# 암호프로토콜 기초 설계 기술

NOTE 05



한국기술교육대학교 컴퓨터공학부 김상진

sangjin@koreatech.ac.kr
www.facebook.com/sangjin.kim.koreatech

## 교육목표

- 암호프로토콜 기초 설계 기술
  - 여분 정보
  - 명명 기법
  - 메시지의 최근성 보장 기법
  - 신뢰 관계(trust relationship)
  - 개체 인증
  - 공개키 사용 프로토콜 설계 원리

# 블록 방식의 대칭 암호알고리즘의 특성 (1/5)

- 블록 암호방식: 정해진 크기의 입력을 받아 입력과 같은 크기를 출력함
  - 평문이 블록크기보다 작으면 채우기 필요, 크면 암호화 모드 사용
- 단일 블록 크기의 서로 다른 메시지 $M$과 $M'$의 암호화 (ECB 모드)

$$E.K(M) = C,\ E.K(M') = C'$$

$len(M) = len(M')$
$= len(C) = len(C')$

  - $C$와 $C'$과 $M$과 $M'$ 간에는 어떤 상관 관계도 없음
  - $C$나 $C'$를 $K'(\neq K)$로 복호화한 결과를 예측할 수 없음
  - 조작된 $C$나 $C'$를 $K$로 복호화한 결과를 예측할 수 없음
  - $M$이 랜덤한 값이면 $K$로 $C$를 복호화하여도 $C$가 $K$를 이용하여 생성한 암호문인지 알 수 없음
  - $M$의 일부가 확인할 수 있는 값이고 복호화한 후에 그것이 확인되면 $C$가 $K$를 이용하여 생성한 암호문인지 확인할 수 있고, 복호화하여 얻은 $M$ 전체의 무결성에 대해 확신할 수 있음
    - 이처럼 암호문을 생성할 때 사용한 암호키를 확인할 수 있게 해주는 평문의 요소를 여분 정보라 함



# 블록 방식의 대칭 암호알고리즘의 특성 (2/5)

- 다중 블록 크기의 메시지 $M$을 암호화한 경우

$$E.K(M_1||M_2) = C_1||C_2$$

  - $M_1$과 $M_2$에 여분 정보가 모두 있어야 $C_1$과 $C_2$를 $K$로 복호화하였을 때 이 암호 블록들이 $K$로 암호화된 블록임을 확신할 수 있음
  - 둘 다 여분정보가 있어도 서로 연결할 수 있는 부분이 없다면 $C_1$과 $C_2$가 하나의 메시지 $M$을 암호화하여 만든 암호 블록임을 확신할 수 없음
    - 공격자는 전송되는 메시지를 다양하게 조작할 수 있음
    - 여분 정보의 유무에 따라 할 수 있는 조작이 달라짐

- 블록 크기, 사용자 식별자, 대칭키의 길이가 모두 16byte라 하자.
  ECB 모드로 암호화한 $\{B||K_{AB}\}.K_{AS} = C_1||C_2$, $\{D||K_{AD}\}.K_{AS} = C'_1||C'_2$가 있을 때, 다음 중 Alice가 복호화하였를 때 조작 여부를 알 수 있는 것은?
  단, Alice는 첫 블록에 $B$ 식별자를 기대하고 있음
  ① $C_1||C'_2$ ② $C'_1||C_2$
  ③ $C_1||X$ ④ $Y||C_2$
  여기서 $X$는 $C_2$의 몇 비트를 수정한 것이고, $Y$는 $C_1$의 몇 비트를 수정한 것임

# 블록 방식의 대칭 암호알고리즘의 특성 (3/5)

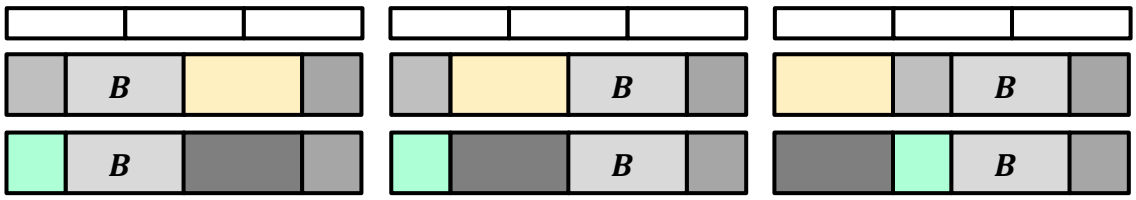
- 블록 크기, 사용자 식별자, 대칭키의 길이는 모두 16byte이고, 난스는 8byte라 하자. ECB 모드로 암호화한 $\{N_A||B||K_{AB}\}.K_{AS} = C$, $\{N_A||K_{AB}||B\}.K_{AS} = C'$, $\{K_{AB}||NA||B\}.K_{AS} = C''$이 있을 때, Alice는 이들을 복호화하여 $N_A$와 $B$를 확인하였고, 채우기도 문제가 없었다. 다음 중 $K_{AB}$의 무결성을 확인할 수 있는 것은?
  ① $C$ ② $C'$ ③ $C''$ ④ 없음

