2026/08/05, 2026 보안 기초 세미나

암호학과 네트워크 보안

-15장 키 관리-

송 민 희(minhee@pel.sejong.ac.kr)

세종대학교 프로토콜공학연구실

목 차

- 개요
- 대칭 키 분배
- 커버로스(Kerberos)
- 대칭 키 합의
- 공개 키 배분

목 차

- 개요
- 대칭 키 분배
- 커버로스(Kerberos)
- 대칭 키 합의
- 공개 키 배분

개요

- 키 관리
  - 암호 통신에 쓰이는 키를 안전하게 생성·교환·저장 및 관리하는 프로세스
- 대칭 키 암호 시스템에서의 키 관리 (1/2)
  - 필요성 (1/2)
    - N^2 문제
      - Alice 한 명이 N명과 각각 1:1로 비밀 통신을 할 경우 N개의 키 필요
      - N명이 집단에 속한 모든 사람들과 1:1로 비밀 통신을 할 경우 두 사람 쌍 마다 키 필요
        - 방향마다 키를 따로 쓸 경우 $N(N-1)$개 필요
        - 한 쌍이 키 하나를 공유할 시 $\frac{N(N-1)}{2}$개 필요
      - 필요한 키의 개수가 N^2에 비례하여 증가하므로 키 관리 문제 발생
        - e.g., $N = 10$이면 90(45)개의 키 필요, $N = 100$이면 9900(4950)개 필요

개요

- 대칭 키 암호 시스템에서의 키 관리 (2/2)
  - 필요성 (2/2)
    - 안전한 키 배분 문제
      - N명의 통신 상대와 서로 안전하게 키 공유할 수단 부재
  - 방법
    - 대칭 키 분배
      - 제 3자가 세션 키를 생성하고 분배하는 방식
      - 모두가 신뢰할 수 있는 제 3자가 존재하는 경우에 사용 가능
      - 제 3자에게 당사자가 키를 미리 공유하고 있다고 전제
    - 대칭 키 합의
      - 두 당사자가 즉석에서 세션 키를 생성하는 방식
      - 사전 등록 없이 사용 가능

*세션 키(Session Key)
하나의 통신 세션에서 메시지를 암호화하기 위해 사용되는 1회용 대칭 키

개요

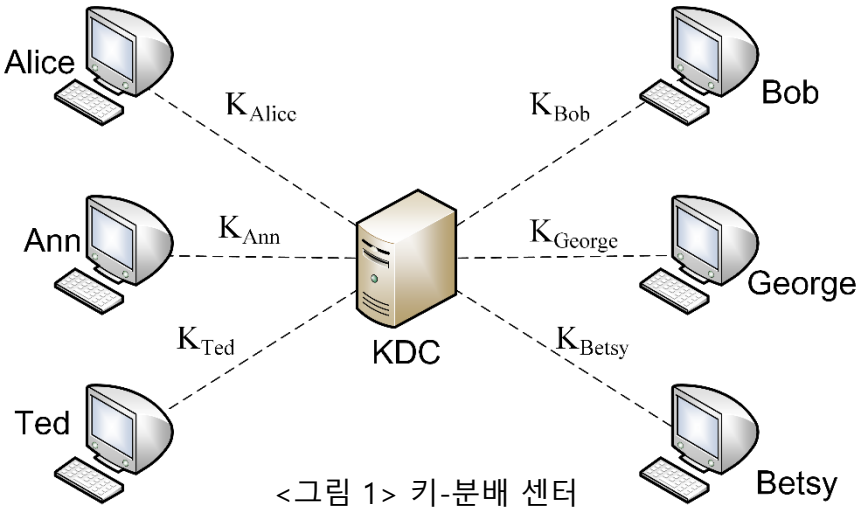
- 공개 키 암호 시스템에서의 키 관리
  - 필요성
    - 위조(진위성) 문제
      - 공개 키는 공개 되어있는 값이므로 유출과 무관
      - 공격자가 가짜 공개 키를 타인의 것으로 위장/교체 가능
      - 특정 공개 키가 실제 소유자의 것임을 보증하는 과정 필요
    - 대규모 신뢰 분산 문제
      - 다수 사용자에게 진위 신뢰를 확장·전달할 체계 필요
      - 단일 기관이 전 세계 모든 사용자 감당 불가능
  - 방법
    - 공개 키 배분 (인증)
      - 공개 키를 프로토콜이나 신뢰받는 제 3자를 통해 인증된 형태로 배포하는 방식

목 차

- 개요
- 대칭 키 분배
- 커버로스(Kerberos)
- 대칭 키 합의
- 공개 키 배분

대칭 키 분배

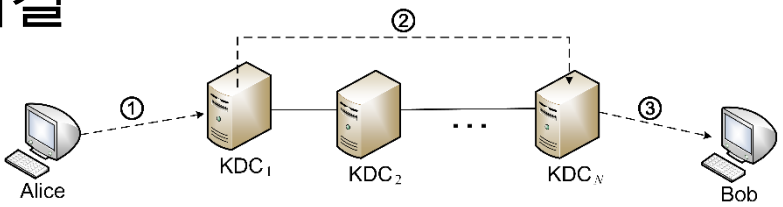
- 키-분배 센터(KDC, Key-Distribution Center) (1/2)
  - 개요
    - 키를 안전하고 효율적으로 분배할 수 있는 신뢰받는 제 3자
    - KDC와 개인이 한 개씩의 비밀 키를 공유
    - 비밀 키를 통해 사용자 인증 후 세션 키를 전달
  - 동작
    1. Alice는 자신이 Bob과 통신하기 위해 세션 키가 필요하다고 KDC에게 요청
    2. KDC는 Alice의 요청을 Bob에게 전송
    3. Bob이 동의 할 경우 두 사람 사이의 세션 키 생성 및 전달



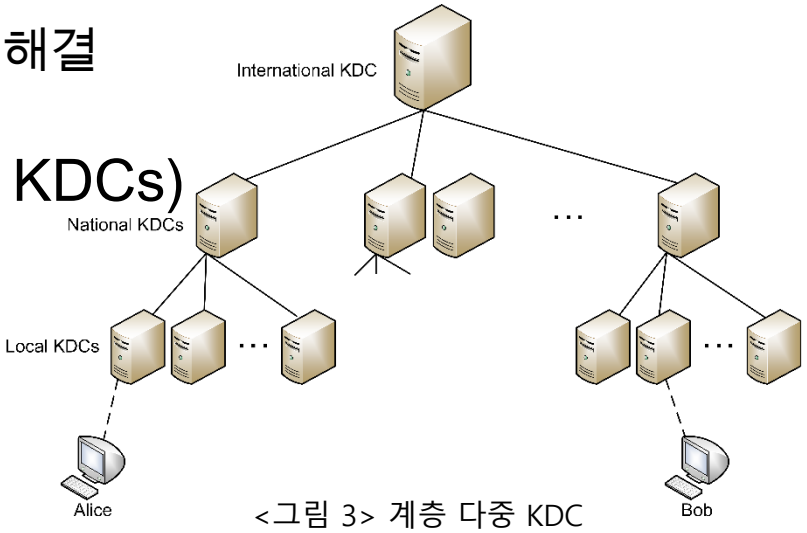
<그림 1> 키-분배 센터

대칭 키 분배

- 키-분배 센터(KDC, Key-Distribution Center) (2/2)
  - 다중 KDC(Multiple KDC)
    - 개요
      - KDC를 이용하는 사람의 수가 증가할 시 병목현상 발생 가능
      - 집단을 여러 개의 도메인으로 나눠 해결
    - 분류
      - 평등 다중 KDC(Flat Multiple KDCs)
        - 여러 개의 KDC를 동일 계층으로 나란히 배치한 구조
        - 사용자 증가로 인한 관리 부담과 병목을 해결
        - 모든 KDC에 대해 연결 필요
      - 계층 다중 KDC(Hierarchical Multiple KDCs)
        - 평등 다중 KDC를 확장하여 KDC를 상하 계층으로 배치한 구조
        - KDC별 관리 범위 명확화 및 신뢰 관계 단순화 가능
        - 경우에 따라 통신 경로 길이 증가



<그림 2> 평등 다중 KDC



<그림 3> 계층 다중 KDC

대칭 키 분배

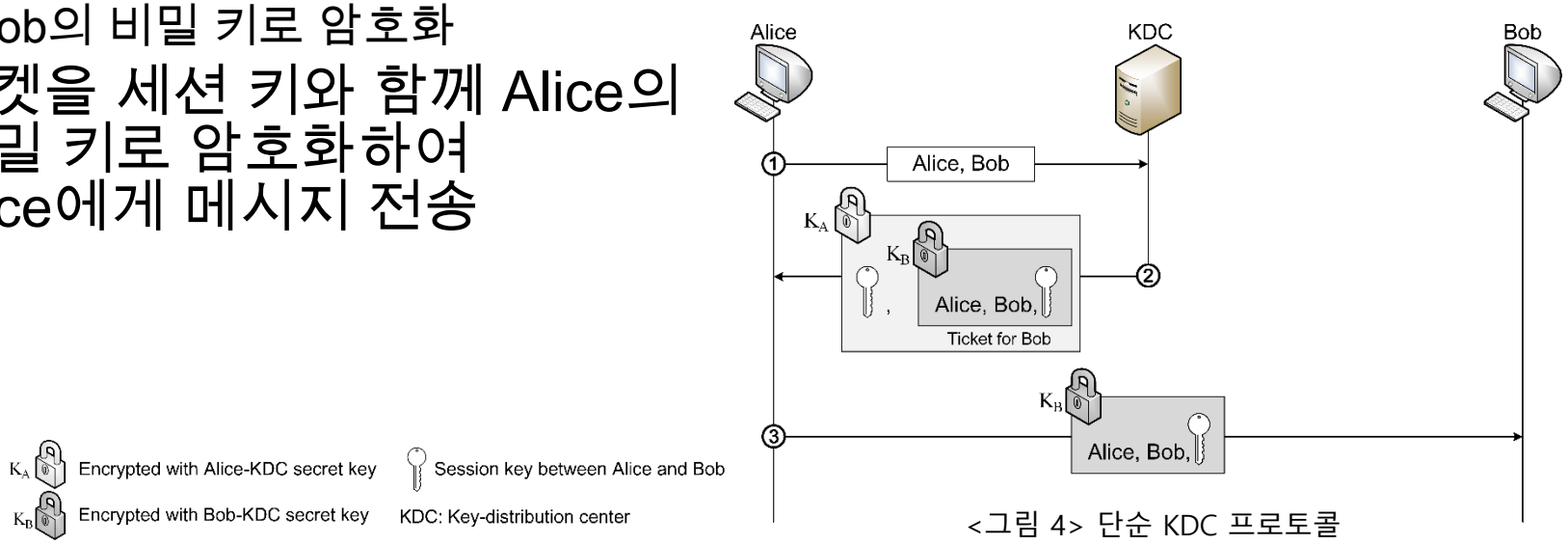
- 키 종류
  - 비밀 키
    - 구성원과 KDC 사이의 장기 키
    - KDC에 구성원임을 인증하고 세션 키를 전달하는데 사용
      - 구성원 간의 통신에서는 사용 불가
  - 세션 키
    - 두 구성원 간의 통신을 위한 1회용 키
      - 통신 종료 후 폐기
    - 실제 통신을 암호화 하는데 사용
- 프로토콜 종류
  - 단순 KDC 프로토콜
  - Needham-Schroeder 프로토콜
  - Otway-Rees 프로토콜

대칭 키 분배

- 단순 KDC 프로토콜 (1/3)
  - 개요
    - 가장 기본적인 형태의 대칭 키 분배 프로토콜
    - 1개의 KDC만을 사용
  - 한계
    - 재전송 공격에 취약
      - Alice가 티켓을 Bob에게 보내는 과정에서 Eve가 메시지를 저장한 후 나중에 재전송 가능

