키 관리와 인증서

컴퓨터시스템보안

금오공과대학교 컴퓨터공학부

최태영

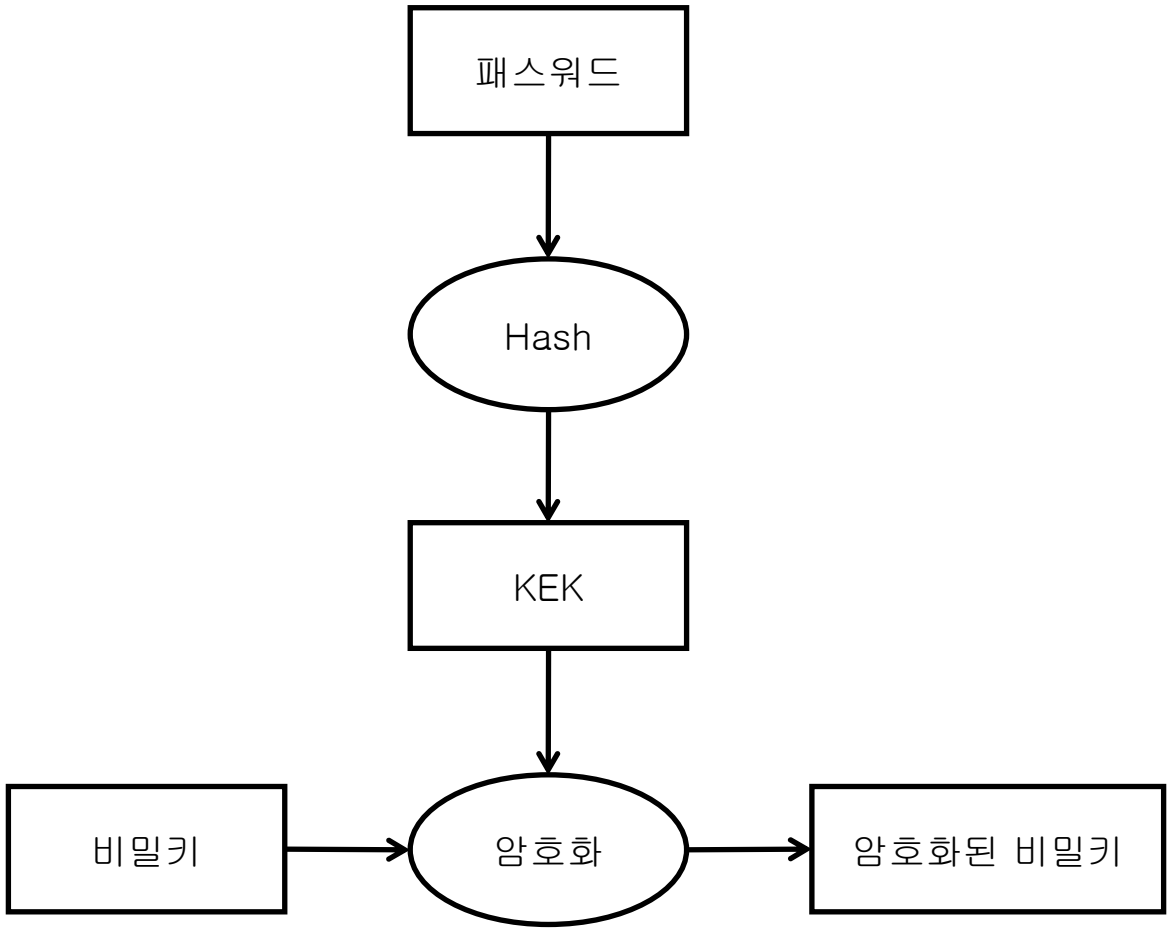
목차

- 비밀키 관리
  - 비밀키 개인 관리
  - 키 관리 센터 (KDC)
- 공개키 관리
  - 공개키 관리의 필요성
  - 공개키 인증서 및 공개키 기반구조
  - 인증서 철회 목록
- 공개키 암호화 표준 (PKCS)
- OpenSSL을 이용한 인증서 관리

비밀키 개인 관리

- 비밀키 도난 방지를 위해 비밀키를 암호화 하여 보관하는 것이 안전
- KEK (Key Encryption Key)
  - 키 암호화에 사용되는 키
  - KEK는 보관되지 않고 필요할 때 생성됨
  - KEK의 생성 : 패스워드와 같이 사용자가 기억하는 정보를 해싱이나 암호화를 통해 생성

