
보안프로그래밍

전자서명 (Digital Signature)

숭실대학교 소프트웨어학부 조효진

Contents

- 전자 서명 (Digital Signature)
- MAC vs. 전자 서명
- RSA 전자 서명
- 공개키 암호 vs. 전자 서명
- 공개키 기반 구조 (Public Key Infrastructure)

전자 서명 (Digital Signature)

□ 종이서명 vs. 전자서명

	종이 서명	전자 서명
작성 형태	문서 내에 서명이 포함 (e.g., 종이문서내에 서명)	문서와 서명이 분리 (e.g., 문서 파일, 서명 파일)
검증 방법	문서내 서명과 기존 서명 기록 대조, 비교	별도의 검증기술을 적용 (검증알고리즘 필요)
서명과 문서의 관계	One-to-Many	One-to-One
서명 검증	검증기관 필요	공개 검증 (with 공인인증서)

전자 서명 (Digital Signature): 용어 정의

□ Public key crypto system

- Public key encryption algorithm (공개키 암호)
 - Public key-based encryption algorithm
- Digital signature (전자서명)
 - (public key-based) digital signature algorithm
- 공개키(public key)와 개인키(private key)
 - 공개키: pk (or K_{pub} or Pub_k), 개인키: sk (or K_{priv} or $Priv_k$)
 - 공개키 암호 알고리즘: 암호화키 pk , 복호화키 sk
 - 전자서명 알고리즘: 검증키 pk , 서명키 sk

□ 비밀키 (Secret key) vs. 개인키 (Private key)

전자 서명 (Digital Signature)

□ 전자 서명 과정

- 공개키 암호알고리즘과 유사
 - 키생성 과정을 통해 검증키 pk , 서명키 sk 생성함
 - $Sig_{sk}(\cdot)$ = 서명 생성 알고리즘, $Ver_{pk}(\cdot)$ = 검증 알고리즘
1. Alice \rightarrow Bob : 메시지 m 에 대한 서명 $s = Sig_{sk}(m)$
 2. Bob : $Ver_{pk}(m, s)$ 로 검증



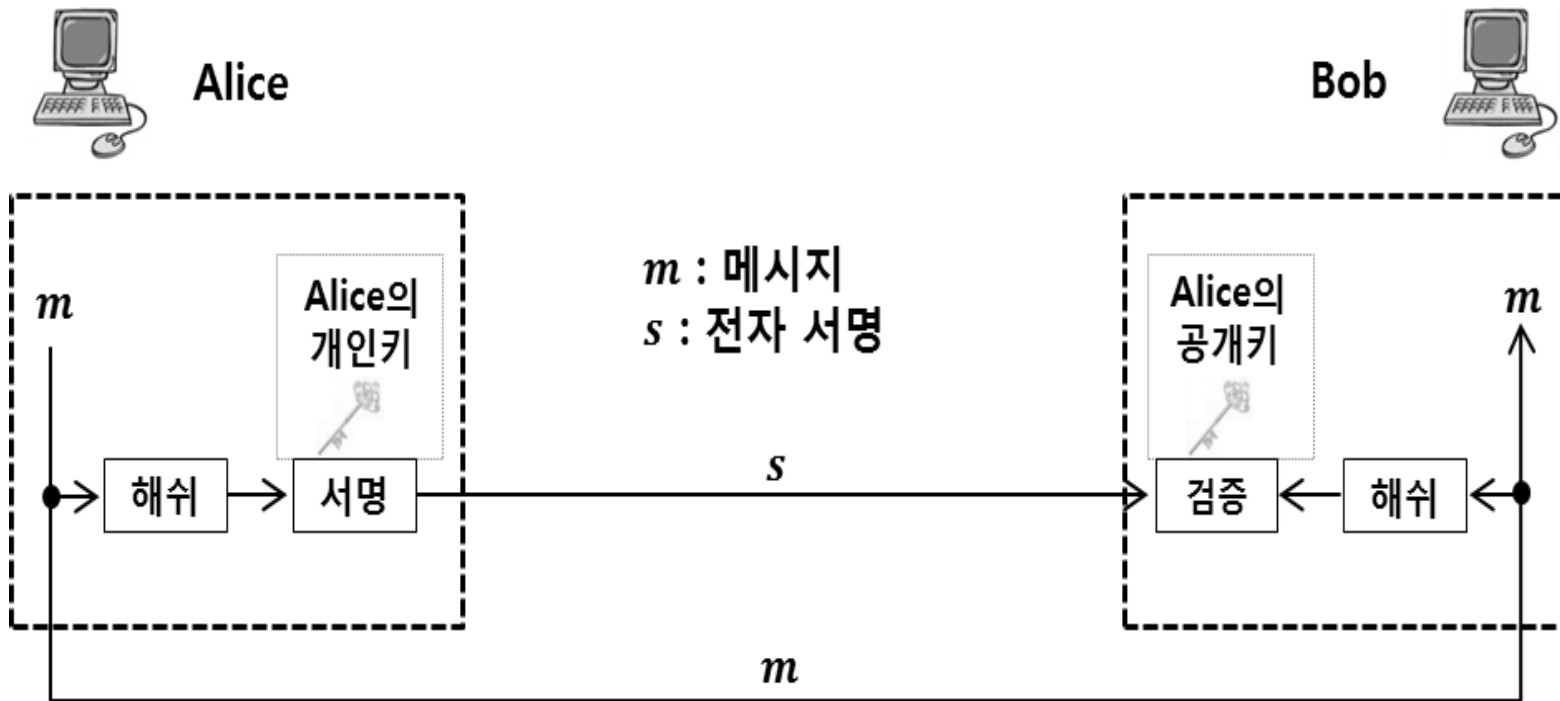
전자 서명 (Digital Signature)

□ 암호학적 해쉬함수 후 서명: $s = \text{Sig}_{sk}(h(m))$

- 파일 $F = f_1 || f_2$,
 - $f_1, f_2 < \text{공개키 크기 (e.g., RSA } n = 2048 \text{ bits)}$
- (효율성) 큰 파일을 서명할 때 효율적임
 - 암호학적 해쉬 함수를 쓰지 않는 경우: $s_1 = \text{Sig}_{sk}(f_1), s_2 = \text{Sig}_{sk}(f_2)$
 - 암호학적 해쉬 함수를 쓰는 경우: $s_1 = \text{Sig}_{sk}(h(F)) = \text{Sig}_{sk}(h(f_1 || f_2))$
- (보안성) 서명의 순서 변경이나 삭제를 방지
 - 암호학적 해쉬 함수를 쓰지 않는 경우
 - 순서 변경: $s_1 = \text{Sig}_{sk}(f_1), s_2 = \text{Sig}_{sk}(f_2)$ vs. $s_1 = \text{Sig}_{sk}(f_2), s_2 = \text{Sig}_{sk}(f_1)$
 - 삭제: $s_1 = \text{Sig}_{sk}(f_1), s_2 = \text{Sig}_{sk}(f_2)$ vs. $s_1 = \text{Sig}_{sk}(f_1)$
 - 암호학적 해쉬 함수를 쓰는 경우
 - 순서 변경: $s_1 = \text{Sig}_{sk}(h(f_1 || f_2))$ vs. $s_1 = \text{Sig}_{sk}(h(f_2 || f_1))$
 - 삭제: $s_1 = \text{Sig}_{sk}(h(f_1 || f_2))$ vs. $s_1 = \text{Sig}_{sk}(h(f_1))$
- 그 밖에 암호학적 해쉬 함수 사용은 전자서명에 대한 위조 공격을 막을 수 있음

전자 서명 (Digital Signature)

□ 해쉬 후 서명: $s = \text{Sig}_{sk}(h(m))$



전자 서명 (Digital Signature)

□ 전자 서명이 제공하는 보안 서비스

- 무결성(message integrity)

- $Sig_{sk}(h(m)) \neq Sig_{sk}(h(m'))$ if $m \neq m'$ { h 는 충돌저항성}

- 메시지 공개 검증(message verification)

- 정당한 sk 를 이용한 서명만이 $Ver_{pk}(m, Sig_{sk}(h(m)))$ 을 통과 (누구나 검증 가능)

- 메시지 인증(message authentication)

- $Sig_{sk}(h(m))$ 에 연관된 메시지 m 이 sk 의 소지자로부터 생성되었다는 것을 인증할 수 있음

- 부인방지(non-repudiation)