대칭 키 분배

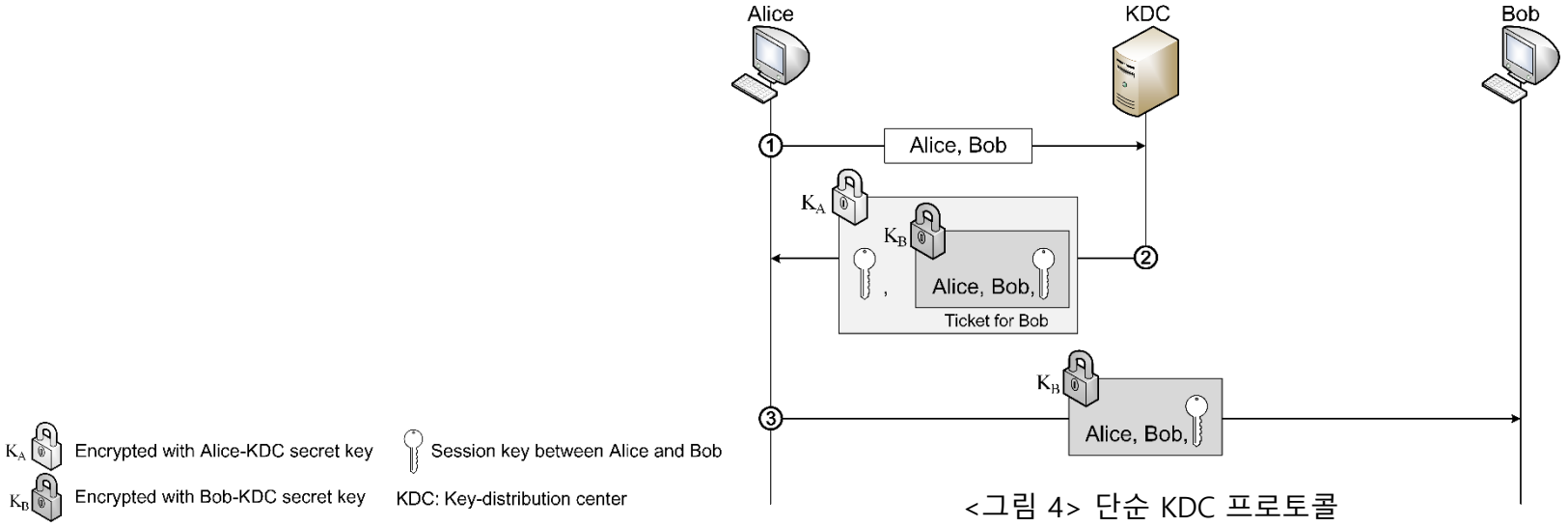
- 단순 KDC 프로토콜 (2/3)
  - 동작 (1/2)
    1. Alice가 Bob과의 세션 키를 KDC에 요청하기 위한 평문 메시지 전송
       - KDC에 등록된 자신의 ID와 Bob의 ID를 메시지로 작성
    2. Alice에게 암호화된 메시지 전송
       1) KDC는 메시지 수신 후 티켓(Ticket)을 생성
          - 티켓에 두 명의 ID와 세션 키 포함
          - Bob의 비밀 키로 암호화
       2) 티켓을 세션 키와 함께 Alice의 비밀 키로 암호화하여 Alice에게 메시지 전송



<그림 4> 단순 KDC 프로토콜

대칭 키 분배

- 단순 KDC 프로토콜 (3/3)
  - 동작 (2/2)
    3. Alice는 메시지를 복호화하고 티켓을 Bob에게 전송
       1) Alice가 자신의 비밀 키로 메시지를 복호화
          - 세션 키 획득
          - KDC에 자신을 인증
       2) Bob이 자신의 비밀 키로 메시지를 복호화
          - Alice가 해당 세션 키로 자신에게 메시지를 보내려고 함을 파악
          - KDC에 자신을 인증



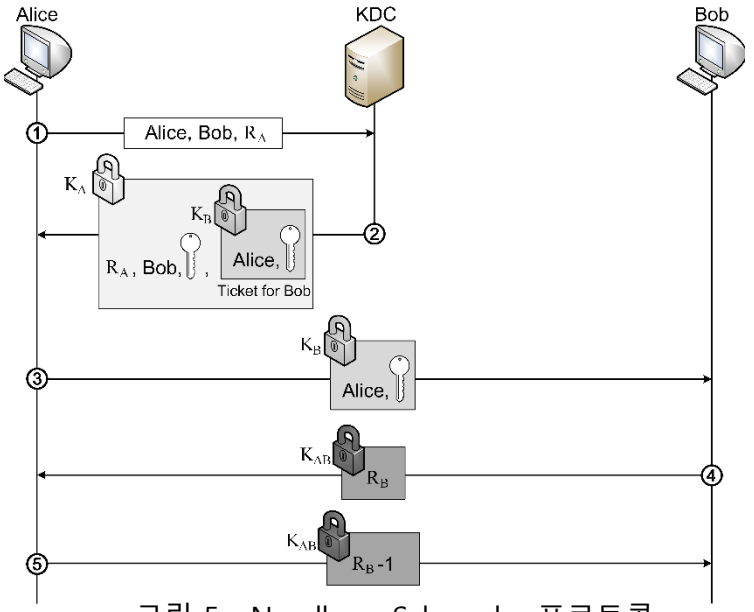
<그림 4> 단순 KDC 프로토콜

대칭 키 분배

- Needham-Schroeder 프로토콜 (1/3)
  - 개요
    - 비표(Nonce)를 이용한 다중 시도-응답으로 재전송 약점 보완
    - 다른 프로토콜의 기초로 사용
      - e.g., 커버로스(Kerberos), Otway-Rees 프로토콜 등
  - 한계
    - 세션 키 재사용 공격(Denning-Sacco 공격)에 취약
      - 티켓에 시간 정보(타임스탬프) 미포함
      - 공격자가 과거에 사용한 세션 키를 알아낼 경우 해당 세션 키가 포함 된 과거의 티켓을 재사용 가능
        - Bob이 해당 티켓이 발급된 시점을 알 수 없으므로 티켓을 수용

대칭 키 분배

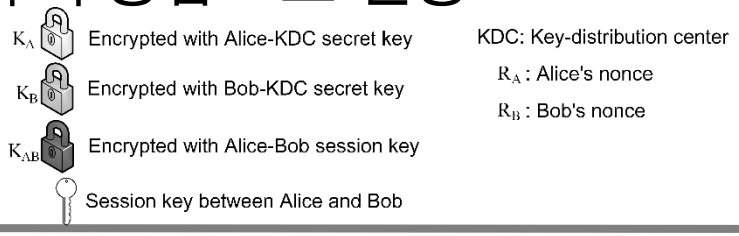
- Needham-Schroeder 프로토콜 (2/3)
  - 동작 (1/2)
    1. Alice는 평문 메시지를 KDC에 전송
       - 자신의 비표, 자신의 ID, Bob의 ID를 메시지에 포함
    2. KDC는 암호화된 메시지를 Alice에 전송
       1) KDC는 메시지 수신 후 티켓(Ticket)을 생성
          - 티켓에 Alice의 ID와 세션 키를 포함
          - Bob의 비밀 키로 암호화
       2) Alice에게 암호화 된 메시지 전송
          - Alice의 비표, Bob의 ID, 세션 키, 티켓을 메시지에 포함
          - Alice의 비밀 키로 암호화하여 전송

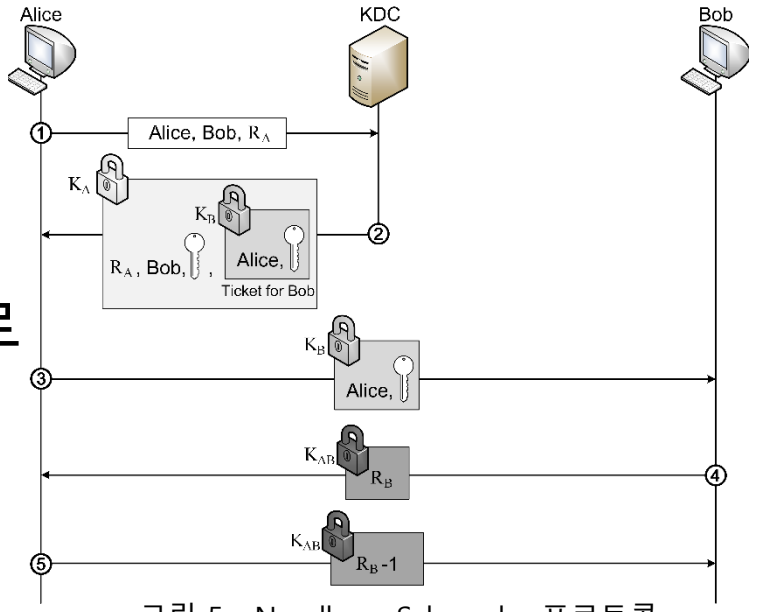


<그림 5> Needham-Schroeder 프로토콜

대칭 키 분배

- Needham-Schroeder 프로토콜 (3/3)
  - 동작 (2/2)
    3. Alice는 티켓을 Bob에게 전송
       - 메시지를 자신의 비밀 키로 복호화
       - 복호화를 통해 얻은 Bob의 티켓을 전송
    4. Bob은 자신의 시도를 Alice에게 전송
       1) 메시지를 자신의 비밀 키로 복호화하여 Alice와 세션 키 확인
       2) 자신의 비표를 세션 키로 암호화하여 시도 생성 후 전송
    5. Alice는 Bob의 시도에 응답
       1) 시도를 세션 키로 복호화
       2) Bob의 비표에서 -1 한 값을 세션 키로 암호화하여 응답으로 전송





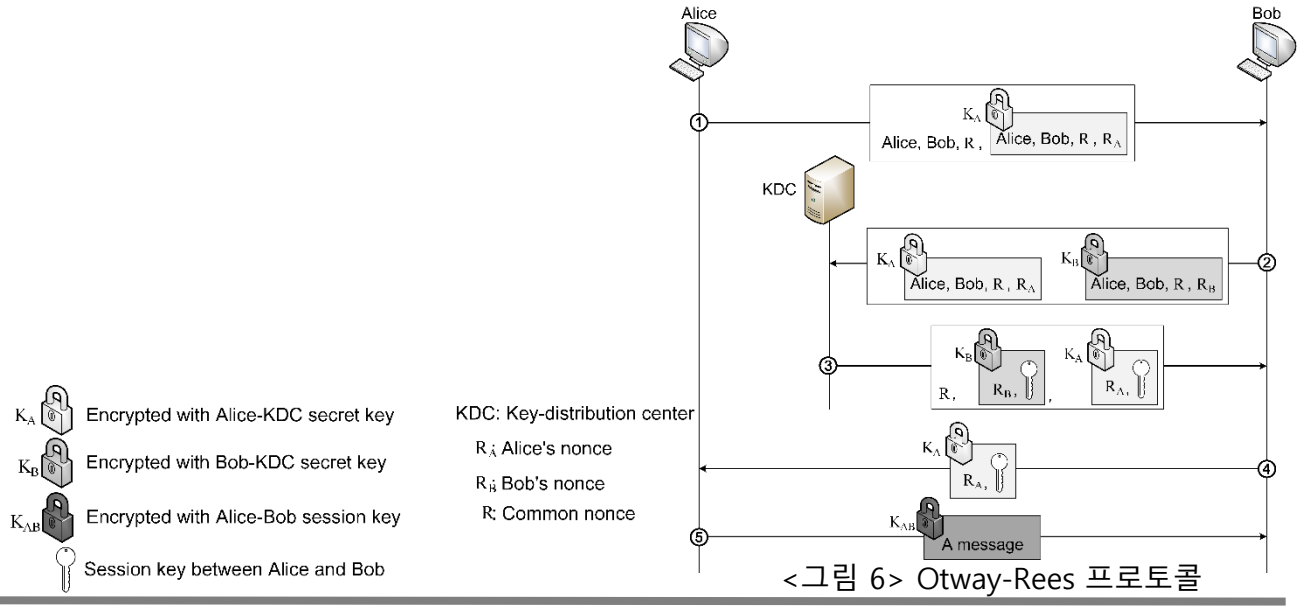
<그림 5> Needham-Schroeder 프로토콜

대칭 키 분배

- Otway-Rees 프로토콜 (1/4)
  - 개요
    - Bob이 KDC에 요청을 중계
    - 공통 비표를 사용하여 세션 키 재사용 방지
      - 세션마다 새롭게 공통 비표 생성
  - 한계
    - 인증 관련 취약점 존재
      - Eve의 메시지 조작으로 Alice와 Bob이 서로 다른, 또는 잘못된 키를 신뢰
      - 상대가 올바른 키를 수신했는지에 대한 여부를 확인하는 과정 부족

대칭 키 분배

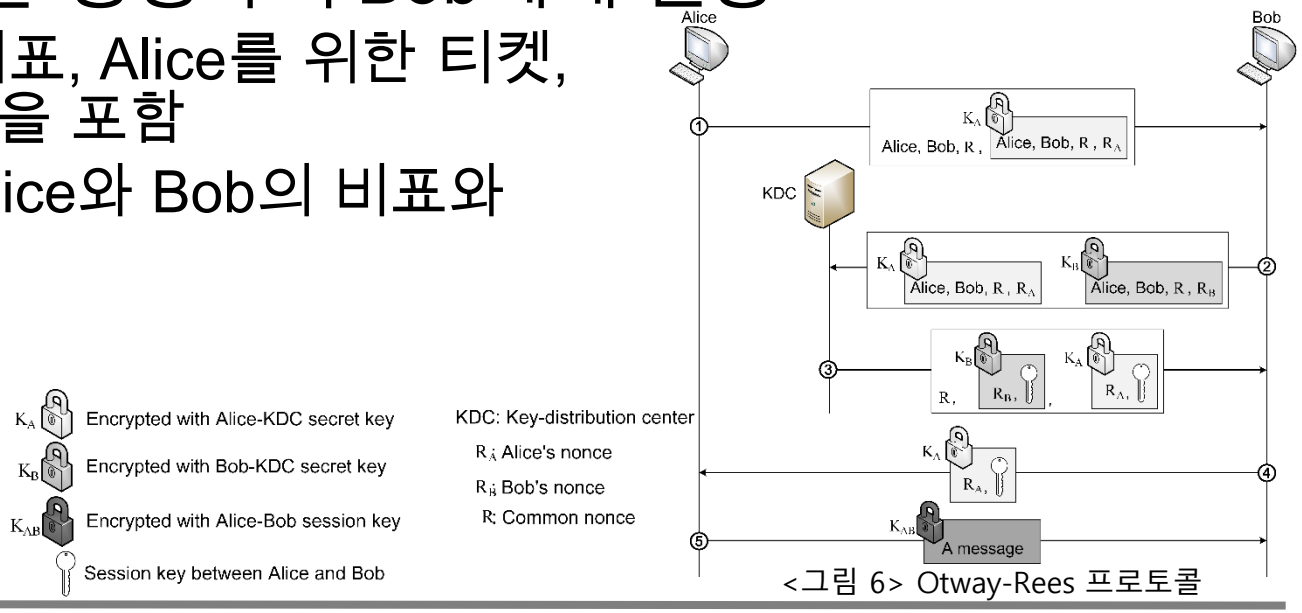
- Otway-Rees 프로토콜 (2/4)
  - 동작 (1/3)
    1. Alice는 메시지를 Bob에게 전송
       1) KDC용 티켓 생성
          - KDC용 티켓에 자신의 ID, Bob의 ID, 공통 비표, 자신의 비표를 포함
          - 자신의 비밀 키로 암호화
       2) KDC용 티켓을 포함한 메시지 생성 후 Bob에게 전송
          - 자신의 ID, Bob의 ID, 공통 비표와 티켓을 메시지에 포함