KEK 생성 및 사용 과정



키 관리 센터

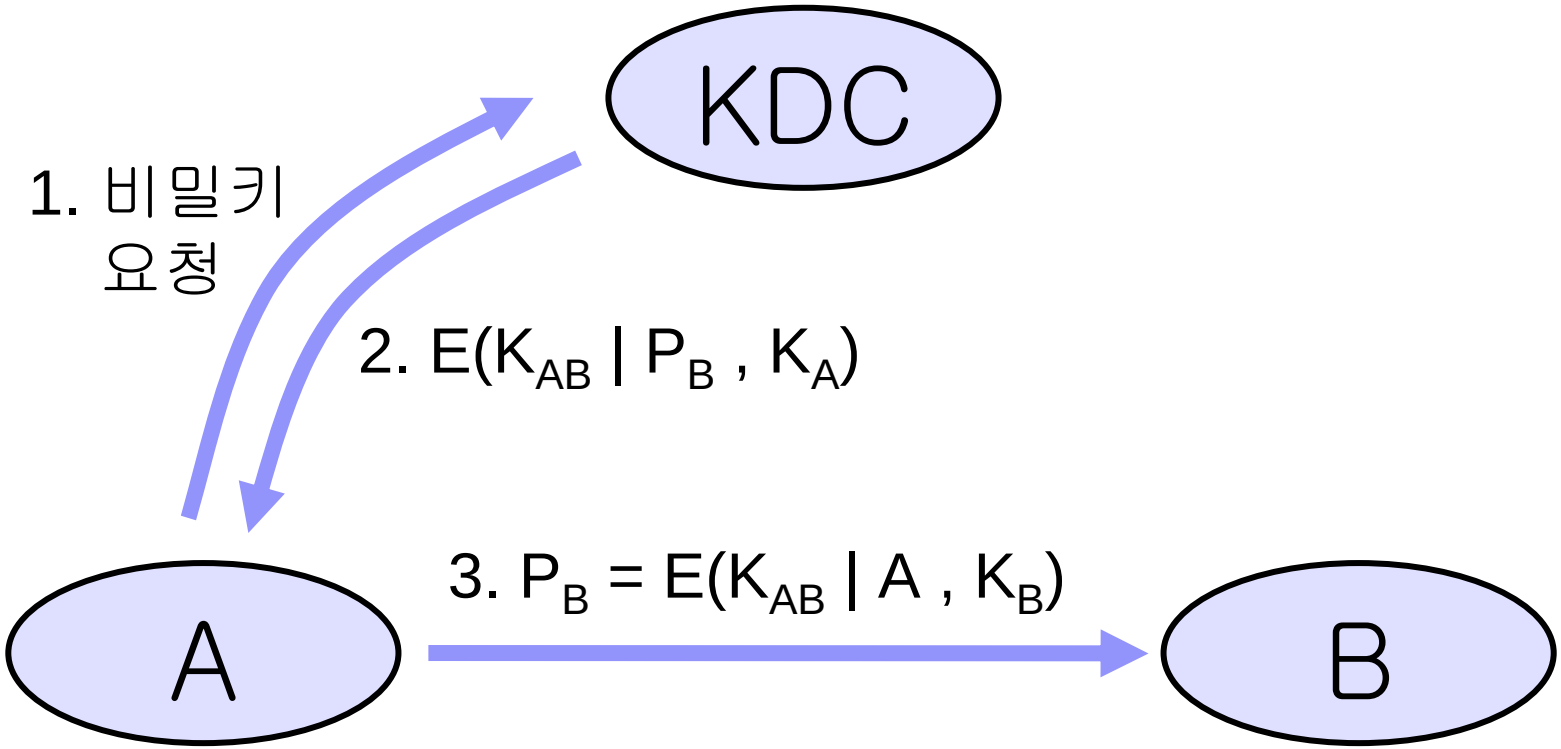
- Key Distribution Center (KDC)
- 신뢰할만한 제 3자로서 키 관리 및 분배를 책임지는 주체
- 사용자들간의 비밀 통신을 보장
- KDC에 대한 사용자 등록 과정
  1. 사용자 A는 KDC에 등록 신청
  2. KDC는 A의 신분 확인
  3. KDC는 A와의 비밀키 K_A를 A에게 전달, (A,K_A)를 저장
- 등록된 사용자는 각각 KDC와의 비밀키를 가지고 있음

KDC를 통한 키 공유

1. 사용자 A는 KDC에 B와 공유할 비밀키 요청
2. KDC는 A의 신원 확인 뒤 $E(K_{AB} \mid P_B, K_A)$를 A에게 전송
   - $P_B = E(K_{AB} \mid A, K_B)$
3. A는 위 암호문을 K_A로 풀어서 K_{AB}를 얻고, P_B를 B에게 전송
4. B는 P_B를 K_B로 풀어서 K_{AB}를 얻음

- 왜 P_B를 직접 KDC가 B에게 보내지 않는가?

KDC를 이용한 키 공유



KDC 분석

- KDC의 단점
  - 단일 실패 지점 (SPoF, Single Point of Failure)
    - KDC가 고장나면 전혀 서비스를 할 수 없음
    - 해결책 : 미러 서버 (mirror server) 가동
      - 서버와 미러 간의 일관성 (consistency) 문제 발생 가능
  - KDC가 공격되면 모든 비밀키가 노출될 수 있음
  - KDC의 신뢰성이 매우 중요함
  - KDC에 대한 병목현상 (bottleneck) 발생
- KDC의 장점
  - 비밀키 방식은 비교적 속도가 빠름
  - 서버를 사용하지 않는 경우보다 키 관리를 쉽게 해줌

공개키 관리의 필요성

- N명 사용자에 대해 N쌍의 키가 필요하므로 비밀키에 비하여 관리가 용이
- 개인키 관리
  - 공격에 의한 노출을 막기 위해 KEK를 통한 암호화가 바람직
- 공개키 관리
  - 소극적인 공격만 있는 경우에는 인터넷과 같은 공개채널을 이용하여 전달이 용이함
  - 적극적 공격에는 안전하다고 할 수 없음
    - 밥이 앨리스에게 보낸 밥의 공개키를 공격자 트루디가 중간에 자신의 공개키로 바꾸어 앨리스에게 전달
    - 앨리스는 트루디의 공개키로 자신의 메시지를 암호화하여 밥에게 전달
    - 트루디는 밥에게 전달되는 메시지를 복호화하여 볼 수 있음
- 공개키 전달 시에 인증과 무결성이 보장되어야 함
  - 전달될 공개키에 대한 전자서명이 필요함
  - 누구의 개인키로 서명을 할 것인가?

공개키 인증서

- Public key certificate
  - 공개키에 전자서명을 첨부한 것
- X.509 표준에 포함
  - X.509 : International Telecommunication Union (ITU)에서 제안한 공개키 기반구조 (Public Key Infrastructure, PKI) 표준의 일부
  - 현재 Version 3이 사용
- Version에 따른 필드 추가
  - Ver.2 : 발행인 및 공개키 소유자 고유번호 추가
  - Ver.3 : 발행인 키 구분자, 소유자 키 구분자, 키 사용범위, 개인키 사용 기간, 인증서 정책, 소유자 별명, 발행인 추가 정보, 소유자의 인증 기관 여부, 이름 제한, 정책 제한 추가

