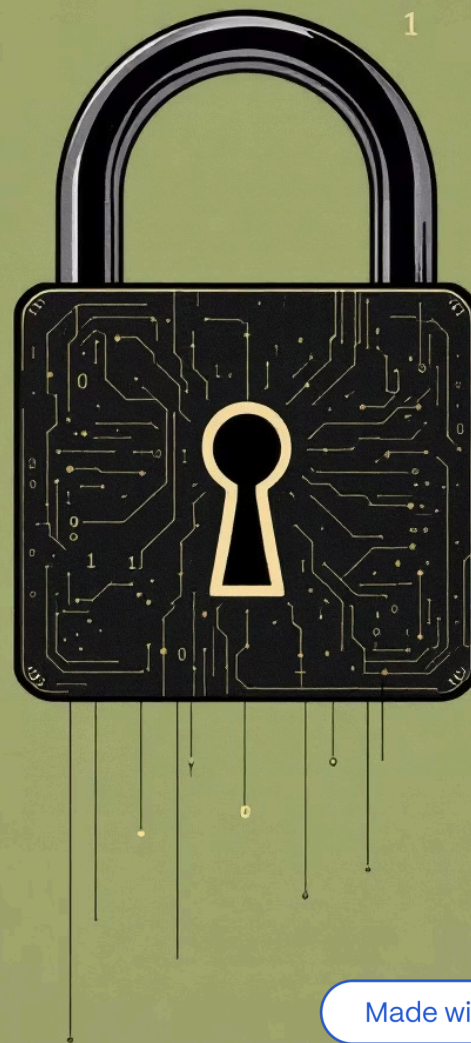


대칭키 vs 공개키(비대칭키)



암호화란 ?

핵심 개념

암호화는 정보를 알아보기 어렵게 바꾸는 기술이며, 허가된 사람만 원래대로 되돌릴 수 있습니다.

마치 비밀 편지를 암호로 쓰는 것과 같다!



1

평문 (Plaintext)

읽을 수 있는 원래 정보

2

암호문 (Ciphertext)

암호화된 알아볼 수 없는 정보

3

복호화 (Decryption)

암호문을 다시 평문으로 되돌림

암호화의 목표: 기밀성

기밀성 (Confidentiality)

허가받지 않은 사람이 정보를 볼 수 없도록 보호

기본 표기법

$C = \text{Enc_key}(M)$

$M = \text{Dec_key}(C)$

- **M (Message / 평문)**: 원래의 메시지, 즉 사람이 읽을 수 있는 평문 데이터.
- **C (Ciphertext / 암호문)**: 암호화된 결과, 즉 아무나 보면 알아볼 수 없는 데이터.
- **Enc_key**: “암호화 함수(Encryption function)” + 그때 쓰는 **키(key)**.
→ 평문 M을 받아서 암호문 C로 변환.
- **Dec_key**: “복호화 함수(Decryption function)” + 그때 쓰는 **키(key)**.
→ 암호문 C를 받아서 원래 메시지 M으로 변환.

암호화는 주로 기밀성을 담당하며, 무결성과 인증은 MAC(메시지 인증 코드)나 디지털 서명과 함께 사용됩니다.



키(Key)의 개념

키란 무엇인가요?

키는 잠금과 해제를 결정하는 비밀 값. 같은 문이라도 열쇠가 다르면 열 수 없는 것과 같다.

- 알고리즘(문을 여닫는 방식) + 키(열쇠) = 보안
- 키 길이가 길수록 무차별 대입 공격에 강함
- 키의 생성·보관·교환이 보안의 절반 이상



대칭키 vs 공개키: 키 사용의 핵심 차이

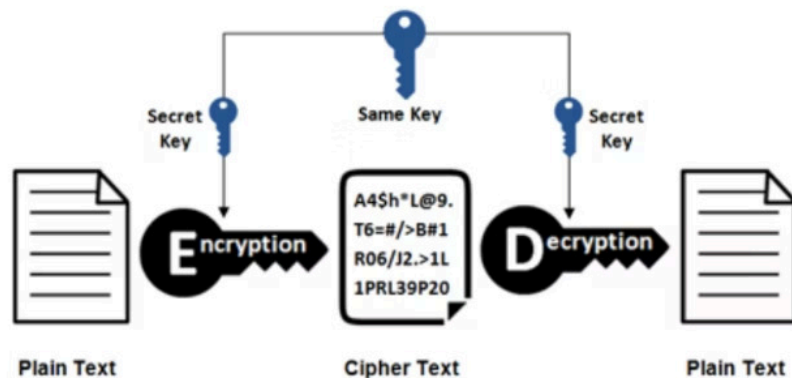


대칭키 암호화 방식

암호화와 복호화에 동일한 하나의 키를 사용.