<그림 6> Otway-Rees 프로토콜

대칭 키 분배

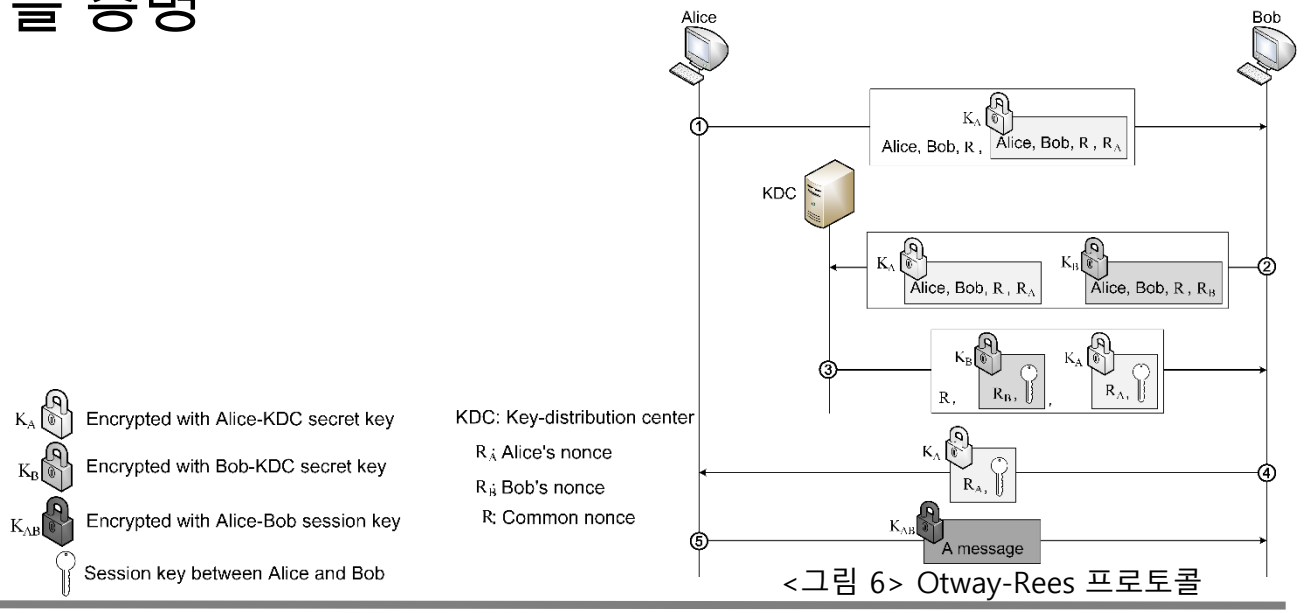
- Otway-Rees 프로토콜 (3/4)
  - 동작 (2/3)
    2. Bob도 동일한 티켓을 생성하여 KDC에 전송
       1) KDC용 티켓 생성
          - Alice의 ID, 자신의 ID, 공통 비표, 자신의 비표를 포함
          - 자신의 비밀 키로 암호화
       2) Alice에게 받은 티켓과 자신이 생성한 티켓을 함께 KDC에 전송
    3. KDC는 메시지를 생성하여 Bob에게 전송
       - 메시지에 공통 비표, Alice를 위한 티켓, Bob을 위한 티켓을 포함
       - 티켓에는 각각 Alice와 Bob의 비표와 세션 키를 포함



<그림 6> Otway-Rees 프로토콜

대칭 키 분배

- Otway-Rees 프로토콜 (4/4)
  - 동작 (3/3)
    4. Bob은 Alice에게 Alice의 티켓을 전송
       - 자신의 비밀 키를 통해 자신의 티켓을 복호화
       - 남은 Alice의 티켓을 전송
    5. Alice는 세션 키로 암호화된 짧은 메시지를 Bob에게 전송
       - 자신의 비밀 키를 통해 티켓을 복호화
       - 세션 키를 획득함을 증명



<그림 6> Otway-Rees 프로토콜

목 차

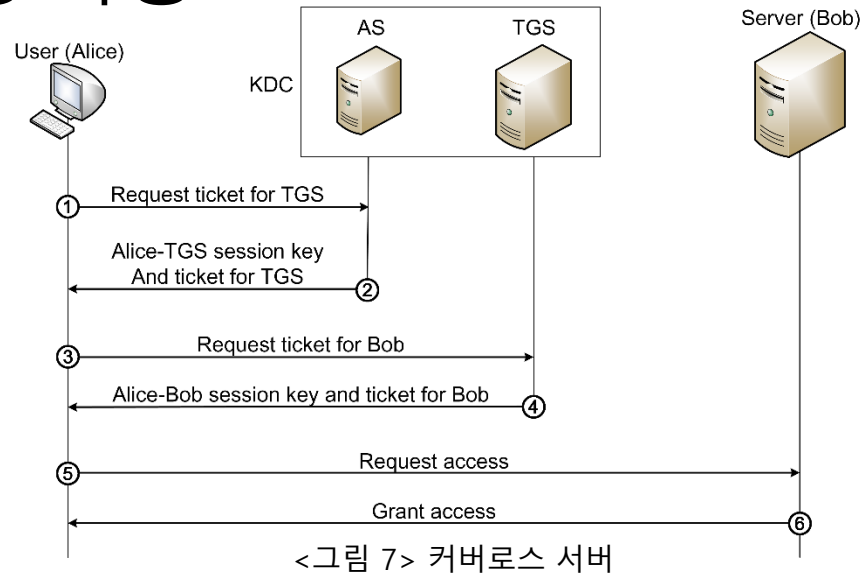
- 개요
- 대칭 키 분배
- 커버로스(Kerberos)
- 대칭 키 합의
- 공개 키 배분

커버로스(Kerberos)

- 정의
  - 티켓 기반 컴퓨터 네트워크 인증 암호화 프로토콜
- 특징
  - 프로토콜이면서 동시에 KDC의 역할
  - 타임스탬프로 재전송 방지
  - 최초 1회 인증으로 여러 서버 이용 가능
- 구성 (1/2)
  - 실질 서버
    - 사용자(Alice)에게 서비스 제공
    - 클라이언트-서버 프로그램용
      - 사용자가 클라이언트 프로세스를 이용하여 서버 프로세스에 접근



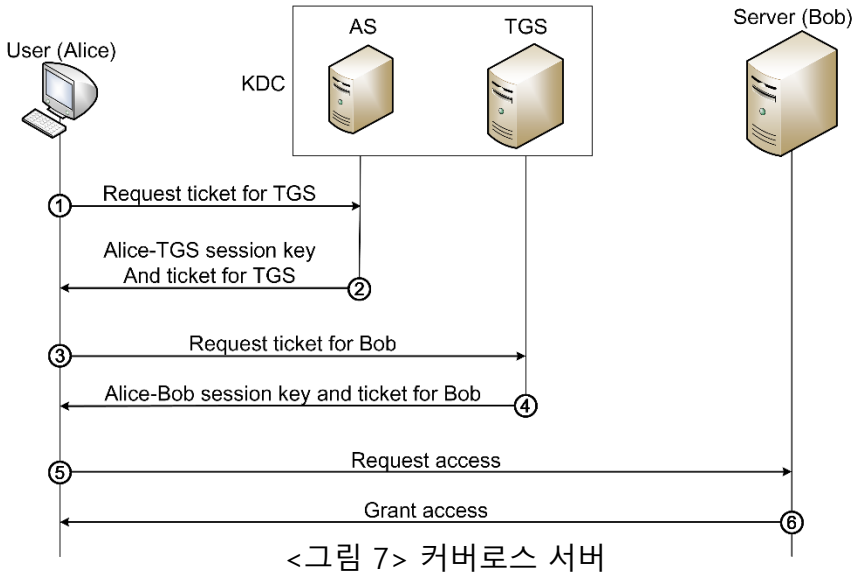
커버로스 서버의
어원인 케르베로스



<그림 7> 커버로스 서버

커버로스(Kerberos)

- 구성 (2/2)
  - 인증 서버(AS, Authentication Server)
    - 사용자 ID와 패스워드를 통해 검증
      - 각 사용자는 AS에 등록 후 사용자 ID와 패스워드 발급
      - 사용자 ID와 대응되는 패스워드에 대한 데이터베이스 소유
  - 티켓-발급 서버(TGS, Ticket-granting Server)
    - 실질 서버(Bob)에 티켓 발급
      - Alice와 Bob 사이에 사용할 세션 키 제공
      - 티켓 발급 시 AS를 통해 인증된 사용자인지 독립적으로 검증 수행

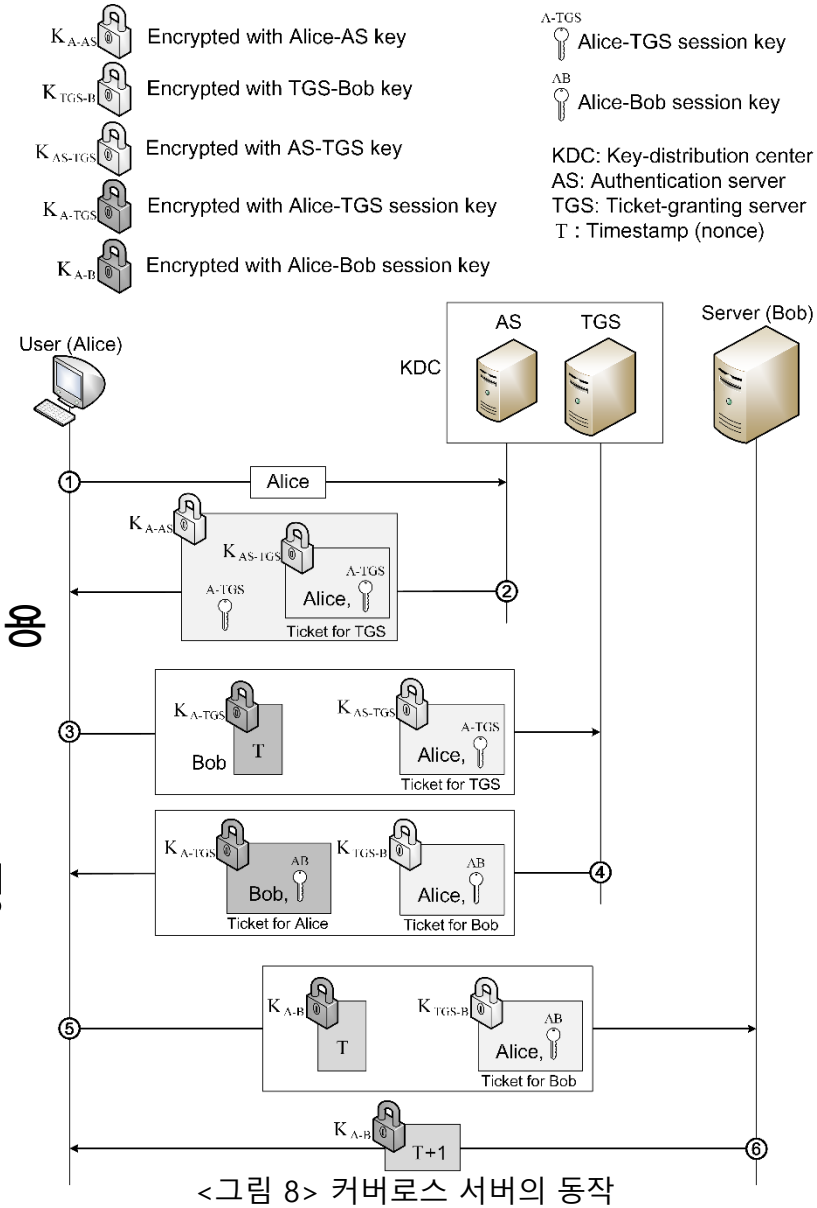


<그림 7> 커버로스 서버

커버로스(Kerberos)

- 동작 (1/3)

1. Alice는 자신의 등록된 ID를 이용하여 평문으로 AS에 자신의 요청을 전송
2. AS는 암호화한 메시지를 Alice에게 전송
   1) Alice와의 비밀 키 생성
      - DB에 저장되어있는 Alice의 패스워드 활용
   2) TGS용 티켓 생성
      - Alice의 ID, Alice-TGS 세션 키 포함
      - TGS와의 비밀 키로 암호화
   3) TGS용 티켓을 포함한 메시지 생성
      - Alice의 TGS 세션 키와 티켓을 포함
      - Alice와의 비밀 키로 암호화