공개키 인증서 구조

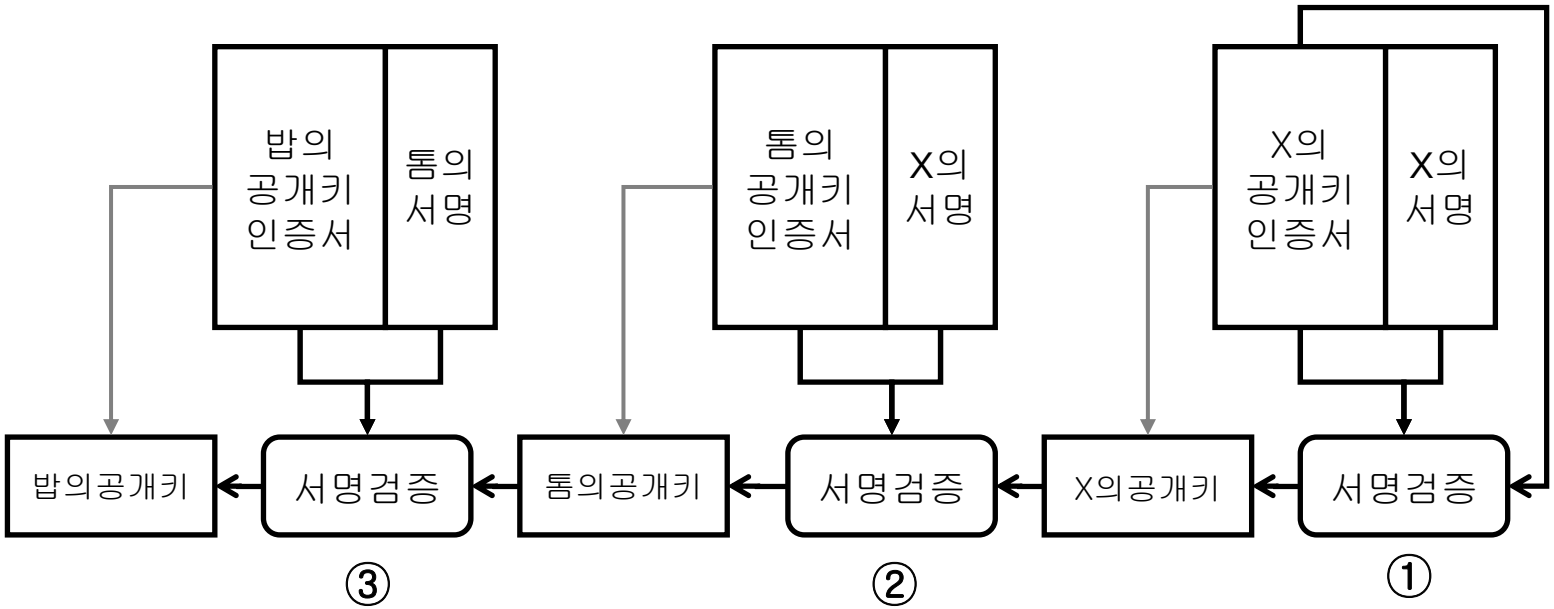

버전 (Version)
일련번호 (Certificate Serial Number)
서명 알고리즘 (Signature Algorithm) f
발행인 이름 (Issuer Name) I
유효기간 (Validity)
공개키 소유자 (Subject Name) A
공개키 (Subject Public Key Information) K^{O}_{A}
발행인 고유번호 (Issuer Unique Identifier)
공개키 소유자 고유번호 (Subject Unique Identifier)
확장 필드 (Extensions)
서명 $f(m , K^{P}_{I})$
m

인증서 각 필드 설명

- Certificate Serial Number : 인증서가 발행된 순서
- Signature Algorithm : 어떠한 공개키 알고리즘을 사용했는지를 기입
- Issuer Name : 인증서에 서명한 사람 또는 기관의 이름
- Validity : 인증서내의 공개키가 유효한 기간
  - 키가 닳기 전에 무효화 시키야 함.
  - 유효기간이 지난 공개키 인증서는 효력이 없음
- Subject Name : 인증서내 공개키의 소유자
- Subject Public Key Information : 공개키
- Issuer Unique Identifier (v2) : 서명한 사람을 식별하기 위한 고유 번호 (여러 사람이 한 공개키를 서명할 수 있음)
- Subject Unique Identifier : 공개키 소유자의 고유 번호
- 서명 : 위의 정보들에 대한 서명
  - 위의 내용들을 발행인 (Issuer)의 개인키를 이용하여 위에서 언급한 서명 알고리즘으로 서명한 전자 서명

공개키 인증서 체인

- 공개키 인증서를 검증하기 위해서는 발행자의 공개키가 있어야 함
- 보통 발행자의 공개키는 그 발행자의 공개키 인증서에 있음.
- 최종 공개키 인증서를 얻을 때까지 이러한 과정을 되풀이함



자기서명 인증서

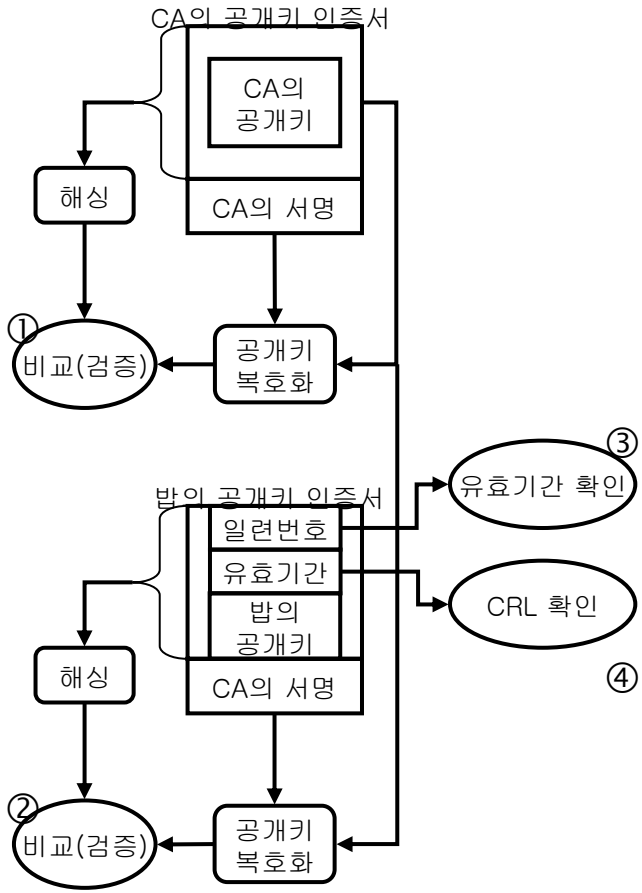
- Self-signed certificate
  - 발행자가 공개키의 소유자와 동일한 인증서
  - 발행자의 개인키로 발행자의 공개키를 서명한 인증서
- 루트 인증서 (Root certificate)
  - 인증기관 (Certificate Authority, CA)가 발행한 자기 서명 인증서
- 공개키 인증서 체인의 마지막에 존재하는 인증서
- 신뢰할 수 있는 상대로부터 신뢰할 수 있는 안전한 채널을 통해 받아야 함
  - 직접 전달 받거나,
  - Fingerprint를 통해 검증함

공개키 기반구조

- Public Key Infrastructure (PKI)
- 인증 기관 (Certificate Authority, CA)
  - PKI에 참여하고 있는 회원들의 공개키를 관리
  - 공개키 보관, 인증서 작성, 인증서 제공
- PKI에서 상대방의 공개키 획득 과정
  1. CA의 루트 인증서 획득 (또는 회원 등록)
  2. CA로부터 밥의 공개키 인증서 획득
  3. CA의 공개키로 밥의 공개키 인증서 검증
- PKI에 대한 회원가입 과정
  1. CA 또는 등록기관 (Registration Authority, RA)에 신원 확인
  2. CA의 루트 인증서 획득
  3. 자신의 공개키와 개인키를 생성하여 공개키와 인증서 신청서를 제출
  4. CA는 공개키 인증서를 생성