- $C$, $C'$, $C''$은 몇 블록?
- $K_{AB}$는 몇 번째 블록?
- $\{N'_A||B||K'_{AB}\}.K_{AS}$, $\{N'_A||K'_{AB}||B\}.K_{AS}$ 과거 암호문을 이용한 조작은?

```
struct msg{
    byte nonce[8];
    char name[16];
    byte key[16];
};
```

```
struct msg{
    byte nonce[8];
    byte key[16];
    char name[16];
};
```

```
struct msg{
    byte key[16];
    byte nonce[8];
    char name[16];
};
```






# 블록 방식의 대칭 암호알고리즘의 특성 (4/5)

- 인증 암호화(encrypt-then-mac) 방법을 사용한 경우

$$C = E.K_1(M), MAC.K_2(C)$$

  - MAC 값이 확인된 경우에만 복호화를 시도함
    - MAC 값이 확인되면 $C$의 무결성을 확신할 수 있음
  - 일반 암호화에서는 복호화한 후 여분 정보를 확인해야 이 암호문이 어떤 키로 암호화된 것인지 확인할 수 있지만 인증 암호화에서는 MAC 값이 확인되고 상대방을 신뢰하면 $C$가 특정키를 이용하여 생성한 암호문인지 확신할 수 있음
    - $C$가 단일 블록이 아니어도 성립함
  - 하지만 인증 암호화를 하더라도 최근성을 포함하여 여러 공격을 방어하기 위한 요소는 여전히 필요함

- 블록 크기, 사용자 식별자, 대칭키의 길이가 모두 16byte라 가정하자. ECB 모드로 인증 암호화한 $\{B||K\}.K_1 = C = C_1||C_2, T = MAC.K_2(C)$, $\{D||K'\}.K_1 = C' = C'_1||C'_2, T' = MAC.K_2(C')$가 있을 때, 다음 중 Alice가 조작 여부를 알 수 있는 것은? 단, Alice는 첫 블록에 $B$ 식별자를 기대하고 있음
  ① $C_1||C'_2, T$ ② $C'_1||C_2, T'$
  ③ $C_1||X, T$ ④ $Y||C_2, T$
  여기서 $X$는 $C_2$의 몇 비트를 수정한 것이고, $Y$는 $C_1$의 몇 비트를 수정한 것임

# 블록 방식의 대칭 암호알고리즘의 특성 (5/5)

- 인증 암호화를 하지 않으면 블록 크기, 평문의 각 요소의 크기, 평문의 구성 등이 바인딩에 영향을 줌
  - 실제 바인딩되었다고 확신하기 어려울 수 있음
- 인증 암호화를 하면 전송 과정의 조작은 발견할 수 있음
  - 송신자가 애초에 메시지를 엉뚱하게 구성하여 전송할 수 있음
    - 이 때문에 신뢰할 수 있어야 함
      - 송신자가 메시지를 엉뚱하게 구성할 이유가 있을까?
  - 상대방을 신뢰할 수 있다면 여분 정보 없이 수신한 암호문이 어떤 키로 암호화되었는지 확신할 수 있음 (사용하는 키 쌍이 고정)
  - 상대방을 신뢰할 수 있다면 바인딩에 대해서도 확신할 수 있음
- 인증 암호화가 모든 문제를 해결해 주는 것은 아님
  - 인증 암호화를 통해 얻을 수 있는 이점이 많기 때문에 하지 않을 이유는 없음
- 이 이후에는 인증 암호화를 하지 않아도 바인딩에 문제가 없다고 생각하고 설명함 (실제는 문제가 있을 수 있음)

$\{A||K_{AB}\}.K_{BS}$



# 여분 정보 (1/2)

- 평문에 여분(redundant) 정보가 없다면 올바른 키를 이용하여 복호화하였는지 판단할 수 없음
  - 여분 정보란 복호화의 정확성을 확인할 수 있도록 해주는 요소
- 여분 정보의 분류
  - 명백한(explicit) 여분 정보: 누구나 확인할 수 있는 정보
    - 예) 사용자 식별자
    - 수동적인 암호해독 공격을 용이하게 해줌
    - 특히, 패스워드와 같이 사전 공격이 가능한 암호키를 이용하거나 비교적 작은 공간에서 선택된 암호키를 이용하여 암호화한 암호문 내에는 명백한 여분 정보를 포함하지 않는 것이 바람직함
  - 함축적(implicit) 여분 정보: 오직 수신자와 송신자만이 알고 있는 정보
- 예) 여분 정보를 포함하는 가장 쉬운 방법
  - $\{M||H(M)\}.K_{AB}$: 여기서 $H(M)$은 명백한 여분 정보