- $Sig_{sk}(h(m))$ 을 생성할 수 있는 사람은 sk 의 소지자

MAC vs. 전자 서명

□ MAC과 전자서명은 모두 무결성과 메시지 인증 제공

□ MAC와 전자서명의 차이점

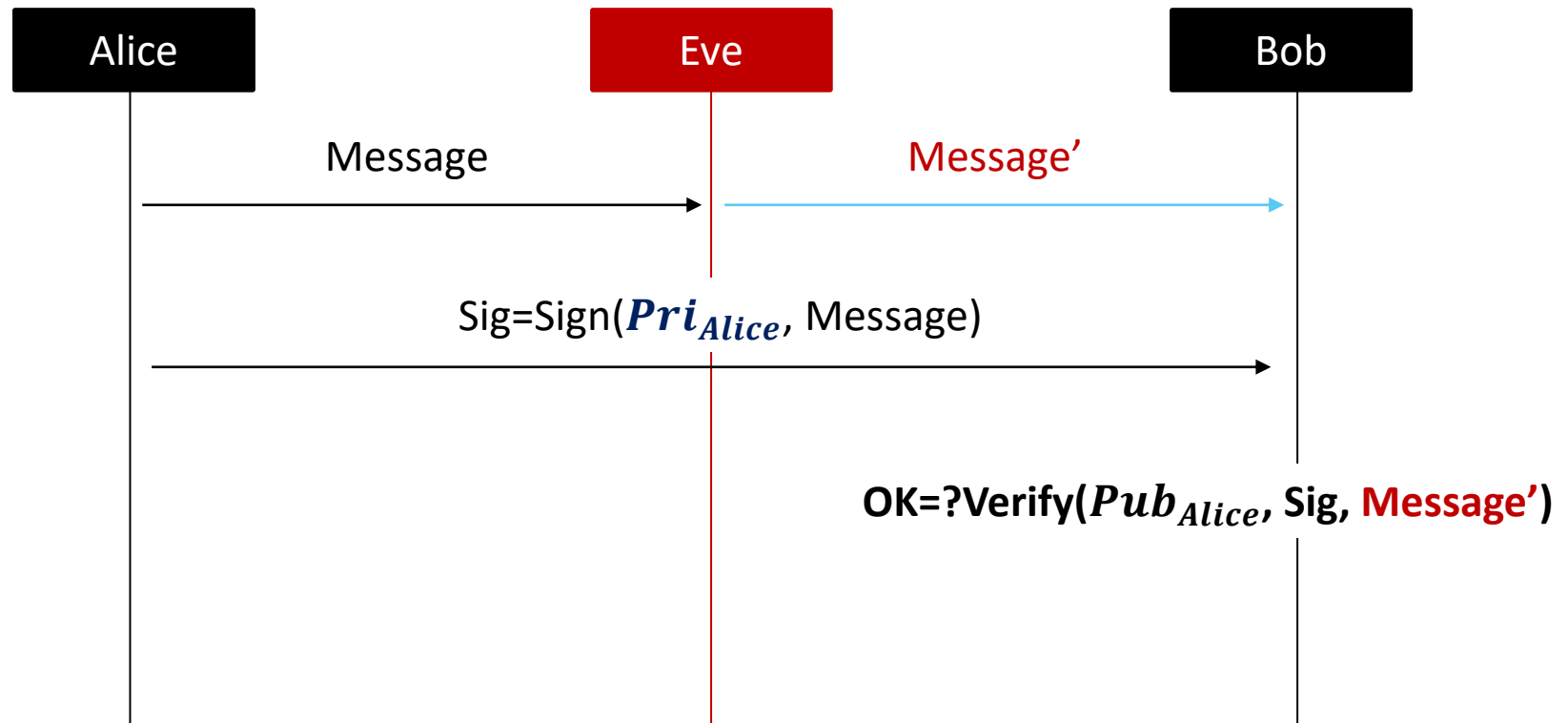
- 공개 검증(public verifiability)
 - pk 는 공개된 정보
- 키 관리
 - MAC의 경우, MAC key를 공유해야 함
- 부인방지(non-repudiation)
 - $Sig_{sk}(h(m))$ 을 생성할 수 있는 사람은 sk 의 소지자
- MAC은 전자서명보다 2~3배 효율적

MAC vs. 전자 서명

□ Digital signature

Alice's public key: Pub_{Alice}

Alice's private key: Pri_{Alice}

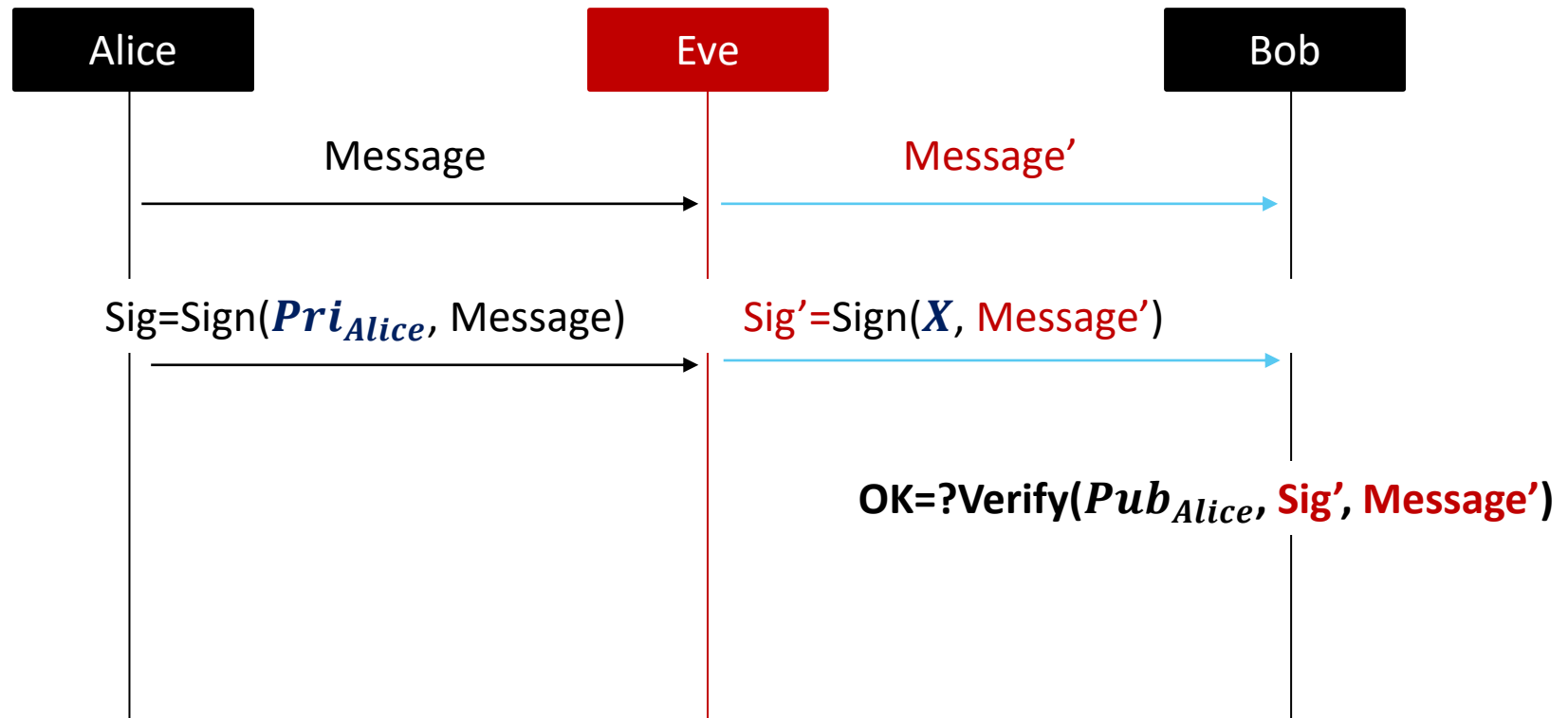


MAC vs. 전자 서명

□ Digital signature

Alice's public key: Pub_{Alice}

Alice's private key: Pri_{Alice}

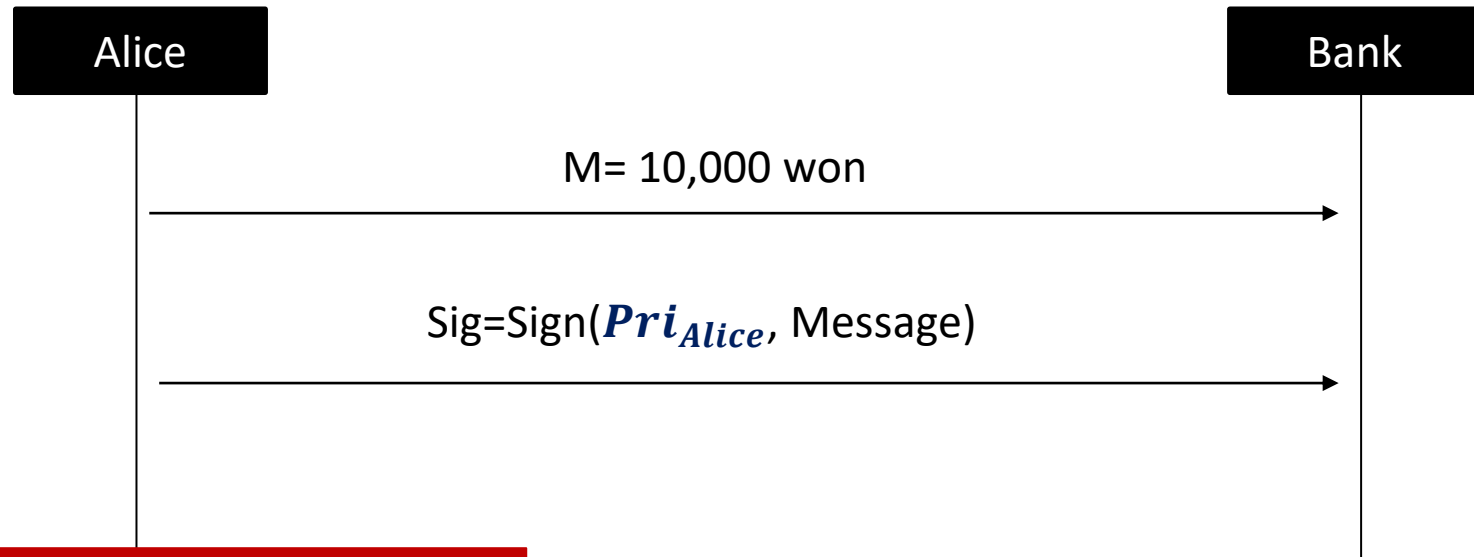


MAC vs. 전자 서명

□ Digital signature provides non-repudiation

- Because a signature can be generated by only one entity who has the corresponding private key

Alice's private key: Pri_{Alice}



I deposited 1,000,000 won.