PKI

- 공개키 인증서 검증 과정 (루트 인증서 소유시)
  1. CA의 공개키 인증서 검증
  2. 상대방의 공개키 인증서를 CA의 공개키로 검증
  3. 상대방의 공개키 인증서 내 유효기간을 확인
  4. 상대방의 공개키 인증서가 인증서 철회 목록 (Certificate Revocation List, CRL)에 없는지 확인



인증서 철회 목록

- Certificate Revocation List (CRL)
  - 이미 철회된 인증서들의 일련번호 모음

Version Signature Algorithm Identifier Issuer name This Update Next Update		
Serial	Revocation date	Extensions
2220002	2009/03/08	개인키 노출
2221001	2009/03/09	회원 탈퇴
Signature		

CRL

- 공개키가 철회되는 이유
  - 사용자의 개인키가 도난과 같은 이유로 노출되었다고 의심되는 경우,
  - 공개키 인증서의 발급 과정에서 문제가 발생한 경우,
  - 새로운 공개키쌍 사용, 또는
  - 자신의 직위가 바뀜에 따라 그 직위에 부여된 공개키가 더 이상 사용되지 않는 경우.
- CRL 전달 방법
  - offline: 주기적으로 사용자에게 email등의 방법으로 전달
  - online : 공개된 서버에 CRL을 게시

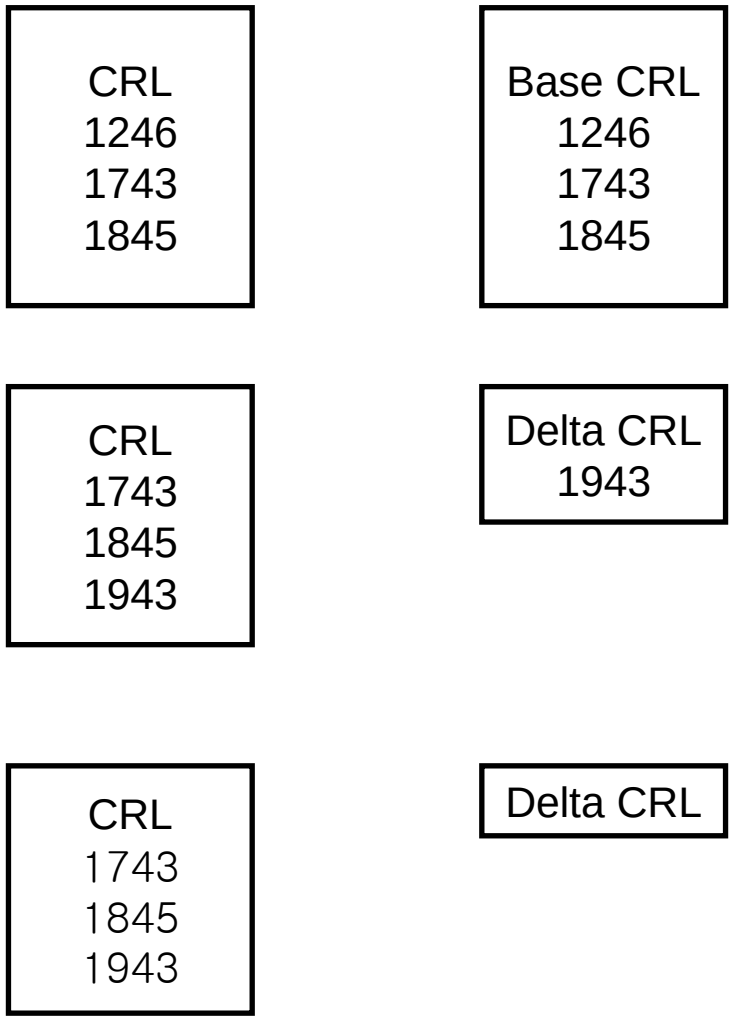
Offline CRL Management

- CRL은 주기적으로 발송되어 CA가 오프라인이라도 키관리가 가능하도록 설계
  - PKI는 CA가 오프라인이라도 가동할 수 있게 구성된 시스템
- 유효기간이 지난 공개키는 CRL에 포함되지 않음
  - 항상 유효기간과 CRL을 같이 검토해야 함
- 공개키를 사용하는 시스템은 항상 CRL을 보관해야 함
  - 카드 결제 시스템 같은 소형 시스템도 포함됨
- 시스템의 용량과 성능에 대한 부하 증가
- CRL 전달시 CA의 병목현상 (bottleneck) 유발
- Delta CRL : 병목현상 감소 목적으로 개발

Delta CRL

- Base CRL과 Delta CRL로 구성
- Base CRL : 처음 발송되는 CRL
- Delta CRL : 이전에 발송한 CRL과의 차이만 포함된 CRL
- Delta CRL은 별도의 필드가 있어야 함
  - Delta CRL 여부를 알려주는 필드
- 장점
  - CRL의 양이 줄어들기 때문에 병목현상이 완화됨
- 단점
  - Delta CRL들 중 하나만 빠져도 일관성 없는 목록이 됨
  - 각 사용자 시스템의 부하는 줄어들지 않음

Delta CRL


CRL
1246
1743
1845
Base CRL
1246
1743
1845
CRL
1743
1845
1943
Delta CRL
1943
CRL
1743
1845
1943
Delta CRL