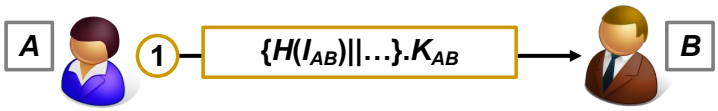
$\{K_{AB}\}.K_{BS}$

$\{A||K_{AB}\}.K_{BS}$

$\{M||\text{MAC}.K(M)\}.K$ | $\{M||\text{MAC}.K_2(M)\}.K_1$ | $\{M\}.K_1, \text{MAC}.K_2(M)$ | $C=\{M\}.K_1, \text{MAC}.K_2(C)$

# 여분 정보 (2/2)

- 예) Alice와 Bob은 둘 만이 공유하고 있는 비밀 정보 $I_{AB}$가 있다. 이때 다음과 같이 메시지를 교환된다면 $H(I_{AB})$는 오직 Alice와 Bob만이 확인할 수 있는 정보이므로 함축적 여분 정보가 됨




  - 제3자 입장에서는 $H(I_{AB})$는 랜덤 값이므로 이를 통해 올바르게 복호화되었는지 확인할 수 없음
- 인증 암호화를 하고, 상대방을 신뢰할 수 있다면 암호문을 생성할 때 사용한 암호키를 확인하기 위한 여분 정보는 필요하지 않음
  - 하지만 키 용도, 메시지 최근성 등을 확인하기 위한 요소는 여전히 필요함



# 명명 기법 (1/3)

- 명명(naming) 기법: 메시지의 의미를 명확하기 위해 참여자의 식별자를 암호문 내에 포함하는 기법
  - 식별자는 항상 명백한 여분 정보임
- 예) 서버가 Alice와 공유하고 있는 대칭키가 $K_{AS}$라 하자. 서버는 Alice의 요청에 의해 Bob와 공유할 대칭키를 다음과 같이 전달해준다고 하자.


  - Alice는 이 메시지만 보면 $K_{AB}$를 누구와 함께 사용해야 하는 키인지 알 수 없음
- 메시지를 받았을 때 해당 메시지를 통해 전달하고자 하는 내용이 메시지 내 명백하게 포함되어 있거나 있는 요소로부터 유추할 수 있어야 함
  - 위 메시지: 서버는 Bob와 사용할 수 있는 새로운 세션키를 Alice에게 비밀로 전달함

# 명명 기법 (2/3)

- 명명 기법의 적용: 대칭 암호알고리즘을 사용하는 경우
  - 예) 서버 *S*가 Alice와 Bob 간에 사용할 세션키 $K_{AB}$를 Alice에게 전달


  - 여기서 *A*는 생략 가능, $K_{AS}$로부터 *A*는 유추 가능
    - 최근성을 제공하지 못함
  - 실제 인증 암호화를 하지 않으면 *B*와 $K_{AB}$의 바인딩을 확신할 수 없음



# 명명 기법 (3/3)

- 명명 기법의 적용: 비대칭 암호알고리즘을 사용하는 경우
  - 예) 같은 용도의 메시지


  - 대칭 암호알고리즘 사용 때와 달리 *A*를 생략할 수 없음.
    생략하면 다음과 같은 공격이 가능함


{{B||K_BC}.–K_S}.+K_C

{{C||B||K_BC}.–K_S}.+K_A

- 특정 암호키로 직접 암호화된 경우에만 해당 암호키로부터 유추할 수 있는 정보는 생략 가능
- 공개키의 경우도 알고리즘에 따라 한번에 암호화할 수 있는 크기가 제한적임
- 특히, 한 번에 암호화할 수 없으면 하이브리드 암호 방식을 보통 사용함