Sig is modified by Bank

What?
I don't know your private key

RSA 전자 서명

□ 전자서명 알고리즘

- RSA 전자 서명
- ElGamal 전자 서명
- Schnorr 전자 서명
- DSA (Digital Signature Algorithm, 미국 표준)
- ECDSA 전자 서명
 - DSA 전자 서명을 타원곡선에 적용함

RSA 전자 서명

□ Textbook RSA 전자 서명

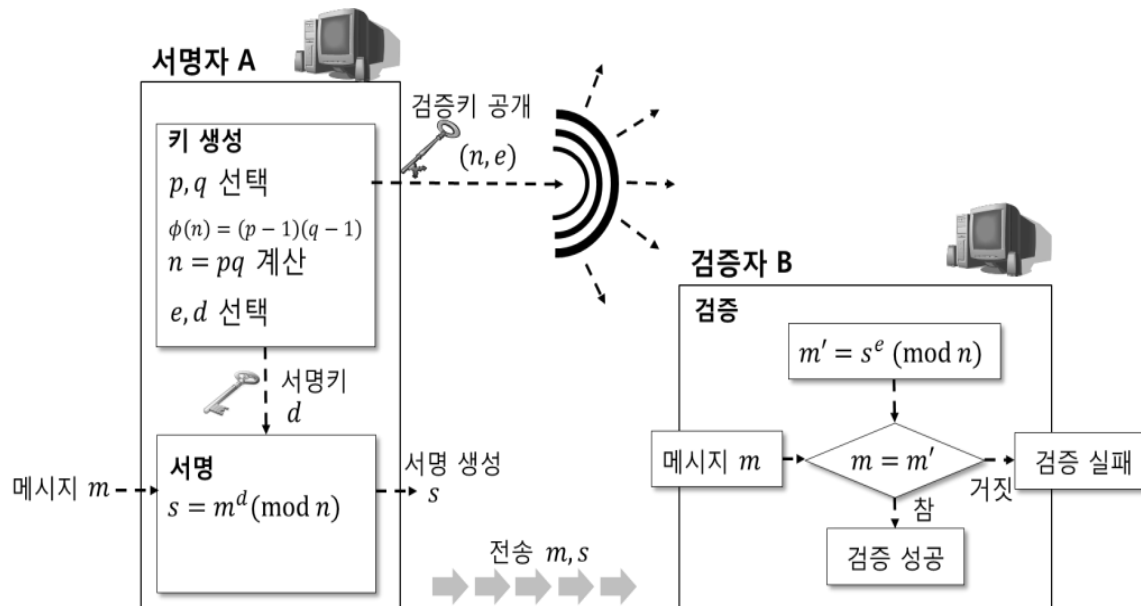
- 키 생성 : RSA 암호와 동일

- 서명 생성 :

- $Sig_{sk}(m) \equiv m^d \equiv s \pmod{n}$

- 서명 검증 :

- $Ver_{pk}(m, s, e) \equiv s^e \pmod{n} \equiv m^{ed} \pmod{n} \equiv m', \quad m' =? m$



RSA 전자 서명

□ Textbook RSA 전자 서명의 안전성

- 알려진 메시지 공격자에 의한 존재적 위조

- 알려진 서명 $s_1 \equiv m_1^d \pmod{n}$ & $s_2 \equiv m_2^d \pmod{n}$ 을 이용해 새로운 서명 생성

- $\rightarrow s_1 s_2 \equiv m_1^d \times m_2^d \equiv (m_1 \times m_2)^d \pmod{n}$









- \rightarrow 즉, $m^{\text{new}} = m_1 \times m_2$ 에 대한 새로운 서명 $s_1 s_2$ 생성 가능함

RSA 전자 서명

□ Textbook RSA 전자서명에 대한 알려진 공격들 때문에, RSA 전자 서명은 안전한 암호학적 해쉬 함수와 같이 사용됨

- $Sig_{sk}(m) \equiv h(m)^d \equiv s \pmod{n}$
- 알려진 메시지 공격자에 의한 존재적 위조
 - 알려진 서명 $s_1 \equiv h(m_1)^d \pmod{n}$ & $s_2 \equiv h(m_2)^d \pmod{n}$
 - $\rightarrow s_1 \times s_2 \equiv h(m_1)^d \times h(m_2)^d \equiv (h(m_1) \times h(m_2))^d$
 $\neq h(m_1 \times m_2)^d \pmod{n}$
 - $\rightarrow (h(m_1) \times h(m_2)) = h(m_3)$ 인 메시지 m_3 을 발견해야 함
 - $\rightarrow h(\cdot)$ 의 역상 저항성(preimage resistance) 때문에 불가능

공개키 암호 vs. 전자 서명

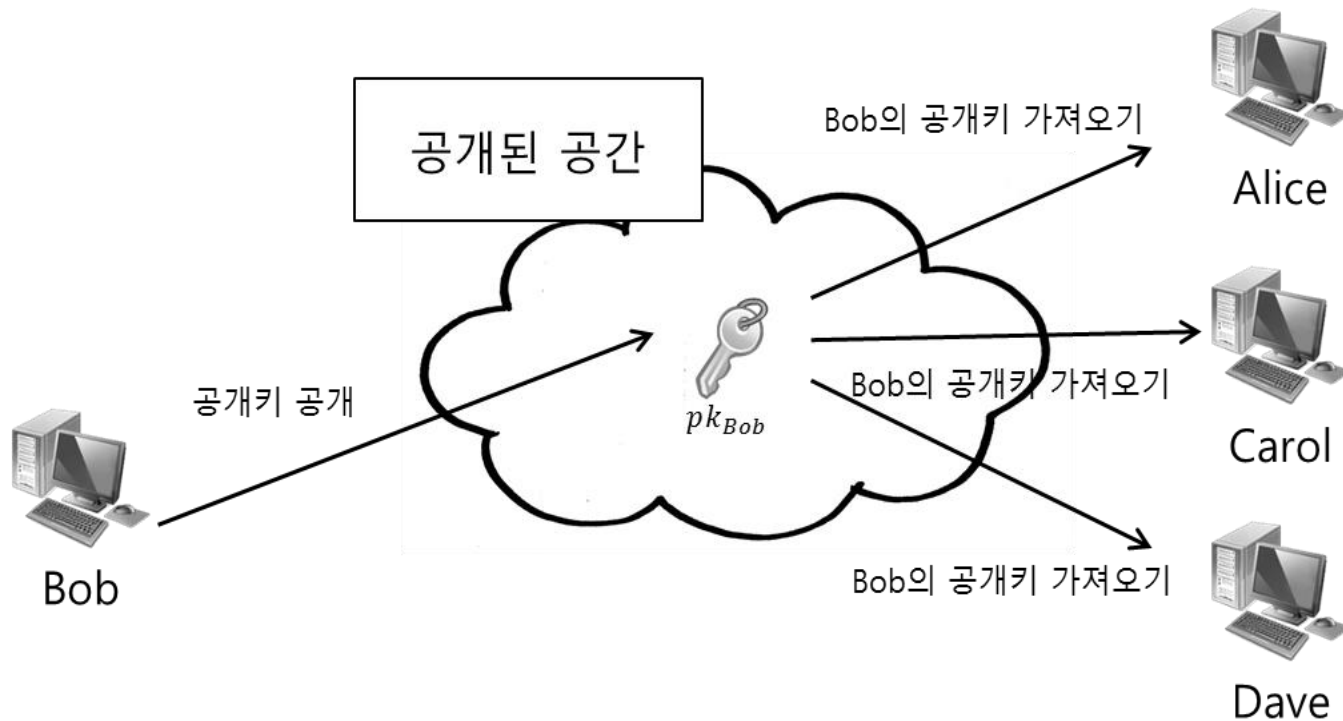
	Sender has 	Recipient has 
 Signing	 Sender private key	 Sender public key
 Encrypting	 Recipient public key	 Recipient private key

[https://docs.microsoft.com/en-us/previous-versions/tn-archive/aa998077\(v=exchg.65\)](https://docs.microsoft.com/en-us/previous-versions/tn-archive/aa998077(v=exchg.65))

공개키 기반 구조 (Public Key Infrastructure)

□ 공개키 암호시스템을 이용한 키 교환

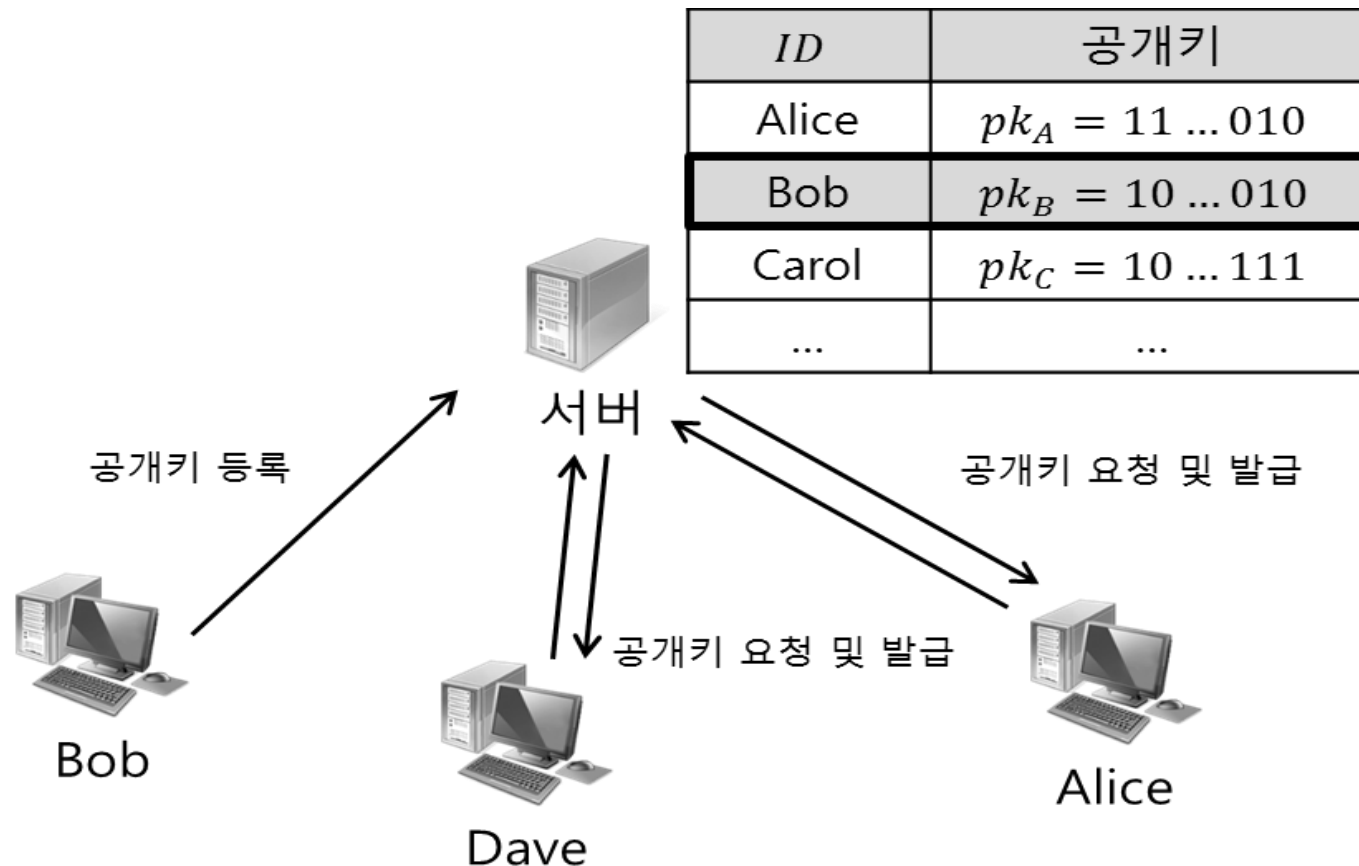
- 공개된 공간을 활용한 공개키 공개 선언 → 신뢰성 문제



공개키 기반 구조 (Public Key Infrastructure)