데이터 전송 속도가 빠르지만, 안전한 키 공유가 중요

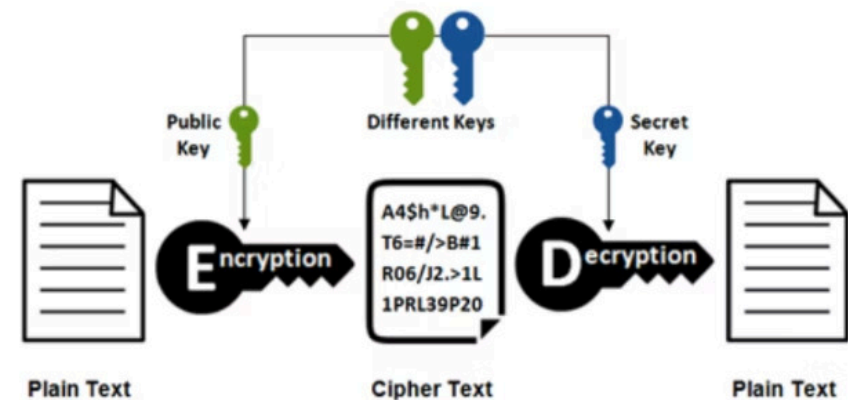
Symmetric Encryption



공개키 암호화 방식

암호화와 복호화에 서로 다른 한 쌍의 키(공개키, 개인키)를 사용 비대칭키 암호화라고도 불리며, 키 공유의 어려움을 해결

Asymmetric Encryption

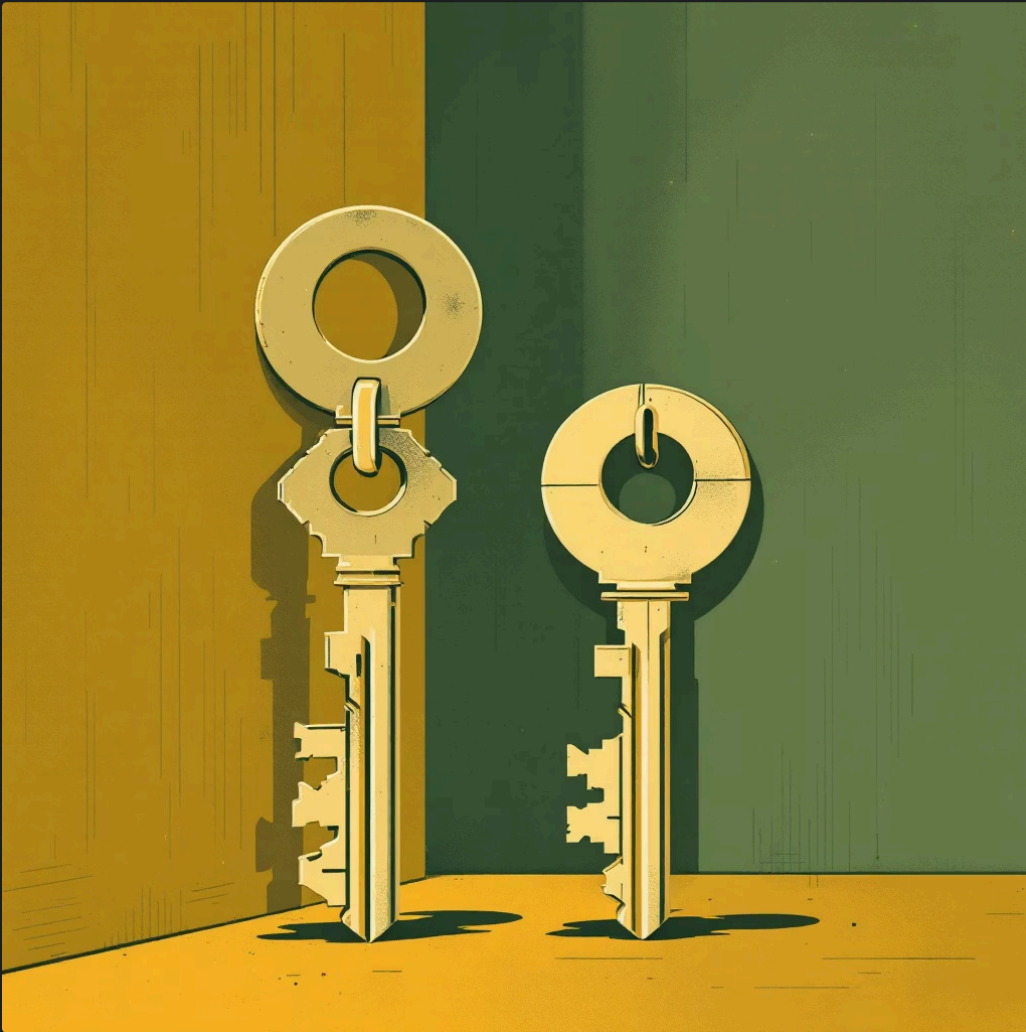


대칭키 vs 공개키(비대칭키) 상세 비교

카테고리	대칭키 (Symmetric Key)	공개키 (Asymmetric Key)
키 구조	하나의 같은 키로 암호화·복호화	키 쌍 (공개키·개인키) 사용
속도	빠르다 (대용량 데이터 적합)	느리다 (큰 수 연산 때문에)
보안성	키가 유출되면 암호 깨짐	공개키는 공개해도 안전, 개인키만 보호
키 배포 문제	키를 상대방과 비밀리에 공유해야 함 → 어려움	공개키는 누구나 배포 가능 → 키 교환 용이

대칭키와 공개키 비유

대칭키



같은 열쇠를 복사해서 서로 나눠 가져야 하는 상황

공개키



누구나 사용할 수 있는 자물쇠(공개키)를 배포하고, 열쇠(개인키)는 오직 나만 보관

대칭키 vs 공개키 동작 과정

대칭키 동작 과정 (양방향 통신)

01

키 공유: A와 B는 서로 같은 대칭키 K 를 미리 안전하게 공유해야 한다.
(직접 만나서 교환하거나, 다른 안전한 방법 필요)

02

A → B 전송

- A는 평문 M 을 키 K 로 암호화 → $C = \text{Enc}_K(M)$
- 암호문 C 를 B에게 보낸다.
- B는 같은 키 K 로 복호화 → $M = \text{Dec}_K(C)$

B → A 도 마찬가지

공개키 동작 과정 (양방향)

01

A와 B는 각자 개인키와 공개키를 가지고 있다.

02

A는 B에게 자신의 공개키를, B도 A에게 자신의 공개키를 전달한다.

03

상대방에게 전달받은 공개키로 암호화해 암호문을 보낸다. 즉 A는 B의 공개키로 암호화하고 B는 A의 공개키로 암호화한다. 이제 암호문을 복호화 할 수 있는건 비밀키를 가진 '상대방' 뿐이다.

04

암호문을 받으면 자신의 비밀키로 복호화한다.