$M = A||B||K_{AB}$ {M||Sig.S(M)}.K, {K}.+$K_A$

# 메시지의 최근성/시기적절성 (1/3)

- 암호프로토콜에서 메시지(암호문)를 수신하면 정해진 다음 해동을 취하기 위해 여러 가지 검사를 하게 됨
  - 이때 보통 메시지의 최근성(freshness, timeliness)을 검사함
- 메시지의 최근성을 검사한다는 것은 메시지가 이전 또는 다른 세션에 사용된 메시지가 아니라 현재 프로토콜 수행을 위해 새롭게 만든 메시지임을 확인하는 것을 말함
- 메시지의 최근성은 메시지의 구성 요소로부터 유추되어야 하며, 어떤 메커니즘을 사용하든 간에 재전송(replay) 메시지에는 포함될 수 없거나 재전송 메시지와 현재 메시지를 구별할 수 있어야 함
- 메시지와 특정 프로토콜 수행과의 연결은 일반적으로 시간적인 관계(temporal relationship) 또는 인과 관계(causal relationship)를 통해 이루어짐
- 시간적 관계 또는 인과 관계는 일반적으로 관계를 증명할 수 있는 식별자를 메시지에 포함함으로써 형성함



# 메시지의 최근성/시기적절성 (2/3)

- 메시지의 최근성을 보장하기 위한 식별자로 가장 널리 사용하는 것은 타임스탬프(timestamp)와 난스(nonce, Number used just Once)임
- 타임스탬프 기반 기법: 시간적 관계를 통해 메시지의 최근성을 보장하는 기법
  - 이 기법은 보통 메시지에 타임스탬프라고 하는 메시지 작성 시간을 포함하여 최근성을 보장함
- 난스 기반 기법: 인과 관계를 통해 메시지의 최근성을 보장하는 기법
  - 이 기법은 주로 시도(challenge)와 그것에 대한 응답(response)을 통해 메시지의 최근성을 보장함
  - 응답이 시도 이후에 만들어질 수밖에 없도록 하여 응답의 최근성을 확인함

# 메시지의 최근성/시기적절성 (3/3)

- 관계를 형성하는 과정에서 참여자의 역할
  - 제공자(supplier): 식별자를 제공하는 참여자
  - 입증자(prover): 제공된 식별자를 메시지에 포함하는 참여자
  - 검증자(verifier): 메시지에 포함된 식별자를 통해 메시지의 최근성을 확인하는 참여자
    - 타임스탬프 기법: 제공자와 입장자가 같은 참여자
    - 난스 기법: 제공자와 검증자가 같은 참여자
- 어떤 특정한 값이 최근성 식별자로서 적합하기 위해서는 반드시 사용한 값이 예측할 수 없어야 하는 것은 아님
  - 타임스탬프는 예측이 가능한 값임
- 더 중요한 것은 식별자를 어떻게 사용하느냐에 따라 불예측성이 필요할 수 있고, 필요하지 않을 수 있음
  - 난스 기법의 경우에도 항상 불예측성이 필요한 것은 아님. 더 중요한 것은 매번 다른 것(이전에 사용하지 않은 것)을 사용하는 것임



# 타임스탬프 (1/5)

- 타임스탬프를 이용한 메시지(암호문)의 최근성 증명


  - Bob은 위 메시지를 수신하면 자신의 지역 시간의 현재 값 $T_B$와 $T_A$를 비교하여 두 시간의 차이 $|T_A-T_B|$가 허용할 수 있는 범위 내에 있으면 메시지가 최근에 생성된 메시지로 인식하게 됨
    - 따라서 두 사용자의 시간이 어느 정도 동기화되어 있어야 함
  - *A*: 제공자/입증자, *B*: 검증자
- 타임스탬프를 유효한 값으로 허용하는 범위를 허용 윈도우(acceptance window)라 하며, 메시지 통신 지연과 시간 동기화(synchronization) 메커니즘에 의해 결정함

# 타임스탬프 (2/5)

- 취약점
  - 허용 윈도우 내에 재전송하는 공격
    - 메시지 중복 검사가 반드시 필요하며, 이를 위해서는 지난 $t$초 ($t$: 허용 윈도우의 길이) 내에 수신한 모든 메시지를 보관함
  - 제공자는 쉽게 특정 순간을 가리키는 타임스탬프를 사용할 수 있음
    - 모든 참여자가 메시지에 타임스탬프를 첨부하고 확인하기 위해서는 타임스탬프의 형태를 알고 있어야 함
      - 타임스탬프의 형태는 공개되어 있을 수밖에 없음
    - 타임스탬프의 값은 미래 또는 과거의 어느 한 순간을 나타냄
    - 검증자는 제공자가 정직하게 타임스탬프를 포함한다는 것을 신뢰해야 함
    - 입증자가 실수로 잘못된 타임스탬프를 포함하지 않는다는 것을 믿을 수 있어야 함
    - 입증자는 이런 능력(competence)을 가지고 있어야 함
  - 시스템 시간이 보호해야 하는 중요한 자산이 됨