□ 공개키 암호시스템을 이용한 키 교환

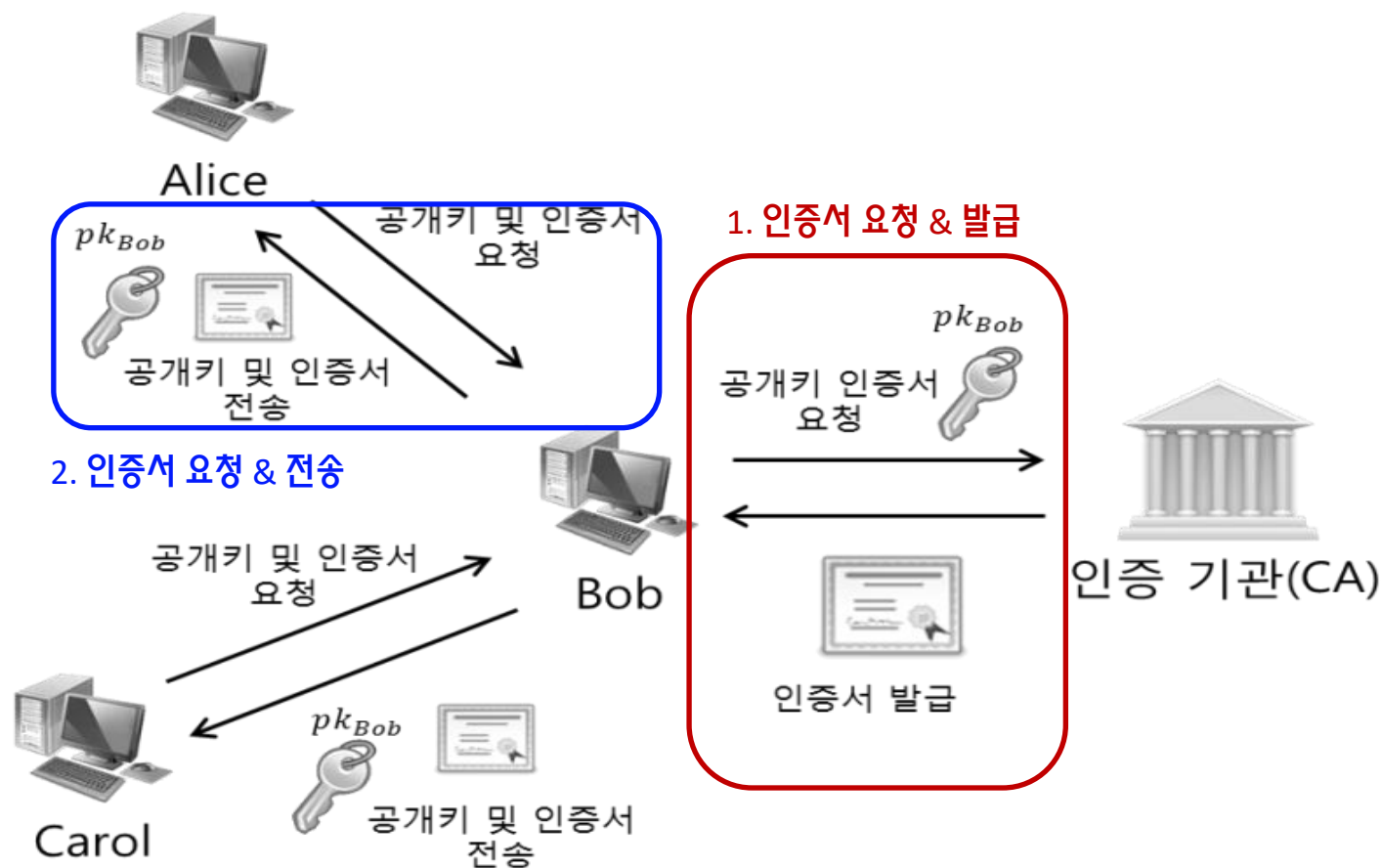
- 신뢰할 수 있는 서버 이용 → 서버에 과부하



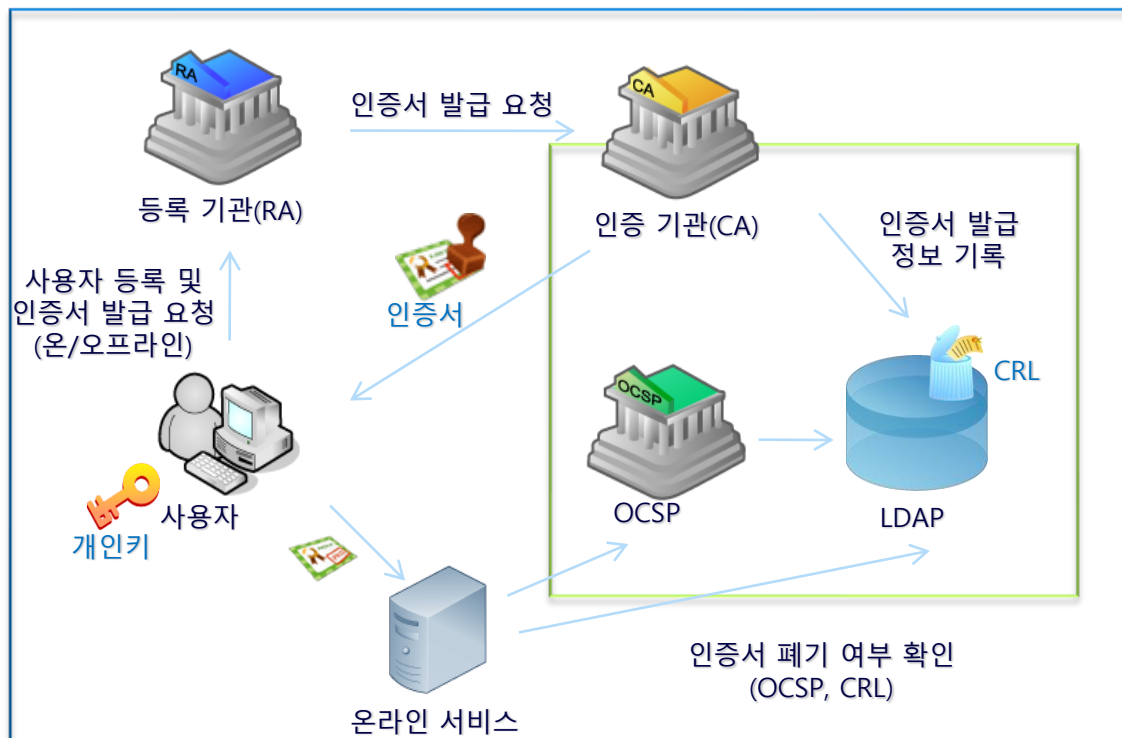
공개키 기반 구조 (Public Key Infrastructure)

□ 공개키 암호시스템을 이용한 키 교환

- 인증서를 이용한 공개키 인증



공개키 기반 구조 (Public Key Infrastructure)