작동 방식과 성능 비교

1

작동 방식

대칭키: 블록/스트림 단위로 데이터를 빠르게 변환

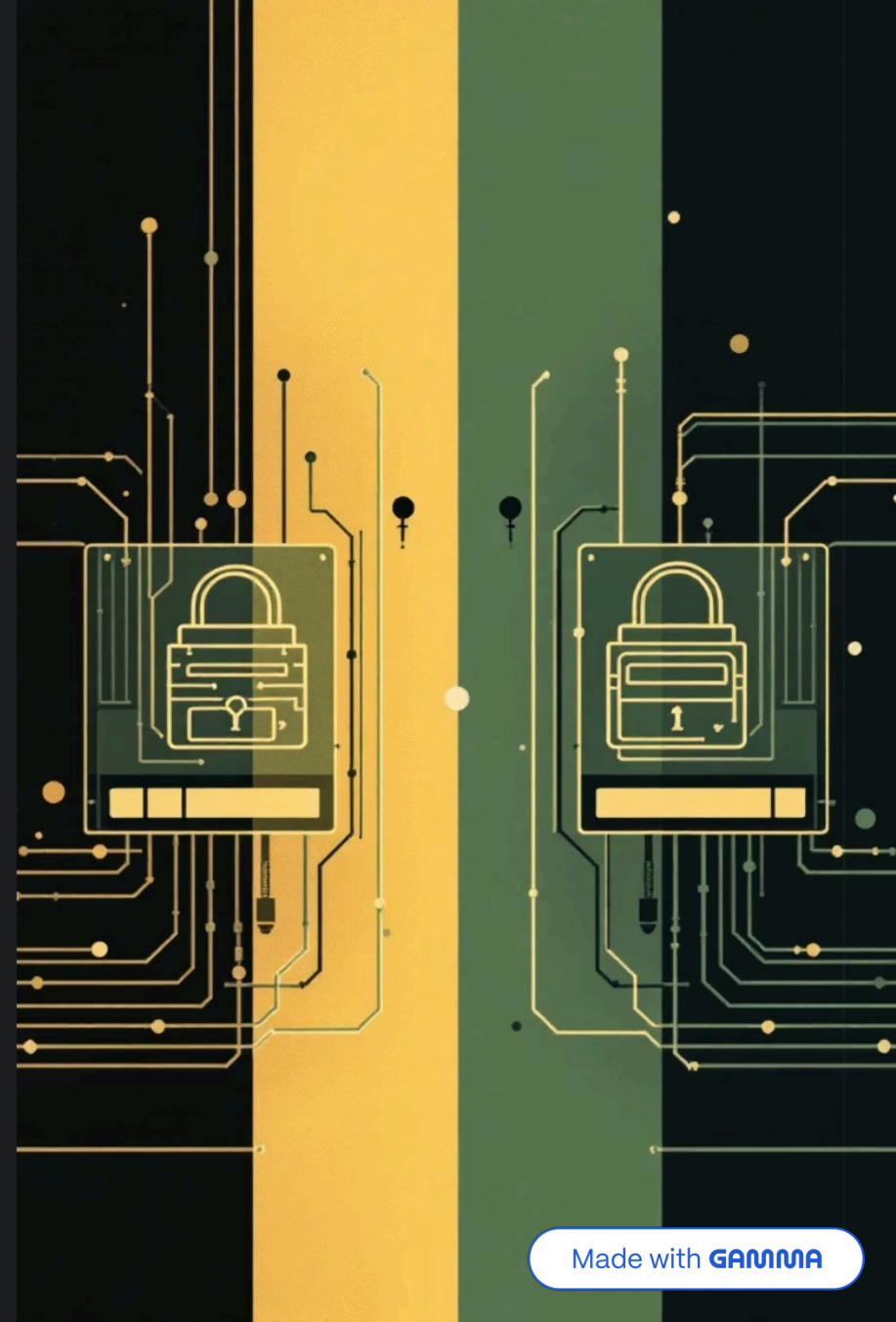
공개키: 수학적 난제(소인수분해, 이산로그)를 이용한 변환

2

성능 차이

대칭키: 매우 빠름, 대용량 데이터에 적합

공개키: 상대적으로 느림, 암호화 길이 제한 존재 ㅅㅅ



암호화의 3가지 보안 목표



기밀성 (Confidentiality)

데이터를 암호화하여 인가된 사용자만이 내용을 볼 수 있도록 하는 것을 목표로 합니다. 중요한 정보가 노출되는 것을 방지하여 프라이버시를 보호합니다.



무결성 (Integrity)

데이터가 전송되거나 저장되는 동안 의도치 않게 변경되거나 손상되지 않았음을 보장합니다. 데이터가 원래 상태 그대로 유지되었음을 확인하여 신뢰성을 확보합니다.



인증 (Authentication)

정보를 주고받는 당사자(사람 또는 시스템)의 신원을 확인하는 목표입니다. 메시지가 정당한 발신자로부터 왔음을 증명하여 사칭을 방지하고 보안 통신을 가능하게 합니다.

대칭키 알고리즘 예시: AES

AES(Advanced Encryption Standard)는 오늘날 가장 널리 사용되는 대칭키 암호화 알고리즘입니다. 평문(원본 데이터)과 암호화 키를 이용하여 암호문을 생성하며, 동일한 키로 복호화가 가능합니다. 128비트 블록 크기를 사용하며, 키 길이에 따라 10, 12, 14 라운드를 반복합니다. 여기서는 128 비트 키를 사용하는 AES-128의 간단한 암호화 과정을 예시로 살펴봅니다.