$\{T_S||B||K_{AB}\}.K_{AS}$



# 타임스탬프 (3/5)

- 예) 수신자의 시간이 송신자의 시간보다 상대적으로 앞서 있다고 가정하자. 이때 송신자의 시간은 정상임
  - 송신자가 올바른 타임스탬프를 기록한 메시지를 전달하면 수신자는 유효한 메시지임에도 불구하고 거부하게 됨
  - 이런 메시지를 선행(predated) 메시지라 함
  - 이 문제점은 시간을 동기화함으로써 해결할 수 있음
  - 참고. 시간은 항상 절대 시간과 동기화되어야 함

# 타임스탬프 (4/5)

- 예) 송신자의 시간이 수신자의 시간보다 상대적으로 앞서 있다고 가정하자. 또한 공격자는 이 사실을 알고 있음. 송신자가 수신자에게 타임스탬프된 메시지를 전송하면 공격자는 이것을 가로채어 보관한 다음에 해당 타임스탬프가 유효한 시기가 되면 이것을 전송하여 공격할 수 있음
  - 이와 같이 원래 시간보다 앞선 시간이 기록된 메시지를 사후(postdated) 메시지라 함
  - 이와 같은 문제를 해결하기 위해 다른 시간을 잘못된 시간과 동기화할 수 있음. 하지만 이 방법은 시간이 실제 시간과 보조를 맞출 수 없게 됨
  - 원래 메시지가 수신 측에 도달하지 않았으므로 단순 메시지 중복 검사를 통해 문제점을 해결할 수 없음

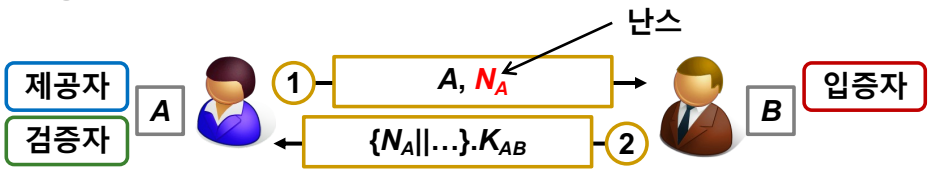


# 타임스탬프 (5/5)

- 장점
  - 제공자와 입증자가 동일인이며, 일방향 통신으로(하나의 메시지로) 최근성을 보장할 수 있음
- 단점
  - 입증자를 신뢰할 수 있어야 함
  - 타임스탬프의 사용은 시스템 간의 시간 동기화가 요구됨
    - 시간 동기화 메커니즘이 시스템 보안성에 미치는 영향이 큼
    - 따라서 암호프로토콜을 이용하여 시간 동기화 메카니즘을 구현하여야 한다는 것을 의미함. 이 프로토콜의 최근성은?
  - 사후 메시지 문제를 해결할 수 없음
- 타임스탬프 기법의 안전성을 결정하는 요인
  - 시간 동기화, 입증자의 정직성, 입증자의 능력

# 난스 (1/2)

- 난스란 특정한 순간을 증명하기 위해 생성된 값을 말함
  - 다른 표현: freshness identifier, liveness indicator
- 난스는 보통 시도-응답 기법을 이용함

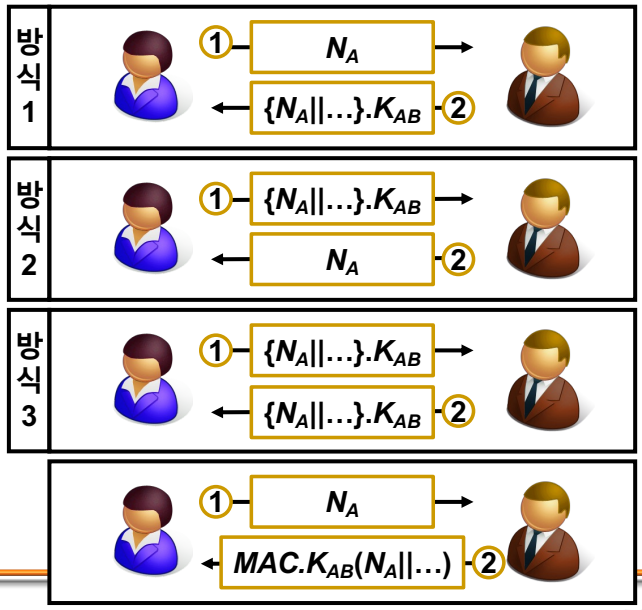



- 시도-응답(challenge-response) 기법이란 메시지 *M*에 대한 응답 메시지는 오직 *M*의 내용을 알고 있을 경우에만 생성 가능
  - 시도와 응답 간의 인과 관계가 성립하며, 두 메시지를 바인딩하는 역할을 함. 이를 통해 응답 시점을 예측할 수 있음
- 일반적으로 제공자와 입증자는 다르며, 제공자와 검증자는 동일인임
- 난스 기법은 최소한 두 개의 메시지를 필요로 함
- 가장 중요한 점. 기존에 사용한 난스 값을 절대 사용하지 않아야 함