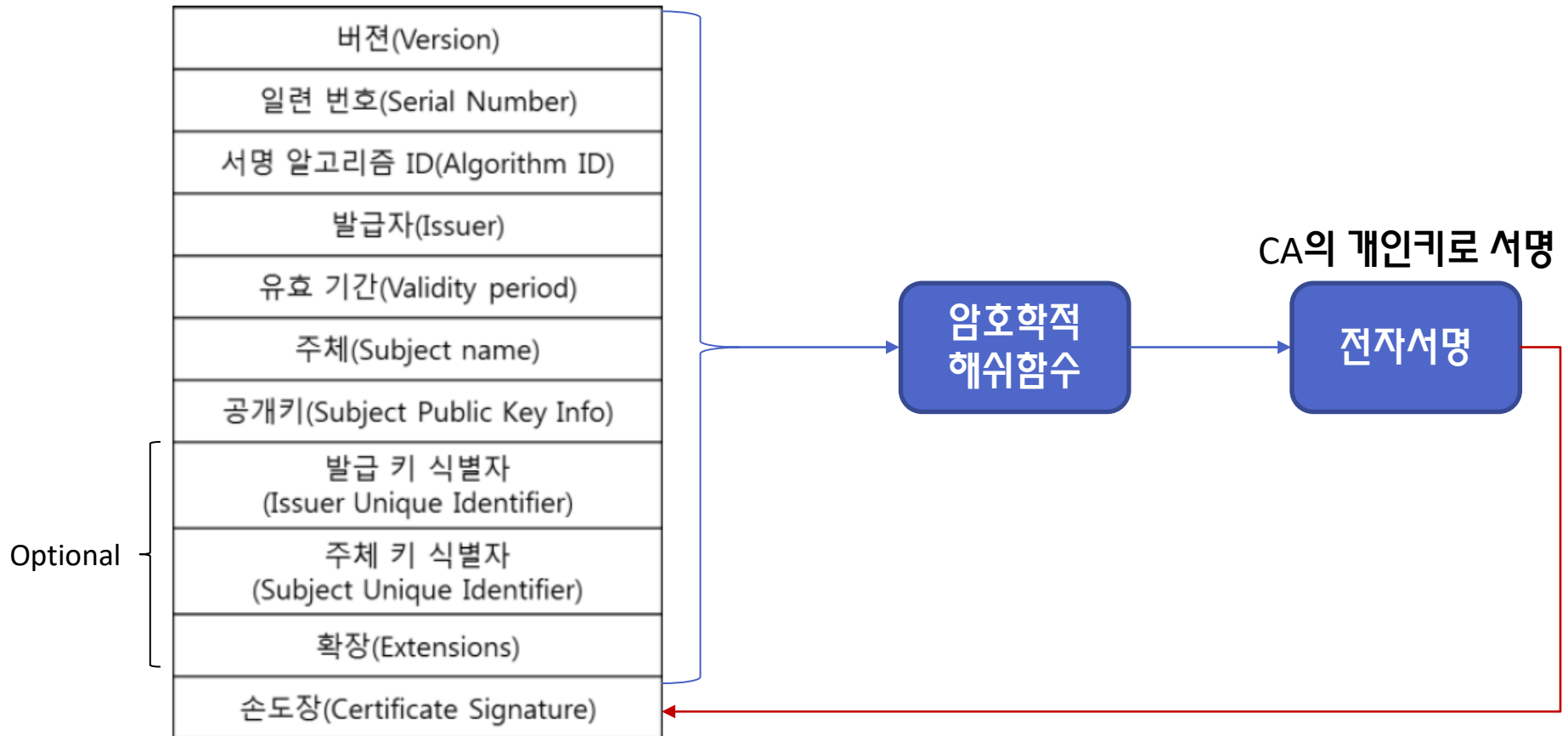


- CA** Certificate Authority
사용자의 인증서를 발급하는 기관
- RA** Registration Authority
사용자와 직접 대면 후 인증 기관에 사용자 정보를 등록해주는 기관
- 인증서** Certificate
CA의 서명이 들어 있는 X.509 표준 규격의 인증서
- 개인키** Private Key
인증서 내의 공개키와 쌍이되는 개인키
- 온라인 서비스** Online Service
PKI 를 통한 사용자 인증을 필요로 하는 서비스 프로바이더(예 : 뱅킹)
- OCSP** Online Certificate Status Protocol
실시간 인증서 상태(예 : 폐기여부) 검증 서비스
- LDAP** Lightweight Dir. Access Protocol
인증서 저장소(LDAP DB)를 액세스 하는 프로토콜(CRL 서비스 수행)

사용되는 국제 표준 규격	설 명
RFC 2459/3280	X.509 인증서와 CRL(Certification Revocation List, 인증서 폐기 목록)에 대한 정의
RFC 2510/2511	CMP 프로토콜에 대한 명세(인증서 발급 과정에 사용 됨)
RFC 2560	실시간 인증서 상태 검증 프로토콜인 OCSP에 대한 명세
RFC 1430/2253	LDAP 프로토콜 명세(LDAP DB 는인증서 저장소로 사용 됨)

공개키 기반 구조 (Public Key Infrastructure)

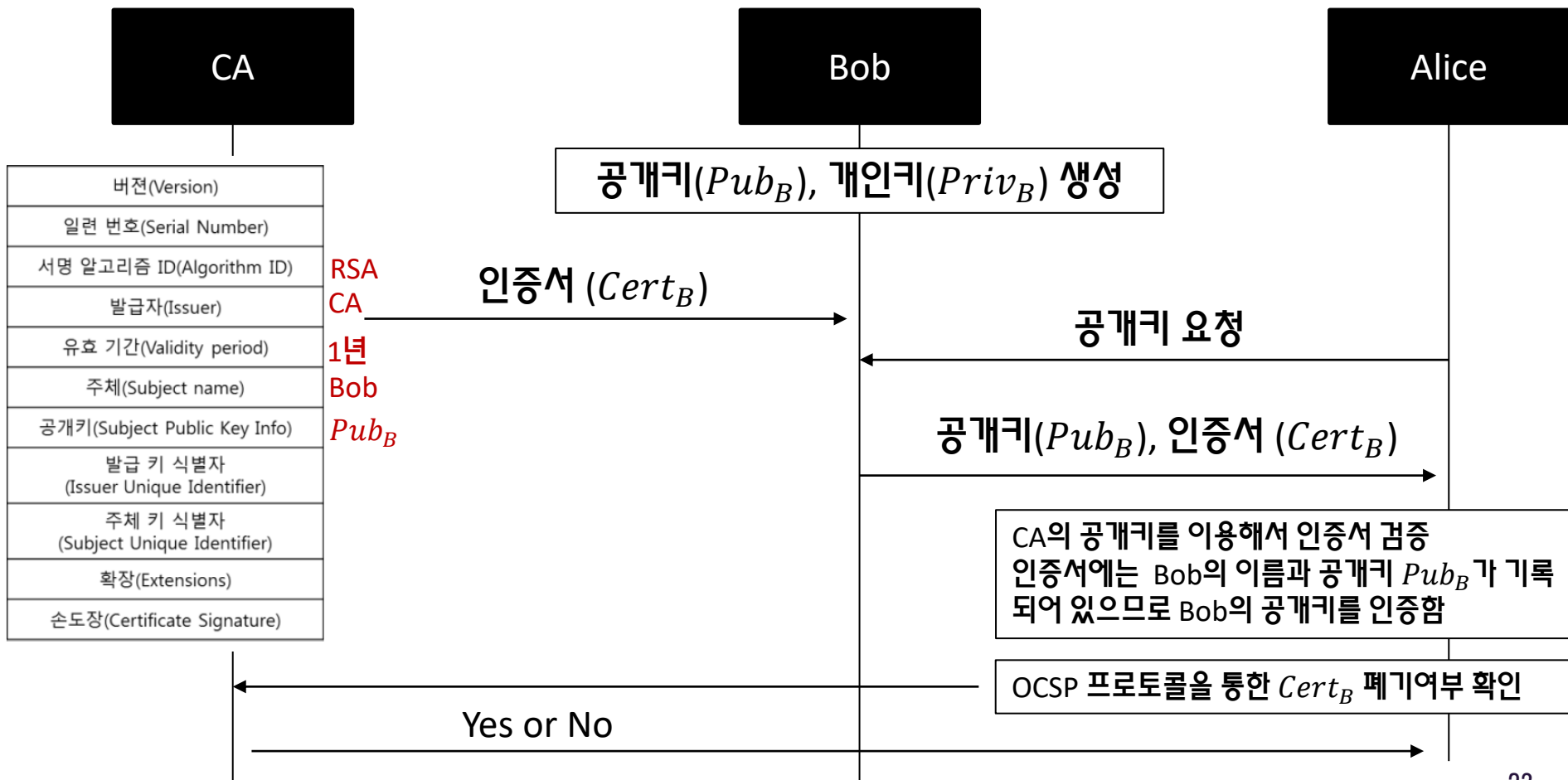
- 공개키 암호시스템을 이용한 키 교환
 - 인증서 형태(X.509)



공개키 기반 구조 (Public Key Infrastructure)

□ CA의 공개키: 모든 디바이스가 알고 있는 값

- E.g., 웹 브라우저에 default로 탑재됨

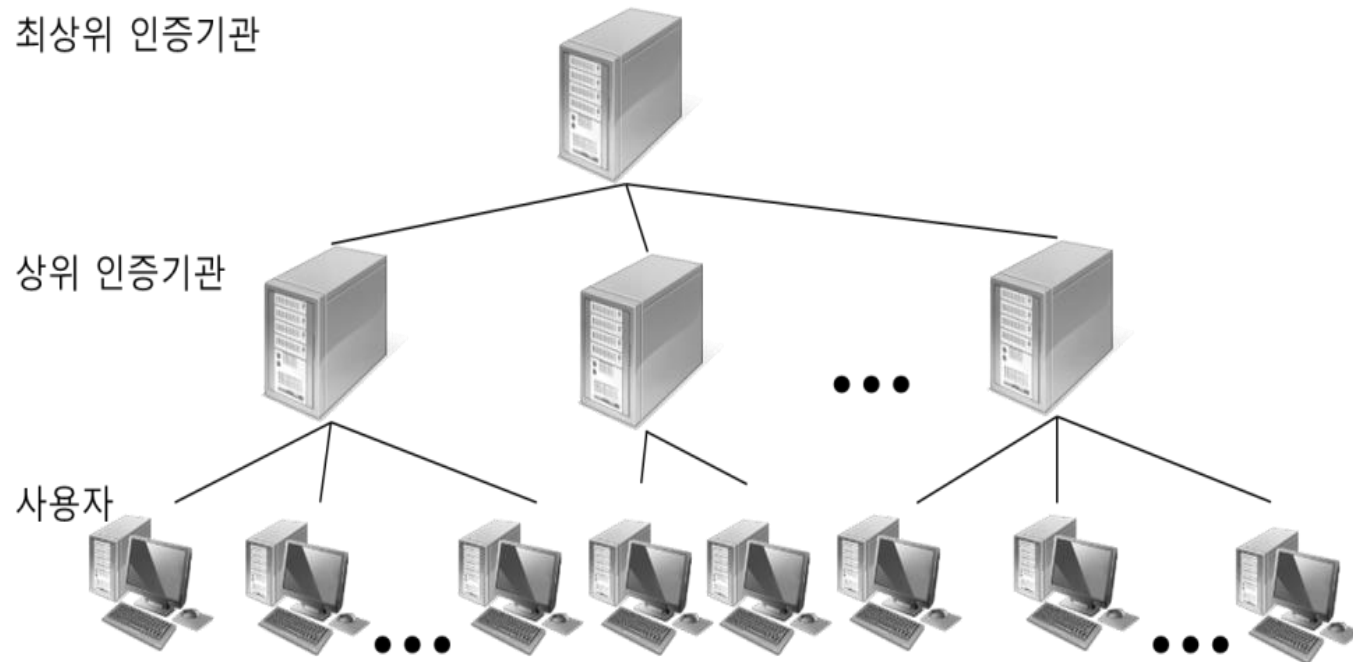


공개키 기반 구조 (Public Key Infrastructure)