AES-128 암호화 과정

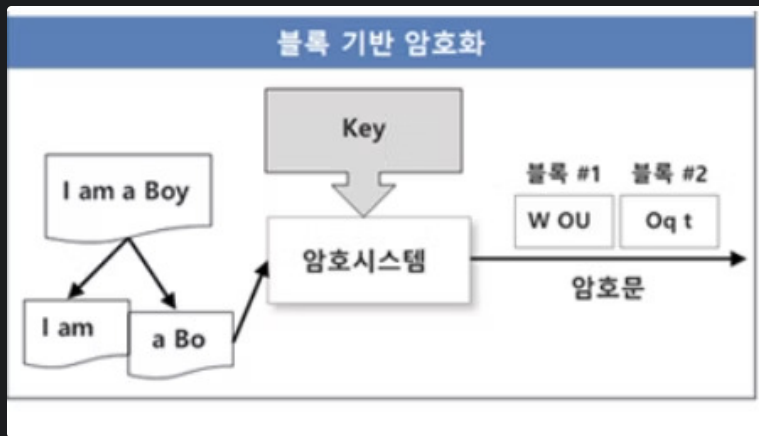
다음은 128비트 평문 한 블록이 128비트 키를 사용하여 암호화되는 과정입니다:

<div>📄</div> <div><h3>평문 (Plaintext) 입력</h3><p>암호화할 원본 데이터 블록. 예: <code>M = "ABCD..."</code> (16바이트 = 128비트)</p></div>	<div>🔑</div> <div><h3>키 (Key) 입력</h3><p>암호화에 사용될 비밀 키. 예: <code>K = 2b7e151628aed2a6abf7158809cf4f3c</code> (16바이트 = 128비트 난수)</p></div>
<div>🔄</div> <div><h3>초기 AddRoundKey</h3><p>평문 <code>M</code>과 키 <code>K</code>를 XOR (\oplus) 연산하여 첫 번째 변형 시작. 결과는 <code>state = M \oplus K</code> 입니다.</p></div>	<div>🔄</div> <div><h3>라운드 반복 (총 9회)</h3><ul style="list-style-type: none">SubBytes: 각 바이트를 S-Box 테이블을 사용하여 다른 값으로 치환하여 난수성을 높입니다. (예: <code>0x19</code> \rightarrow <code>0xd4</code>)ShiftRows: 바이트 행을 좌측으로 이동시켜 데이터의 확산(diffusion)을 증대시킵니다.MixColumns: 열 단위로 선형 변환을 적용하여 바이트 간의 복잡한 관계를 만들고 패턴을 제거합니다.AddRoundKey: 이번 라운드에 해당하는 라운드 키와 현재 상태를 다시 XOR 연산합니다.<p>이 과정을 9라운드 동안 반복합니다.</p></div>
<div>🏁</div> <div><h3>마지막 라운드 (10번째)</h3><p>마지막 라운드에서는 MixColumns 단계를 제외한 세 가지 변환만 수행합니다.</p><ul style="list-style-type: none">SubBytesShiftRowsAddRoundKey</div>	<div>📄</div> <div><h3>암호문 (Ciphertext) 생성</h3><p>모든 라운드를 거쳐 최종 암호문이 생성됩니다. 예: <code>C = 3925841d02dc09fbdc118597196a0b32</code></p></div>

직관적인 비유

암호화 과정을 요리하는 것에 비유할 수 있습니다.

- 평문은 "깨끗한 물컵"과 같습니다.
- 키는 요리의 맛을 완전히 바꿀 "비밀 조미료"입니다.
- AES 알고리즘은 "10번의 레시피 단계(치환·섞기·비비기)"를 거쳐 물컵의 내용물을 변형시킵니다.
- 결과는 "전혀 원래 맛을 알 수 없는 음료(암호문)"가 됩니다.
- 이 비밀 조미료를 이용해 같은 단계를 거꾸로 따라가면, 원래의 깨끗한 물맛(평문)이 다시 나옵니다.