# 난스 (2/2)

- 난스 값으로 사용하는 것
  - 타임스템프, 카운터, pseudo-random number
- 장점. 검증자는 입증자의 정직성과 능력에 대해 신뢰할 필요가 없음
- 사용 방식
  - 난스는 항상 명백한 여분 정보임
  - 인증 측면에서 방식 2의 경우에는 $N_A$를 예측할 수 없어야 함
    - 예측이 가능하면 누구나 응답할 수 있음
    - 최근성 보장이 목적이 아닐 수 있음
  - 방식 3에서 두 메시지를 구분할 수 있어야 함



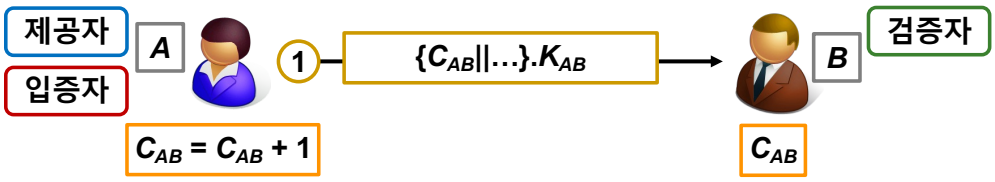
# Timestamp VS. Nonce

| Timestamp | Nonce |
|---|---|
| 제공자 = 입증자 ≠ 검증자 | 제공자 = 검증자 ≠ 입증자 |
| 입증자의 정직성이 요구됨 | 값의 선택은 제공자의 책임 |
| 하나의 메시지로 최근성을 보장함 | 최소한 두 개의 메시지가 필요 |
| 클럭 동기화 메커니즘 필요 | 클럭 동기화 메커니즘 불필요 |
| 메시지 바인딩 기능 없음 | 메시지 바인딩 기능 제공 |



# 메시지 최근성을 보장하는 기타 다른 방법

- 카운터 사용
  - 카운터의 사용은 쌍방 간의 동기화된 카운터의 존재를 가정하며, 실제 사용은 타임스탬프와 차이가 없음

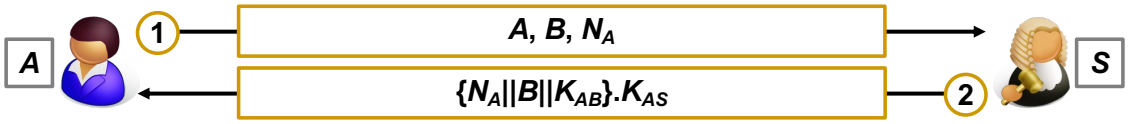



  - 장점. 실제 시간과 독립적으로 동기화 가능
  - 단점. 모든 사용자 간에 동기화된 별도 카운터가 필요함. 사용자 간에는 힘들지만 서버와 사용자 간에는 활용 가능
  - 무한 카운터를 사용할 수는 없음
  - 카운터도 중요한 보호 자산

- 최근성이 보장된 암호키 사용
  - 최근에 생성된 키를 사용하여 생성된 메시지(예: 암호화)는 기본적으로 최근에 생성된 메시지가 됨

# 신뢰 관계 (1/3)

- 예) Alice와 서버 간에 다음과 같이 메시지가 교환되었다고 하자.




  - 바인딩에 문제가 없다고 가정
  - 복호화하여 $N_A$와 $B$를 확인하였고, 이를 통해 메시지 2의 암호문이 $K_{AS}$로 암호화된 암호문임을 확인하였음
    - 서버가 생성한 암호문임을 확인한 것임
  - $N_A$의 확인을 통해 메시지 2의 암호문이 최신에 생성한 것임을 확신할 수 있음 (서버에 대한 신뢰 불필요)
  - $K_{AB}$는 식별자 $B$, 난스 $N_A$와 바인딩되어 있으므로 Bob와 공유하기 위한 서버가 최근에 생성한 세션키임을 확신할 수 있음 (서버에 대한 신뢰 필요)