□ PKI 구조

▪ 인증 기관들 간 신뢰 모델

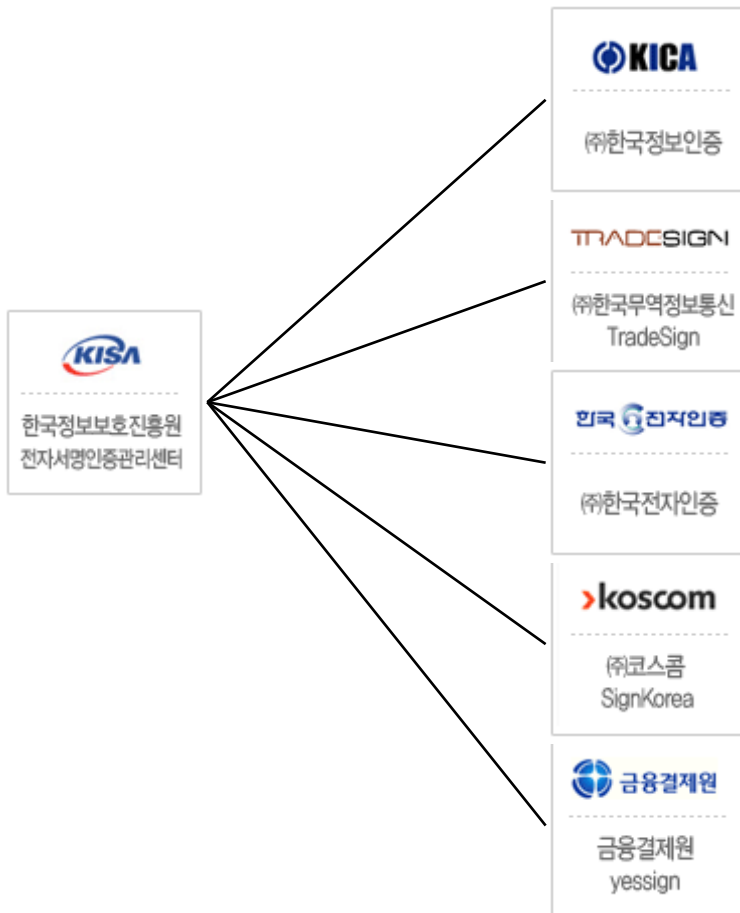
- 계층 모델(Hierarchical Model) - 국내모델
- 상위 계층이 바로 아래 계층의 인증서를 발급, 최상위 계층인 루트 인증 기관은 self-signing



공개키 기반 구조 (Public Key Infrastructure)

PKI 구조

인증 기관(Certificate Authority, CA)



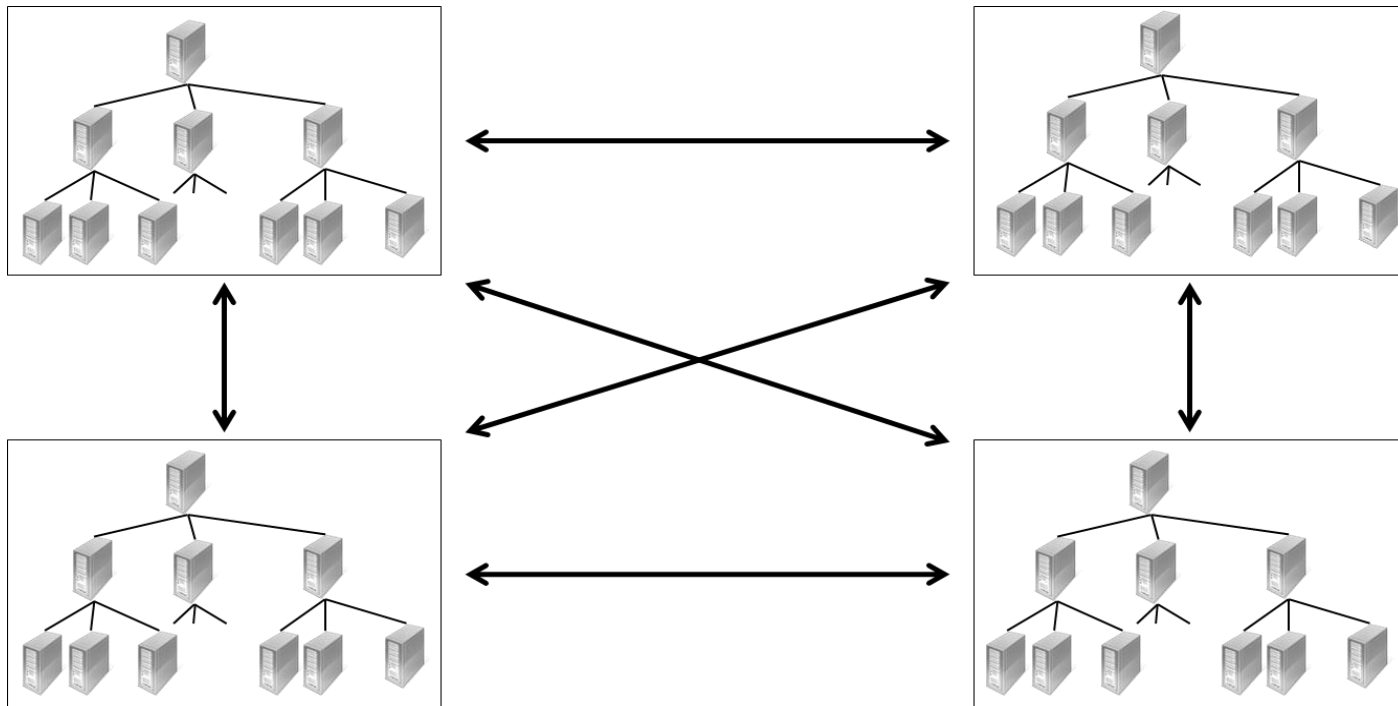
최상위인증기관 인증서 - KISA RootCA 1(RSA)			
발급일자	2005-08-24	인증서버전	X.509 v3
만료일자	2025-08-24	인증서일련번호	04
인증서 다운로드		<input type="button" value="PEM 다운로드"/> <input type="button" value="DER 다운로드"/>	
인증서 내용			
<div>Root CA Certificate</div> <div>Data: Version: 3 (0x2) Serial Number: 4 (0x4) Signature Algorithm: sha1WithRSAEncryption Issuer: C=KR, O=KISA, OU=Korea Certification Authority Central, CN=KISA RootCA 1 Validity Not Before: Aug 24 08:05:46 2005 GMT Not After : Aug 24 08:05:46 2025 GMT Subject: C=KR, O=KISA, OU=Korea Certification Authority Central, CN=KISA RootCA 1 Subject Public Key Info: Public Key Algorithm: rsaEncryption RSA Public Key (2048 bit) Modulus (2048 bit): 00:bc:04:e4:fa:13:39:f0:34:96:20:6b:6c:68:bb: fa:db:77:ff:27:f7:ac:ec:2f:e7:fd:0:7f:6d:6f: 8c:2a:c:d:25:09:5b:24:f4:a1:68:fc:28:ec:c9:25: e2:ac:ed:de:c8:33:84:f5:b0:a5:09:3a:a7:b1:47: 48:c5:cc:4f:8c:79:9c:f9:06:57:7d:dd:ee:38:f6: cf:14:b2:9c:ea:d3:c0:5d:77:62:f0:47:0d:b9:1a: 40:53:5c:64:70:af:08:5a:c0:f7:cf:75:f9:6c:8d: 64:28:1e:20:fe:b7:1b:19:d3:5a:66:83:72:e2:b0: 9b:bd:d3:25:15:0d:32:6f:64:37:94:85:46:c8:72: be:77:d5:6e:1f:28:2f:c7:69:ed:e7:83:89:33:58: d3:de:a0:bf:40:e8:43:50:ee:dc:4d:6b:bc:a5:ea: a6:c8:61:8e:f5:c3:64:af:06:15:dc:23:8b:3f:75: 8c:bc:71:44:db:fc:ad:b5:17:1d:6d:89:83:cf:c6: 33:bd:bf:45:a2:fe:0a:9f:a3:11:5f:0f:b9:1f:9c: 1a:c2:46:cc:9c:28:66:9f:70:26:3c:2e:df:aa:80: fe:8c:c5:04:03:25:4f:cd:93:47:3c:37:ea:02:67: 92:fe:fc:22:24:5c:ac:d2:2c:e0:5c:01:33:8a:c1:</div>			

KISA Root 인증서

공개키 기반 구조 (Public Key Infrastructure)

□ PKI 구조

- 인증 기관들 간 신뢰 모델
 - 메쉬 모델(Mesh Model): 국가간 상호인증



공개키 기반 구조 (Public Key Infrastructure)