- **DES (Data Encryption Standard)**

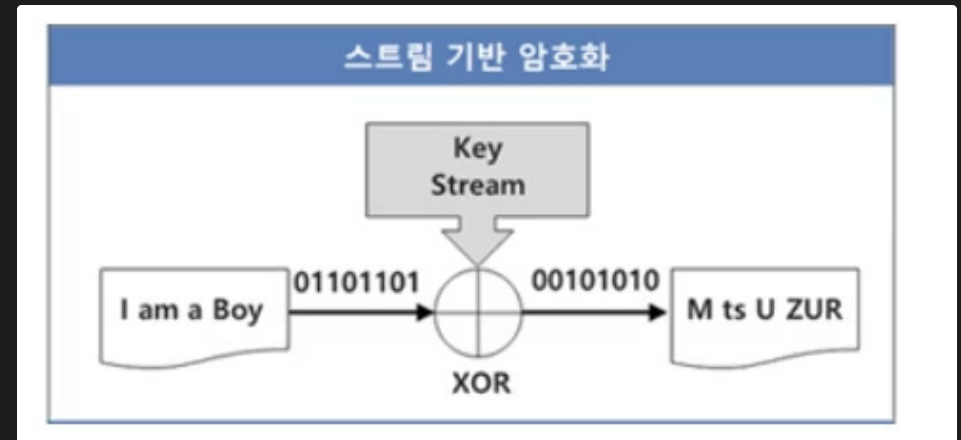
64비트 블록, 56비트 키 사용

1970~90년대 표준이었지만 지금은 키 길이가 너무 짧아서 더 이상 안전하지 않음

- **3DES (Triple DES)**

DES를 3번 반복해 보안 강화

하지만 느리고, 현재는 더 이상 권장되지 않음



- **RC4**

오래된 스트림 암호, SSL/TLS에도 사용된 적 있음

현재는 여러 취약점 때문에 금지

- **Salsa20**

빠르고 안전한 현대적 스트림 암호

- **ChaCha20**

Salsa20 개선판, 구글과 IETF에서 채택

TLS(HTTPS)에서 AES 대신 사용되기도 함

ChaCha20-Poly1305 조합은 **AEAD**(기밀성+무결성 동시 제공)

공개키 알고리즘 예시: RSA

RSA (Rivest-Shamir-Adleman)는 현대 암호 시스템에서 가장 널리 사용되는 공개키 암호화 알고리즘 중 하나입니다. 공개키와 개인키 한 쌍을 사용하여 데이터를 암호화하고 복호화하며, 전자서명에도 활용됩니다. RSA의 보안성은 큰 숫자를 소인수 분해하는 것이 어렵다는 수학적 난제에 기반합니다.

1) 키 생성 과정

- 아주 큰 소수 두 개 p, q 를 뽑는다.
- $n = p \times q$ 계산 → 공개키/개인키 공통의 모듈러 값.
- 오일러 함수 $\phi(n) = (p - 1)(q - 1)$ 계산.
- 암호화 지수 e 를 선택 (보통 65537 사용). 이때, $1 < e < \phi(n)$ 이고 e 와 $\phi(n)$ 은 서로소여야 한다.
- 복호화 지수 d 를 계산: $d \equiv e^{-1} \pmod{\phi(n)} \rightarrow$ 즉, $e \times d \equiv 1 \pmod{\phi(n)}$ 성립.
- 공개키: (n, e)
- 개인키: (n, d)

2) 암호화 과정

- 평문 M (숫자로 표현된 메시지)을 공개키 (n, e) 로 암호화:
- $C = M^e \pmod{n}$
- 누구나 공개키를 가지고 이 연산 가능 → 암호문 C 생성.

3) 복호화 과정

- 암호문 C 를 개인키 (n, d) 로 복호화:
- $M = C^d \pmod{n}$
- 오직 개인키 소유자만 이 연산이 가능.

수학적 기반에 따른 분류

- **RSA 계열** → 큰 정수 소인수분해 문제 기반
- **ECC 계열(Elliptic Curve)** → 타원곡선 위의 이산로그 문제 기반 (예: ECDH, ECDSA)
- **ElGamal 계열** → 이산로그 문제 기반

용도에 따른 분류

- **암호화(Encryption/KEM)**: RSA-OAEP, ECIES 등
- **키 교환(Key Exchange)**: Diffie-Hellman, ECDH
- **전자서명(Signature)**: RSA-PSS, ECDSA, Ed25519


무결성 (Integrity) 보장 방법

1) 개념


- 무결성 = "메시지가 전송 중에 변조되지 않았음을 보장"
- 받는 쪽에서 확인했을 때 원래 발신한 그대로인지 검증 가능해야 한다.