# 신뢰 관계 (2/3)

- 일반적으로 암호프로토콜을 설계할 때에는 참여자 간의 어떤 신뢰 관계를 가정하게 되며, 프로토콜의 안전성이 이와 같은 관계에 의존하게 됨
- 어떤 참여자 $A$가 $B$를 신뢰한다는 것은 $A$는 $B$가 어떤 특정 상황에서 특정한 행동을 할 것으로 믿는다는 것을 말함
  - 정해진 프로토콜 규칙에 따라 행동함
- 신뢰와 관련된 주장(또는 가정)은 잘못되었다고 말할 수 없지만 적절하지 않다고 말할 수는 있음
- 프로토콜 수행에 대해 이해관계가 있는 참여자들은 서로를 신뢰하기 어려움
  - 예) 전자상거래에서 상점과 고객
- 신뢰는 서버에 대한 신뢰가 아니라 서버를 운영하는 기관에 대한 신뢰를 말함
- 따라서 프로토콜의 보안 요구사항을 충족시키기 위해 제3의 신뢰기관을 사용해야 하는 경우가 많음
  - 하지만 응용에 따라 적절한 신뢰 기관을 찾는 것이 어려운 경우도 많음
  - 보통 정부기관에서 운영 또는 승인한 기관만 신뢰하는 경우가 많음

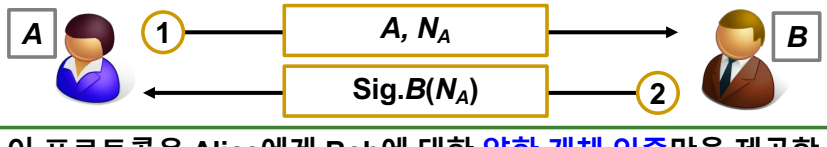
# 신뢰 관계 (3/3)

- 예) 클라우드 컴퓨팅에서 사용자가 클라우드 서비스를 제공하는 기관을 신뢰할 수 있나?
- 현재 널리 사용하는 신뢰 기관
  - 인증기관
    - 공인인증기관, 사설인증기관
    - 탈중앙
  - 전자상거래에서 신용카드 회사



# 개체 인증 (1/2)

- 개체 인증의 분류
  - 약한 개체 인증: 상대방이 현재 온라인에 있다는 것이 확인된 경우
  - 강한 개체 인증: 상대방이 현재 온라인 상태이며, 본인을 통신의 상대방으로 인식하고 있다는 것이 확인된 경우
- 예)

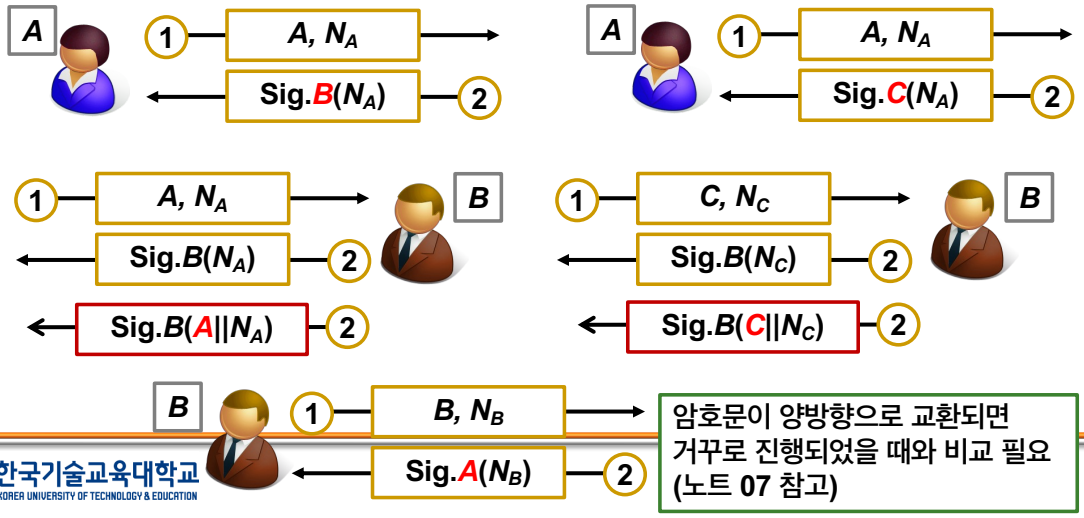



- 이 프로토콜은 Alice에게 Bob에 대한 약한 개체 인증만을 제공함
- Alice는 Bob이 전달한 서명이 최근에 생성한 것임을 확신할 수 있음


- 공격자는 메시지를 중계하고 있음
- Alice는 수신한 서명이 Bob이 생성한 것임을 확인할 수 있지만, Bob이 Alice에 대한 요청에 대한 응답으로 이 메시지를 생성했다는 것은 알 수 없음
- 약한 개체 인증만 제공하고 있는 것임