Offline CRL

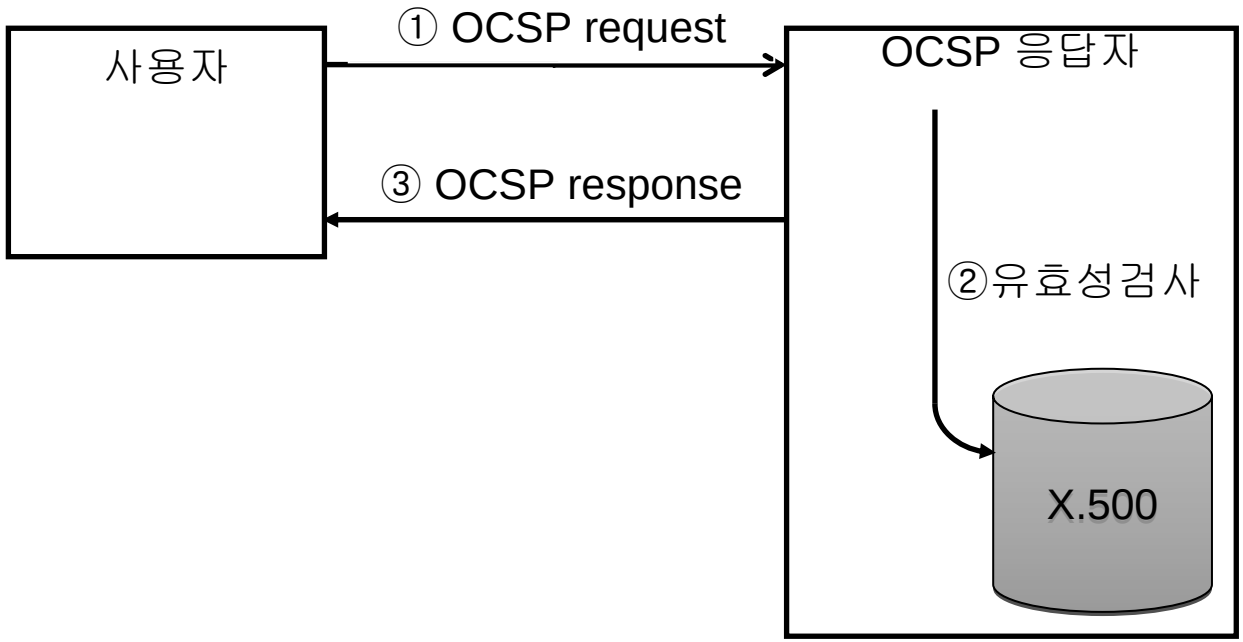
- CRL 발행 후에 철회된 공개키는 확인하지 못함
  - 오프라인으로서의 문제점
  - Ex) CRL이 3/10에 발급되었고 공개키 인증서가 3/14에 무효화되면, 3/15에 그 인증서를 받은 사람은 무효화된 인증서를 이용하게 됨
- 갱신될 내용이 없어도 다음 CRL이 발급되어야 함
  - 주기적 갱신으로서의 문제점
  - 유효기간이 남아있는 철회된 공개키 인증서 목록이 모두 발급됨
  - Delta CRL로 어느 정도 부하를 줄일 수는 있음

온라인 인증서 철회 관리

- 오프라인 방식을 온라인 방식으로 변경
- Online Certificate Status Protocol (OCSP)
  - 검증시마다 인증서의 철회 여부를 서버에게 문의
  - OCSP responder가 철회된 인증서의 목록을 관리하며 질의에 응답함
    - OCSP response : 인증서의 상태 (good, revocated, unknown), 일시, (철회이유,) 전자서명
  - http 또는 email 같은 기존의 인터넷 통신 프로토콜 사용
- Simple Certificate Validation Protocol (SCVP)
  - Delta CRL과 인증서 체인에 필요한 목록 제공

OCSP

1. 사용자는 OCSP 응답자에게 한 인증서의 정당성 여부를 묻는 요청 (OCSP request)를 보낸다.
2. OCSP 응답자는 자신의 X.500 디렉토리에 들어있는 해당 인증서의 유효여부를 검사한다.
3. OCSP 응답자는 검사 결과를 OCSP response의 형태로 사용자에게 돌려준다.



공개키 인증서의 종류

- 이메일 인증서
- 서버측 인증서
- 사용자측 인증서
- 코드 인증서
- 로밍 인증서

공개키 인증서 (계속)

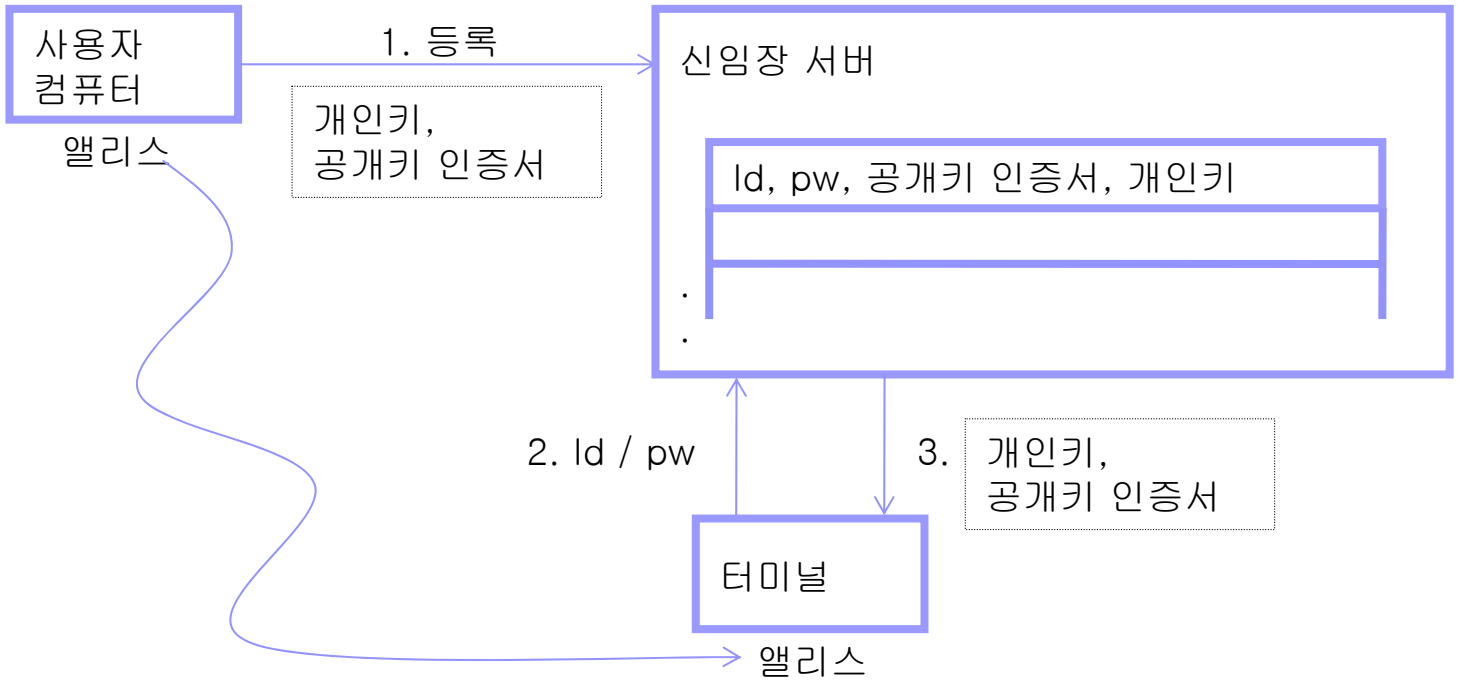
- 이메일 인증서
  - 이메일의 신원을 보장하기 위해 사용
  - VBR (Vouch by Reference) : 제 3자가 이메일 발송지의 신원을 보장하는 방법
- 서버측 인증서
  - 서버가 자신에게 암호문을 보낼 수 있도록 클라이언트에게 보내는 인증서
  - 서버의 신원을 보장하는 용도로도 사용
- 사용자측 인증서
  - 서버가 클라이언트의 신원을 확인하기 위해 사용