2) 주요 방법

(1) MAC (Message Authentication Code)

- 대칭키 기반 검증 방식.
- 송신자: $\text{Tag} = \text{MAC}_K(M)$ 계산 후 메시지와 함께 전송.
- 수신자: 같은 키 K 로 $\text{MAC}_K(M)$ 재계산 → 비교.
- 다르면 변조됨 판정.
-  예: HMAC-SHA256

(2) AEAD (Authenticated Encryption with Associated Data)

- 암호화 + 무결성 검증을 동시에 제공하는 현대적 방식.
- 송신자: Ciphertext + Tag 생성.
- 수신자: 복호화 시 태그 검증 → 불일치하면 즉시 거부.
-  예: AES-GCM, ChaCha20-Poly1305
- 오늘날 TLS(HTTPS), VPN, 메신저에서 표준적으로 사용.

(3) 해시 기반 체크 (단독으로 불충분)

- 메시지 해시값(SHA-256 등)을 같이 보내는 방법.
- 단, 키가 없으면 공격자가 해시도 바꿔치기 가능 → 단독 사용은 안전하지 않음.
- 그래서 보통 **MAC(HMAC)**처럼 "해시 + 키" 조합을 사용.

3) 비유

편지에 봉인 테이프를 붙여서, 도착했을 때 찢어지지 않았는지 확인하는 것. 중간에 누가 몰래 열고 다시 붙였다면 바로 표시가 난다.

인증 (Authentication) 보장 방법

1) 개념


- 인증 = "이 메시지를 누가 보냈는지 확인하는 것"
- 무결성이 단순히 "안 바뀌었다"라면, 인증은 "이게 진짜 보낸 사람이 맞다"까지 확인.
- 경우에 따라 부인방지(Non-repudiation)도 제공 → 발신자가 "난 안 보냈다"라고 주장할 수 없게 함.

2) 주요 방법

(1) 대칭키 기반 인증 (MAC 이용)

- 송신자와 수신자가 같은 키 K 를 공유.
- 메시지 + MAC 태그가 맞으면 "키를 가진 사람"이 보냈다는 걸 알 수 있음.
- 하지만 키를 둘 다 가지고 있으므로, 누가 보냈는지 구체적으로 특정은 불가.
- → "상호 인증"에는 쓸 수 있지만, "부인방지"는 불가능.

(2) 비대칭키 기반 인증 (디지털 서명)

- 발신자: 메시지 해시값을 개인키(SK)로 서명.
- 수신자: 공개키(PK)로 서명 검증.
- 검증 성공 = 이 메시지는 해당 개인키 소유자가 보낸 것.
-  대표 알고리즘: RSA-PSS, ECDSA, Ed25519
- 장점: 발신자 특정 + 부인방지까지 제공.
- HTTPS, 전자계약, 블록체인 서명 등에서 핵심.

3) 비유

- MAC 기반: 두 사람이 같은 도장을 공유하고 찍는 것. → 도장이 맞으면 둘 중 누군가 찍은 건 알 수 있지만, 정확히 누군지는 특정 못 함.
- 디지털 서명: 발신자가 자기 고유 도장을 찍는 것. → 수신자는 그 도장이 해당 사람만의 것임을 확인 가능. → 발신자는 나중에 "내가 안 찍었다"라고 부정할 수 없음.

실제 사용처와 활용 분야



대칭키 사용처

- 저장 데이터 암호화
- VPN과 디스크 암호화
- 대량 데이터 전송



공개키 사용처

- 키 교환과 세션 설정
- 디지털 서명과 인증서
- HTTPS 초기 핸드셰이크

❏ **실전 패턴:** 공개키로 세션키를 안전하게 교환한 후, 대칭키로 본문을 빠르게 암호화하는 하이브리드 방식을 주로 사용합니다.

결론: 하이브리드가 정답!

- 1 — 공개키로 신뢰 수립
안전한 채널 구성
- 2 — 세션키 교환
일회성 대칭키 전달
- 3 — 대칭키로 데이터 처리
빠르고 효율적인 암호화

두 방식의 장점을 결합하여 보안성과 성능을 모두 확보하는 것이 현대 암호화의 핵심입니다. HTTPS, VPN 등 우리가 매일 사용하는 기술들이 모두 이런 하이브리드 방식을 사용하고 있어요!

HYBRID ENCRYPTION

