '알고 있는 어떤 것(PIN)'을 둘 다 요구
- 세 개의 '어떤 것' 중에서 두 개를 요구하는 인증 방법을 이중 인증(two-factor authentication)이라고 함

6.7 통합인증과 웹 쿠키

■ 통합인증(single sign-on)

- 사용자들은 패스워드와 같은 인증 정보를 반복해서 입력하는 것을 힘들어함
- 편리한 해결책 : 초기의 인증에는 앨리스(사용자)가 참여해야 하지만 이후에 따라 나오는 인증은 표면 뒤에서 일어나도록 하는 것 = 통합인증

■ 웹 쿠키(web cookies)

- 앨리스가 웹 서핑을 할 때 웹 사이트는 앨리스의 브라우저에 웹 쿠키를 제공함
- 이러한 웹 쿠키는 앨리스의 브라우저에 의해 저장되고 관리되는 숫자 값임
- 웹 사이트는 앨리스가 웹 쿠키를 가지고 있는지 아닌지에 근거해서 '앨리스'를 인증할 수 있는 것임
- 더 강력한 버전에서는 처음에 앨리스를 인증하기 위해 패스워드를 사용하고 그 후에는 쿠키만으로 충분하다고 여김

6.8 요약

■ 기계에 대한 사람의 인증

- '알고 있는 어떤 것(=패스워드),' '가지고 있는 어떤 것,' 또는 '본인 자체의 어떤 것(=생체인증)'을 기반으로 이루어짐
  - 패스워드 : 인증의 이상적인 방법과는 거리가 멀지만 가장 적은 비용이 든다는 측면에서 앞으로도 계속 인기가 있을 것임
  - 생체인증은 패스워드보다 더 강력한 보안을 제공한다는 것은 명확함
  - 하지만 비용이 많이 들며, 문제점이 완전히 없는 것도 아님
- 이중 인증
- 통합인증과 웹 쿠키의 역할과 개념

6.9 지문 인식 실습

- **비밀번호는 잊거나 도난당할 수 있지만, 지문은 항상 내 손가락에 있음**
- **지문 무늬는 쌍둥이도 다르며 평생 변하지 않음**
- **따라서 “나 자신”이 곧 인증 수단이 됨**

- **생체인증의 세 가지 방식**

인증 방식	의미	예시
내가 **아는 것**	지식 기반	비밀번호, PIN
내가 **가진 것**	소유 기반	카드, OTP
내가 **바로 나인 것**	생체 기반	지문, 얼굴, 홍채

- **지문 인식의 흐름**
  - 지문 사진 촬영 -> 전처리(명암 조정, 노이즈 제거)
    -> 골격화(선만 남김) -> 특징점 추출(길 끝, 갈라짐) -> 비교 /판정

6.9 지문 인식 실습

■ FVC2022

- 국제 생체인식 대회(Fingerprint Verification Competition)의 공개 데이터셋
- 연구·교육용으로 전 세계 대학에서 사용.4개 데이터베이스(DB1~DB4), 각 880장(110명 × 8회).
- 지문 인식 알고리즘 성능을 평가하는 표준.

- FVC2002 - Second International Fingerprint Verification Competition

■ 환경세팅

- opencv-python scikit-image matplotlib numpy

6.9 지문 인식 실습

■ 이미지 확인 해 보기 (지문 이미지 불러와 보기)

```python
import cv2, matplotlib.pyplot as plt

img = cv2.imread('DB1_B/101_1.tif', cv2.IMREAD_GRAYSCALE)
plt.imshow(img, cmap='gray')
plt.title("Original Fingerprint")
plt.axis('off')
plt.show()
```

6.9 지문 인식 실습

■ 이미지 확인 해 보기 (이진화)

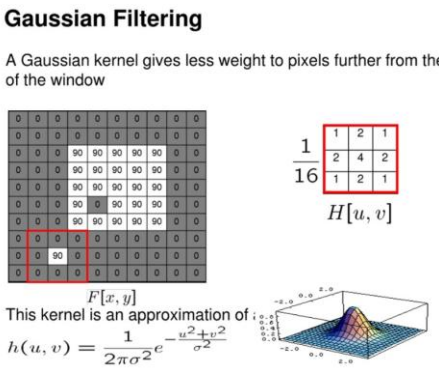
```python
import cv2, matplotlib.pyplot as plt

img = cv2.imread('DB1_B/101_1.tif', cv2.IMREAD_GRAYSCALE)
blur = cv2.GaussianBlur(img, (5,5), 0)
binary = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                               cv2.THRESH_BINARY_INV, 15, 10)
plt.imshow(binary, cmap='gray')
plt.title("Binarized Image")
plt.show()
```