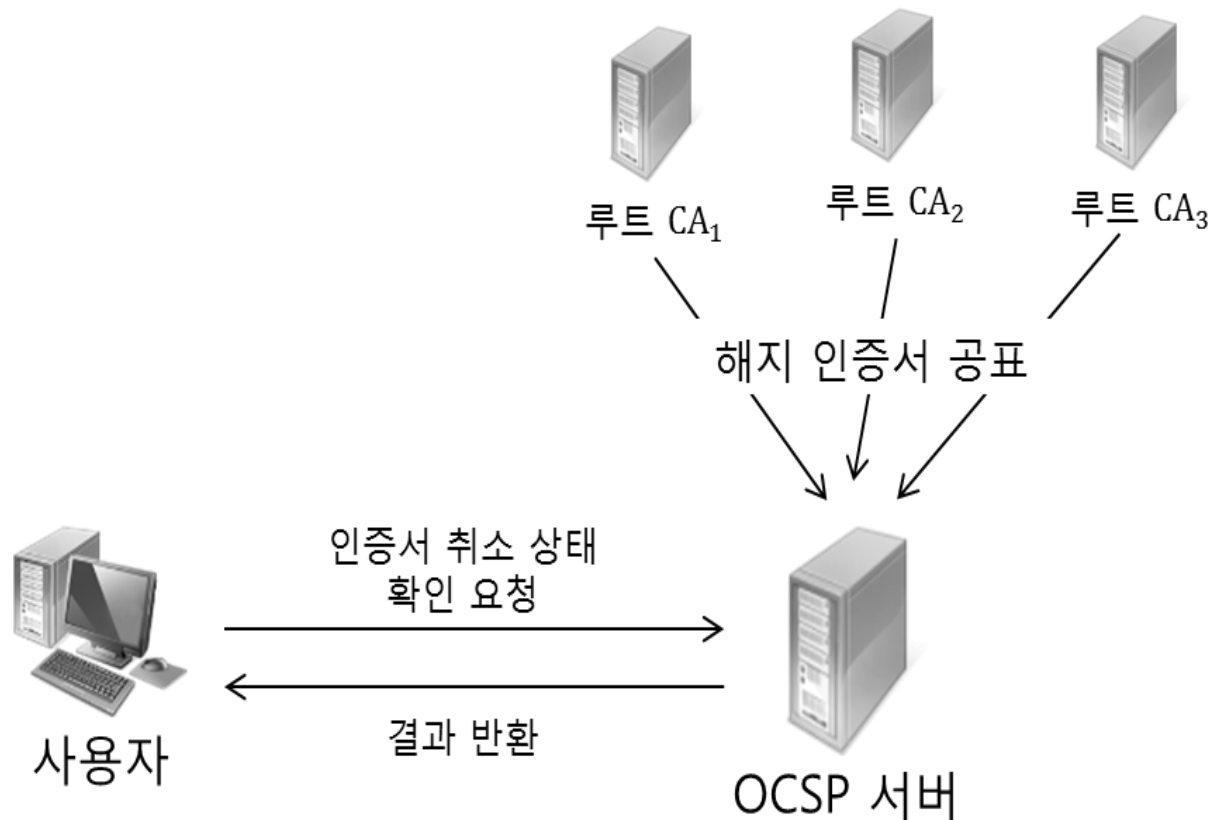
□ 인증서 취소 상태 확인

- 인증서 해지 목록 (Certificate Revoked List, CRL)

서명 알고리즘 ID
발행자 이름
금번 업데이트 시간
다음 업데이트 일자
첫 번째 폐지 인증서
...
마지막 폐지 인증서
서명

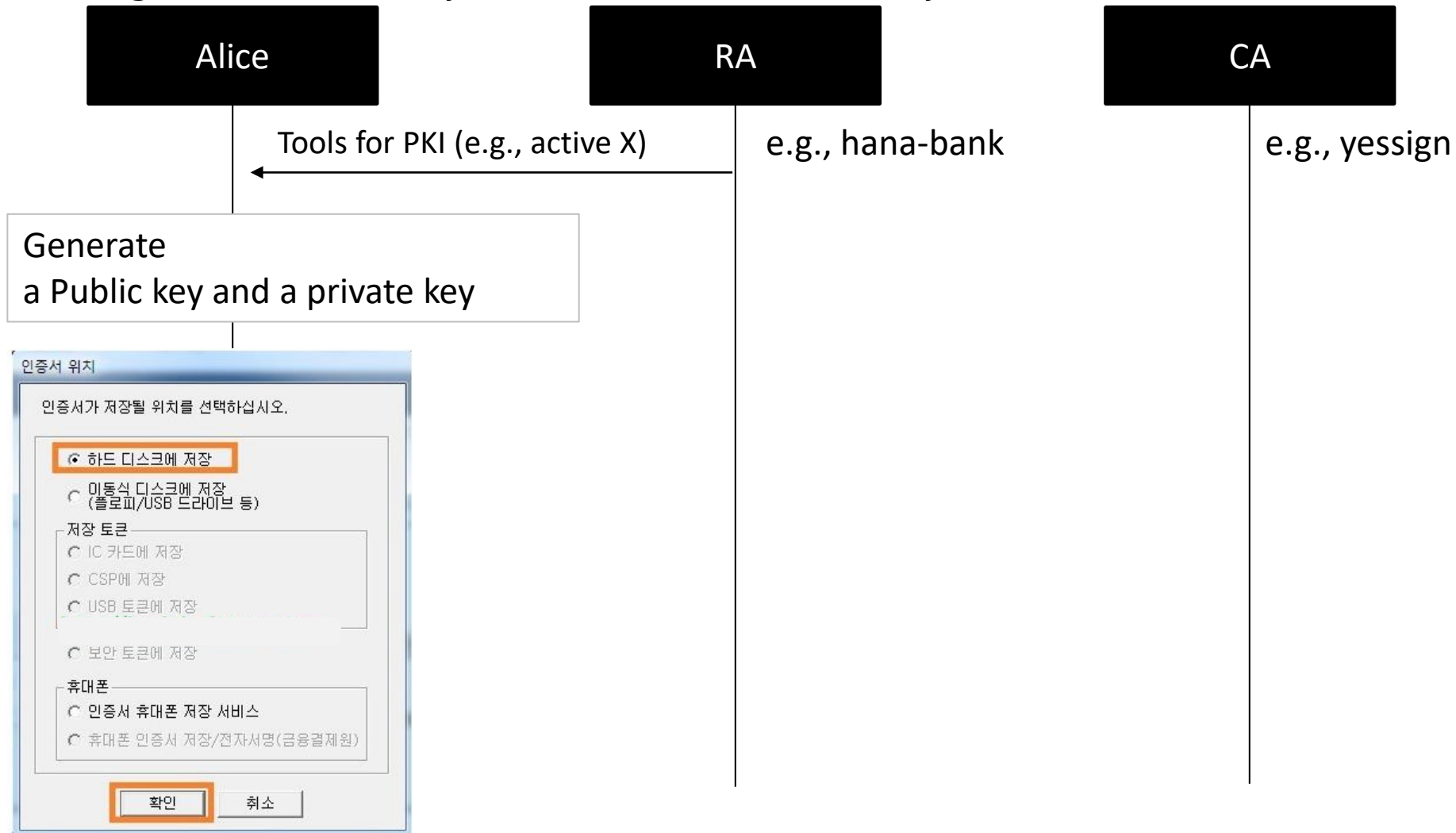
공개키 기반 구조 (Public Key Infrastructure)

- 실시간 인증서 상태 확인 기술(Online Certificate Status Protocol, OCSP)



공개키 기반 구조 (Public Key Infrastructure)

RA = Registration Authority; CA=Certification Authority



공개키 기반 구조 (Public Key Infrastructure)

RA = Registration Authority Authority; CA=Certification Authority

Alice

RA

CA

Tools for PKI (e.g., active X)

e.g., hana-bank

e.g., yesign

Generate
a Public key and a private key

Encrypt the private key

인증서 암호 입력



발급하신 인증서에 사용될 암호를 입력하십시오.
인증서 암호는 본인에게 누락되지 않도록 하시고, 반
드시 8자 이상 문자/숫자 조합으로 입력하십시오.

인증서 암호 입력

인증서 암호 :

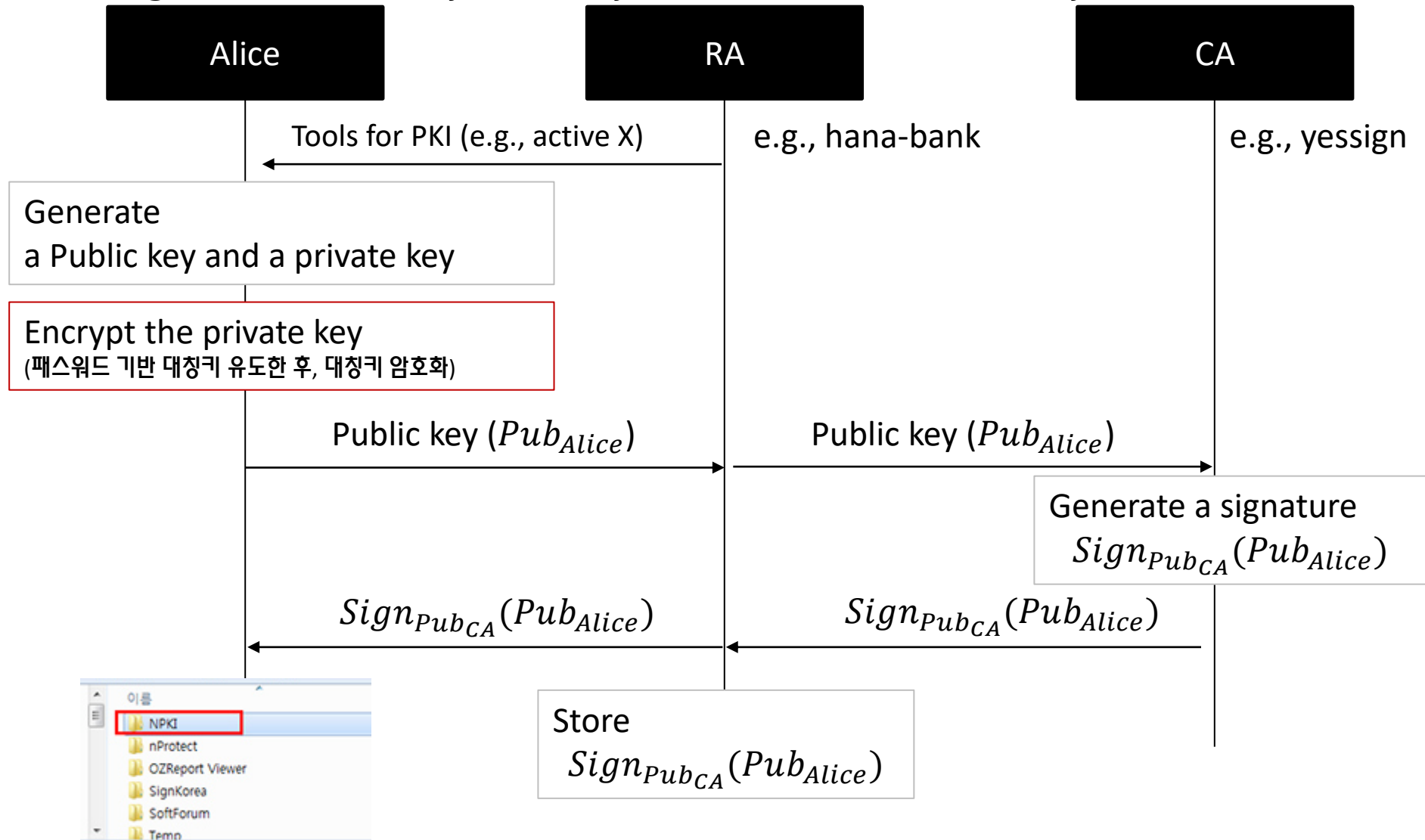
인증서 암호 확인 :

