암호프로토콜 개요

NOTE 04

한국기술교육대학교 컴퓨터공학부 김상진

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

교육목표

- 프로토콜과 암호프로토콜에 대한 기초적 이해
- 암호프로토콜의 분류에 대한 이해
- 암호프로토콜의 예: 키 확립 프로토콜
- 암호프로토콜에서 대칭, 비대칭 암호알고리즘의 사용
- 암호프로토콜의 안전성
- 암호프로토콜의 효율성

프로토콜 (1/2)

- 프로토콜(protocol): 어떤 목적을 달성하기 위해 2명 이상이 참여하는 유한한 일련의 단계(sequence of steps)를 말함
  - 알고리즘(algorithm): 홀로 어떤 목적을 달성하기 위해 수행하는 일련의 단계
  - 일련의 단계란 단계를 실행하는 순서가 있으며, 한 단계가 끝나야 다음 단계를 수행함
- 이 교과에서 고려하는 프로토콜은 모두 통신 프로토콜임
  - 원격 사용자 간의 메시지 교환을 통해 이루어지는 프로토콜
- 통신 프로토콜을 기술할 때에는 각 참여자가 교환하는 메시지의 형태 뿐만 아니라 메시지를 수신하였을 때 참여자가 취하게 되는 내부 행동 (메시지가 도착하지 않은 경우, 잘못된 형태의 메시지가 수신된 경우 등)까지 포함되어 있어야 함
  - 하지만 축약하여 기술할 때에는 보통 교환되는 메시지의 형태만 기술하는 경우가 많음 (이 수업에서는 이 방식을)

프로토콜 (2/2)

- 프로토콜의 특성
  - 사전에 알고 있어야 함
  - 참여자 간에 동의하고 있어야 함
  - 모호하지 않아야 함
  - 완전성(completeness): 완전해야 함 (목적이 달성되어야 함)
- 프로토콜 수행(protocol run): 프로토콜의 임의의 단일 실행
  - 여러 프로토콜 수행이 병행으로 수행되고 있을 수 있음
  - 트랜스크립트: 한 수행에서 교환된 메시지들
- 암호프로토콜(cryptographic protocol): 암호기술을 사용하는 프로토콜
- 암호기술을 사용하는 목적: 프로토콜의 각 메시지의 의미를 보장하고, 참여자 또는 제3자가 부당한 이득을 얻지 못하도록 하기 위함
- 일반 프로토콜과의 차이점: 공격자의 존재를 가정함
- 일반 프로토콜: 완전성
- 암호프로토콜: 완전성 + 안전성(secureness)

암호프로토콜에 대한 공격

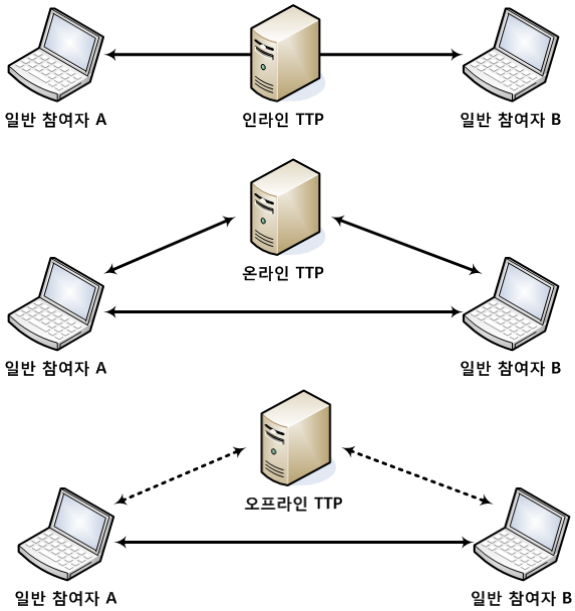
- 보안을 위협하는 자
  - 공격자: attacker, intruder, penetrator, interloper, adversary, enemy, eavesdroper, interceptor, tapper
- 공격의 종류
  - 수동 공격(passive attack): 프로토콜의 진행을 방해하지 않는 공격
    - 데이터의 비밀성에 대한 위협
    - 트래픽 분석이 목적일 수 있음
    - 수동 공격을 하는 자를 보통 도청자(eavesdropper)라 함
  - 능동 공격(active attack): 프로토콜의 진행에 개입(메시지 삭제(차단), 삽입, 변경, 재전송)하는 공격
    - 데이터의 비밀성뿐만 아니라 인증, 무결성에 대한 위협
    - 공격을 방어하기 위한 기본. 공격이 이루어지고 있다는 것을 인식하는 것