공개키 인증서 (계속)

- 코드 인증서
  - 자바 애플릿이나 액티브-액스는 트로이 목마의 가능성을 포함하고 있음
  - 다운로드 받은 프로그램의 신뢰성을 보장해주기 위해 사용
  - 프로그램의 서명과 그 서명을 검증할 수 있도록 첨부된 공개키 인증서
- 로밍 인증서 (Roaming certificate)
  - 인증서를 휴대하지 않고도 외부에서 필요한 인증서를 얻는 방법

로밍 인증서

- 신임장 서버 (Certificate server)에 공개키 인증서와 개인키를 저장해 두고 필요할 때마다 접속하여 인증서를 획득



개인키 관리

- 개인키 관리의 어려움
  - 개인 컴퓨터에 보관된 개인키는 공격자의 컴퓨터 침입에 대비해야 함
  - 휴대하는 개인키는 분실 또는 복제의 위험성에 노출됨
- 개인키 공격에 대한 방어
  - 패스워드 : KEK 등을 이용하여 개인키를 암호화함, 패스워드 추측공격을 당할 수 있음
  - 암호화한 이동형 저장장치 : 저장장치의 내용에 접근하기 위해서 pin 등으로 암호화함
- PKCS#12는 개인키와 인증서를 암호화하여 저장하는 표준임

개인키 관리 (계속)

- 복호화를 위한 개인키와 서명을 위한 개인키는 구분할 필요가 있음 (8장에서 다룸)
- 용도에 따른 개인키 보관 시기가 다름
  - 복호화용 개인키
    - 인증서가 무효화되었더라도 보관해야 함
    - 무효화되기 이전에 생성된 암호문이 무효화된 이후에 도착할 수 있음,
    - 또는 이전에 저장된 암호문이 나중에 참조될 수 있음
  - 서명용 개인키
    - 인증서가 무효화되는 즉시 폐기해야 함 (악용 가능성)
  - 공개키 인증서
    - 유효기간이 지나더라도 CA에서 보관
      - 소송 등의 증거로 사용됨

공개키 암호화 표준

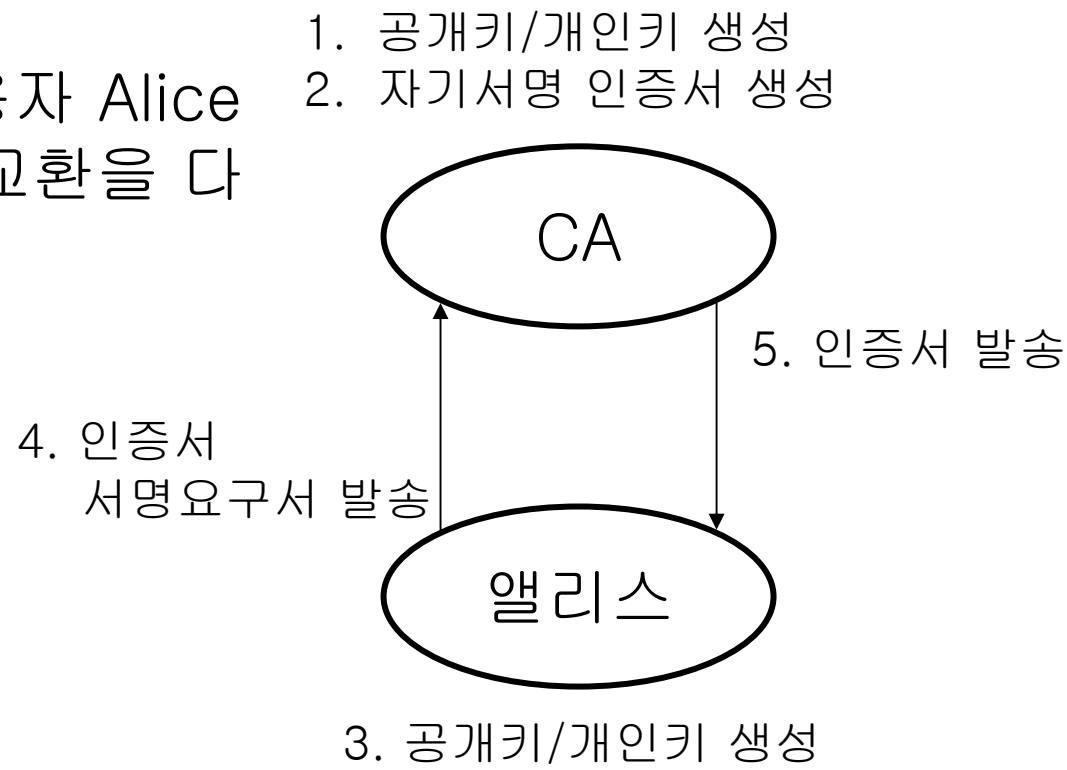
- Public Key Cryptography Standard (PKCS)
  - RSA Laboratories에서 제안
  - 공개키 기반구조 (PKI) 표준
  - 15개의 항목으로 구성
  - 메시지 포맷, 알고리즘, API 등을 포함
  - 2개 항목은 개정 과정에서 통합되었으며 2개 항목은 개발중임
- X.509
  - Telecommunication Standardization Sector (ITU-T)에서 제안한 X.500 표준의 일부
  - 공개키 인증서, CRL, attribute certificate (AC, authorization certificate) 등을 주로 다룸
    - AC : 특정 자원이나 서비스를 이용할 수 있는 허가서