6.9 지문 인식 실습

■ 이미지 확인 해 보기 (이진화)

- cv2.imread(..., cv2.IMREAD_GRAYSCALE)
  - 이미지는 pixel로 이루어지며, 각 픽셀은 0~255사이의 정수 값을 가짐
    - 0 = 검정, 255 = 흰색
  - 색상 정보(RGB)는 제거하고, 명암만 남김 / 지문에서는 색보다 밝기 대비가 중요
- cv2.GaussianBlur(img, (5,5), 0)
  - 블러(Blur) 는 노이즈 제거를 위한 저주파 필터링(Low-pass Filtering) 기법
    - 지문에는 먼지나 조명 반사 등으로 인한 작은 점 형태의 노이즈가 존재
    - 이 노이즈가 이진화 단계에서 잘못된 픽셀로 인식되면 지문선이 끊기거나 불필요한 점이 생김
  - 가우시안 필터는 주변 픽셀의 값을 '정규분포 ' 로 가중 평균하여 중심 픽셀의 새 값을 계산



6.9 지문 인식 실습

■ 이미지 확인 해 보기 (이진화)

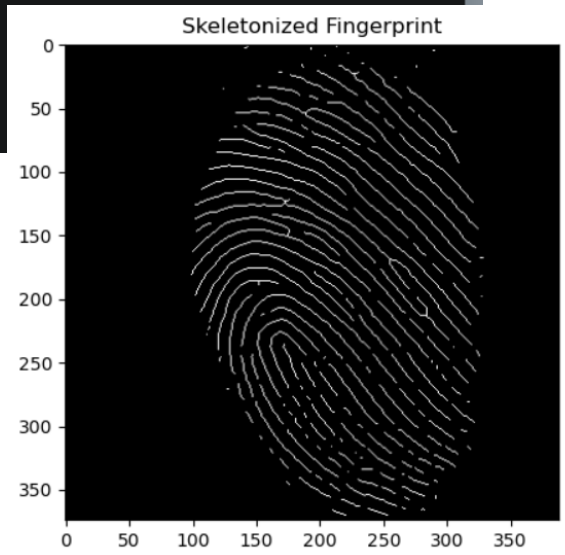
- cv2.adaptiveThreshold(...)
  - 전통적 임계값 이진화의 한계
    - 기본적인 이진화는 단일 임계값 T 사용
    - 방식은 조명이 고르지 않은 이미지에서 실패
    - 왼쪽이 밝고 오른쪽이 어두운 지문에서는한쪽은 전부 흰색, 다른 한쪽은 전부 검정이 되는 현상이 생김
  - 적응형 임계값
    - $T(x, y) = mean\ of\ neighborhood\ around\ (x, y) - C$
    - 셀 주변의 평균 밝기값을 기준으로,해당 지역의 조명 상태에 맞게 동적으로 임계값 설정
- 이진화는 $Non - Linear\ Transformation$ 정보 손실이 있지만 ,$Feature\ Extraction$ 단계에서 이득

6.9 지문 인식 실습

■ 이미지 확인 해 보기 (골격화)

```python
import cv2, matplotlib.pyplot as plt
from skimage.morphology import skeletonize

img = cv2.imread('DB1_B/101_1.tif', cv2.IMREAD_GRAYSCALE)
blur = cv2.GaussianBlur(img, (5,5), 0)
binary = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                               cv2.THRESH_BINARY_INV, 15, 10)
skeleton = skeletonize(binary//255)
plt.imshow(skeleton, cmap='gray')
plt.title("Skeletonized Fingerprint")
plt.show()
```



6.9 지문 인식 실습

■ 이미지 확인 해 보기 (골격화)

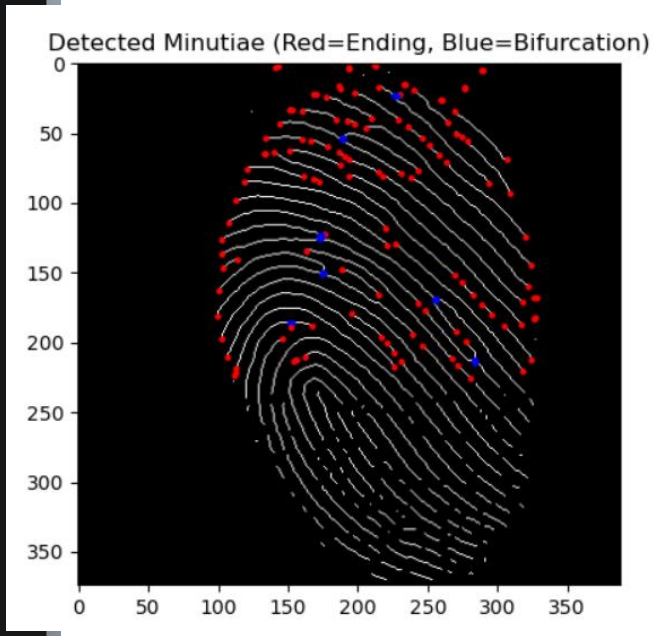
- Skeletonization의 원리
  - 형태학적 연산(Morphological Operation)
  - 지문선의 외곽부터 한겹씩 침식시키되 선이 끊기지 않도록 반복
  - 지문의 topology를 보존한다는 점에서 유의미