# 다른 분석 방법

- 2자간 프로토콜의 경우 프로토콜을 시작하는 참여자(개시자)와 그것에 대해 반응하는 참여자(반응자)로 구분할 수 있음
- 이때 프로토콜이 안전하기 위해서는 개시자가 서로 다른 두 명의 반응자와 프로토콜을 수행할 때 두 수행을 구분할 수 있어야 하며,
- 반대로 반응자 입장에서 서로 다른 두 명의 개시자와 프로토콜을 수행할 때 두 수행을 구분할 수 있어야 함

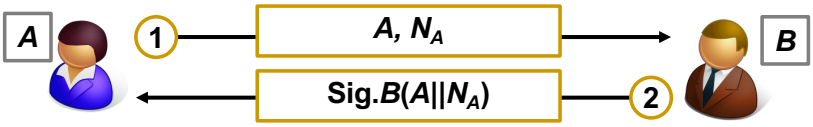



암호문이 양방향으로 교환되면 거꾸로 진행되었을 때와 비교 필요 (노트 07 참고)



# 개체 인증 (2/2)

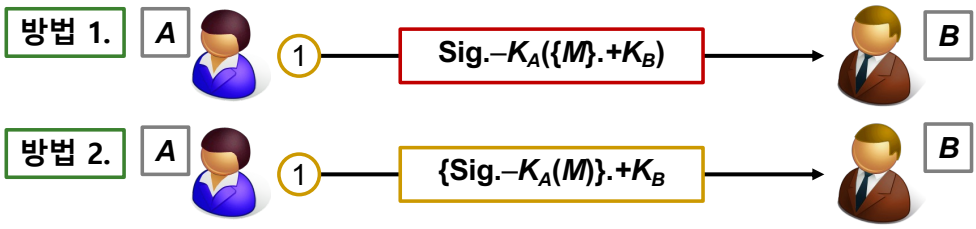
- 예) Alice에게 Bob에 대한 강한 개체 인증을 제공하는 프로토콜




- 상호 인증(mutual authentication): 같은 프로토콜에서 두 참여자가 서로를 인증할 수 있는 경우
- 일방향 인증(unilateral authentication): 같은 프로토콜에서 한 참여자만 다른 참여자를 인증할 수 있는 경우
- 질문. ATM 기기는 일방향 인증? 상호 인증?

# 공개키 사용 프로토콜 설계 원리 (1/2)

- Anderson과 Needham의 제시한 원리
  - 원리 1. 암호화하기 전에 서명을 해야 함
    - 예)




- 두 방법 모두 오직 Bob만 $M$을 얻을 수 있음
- 방법 1에서 Bob은 Alice가 $M$을 알고 있다고 확신할 수 없지만 방법 2는 확신할 수 있음
- 방법 2에서 Bob은 Alice가 이 메시지를 전송하였다고 확신할 수 없음
- 방법 1과 방법 2는 서로 장단점이 있음

비고. 이 장에서는 서명 확인에 원 메시지가 필요한 서명 기법을 사용한다고 가정함

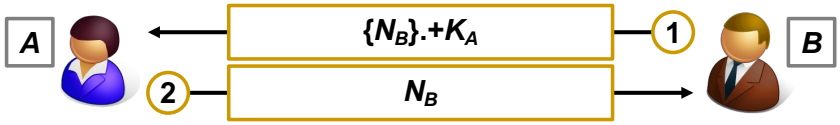
$C$={$M$}.$K$, {$K$}.+$K_B$, Sig.–$K_A$($H$(C))

{$M$}.$K$, {$K$}.+$K_B$, Sig.–$K_A$($H$($M$))

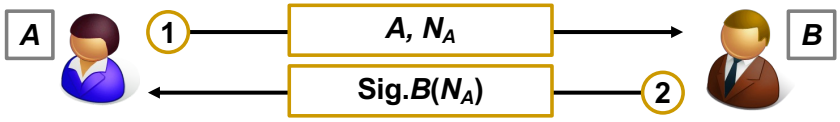


# 공개키 사용 프로토콜 설계 원리 (2/2)

- 원리 2. 개인키로 서명하거나 복호화할 때 상대방이 자신을 오라클(oracle)로 활용할 수 있음으로 주의해야 함
  - 예)




- 이 프로토콜을 통해 Bob은 Alice가 존재한다는 것을 확신할 수 있음
- Alice가 복호화한 값에 대해 특별한 주의를 기울이지 않고, 중요한 비밀 정보를 교환할 때에도 +$K_A$를 사용한다면 이 프로토콜을 이용하여 많은 사용자들은 {$M$}.+$K_A$ 형태의 암호문을 복호화할 수 있음




암호세계에서 오라클: 물어본 것에 대해 항상 올바른 답을 주는 개체