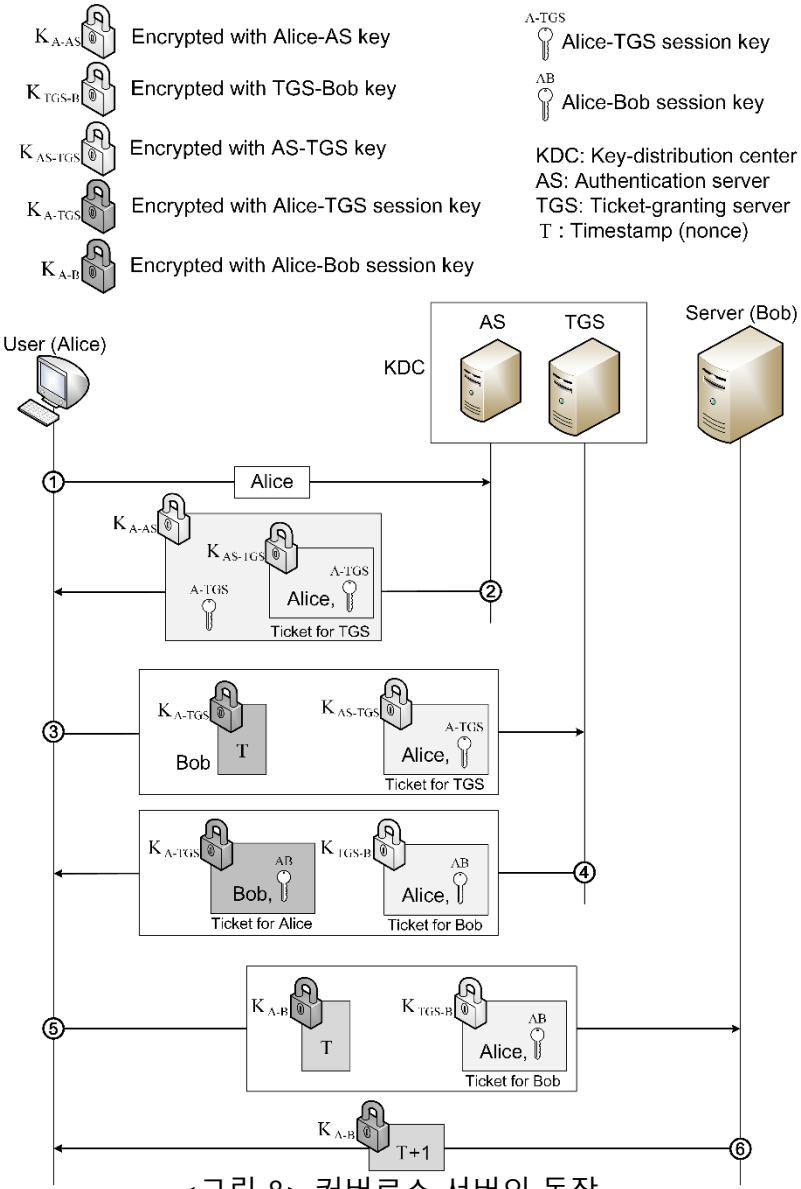


<그림 8> 커버로스 서버의 동작

커버로스(Kerberos)

- 동작 (2/3)

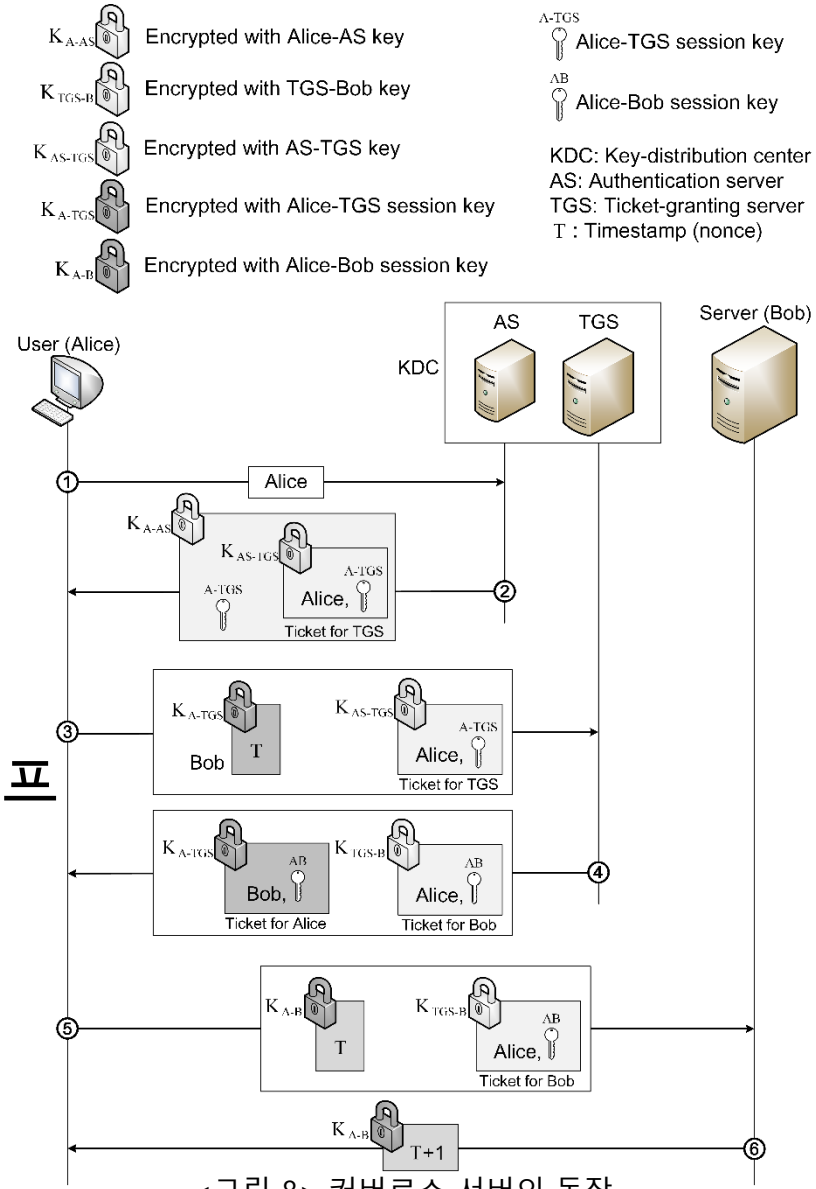
3. Alice는 TGS에 메시지 전송
   1) 자신의 비밀 키 생성하여 복호화
      - 메시지를 받은 후 대칭 패스워드 입력
      - 입력된 패스워드가 올바를 시 적합한 알고리즘으로 대칭 키 생성
        - 생성 후 입력 값 파기
   2) TGS에 메시지 전송
      - AS에서 받은 TGS용 티켓
      - 실질 서버의 이름 (Bob)
      - TGS와의 비밀 키로 암호화 된 타임스탬프
        - Eve의 재전송 공격 방지



<그림 8> 커버로스 서버의 동작

커버로스(Kerberos)

- 동작 (3/3)
  4. TGS는 2개의 티켓 전송
     - 각각의 티켓은 Alice와 Bob 사이의 세션 키 포함
       - Alice를 위한 티켓
         - Alice와 TGS 사이의 세션 키로 암호화
       - Bob을 위한 티켓
         - Bob와의 대칭 키로 암호화
     - Eve가 티켓을 암호화하고 있는 키 확보 불가능
       - 세션 키를 훔칠 수 없으므로 타임스탬프 수정을 통한 재전송 공격 불가능
  5. Alice는 Bob용 티켓과 암호화한 타임스탬프를 Bob에게 전송
  6. Bob은 수신된 타임스탬프에 1을 더해서 수신 확인



<그림 8> 커버로스 서버의 동작

커버로스(Kerberos)

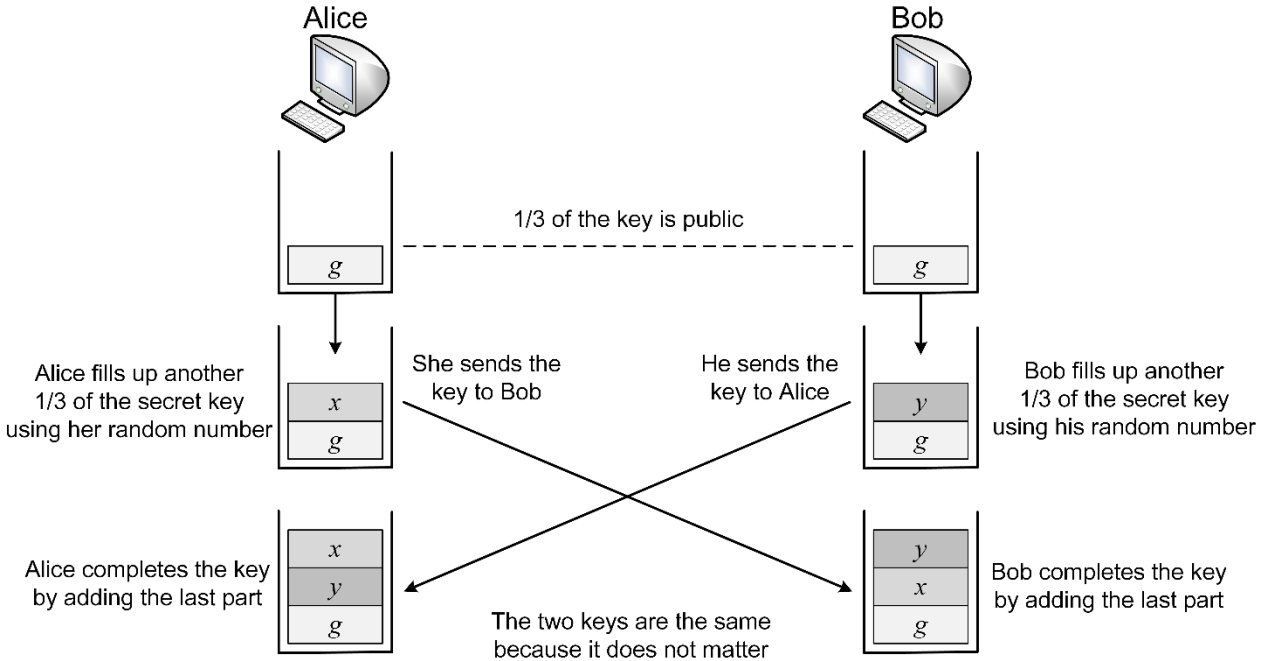
- 공동체(Realm)
  - 개요
    - 커버로스는 AS와 TGS가 전 세계에 분산된 환경으로 구성
    - 하나의 커버로스(AS, TGS)가 관리하는 독립된 영역
    - 전체 집단 내 각 시스템 단위
    - 각 공동체는 자체 KDC(AS·TGS)로 자기 영역의 사용자와 서버 관리
  - 다른 realm의 서버 이용
    - 지역 TGS가 원격 realm의 TGS에 티켓 요청
      - 단, 원격 TGS가 지역 TGS에 등록 되어 있는 경우만 가능
    - 등록 시 지역 TGS가 원격 TGS용 티켓을 발급하여 원격 실질 서버 접속 가능

목 차

- 개요
- 대칭 키 분배
- 커버로스(Kerberos)
- 대칭 키 합의
- 공개 키 배분

대칭 키 합의

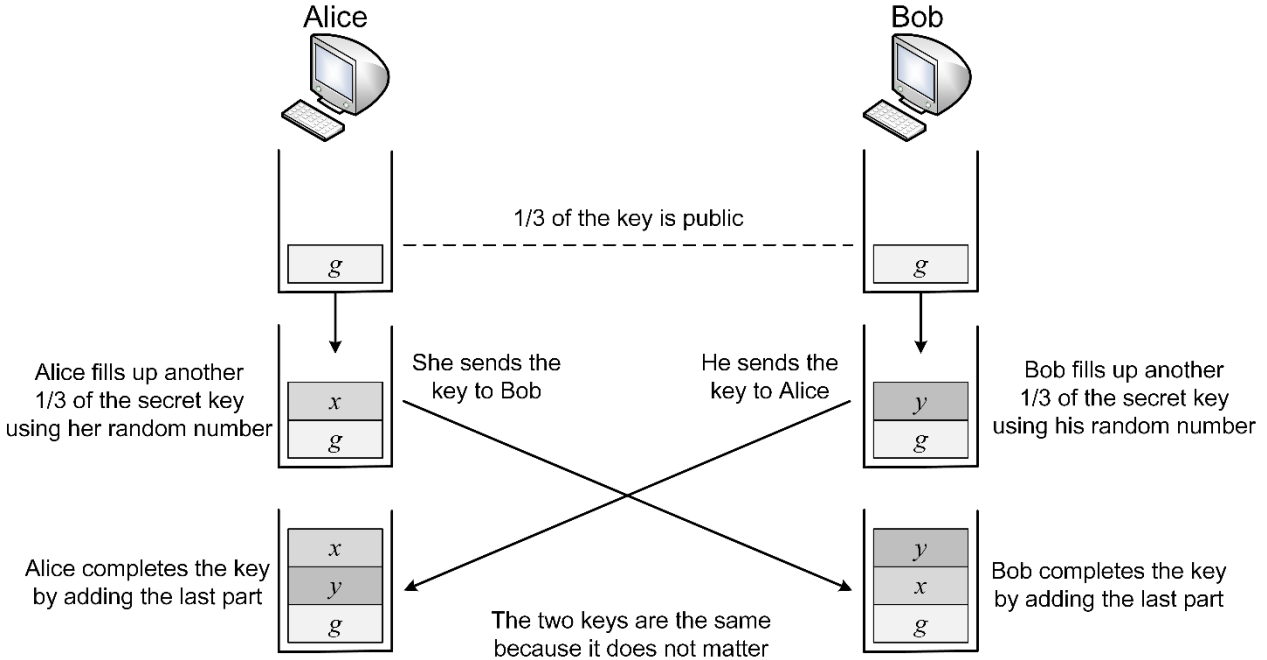
- Diffie-Hellman 프로토콜
  - 동작 원리 (1/2)
    - 비밀 키 K는 g, x, y 세 부분으로 구성
      - g: 공개 되는 부분
      - x, y: Alice와 Bob이 각각 채우는 비밀 수
    - 상대의 비밀 수를 수신해 마지막 1/3을 채워 키 완성