6.9 지문 인식 실습

■ 이미지 확인 해 보기 (특징점 추출)

```python
import cv2, matplotlib.pyplot as plt
from skimage.morphology import skeletonize
import matplotlib.pyplot as plt
import numpy as np
def extract_minutiae(skel):
    h, w = skel.shape
    points = []
    for y in range(1,h-1):
        for x in range(1,w-1):
            if skel[y,x]:
                n = np.sum(skel[y-1:y+2, x-1:x+2]) - 1
                if n==1: points.append((x,y,'ending'))
                elif n==3: points.append((x,y,'bifurcation'))
    return points

img = cv2.imread('DB1_B/101_1.tif', cv2.IMREAD_GRAYSCALE)
blur = cv2.GaussianBlur(img, (5,5), 0)
binary = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                               cv2.THRESH_BINARY_INV, 15, 10)
skeleton = skeletonize(binary//255)
features = extract_minutiae(skeleton)
print(f"Detected {len(features)} minutiae points")
plt.imshow(skeleton, cmap='gray')
for x,y,t in features[:150]:
    color = 'r' if t=='ending' else 'b'
    plt.scatter(x,y,c=color,s=5)
plt.title("Detected Minutiae (Red=Ending, Blue=Bifurcation)")
plt.show()
```



6.9 지문 인식 실습

■ 지문비교

```python
import cv2, matplotlib.pyplot as plt
from skimage.morphology import skeletonize
import matplotlib.pyplot as plt
import numpy as np
from scipy.spatial.distance import cdist


def extract_minutiae(skel):
    h, w = skel.shape
    points = []
    for y in range(1,h-1):
        for x in range(1,w-1):
            if skel[y,x]:
                n = np.sum(skel[y-1:y+2, x-1:x+2]) - 1
                if n==1: points.append((x,y,'ending'))
                elif n==3: points.append((x,y,'bifurcation'))
    return points

img = cv2.imread('DB1_B/101_1.tif', cv2.IMREAD_GRAYSCALE)
blur = cv2.GaussianBlur(img, (5,5), 0)
binary = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                               cv2.THRESH_BINARY_INV, 15, 10)
skeleton = skeletonize(binary//255)
features = extract_minutiae(skeleton)
print(f"Detected {len(features)} minutiae points")
plt.imshow(skeleton, cmap='gray')
for x,y,t in features[:150]:
    color = 'r' if t=='ending' else 'b'
    plt.scatter(x,y,c=color,s=5)
plt.title("Detected Minutiae (Red=Ending, Blue=Bifurcation)")
plt.show()

def match_score(f1, f2):
    A = np.array([(x,y) for x,y,_ in f1])
    B = np.array([(x,y) for x,y,_ in f2])
    dist = cdist(A, B)
    return np.mean(np.min(dist, axis=1))

# 같은 사람의 다른 지문
img2 = cv2.imread('DB1_B/101_2.tif', 0)
f2 = extract_minutiae(skeletonize(
     cv2.adaptiveThreshold(cv2.GaussianBlur(img2,(5,5),0),
     255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV,15,10)//255))

print("매칭 점수 (낮을수록 유사):", match_score(features, f2))
```

지표	뜻	결과
FAR (=False Acceptance Rate)	남의 지문을 내 것으로 착각	보안에 취약
FRR (=False Rejection Rate)	내 지문을 못 알아봄	불편함 증가
EER (=Equal Error Rate)	두 확률이 같을 때	정확도의 기준

6.9 지문 인식 실습

■ 보안적으로 안전하게 쓰려면

- 원본 지문 이미지는 저장하지 말 것.
- 특징점 좌표를 암호화(해시) 해서 저장.
- 가짜 손가락 방지(Liveness Detection) 기술 필요.
- 지문 + PIN = 2단계 인증(MFA) 이 이상적.