- PKCS#1 (RSA Encryption Standard)
  - 공개키쌍의 수학적 특성 기술
  - RSA 암호화 및 복호화 알고리즘
  - 패딩방식
  - 전자 서명 및 검증 알고리즘
- PKCS#2 (PKCS#1에 포함됨)
- PKCS#3 (Diffie-Hellman Key Agreement Standard)
- PKCS#4 (PKCS#1에 포함됨)
- PKCS#5 (Password-based Encryption Standard)
  - KEK 생성 알고리즘
- PKCS#6 (Extended Certificate Syntax Standard)
  - X.509 v3이 이를 대체함
- PKCS#7 (Cryptographic Message Syntax Standard)
  - 암호화, 서명 등을 포함하는 경우와 포함하지 않는 경우의 메시지 표준 양식이 포함됨

- PKCS#8 (Private-Key Information Syntax Standard)
  - 개인키를 (휴대하고) (암호화하여) 보관하는 방식
- PKCS#9 (Selected Attribute Types)
  - PKCS#6의 확장된 인증서, PKCS#7의 전자서명 메시지, PKCS#8의 개인키 정보, PKCS#10의 인증서-서명 요구에 관련된 특성들에 대한 정의를 다룸
- PKCS#10 (Certification Request Standard)
  - 인증서 요구 메시지의 양식과 프로토콜
- PKCS#11 (Cryptographic Token Interface, Cryptoki)
  - 암호화 토큰에 대한 API 표준
- PKCS#12 (Personal Information Exchange Syntax Standard)
  - 공개키 인증서, 개인키, 공개키 저장을 위한 파일 포맷
- PKCS#13 (Elliptic Curve Cryptography Standard)
- PKCS#14 (Pseudo-random Number Generation)
- PKCS#15 (Cryptographic Token Information Format Standard)
  - 암호화 토큰의 사용자가 자신의 신분을 토큰에게 알리는 방법

OpenSSL을 이용한 인증서 관리

- OpenSSL command line 명령어를 사용
- 인증기관 CA와 사용자 Alice 간의 파일 생성 및 교환을 다룸



루트 인증서 생성

- CA의 공개키 인증서 (root certificate) 생성의 예

```
$ openssl genrsa -out CAPriv.pem 1024

$ openssl req -new -key CAPriv.pem -out CAReq.pem

$ openssl x509 -req -days 3650 -in CAReq.pem -signkey \
CAPriv.pem -out CACert.pem
```

  - 공개키 인증서는 인증서 요청서 (CAReq.pem)를 통해 생성함

OpenSSL **req** command

- 인증서 서명 신청서 작성 명령어
- PKCS#10 표준을 따름
- -new : 새로운 서명 신청서를 만든다는 옵션
- -key <file> : 신청서에는 공개키도 포함되고 신청서 내용에 대한 서명이 첨부됨
  - 서명은 공개키 소유자의 개인키로 서명됨
- -out <file> : 신청서 파일명
- -in <file> : 기존의 신청서 파일 내용을 참조함을 의미함
- -newkey arg : 새로운 키를 만들어 사용함을 의미함
  - Ex) -newkey rsa:1024
- -nodes : 새로운 개인키를 만들었을 경우 이를 암호화하여 저장하지 않겠다는 의미
- -subj arg : 인증서 소유자 관련 정보가 포함됨

Subject 정보 입력 예

$ openssl req -new -key CAPriv.pem -out CAReq.pem
Country Name (2 letter code) [AU]:KR
State or Province Name (full name) [Some-State]:CA-State
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Hanti
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:Gil-Dong
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

OpenSSL x509 command

- 인증서 내용을 보여주거나, 다른 형식으로 변환하거나, 서명을 하여 인증서를 만드는 명령어
- -req : 인증서 서명 신청서를 입력으로 받아들임을 뜻하는 옵션
- -days <day> : 현재 시각부터 <day>까지를 만들어질 인증서의 유효기간으로 함을 뜻함
- -signkey <file> : 개인키가 들어있는 파일을 가리킴
  - ex) -signkey CAPriv.pem
- -fingerprint : 인증서의 DER 형식에 대한 digest (or hash)를 출력한다.

```
$ openssl x509 -noout -in CACert.pem -fingerprint
SHA1 Fingerprint=90:45:5F:BF:BC:C1:15:32:BB:A2:35:7B:3F:
19:57:54:14:90:32:8C
```

공개키 인증서의 검증

```
$ openssl verify -CAfile CACert.pem CACert.pem
```

- verify command
  - 공개키 인증서로 다른 공개키 인증서를 검증
  - 위 예는 root 인증서를 검증한 경우
  - -CAfile : 인증서를 검증할 공개키가 들어있는 인증서
    - 위 예의 마지막 파라미터는 검증될 공개키 인증서
  - -CApath <dir> : 신뢰할만한 인증서가 hash.0이란 이름을 가질 경우 그 디렉터리를 <dir>에 기입하면 인증서 파일명을 적을 필요가 없음
  - -untrusted <file> : 신뢰하지 못하는 인증서들이 있을 때 이들의 이름을 여기에 기입함

개인 사용자를 위한 인증서 발급

```
$ openssl req -new -keyout AlicePriv.pem -subj \
'/C=KR/ST=CA-State/O=Hanti/CN=Alice' -out AliceReq.pem
Generating a 1024 bit RSA private key
.............++++++
.......++++++
writing new private key to 'AlicePriv.pem'
Enter PEM pass phrase: Alice is making cert
Verifying - Enter PEM pass phrase: Alice is making cert
-----
```

- req command : 사용자의 개인키와 인증서 서명 신청서를 한꺼번에 만들 수 있는 명령어
- -subj : 파라미터로 인증서 소유자의 정보를 입력할 수 있음
- pass phrase : password 대용으로 사용되는 비밀 문장

개인 사용자용 인증서 발급

- 사용자는 서명 신청서를 CA에게 발송
- CA는 서명 신청서가 본인인지 확인
  - 사용자가 직접 CA를 방문하여 신청서 제출
  - 전화 등을 통해 신청서 일부의 일치 여부 확인
- CA는 서명 신청서를 바탕으로 공개키 인증서를 생성
  - CA의 개인키로 사용자의 인증서에 서명함
  - 사용자의 인증서 서명 신청서, CA의 개인키와 인증서가 필요함

CA의 사용자 인증서 생성

```
$ openssl x509 -req -days 365 -CA CACert.pem -CAkey \
CAPriv.pem -CAcreateserial -in AliceReq.pem -out
AliceCert.pem
Signature ok
subject=/C=KR/ST=CA-State/O=Hanti/CN=Alice
Getting CA Private Key
```