<그림 9> Diffie-Hellman 프로토콜의 동작 원리

대칭 키 합의

- Diffie-Hellman 프로토콜
  - 동작 원리 (2/2)
    - 채우는 순서와 무관하게 두 키는 동일
      - $g^{xy} = g^{yx}$
    - 상대의 비밀 수는 파악 불가
      - $mod\ p$연산으로 g^x만 전달되고 이산대수 문제로 x는 역산 불가



<그림 9> Diffie-Hellman 프로토콜의 동작 원리

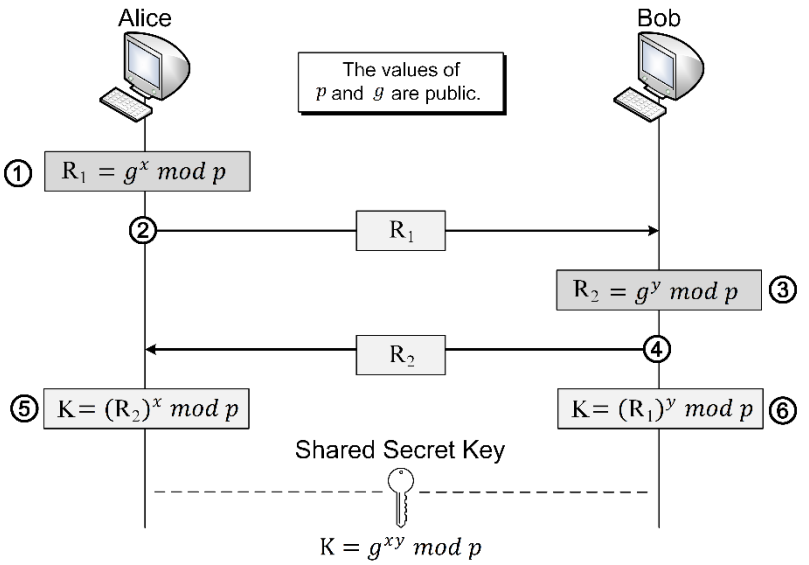
대칭 키 합의

- Diffie-Hellman 프로토콜
  - 특징
    - 양쪽 통신 주체가 계산한 K를 세션 대칭 키로 사용
      - $K = (g^x \ mod \ p)^y \ mod \ p \ = \ (g^y \ mod \ p)^x \ mod \ p \ = \ g^{xy} \ mod \ p$
    - 두 사람이 각각 x와 y 중 하나만 알아도 모두 동일한 K 생성 가능
  - 준비 과정
    - 대칭 키 생성 전 양 측은 두 개의 수 p와 g 선택
      - p: 300자리가 넘는 10진수의 매우 큰 소수
      - g: 군 $< Z_p^*, \times >$의 원소로 위수가 $p-1$인 생성자
      - p와 g는 공개 되어도 무방

*군 $< Z_p^*, \times >$
1부터 $p-1$까지의 수를 $mod \ p$ 곱셈으로 묶은 군
*군의 위수
원소의 개수 $(= p-1)$
*원소의 위수
해당 원소를 거듭제곱 해서 1이 될 때까지의 최소 횟수

대칭 키 합의

- Diffie-Hellman 프로토콜
  - 동작
    1. Alice는 임의의 큰 수 x를 $0 \leq x \leq p-1$ 안에서 선택하고 $R_1 = g^x \ mod \ p$를 계산
    2. Bob은 다른 임의의 큰 수 y를 $0 \leq y \leq p-1$ 안에서 선택하고 $R_2 = g^y \ mod \ p$를 계산
    3. Alice는 R_1을 Bob에게 전송
    4. Bob은 R_2를 Alice에게 전송
    5. Alice는 $K = (R_2)^x \ mod \ p$를 계산
    6. Bob은 $K = (R_1)^y \ mod \ p$를 계산



<그림 10> Diffie-Hellman 프로토콜의 동작

대칭 키 합의

- Diffie-Hellman 프로토콜
  - 예제 15.1

Diffie-Hellman 프로토콜을 적용할 때는 매우 큰 수를 이용해야 한다.
그러나 이해를 위해 작은 수를 적용해보자.
$g = 7$이고, $p = 23$이라고 할 때 Diffie-Hellman 절차를 적용하고 K를 구하라.

1. Alice는 $x = 3$을 선택했다고 가정
   - $R_1 = 7^3 \ mod \ 23 = 21$
2. Bob은 $y = 6$을 선택했다고 가정
   - $R_2 = 7^6 \ mod \ 23 = 4$
3. Alice는 21을 Bob에게, Bob은 4를 Alice에게 전송
4. Alice는 $K = 4^3 \ mod \ 23 = 18$을 계산
5. Bob은 $K = 21^6 \ mod \ 23 \ = \ 18$을 계산

6. $g^{xy} \ mod \ p \ = \ 7^{18} \ mod \ 23 \ = \ 18$

$\therefore K = 18$

대칭 키 합의

- Diffie-Hellman 프로토콜
  - 예제 15.2

실제와 가까운 예를 들기 위해 프로그램을 이용하여 512비트의 난수를 생성한다. 154자리 수 p를 포함하여 g, x, y가 [표 1]과 같을 때 R_1, R_2, K를 나타내라.

- 프로그램 코드는 부록#2에 첨부
- 프로그램 동작 결과는 <그림 11> 과 같음

[표 1] Diffie-Hellman 프로토콜 예제

p	76462429856349357218249376595 50305074763380967269497489235 73772860925235666660755423637 42330966118003333810619473013 09504147387009991780436548785 807987581
g	2
x	557
y	273

```
g = 2, x = 557, y = 273

R1 =    584492028420566550521617294749103509414343369852001266086286363106767361995928082858670080213185929094514021750031997331294583608382194306596602015795535 4

R2 =    151771007205135083665582961470587414581438034300948400097797844510851897281656913 92

K  =    344504476795043415881721219424755310311189256780610529919325172155950715202998789367793701400944331954497785217989768767762303304298373409902790732284617 7

Alice == Bob : True
```

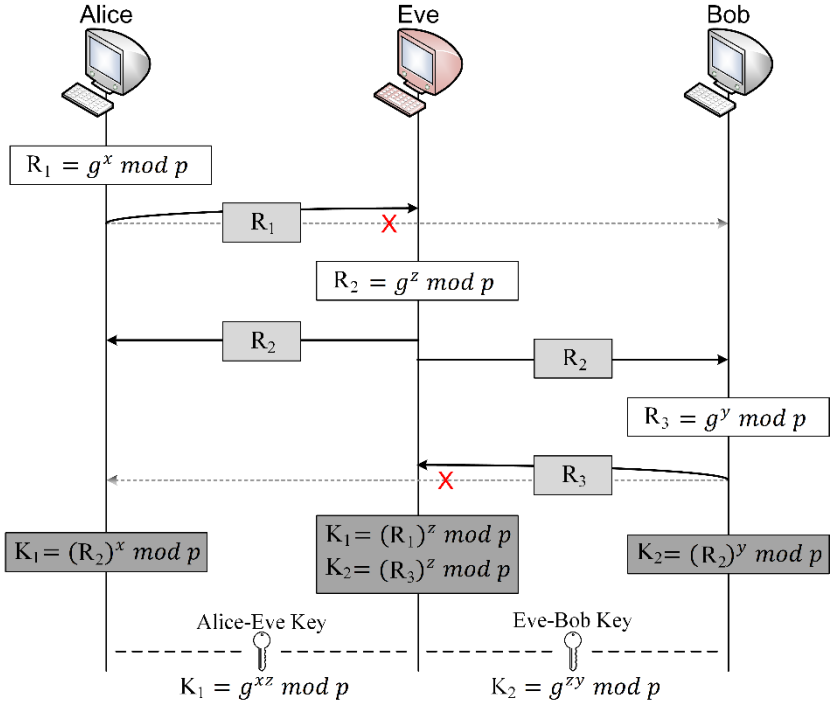
<그림 11> 프로그램 동작 결과

대칭 키 합의

- Diffie-Hellman 프로토콜
  - 안전성 (1/3)
    - 이산 대수 공격(Discrete Logarithm Problem)
      - 개요
        - Eve가 R_1과 R_2를 가로채 비밀 키를 계산하여 획득하는 공격
      - 공격 절차
        1. Eve가 R_1과 R_2를 획득
        2. $R_1 = g^x \ mod \ p$로부터 x, $R_2 = g^y \ mod \ p$로부터 y를 계산
        3. 획득한 x와 y로 $K = g^{xy} \ mod \ p$를 계산하여 비밀 키를 획득
      - 필수 조치
        - 소수 p는 매우 큰 수 (10진수로 300자리 이상)
        - 소수 p를 선택할 때 $p-1$이 적어도 하나의 큰 소인수(10진수로 60자리 이상)
        - 생성자 g는 군 $< Z_p^*, \times >$에서 선택
        - Bob과 Alice는 대칭 키 생성 이후 x와 y를 각자 폐기
        - x와 y의 값은 한 번만 사용

대칭 키 합의

- Diffie-Hellman 프로토콜
  - 안전성 (2/3)
    - 중간자 공격(MITM, Man-in-the-middle attack) (1/2)
      - 개요
        - Eve가 임의의 두 개의 키를 만들어 Alice와 Bob에게 전송
        - 양동이 소방대 공격(Bucket Brigade Attack)이라고도 호칭
      - 공격 절차 (1/2)
        1. Alice는 x를 선택한 다음 $R_1 = g^x \bmod p$를 계산하여 Bob에게 전송
        2. Eve는 중간에서 R_1을 획득하고 z를 선택한 다음 $R_2 = g^z \bmod p$를 계산하여 Alice와 Bob에게 전송
        3. Bob은 y를 선택한 다음 $R_3 = g^y \bmod p$를 계산하여 Alice에게 전송 중 Eve가 R_3를 가로채 Alice는 R_3 획득 불가

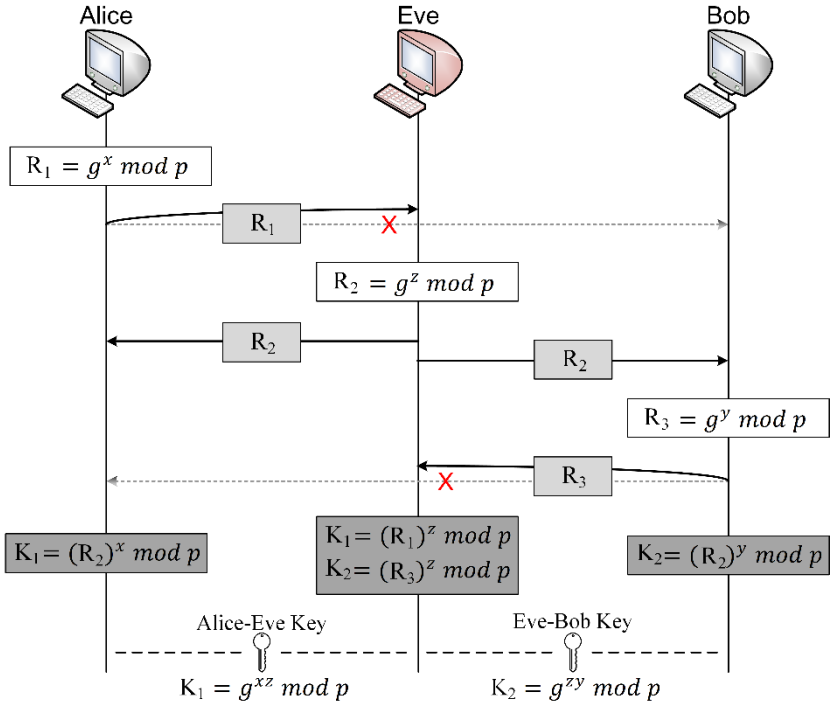


<그림 12> Diffe-Hellman 프로토콜의 중간자 공격 과정

대칭 키 합의

- Diffie-Hellman 프로토콜
  - 안전성 (3/3)
    - 중간자 공격(MITM, Man-in-the-middle attack) (2/2)
      - 공격 절차 (2/2)