프로토콜의 참여자

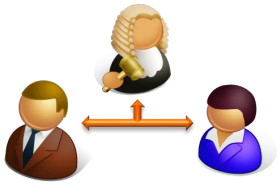
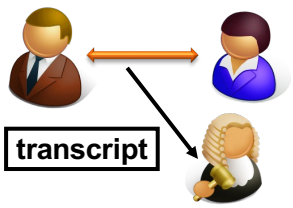
- 참여자(principal): 실제 사용자를 의미할 수 있고, 사용자가 사용하는 장치 또는 소프트웨어일 수 있음
- 참여자의 분류: 일반 참여자, 제3의 신뢰기관(TTP, Trusted Third Party)
  - 일반 참여자: 프로토콜을 통해 얻고자 하는 것이 있는 이해 당사자
  - TTP: 프로토콜의 실행 결과에 대한 어떤 이해 관계가 없고, 어떤 참여자와도 특별한 협력 관계가 없지만 원활한 프로토콜의 수행을 위해 참여하는 참여자
    - 신뢰(trust)한다는 것은 프로토콜에서 정한 규칙대로 프로토콜을 수행한다는 것을 말함
      - 어떤 부정 행위도 하지 않는다는 것은 아님
    - 다른 말로 중재자(arbitrator)라 함
- 암호프로토콜에서는 항상 공격자의 존재를 가정함

TTP의 종류

- 인라인(inline) TTP: 프로토콜의 모든 과정에 항상 참여
  - 모든 메시지는 항상 TTP를 경유함
- 온라인(online) TTP: 프로토콜의 일부 과정에서는 항상 참여
- 오프라인(offline) TTP: 실제 프로토콜 수행과정에서 참여하지 않지만 사전에 또는 사후에 필요에 따라 참여



암호프로토콜의 분류 (1/2)

		
자체 강화 프로토콜	중재 프로토콜	판결 프로토콜

• 중재자(arbitrator)가 필요 없는 프로토콜
  ◦ 가장 이상적인 형태
  ◦ 모든 상황을 이 형태로 해결할 수 없음
	• 중재자가 프로토콜에 항상 참여하는 프로토콜
	
	평상시에 수행하는 프로토콜(nonarbitrated subprotocol)과 분쟁이 발생하였을 때 수행하는 프로토콜(adjudicated subprotocol)로 나누어짐
		• 분쟁이 발생한 경우에만 판결자(adjudicator)가 참여하는 프로토콜
		• 프로토콜은 분쟁을 해결할 수 있도록 증거 자료를 남겨야 하며, 중재자는 제시된 증거를 검사하여 분쟁을 명확하게 해결할 수 있어야 함

암호프로토콜의 분류 (2/2)

- 신뢰하는 제3의 중재자를 사용할 경우 발생할 수 있는 문제
  - 중재자가 동작하지 않거나 중재자와 다른 참여자 간의 통신이 중단되면 프로토콜 자체를 수행할 수 없음
    - 이 문제를 단일 실패점(single-point-of-failure)이라 함
  - 중재자의 신뢰성 문제
  - 중재자를 유지하고 관리하는 추가 비용 소요
  - 추가적인 통신 지연
  - 모든 프로토콜 수행에 관여하여야 하므로 병목현상 발생 가능
  - 좋은 공격 대상
- 중재 프로토콜의 중재자는 인라인 또는 온라인 TTP이며, 판결 프로토콜의 판결자는 오프라인 TTP임