확인

취소

공개키 기반 구조 (Public Key Infrastructure)

RA = Registration Authority Authority; CA=Certification Authority



공개키 기반 구조 (Public Key Infrastructure)

RA = Registration Authority Authority; CA=Certification Authority

Alice

RA

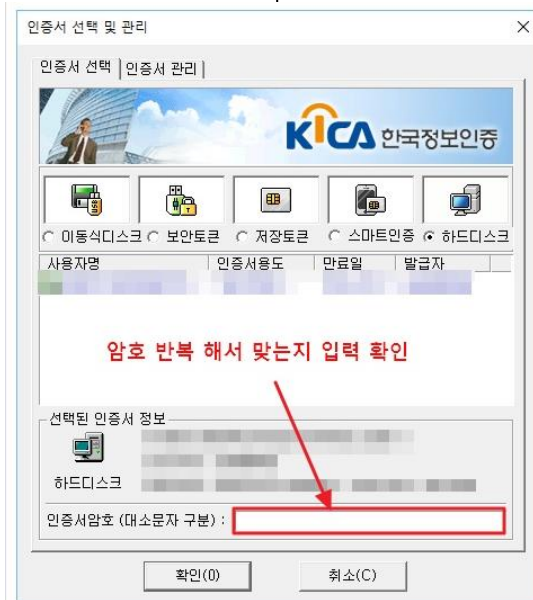
CA

Do transactions
e.g., Deposit/Withdraw

e.g., hana-bank

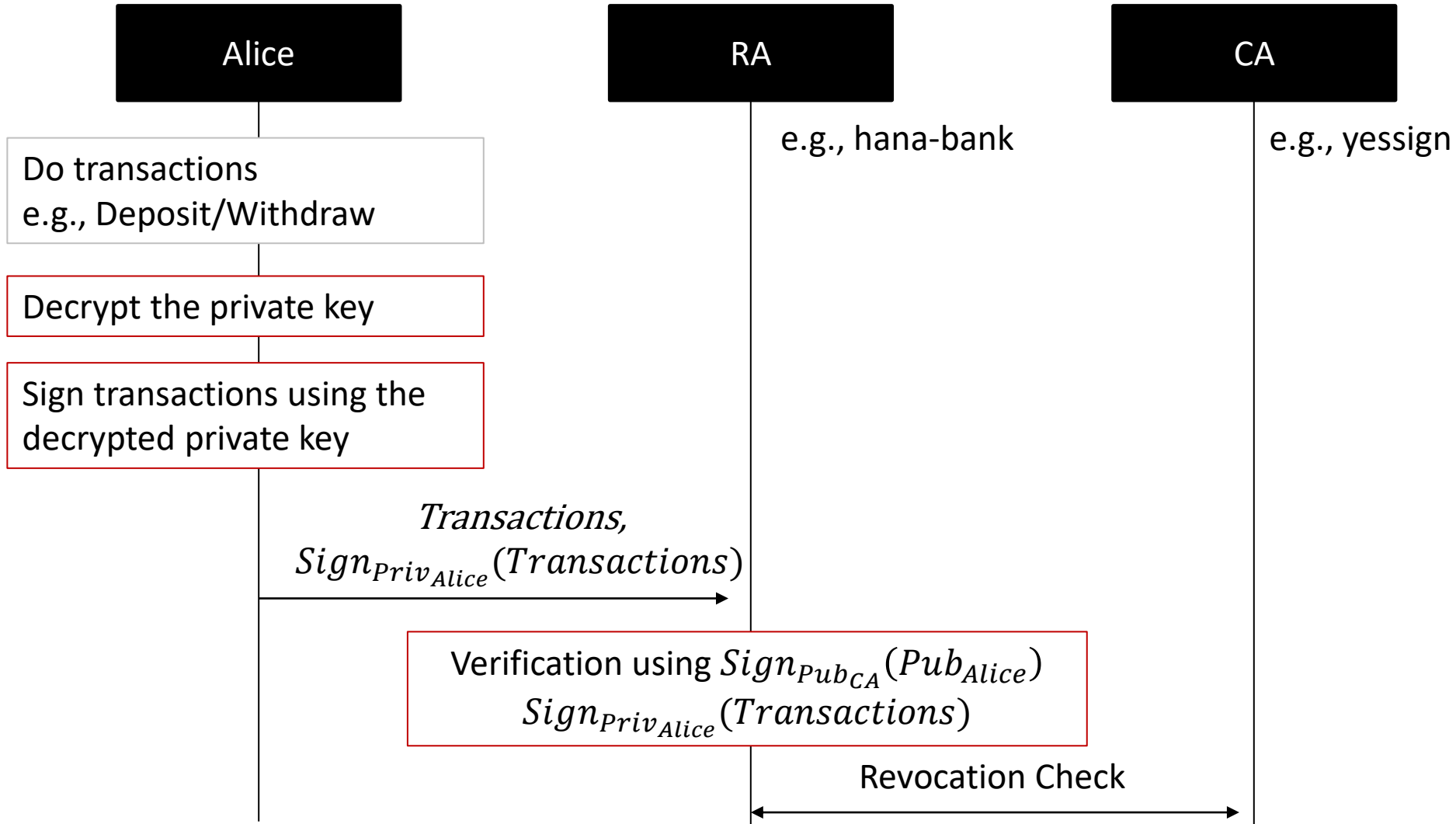
e.g., yessign

Decrypt the private key
(패스워드 기반 대칭키 유도한 후, 대칭키 복호화)



공개키 기반 구조 (Public Key Infrastructure)

RA = Registration Authority Authority; CA=Certification Authority



Signature

- ❑ This Signature class is used to provide applications the functionality of a digital signature algorithm.
 - Digital signatures are used for authentication and integrity assurance of digital data.

- ❑ The signature algorithm can be, among others, the NIST standard DSA, using DSA and SHA-1.
 - The DSA algorithm using the SHA-1 message digest algorithm can be specified as **SHA1withDSA**.

 - In the case of RSA, there are multiple choices for the message digest algorithm, so the signing algorithm could be specified as, for example, **MD2withRSA**, **MD5withRSA**, or **SHA1withRSA**. The algorithm name must be specified, as there is no default.

Signature - use

1. Initialization, with either

- a public key, which initializes the signature for verification ([initVerify](#)), or
- a private key (and optionally a Secure Random Number Generator), which initializes the signature for signing ([initSign\(PrivateKey\)](#) and [initSign\(PrivateKey, SecureRandom\)](#)).

2. Updating

- Depending on the type of initialization, this will update the bytes to be signed or verified. See the [update](#) methods.

3. Signing or Verifying a signature on all updated bytes. See the [sign](#) methods and the [verify](#) method.

Signature - example

```
public class SignTest {  
  
    public static int BUF_SIZE=4096;  
    RSAPrivateCrtKey privKey;  
    RSAPublicKey pubKey;
```

Signature – key setting

```
public void init() throws Exception {  
  
    /* Generate a RSA key pair */  
    KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");  
    SecureRandom random = SecureRandom.getInstance("SHA1PRNG");  
  
    keyGen.initialize(1024, random);  
  
    KeyPair pair = keyGen.generateKeyPair();  
  
    privKey = (RSAPrivateCrtKey)pair.getPrivate();  
    pubKey = (RSAPublicKey)pair.getPublic();  
  
}
```

Signature – example - sign

```
public byte[] testSign(String fileName) throws Exception {

    Signature sign = Signature.getInstance("SHA1withRSA");

    sign.initSign(privKey);

    byte[] buffer = new byte[BUF_SIZE];
    FileInputStream fis = new FileInputStream(fileName);
    int read = BUF_SIZE;

    while ((read = fis.read(buffer, 0, BUF_SIZE)) > 0) {

        sign.update(buffer);
    }

    byte[] bytesSign = sign.sign();

    return bytesSign;
}
```

Signature – example - verify

```
public boolean testVerify(String fileName, byte[] bytesSign) throws Exception {

    Signature sign = Signature.getInstance("SHA1withRSA");

    sign.initVerify(pubKey);

    byte[] buffer = new byte[BUF_SIZE];

    FileInputStream fis = new FileInputStream(fileName);

    int read = BUF_SIZE;

    while ((read = fis.read(buffer, 0, BUF_SIZE)) > 0) {

        sign.update(buffer);
    }

    return sign.verify(bytesSign);
}
```

Signature – example - main

```
public static void main(String[] args) {

    SignTest signTest = new SignTest();
    try {
        signTest.init();

        System.out.println("서명을 생성합니다.");
        byte[] bytesSign = signTest.testSign("01.pdf");
        System.out.println("Sign : " + Utils.toHexString(bytesSign));

        System.out.println("\n서명을 검증합니다.");
        if (signTest.testVerify("01.pdf", bytesSign))
            System.out.println("정당한 서명입니다.");
        else
            System.out.println("정당하지 않은 서명입니다.");

    } catch (Exception e) {
        e.printStackTrace();
    }

}
```


Thank you 