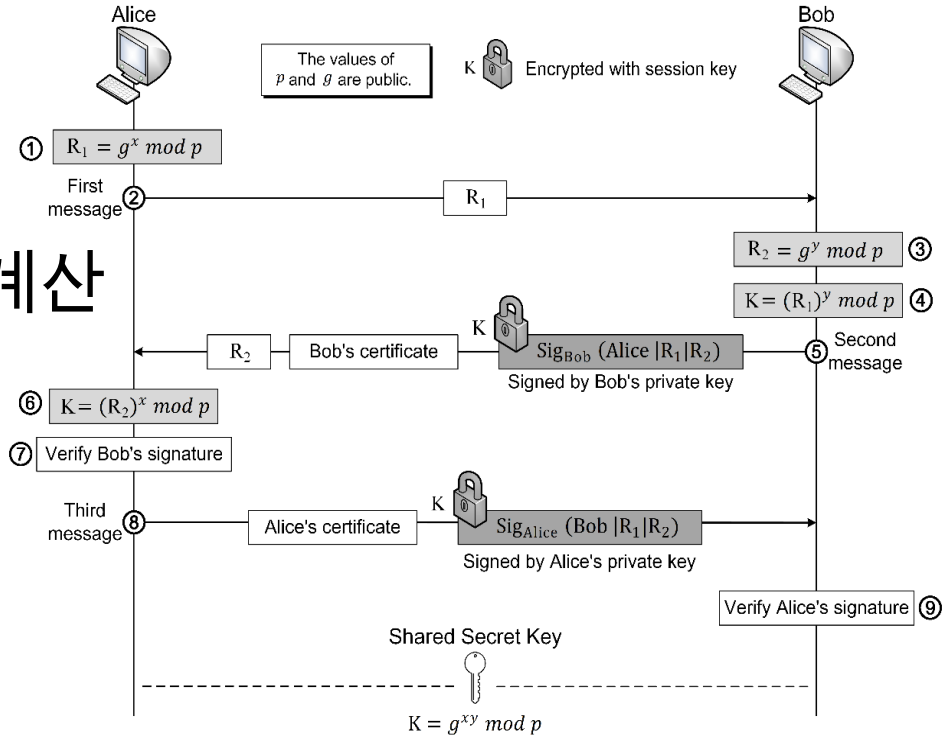
4. Alice와 Eve는 $K_1 = g^{xz} \ mod \ p$를 계산
   - K_1은 Alice와 Eve가 공유하는 키
   - Alice의 입장에서는 Bob과 공유한다고 생각
5. Eve와 Bob은 $K_2 = g^{zy} \ mod \ p$를 계산
   - K_2은 Eve와 Bob이 공유하는 키
   - Bob의 입장에서는 Alice와 공유한다고 생각



<그림 12> Diffe-Hellman 프로토콜의 중간자 공격 과정

대칭 키 합의

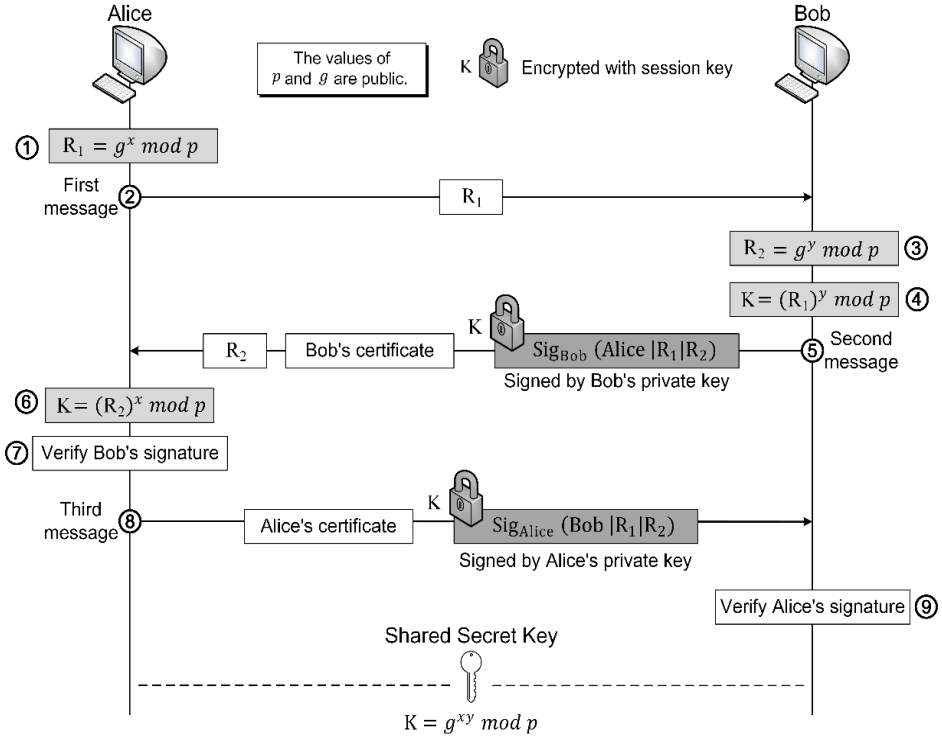
- 지국-대-지국 프로토콜(Station-to-station Protocol)
  - 개요
    - Diffie-Hellman 프로토콜을 기반으로 세션 키 생성
    - 상호 인증을 위해 공개 키 인증서 기반 디지털 서명을 활용
      - 중간자 공격 방지
  - 동작 (1/2)
    1. Alice는 x를 설정하고 R_1을 계산
    2. Alice는 R_1을 Bob에게 평문으로 전달
    3. Bob은 y를 설정하고 R_2를 계산
    4. Alice에게서 수신한 R_1을 사용하여 세션 키 K 계산



<그림 13> 지국-대-지국 프로토콜의 동작

대칭 키 합의

- 지국-대-지국 프로토콜(Station-to-station Protocol)
  - 동작 (2/2)
    5. Bob은 Alice의 ID, R_1, R_2를 이어 붙이고 자신의 개인키로 서명 후 R_2, 공개 키 인증서, 서명을 전송
       - 서명은 세션 키 K로 암호화
    6. Alice는 R_2로 K 계산
    7. Alice는 Bob의 서명 검증
    8. 검증 후 Bob의 ID, R_1, R_2를 이어 붙이고 자신의 개인 키로 서명하여 Bob에게 전송
       - 서명은 세션 키 K로 암호화
    9. Bob은 Alice의 서명 검증



<그림 13> 지국-대-지국 프로토콜의 동작

대칭 키 합의

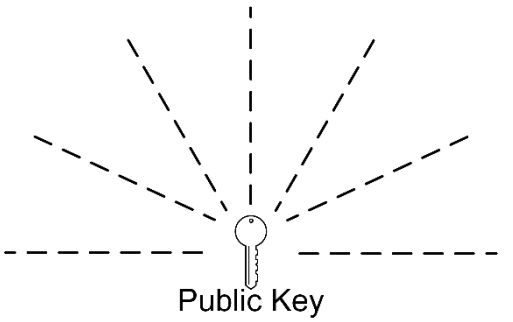
- 지국-대-지국 프로토콜(Station-to-station Protocol)
  - 안전성
    - 중간자 공격 불가능
      - Eve가 R_1을 가로채도 자신의 R_2를 Bob이 보낸 것 처럼 위조 불가능
        - Bob의 개인 키를 알 수 없어 서명 생성 불가능
      - Eve가 서명을 꾸며내더라도 인증서의 Bob의 공개 키로 검증 불가능
      - Eve가 Alice가 보낸 3번째 메시지에 대한 서명 위조 불가능
        - Eve가 Alice 개인키를 알 수 없어 서명 생성 불가능

목 차

- 개요
- 대칭 키 분배
- 커버로스(Kerberos)
- 대칭 키 합의
- 공개 키 배분

공개 키 배분

- 공개 선언
  - 개요
    - 공개 키를 공개적으로 선언하는 방식
      - e.g., 자신의 웹사이트 게시, 신문 발표
    - 가장 간단하고 쉬운 방식

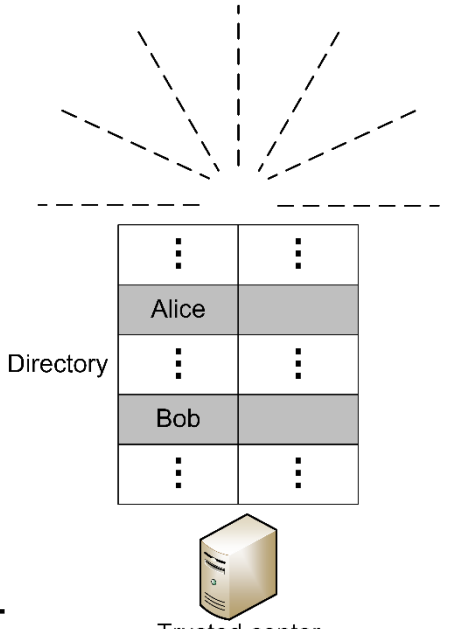


<그림 14> 공개 선언

  - 한계
    - 위조에 매우 취약
      - Eve가 자신의 공개 키를 Bob의 공개 키라고 발표 가능
        - Bob이 대처하기 전에 피해 발생 가능
      - Eve가 Bob의 공개 키를 Alice의 공개 키로 변경
        - Bob에게 가야 하는 메시지가 Alice에게 전달
      - Eve가 Bob의 공개 키를 자신의 공개 키로 변경
        - 자신의 개인 키로 문서에 서명한 뒤 Bob이 서명한 것처럼 위조

공개 키 배분

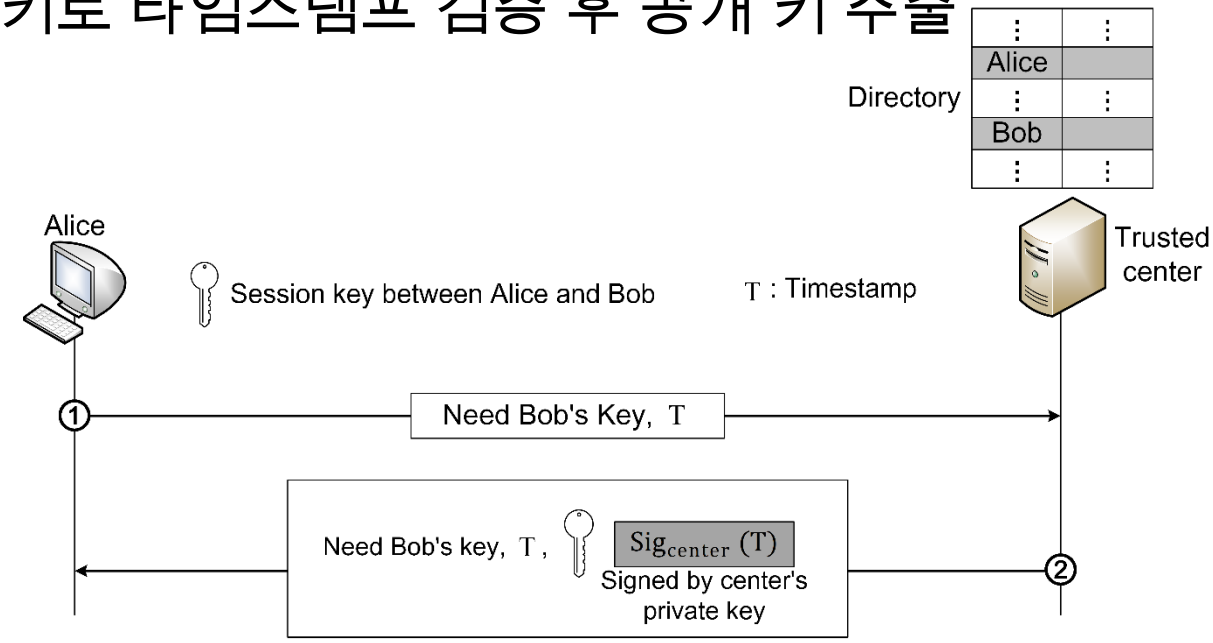
- 신뢰받는 센터
  - 개요
    - 신뢰받는 센터가 공개 키 디렉터리를 관리하는 방식
    - 모든 사용자는 센터에 공개 키를 등록하고 신분을 증명
    - 센터가 디렉터리를 공개적으로 배포하여 센터를 통해 공개 키 조회 가능
  - 한계
    - 실시간 접속 필요
      - 공개 키가 필요할 때마다 센터에 접속 필요
    - 센터에 부하 집중
      - 요청이 많아지면 센터에 병목 발생
    - 응답 변조 취약
      - 센터의 응답을 중간에서 가로채거나 수정 가능