- -CA <file> : 서명에 사용될 개인키에 대응되는 인증서, 즉 CA의 root certificate
- -CAkey <file> : 서명에 사용될 개인키 파일
  - 위 예제에서 CA의 개인키가 사용됨
- -CAcreateserial : 일련번호 부여 파일 생성
  - 인증서 파일명에 따라 파일명 생성 : 인증서 파일 CACert.pem에 대해 CACert.srl 일련번호 파일이 생성됨
- -CAserial <file> : <file>을 일련번호 부여를 위해 사용
  - 한번씩 사용될 때 마다 값이 1씩 증가함

사용자의 인증서 검증

```
$ openssl verify -CAfile CACert.pem CACert.pem
$ openssl verify -CAfile CACert.pem AliceCert.pem
```

- CA에 등록할 때 받은 root certificate CACert.pem과 CA에게 신청하여 받은 자신의 인증서 AliceCert.pem을 검증

새로운 사용자에 대한 인증서 발급

- Bob:

```
$ openssl req -new -keyout BobPriv.pem -subj \
'C=KR/ST=CA-State/O=Hanti/CN=Bob' -out BobReq.pem \
-nodes
```

- CA:

```
$ openssl x509 -req -days 365 -CA CACert.pem \
  -CAkey CAPriv.pem -CAserial CACert.srl –in \
  BobReq.pem -out BobCert.pem
$ openssl x509 -text -noout -in BobCert.pem
```

- req command의 –nodes : 생성되는 개인키를 암호화하지 않고 저장함
- x509 command의 –text : 인증서 파일의 내용을 읽을 수 있는 형태로 출력함
- –CAserial <file> : 일련번호가 있는 <file>에서 일련번호를 읽어 사용하고 그 값을 1 증가시킴

문서를 서명하고 검증하는 예

```
Alice:
$ openssl smime -sign -in test.txt -inkey AlicePriv.pem \
-signer AliceCert.pem -out mail.msg
Bob:
$ openssl smime -verify -in mail.msg -signer AliceCert.pem \
-CAfile CACert.pem -out recover.txt
```

- Alice가 평문 mail.msg를 서명한 문서 mail.msg를 Bob에게 보내고, Bob은 이를 검증함
- -sign : 전자 서명을 생성
- -inkey <file> : 서명에 사용될 개인키 파일
- -signer <file> : 서명자의 공개키 인증서 파일, 서명자의 정보를 서명에 기입하기 위해 필요
- -CAfile <file> : root certificate filename

인증서를 이용한 메시지 서명 및 암호화

- Alice의 서명 및 암호화

```
$ openssl smime -sign -in test.txt -signer
  AliceCert.pem -inkey AlicePriv.pem -text -out
  signed.msg
$ openssl smime -encrypt -in signed.msg -out sien.msg -
  from Alice -to Bob -subject "Sign and Encryption
  test" -aes128 BobCert.pem
```

- smime command : 파일 서명 및 암호화 수행 명령어
  - -encrypt : 입력파일을 암호화 함, 받을 사람의 공개키 인증서 필요
  - -from <ID> : 메일을 보낼 사람의 ID나 email address
  - -to <ID> : 받을 사람의 ID나 email address
  - -subject <str> : 메일의 제목
  - -[cryptography algorithm] : 암호알고리즘
  - [certificate] : 받을 사람의 공개키 인증서 (전자봉투 작성에 사용)

서명 및 암호화된 메시지 처리

- Bob의 복호화 및 검증
  - `$ openssl smime -decrypt -in sien.msg -out decryp.msg -recip BobCert.pem -inkey BobPriv.pem`
  - `$ openssl smime -verify -in decryp.msg -out verif.txt -signer AliceCert.pem -CA file CACert.pem`
- 관련된 smime parameter
  - -decrypt : 복호화 작업을 수행할 것을 지시함
  - -recip <file> : 받는 사람의 공개키 인증서, 자신이 그 메시지의 수신자인지 확인하기 위해 필요
  - -inkey <file> : 개인키 파일, 암호화된 대칭키 (비밀키)의 복호화를 위해 필요

Project P4a

- 난이도 : 높음 (OpenSSL 홈페이지의 매뉴얼을 참조해야 함)
- 내용 : 작은 CA 서버와 클라이언트 구축.
  - Hint : 유닉스의 system() 함수와 OpenSSL 커멘드 라인 명령어를 이용하여 구현.
- CA 서버가 하는 일
  - 사용자 프로세스가 CA의 루트 인증서를 요청하면 그 인증서를 전달한다.
  - 사용자 프로세스가 인증서 서명 신청서를 제출하면 이를 검증하고 공개키 인증서를 생성하여 보내 준다.
  - 사용자 프로세스가 어떤 사용자의 공개키 인증서를 요구하면 그 공개키 인증서를 보내 준다. 만약 그 인증서가 없으면 없다는 답변을 보낸다.
  - 인증서를 모두 보관한다.

P4a (Cont.)

- 사용자 프로세스가 하는 일은 다음과 같다.
  - 사용자 프로세스는 사용자의 정보를 보관하고 있으며 사용자의 키보드 입력에 따라 위의 작업을 수행한다.
  - 자신의 공개키 인증서가 없으면 자신의 개인키와 인증서 서명 신청서를 작성하여 CA에게 신청한다.
  - 자신의 또는 다른 사용자의 공개키 인증서를 신청하여 받으면 다음 과정을 통하여 검증한다.
    - 루트 인증서가 없으면 CA에게 요청하여 받는다.
    - 루트 인증서를 검증한다.
    - 상대방의 공개키 인증서를 검증한다.
  - Hint : OpenSSL 커멘드 라인에서 ca 명령어는 CA 서버의 인증서 생성과 리스트 관리를 도와주므로 CA 서버 관리를 보다 용이하게 하는 명령어임

Project P4b

- 난이도 : 매우 높음 (OpenSSL 홈페이지의 매뉴얼을 참조해야 함)
- P4a의 기능에 다음을 추가하시오.
  - CA 서버의 경우:
    - 사용자 프로세스가 자신의 인증서 유효기간보다 빨리 새로운 인증서 서명 신청서를 제출하면 그 사용자의 인증서를 CRL에 넣고 새로운 인증서를 생성하여 보내 준다.
    - 사용자 프로세스가 CRL을 요구하면 이를 보내 준다.
    - 유효한 인증서와 폐기된 인증서를 모두 보관한다.
  - 사용자 프로세스의 경우:
    - P4a에서 3단계의 검증과정 중 다음을 포함한다.
      - CRL을 요청하여 받은 후 공개키 인증서가 CRL에 포함되어 있는지 확인한다.