<그림 15> 신뢰받는 센터

공개 키 배분

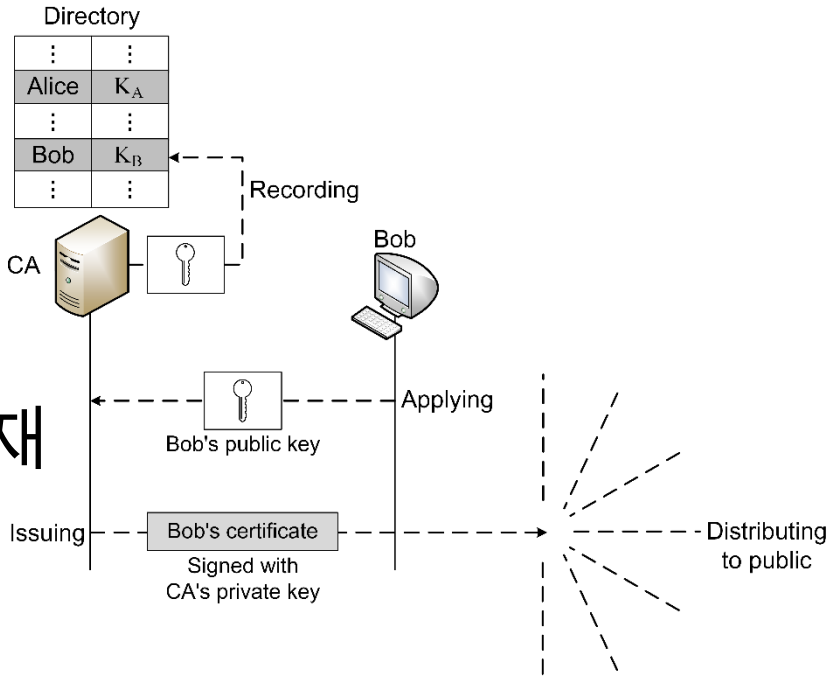
- 통제된 신뢰받는 센터
  - 개요
    - 신뢰받는 센터에 통제를 추가한 방식
      - 센터 응답에 타임스탬프와 서명 추가
    - Alice가 Bob의 이름과 타임스탬프로 센터에 요청 시 센터가 Bob의 공개 키, 원 요청, 서명한 타임스탬프로 응답
      - Alice는 센터의 공개 키로 타임스탬프 검증 후 공개 키 추출
  - 한계
    - 센터에 부하 집중
      - 요청이 많아지면 센터에 병목 발생



<그림 16> 통제된 신뢰받는 센터

공개 키 배분

- 인증 기관(CA, Certification Authority)
  - 개요
    - CA가 공개 키 인증서(Public-key Certificate)를 발급하는 방식
    - 공개 키와 소유자를 연관시켜 인증서 생성
      - CA가 자신의 개인 키로 인증서에 서명
      - 사용자는 인증서를 다운로드해 CA의 공개 키로 서명 검증 후 공개 키 획득
    - 인증서 1회 발급 시 자유롭게 배포 및 재사용 가능
  - 한계
    - 기관마다 인증서 형식에 차이 존재
      - 프로그램이 다른 기관 인증서를 자동 처리 불가



<그림 17> 인증 기관

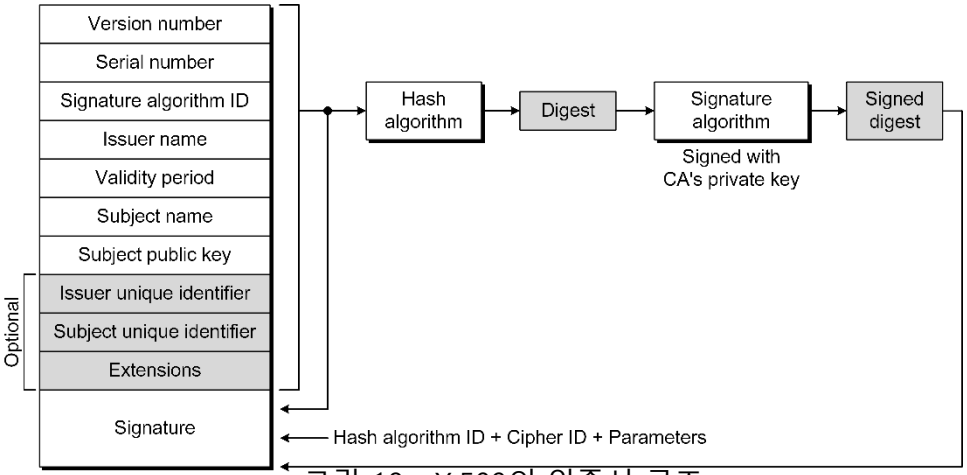
공개 키 배분

- X.509
  - 개요
    - 인증서를 구조적으로 표현하는 표준 형식
    - ITU가 설계하여 ASN.1 표기법 사용
    - 인증서 형식을 통일해 상호운용성 확보

*ITU (International Telecommunication Union)

*ASN.1 (Abstract Syntax Notation 1)
데이터 구조를 정의하는 표준 표기법

  - 인증서 구조 (1/4)
    - 버전 번호(Version Number)
      - 인증서의 X.509 버전 정의
    - 순서 번호(Serial Number)
      - 인증서 발행기관에서 각 인증서에 부여한 번호 표기

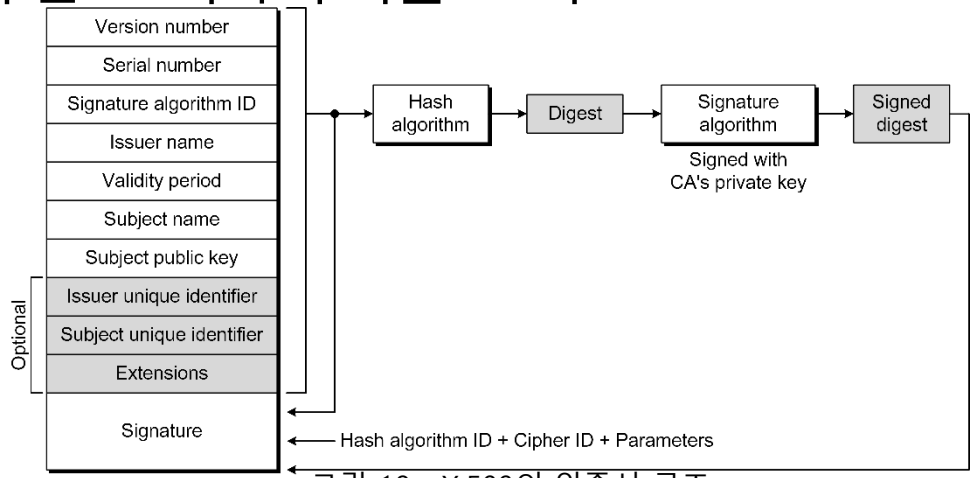


<그림 18> X.509의 인증서 구조

공개 키 배분

- X.509
  - 인증서 구조 (2/4)
    - 서명 알고리즘 ID(Signature Algorithm ID)
      - 인증서를 서명하는 알고리즘과 서명에 필요한 매개변수 정의
    - 발행자 이름(Issuer Name)
      - 인증서를 발급한 인증기관의 식별 정보를 스트링 계층 구조로 표기
        - e.g., 국가, 지방, 조직, 부서 등
    - 유효기간(Validity Period)
      - 인증서가 유효한 시작 시각부터 만료 시각까지를 표기
    - 주체 이름(Subject Name)
      - 공개 키 소유자의 식별 정보를 스트링 계층 구조로 표기
      - 일반 이름(Common Name) 필드가 포함
        - 키를 소유하는 사람의 실제 이름

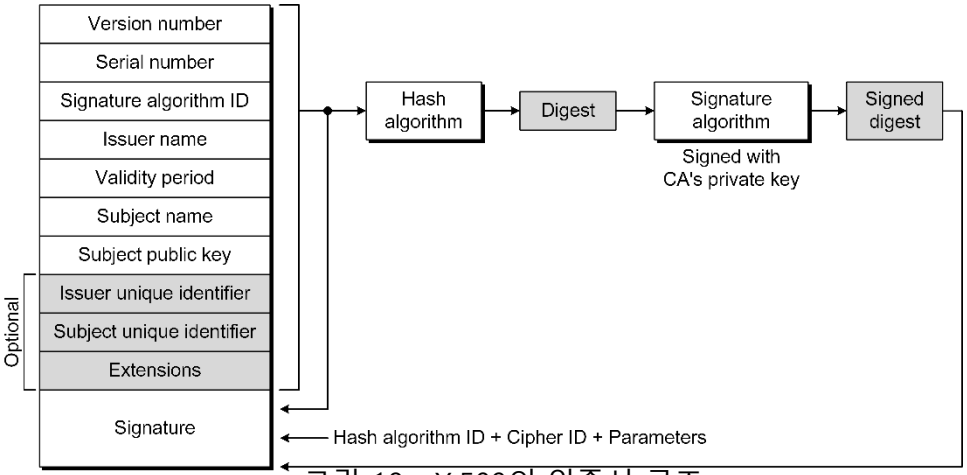
*스트링(String) 계층 구조
각 계층을 문자열로 담은 계층 구조
e.g., 국가(C) = “KR”, 조직(O) = “SJU”



<그림 18> X.509의 인증서 구조

공개 키 배분

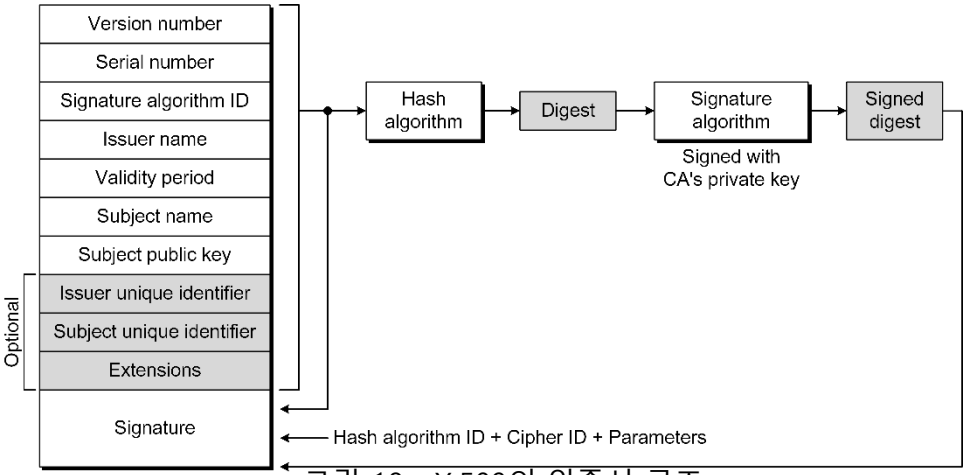
- X.509
  - 인증서 구조 (3/4)
    - 주체 공개 키(Subject Public Key)
      - 소유자의 공개 키 식별 정보
      - 대응하는 공개 키 알고리즘과 해당 매개변수 포함
    - 발행자 유일 식별자(Issuer Unique Identifier)
      - 옵션 필드
      - 발행자 이름이 동일할 경우 발행자 유일 식별자로 구분 가능
    - 주체 유일 식별자(Subject Unique Identifier)
      - 옵션 필드
      - 주체 이름이 동일할 경우 주체 유일 식별자로 구분 가능



<그림 18> X.509의 인증서 구조

공개 키 배분

- X.509
  - 인증서 구조 (4/4)
    - 확장(Extensions)
      - 옵션 필드
      - 발행자가 인증서에 부가적인 정보 추가 가능
    - 서명(Signature)
      - 3개의 부분으로 구성
        1. 인증서의 모든 필드
        2. CA의 공개 키로 암호화 된 인증서의 첫번째 부분의 다이제스트
        3. 두번째 부분을 생성하기 위해 사용한 알고리즘 식별자



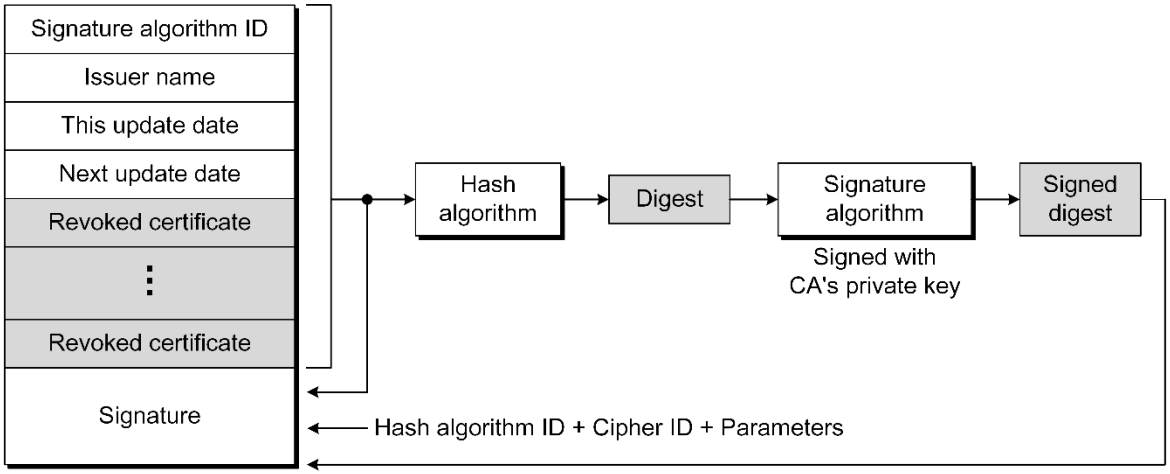
<그림 18> X.509의 인증서 구조

공개 키 배분

- X.509
  - 인증서 갱신
    - 인증서의 유효 기간이 끝나기 전 새로운 인증서 발급
      - 인증서에 문제가 없다는 전제
  - 인증서 폐지 (1/2)
    - 인증서의 유효기간이 끝나기 전 폐지가 필요한 경우 존재
      - 사용자(주체)의 개인 키가 침해 당한 경우
      - CA가 사용자를 더 이상 인증하지 않는 경우
        - e.g., 특정 기관에서 더 이상 일을 하지 않게 된 사용자의 기관 관련 인증서
      - 인증서를 검증하는데 필요한 CA의 개인 키가 침해 받은 경우
        - 유효기간이 남아있는 모든 인증서를 폐지
  - 인증서 폐지 목록(CRL, Certificate Revocation List) 발행
    - CRL이 발행될 때 유효기간이 남아있는 모든 폐지 대상 인증서 포함
    - 사용자가 인증서를 사용하고자 할 경우 해당 CA의 최신 CRL을 확인

공개 키 배분

- X.509
  - 인증서 폐지 (2/2)
    - 인증서 폐지 목록
      - 필드
        - 서명 알고리즘 ID
        - 발행자 이름
        - 금번 업데이트 일자(This Update Date): 해당 목록이 발표된 날짜
        - 다음 업데이트 일자(Next Update Date): 새로운 목록이 발표될 날짜
        - 폐지 인증서(Revoked Certificate): 사용자 인증서 순서번호와 폐지 일자 목록
        - 서명



<그림 19> X.509의 인증서 폐지 목록

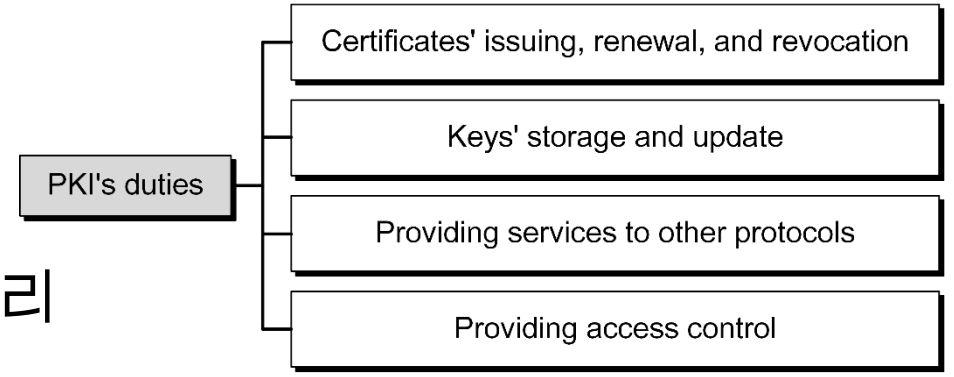
    - 델타 인증서 폐지 목록(Delta CRL)
      - 금번 업데이트 일자와 다음 업데이트 일자 이후에 변경이 생길 경우 공표하는 목록
      - CRL을 발행하는 주기 외에 폐지할 인증서 발생시 델타 CRL 발행
      - 가장 최근에 발행한 CRL 이후에 발생한 변경사항만 포함

공개 키 배분

- 공개 키 기반구조(PKI, Public-Key Infrastructure)
  - 개요
    - X.509를 기반으로 인증서를 생성, 배분, 폐지하는 체계

*IPSec (Internet Protocol Security)
*TLS (Transport Layer Security)

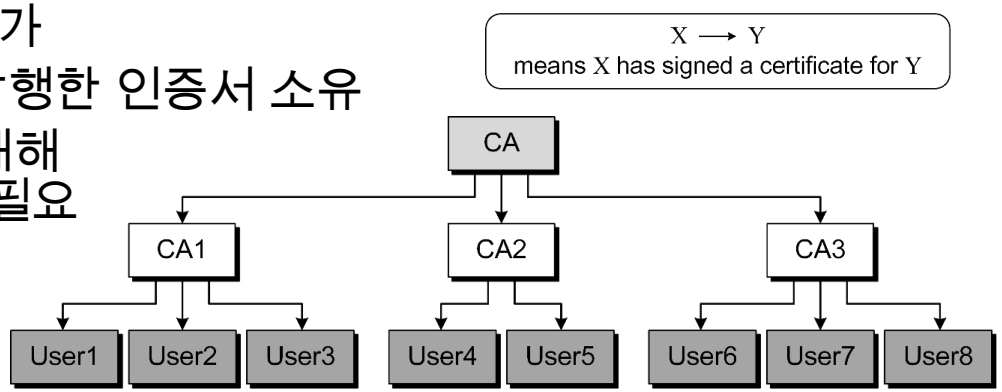
  - 역할
    - 인증서 발급, 갱신 및 폐지
      - 인증서와 관련된 모든 역할 처리
    - 키 저장 및 갱신
      - 개인 키를 안전하게 보관하려는 사용자에게 저장소 제공
      - 구성원의 키 갱신 요청 시 갱신 처리
    - 다른 프로토콜에 대한 서비스 제공
      - IPSec이나 TLS와 같은 인터넷 보안 프로토콜에 인증서 제공
    - 접근제어 제공
      - 데이터베이스에 저장된 정보에 대해 접근 수준 차등 관리



<그림 20> PKI의 역할

공개 키 배분

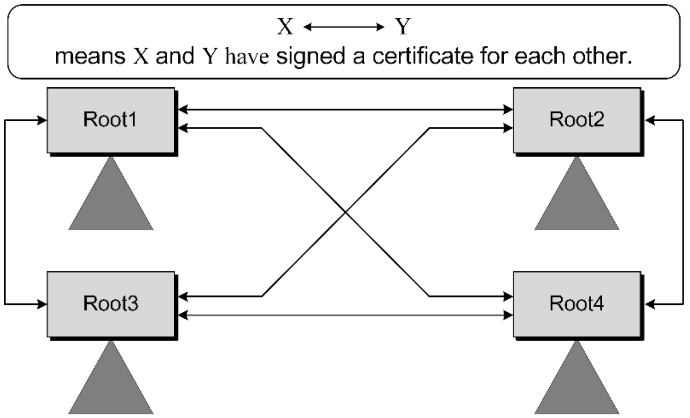
- 공개 키 기반구조(PKI, Public-Key Infrastructure)
  - 표기
    - 기관 X가 개체 Y에게 인증서를 발급할 경우 X<<Y>>로 표기
  - 신뢰 모델(Trust Model) (1/4)
    - 개요
      - 사용자가 한 CA로부터 발급받은 인증서를 검증하는 방식을 정의
    - 종류 (1/2)
      - 계층 모델(Hierarchical Model)
        - 기본적으로 루트 CA, 하위 CA, 사용자의 3계층으로 구성
          - 필요 시 하위 CA의 계층 증가
        - 루트 CA는 자체 서명 및 자체 발행한 인증서 소유
          - 자체 서명/발행 인증서에 대해 다른 CA나 사용자의 신뢰 필요



<그림 21> 계층 모델의 구조

공개 키 배분

- 공개 키 기반구조(PKI, Public-Key Infrastructure)
  - 신뢰 모델(Trust Model) (2/4)
    - 종류 (2/2)
      - 메쉬 모델(Mesh Model)
        - 여러 계층 구조의 루트가 서로를 인증하는 방식
        - 루트는 상호 인증서를 통해 서로를 인증
        - 각 루트는 나머지 모든 루트와 연결
          - 루트가 N개면 $N(N-1)$개의 인증서 필요



<그림 22> 메쉬 모델의 구조

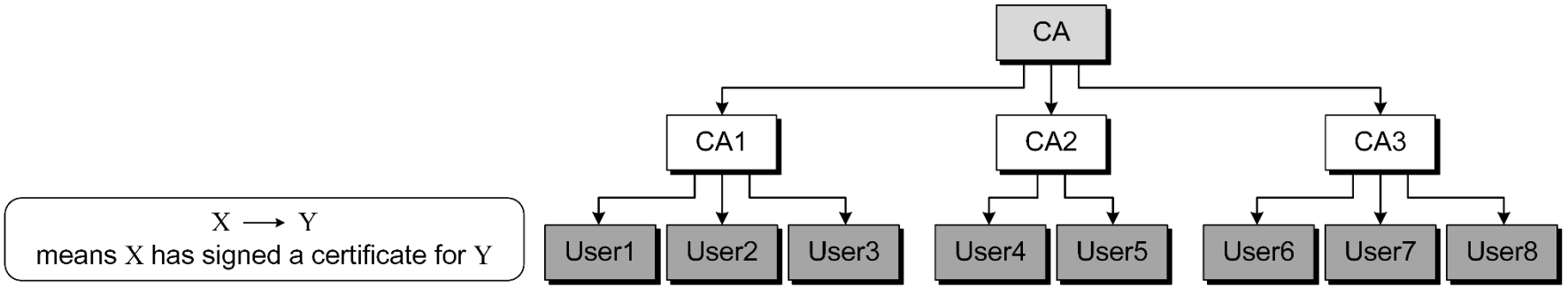
      - 신뢰 망(Web of Trust)
        - CA 없이 사용자들이 서로의 공개 키를 직접 인증하는 방식
        - 전자메일의 보안 서비스인 PGP(Pretty Good Privacy)에서 사용

공개 키 배분

- 공개 키 기반구조(PKI, Public-Key Infrastructure)
  - 신뢰 모델(Trust Model) (3/4)
    - 예제 15.3

계층 모델에서 User1이 루트 CA의 공개 키만을 알고 있을 때 어떻게 User3의 적법한 공개 키를 획득하는지 설명해보시오.

- User3은 User1에게 인증서 체인 {CA<<CA1>>, CA1<<User3>>}을 전송
- User1은 CA의 공개 키를 이용하여 CA<<CA1>>을 검증
- User1은 CA<<CA1>>로부터 CA1의 공개 키를 추출
- User1은 CA1의 공개 키를 이용하여 CA1<<User3>>을 검증
- User1은 CA1<<User3>>으로부터 User3의 공개 키를 추출



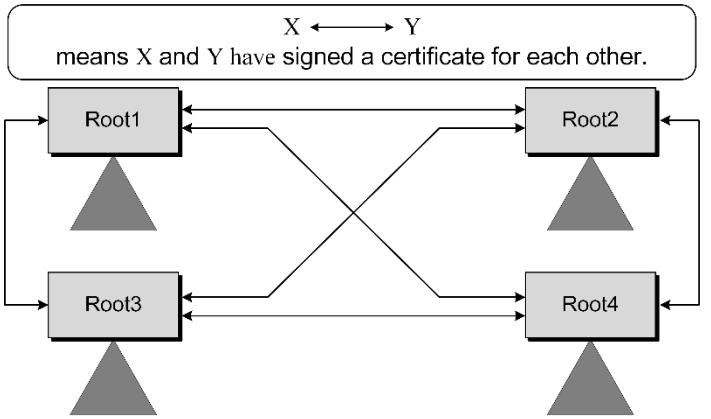
<그림 21> 계층 모델의 구조

공개 키 배분

- 공개 키 기반구조(PKI, Public-Key Infrastructure)
  - 신뢰 모델(Trust Model) (4/4)
    - 예제 15.4

메쉬 모델에서 Alice가 기관 Root1에 속해있고, Bob은 기관 Root4에 속해있다고 해보자. 어떻게 Alice는 Bob의 적법한 공개 키를 획득할 수 있을까?

- Bob은 Root4를 통해 Alice에게 인증서 체인을 전송
- Alice는 인증서 Root1<<Root1>>과 Root1<<Root4>>를 구하기 위해 Root1의 디렉터리를 확인



<그림 22> 메쉬 모델의 구조

Thanks!

송 민 희(minhee@pel.sejong.ac.kr)

부록#1

- 커버로스 버전 4 V.S. 버전 5
  - 버전 5의 업데이트 특징
    - 편의성, 확장성을 오버헤드와 트레이드
      - 편의성: 수명과 갱신
      - 확장성: 알고리즘과 표기 방식
  - 버전 4부터 공개
    - 버전 3 이하는 MIT 내부용이라 미 배포

[표 2] 커버로스 버전 비교

항목	버전 4	버전 5
티켓 수명	짧음	길
티켓 갱신	불가	가능
대칭 키 알고리즘	제한적 (DES 중심)	모든 대칭 키 알고리즘 수용
데이터 유형 기술	자체 방식	다른 프로토콜(ASN.1) 수용
오버헤드	작음	큼

부록#2

- 예제 15.2 코드

```
1.  def diffie_hellman(p, g, x, y):
2.      R1 = pow(g, x, p)          # Alice: R1 = g^x mod p
3.      R2 = pow(g, y, p)          # Bob:   R2 = g^y mod p
4.      K_alice = pow(R2, x, p)    # Alice: K = R2^x mod p
5.      K_bob = pow(R1, y, p)      # Bob:   K = R1^y mod p
6.      return R1, R2, K_alice, K_bob

7.  def wrap(n, width=64): # 긴 정수를 width 자리마다 줄바꿈해 정렬 출력용 문자열로 변환
8.      s = str(n)
9.      lines = [s[i:i + width] for i in range(0, len(s), width)]
10.     return ("\n" + " " * 8).join(lines)

11. def main():
12.     p = int("76462429856349357218249376595503050747633809672694974892357372860925 2"
13.             "35666607554236374233096611800333381061947301309504147387009991780436 5"
14.             "48785807987581"    )
15.     g, x, y = 2, 557, 273
16.     R1, R2, K_a, K_b = diffie_hellman(p, g, x, y)
21.     print(f"g = {g}, x = {x}, y = {y}\n")
22.     print(f"R1 =    {wrap(R1)}\n")
23.     print(f"R2 =    {wrap(R2)}\n")
24.     print(f"K  =    {wrap(K_a)}\n")
25.     print(f"Alice == Bob : {K_a == K_b}")

26. if __name__ == "__main__":
27.     main()
```