표기법

표기	의미	
A, B, C, S	A, B	일반 참여자
	C	공격자
	S	신뢰하는 서버
CA	인증기관	
K_{AB}	A와 B간에 공유한 대칭키	
$+K_A$	A의 공개키	
$-K_A$	A의 개인키	
T_A	A가 기록한 타임스템프	
N_A	A가 생성한 nonce	

표기	의미
$H(M)$	메시지 M에 대한 해시값
$\{M\}.K$	암호키 K를 이용한 메시지 M의 암호화
$\text{Sig}.K(M)$	키 K를 이용한 메시지 M에 대한 서명
$\text{Sig}.A(M)$	사용자 A의 메시지 M에 대한 서명
$\text{MAC}.K(M)$	메시지 M에 대한 키 K를 이용한 MAC값
$M_1 \| M_2$	메시지 M_1과 메시지 M_2의 비트 결합
$\oplus$	비트 XOR
Cert_A	A의 공개키 인증서

암호프로토콜과 암호알고리즘

- 암호알고리즘은 암호프로토콜의 각종 요구사항을 충족하기 위해 사용하는 가장 기본적이며 핵심 도구임
- 따라서 암호알고리즘의 안전성은 암호프로토콜의 안전성에 큰 영향을 주는 것은 당연함
- 하지만 암호프로토콜을 설계할 때에는 이상적인 암호알고리즘의 존재를 가정함
  - 이상적인 암호알고리즘이란 각 암호알고리즘이 만족해야 하는 요구조건이 무조건적으로 만족된다는 것을 의미함
  - 프로토콜 설계의 복잡성을 줄이기 위함
  - 하지만 실제 어떤 수준의 안전성이 요구되는지는 구체적으로 기술되어야 하며, 암호프로토콜을 구현할 때에는 반드시 고려되어야 함

암호프로토콜 설계 절차

- 설계 순서
  - 달성 목적을 포함한 프로토콜의 정의
    - 수행 환경, 참여자, 사용하는 장치 등
  - 요구사항 분석
    - 암호프로토콜마다 달성해야 하는 목적이 다르기 때문에 만족해야 하는 요구사항도 다름
  - 가정 분석
    - 참여자에 대한 가정: 참여 빈도, 신뢰 관계 등
    - 환경에 대한 가정: 네트워크, 장치, 이동성 등
    - 공격자에 대한 가정: 공모 공격 포함
  - 설계
  - 증명: 요구사항이 충족됨을 증명해야 함
  - 구현
    - 소프트웨어 구현 결과에 대한 검증이 반드시 필요함

프로토콜의 가정

- 참여자에 대한 가정
  - 보통 일반 참여자들은 서로를 신뢰하지 않음
  - 신뢰 서버는 보통 서버가 수행해야 하는 모든 기능에 대해 신뢰 있게 행동한다고 가정함
    - 응용에 따라 특정 행동에 대해서만 신뢰하는 경우도 있음
- 공격자에 대한 가정
  - 가정 1. 프로토콜을 통해 교환되는 모든 메시지를 확보할 수 있음
  - 가정 2. 프로토콜의 진행을 방해할 수 있음. 예를 들어 메시지를 변경, 삽입, 차단할 수 있으며, 다른 목적지로 전달할 수 있음
  - 가정 3. 프로토콜에 정상적으로 참여할 수 있는 사용자일 수 있으며, 제3자일 수 있음
  - 가정 4. 오래된 세션키는 공격자에게 노출될 수 있음
    - 세션키 간의 독립성이 필요함
  - 가정 5. 공모 공격을 할 수 있음

키 확립 프로토콜

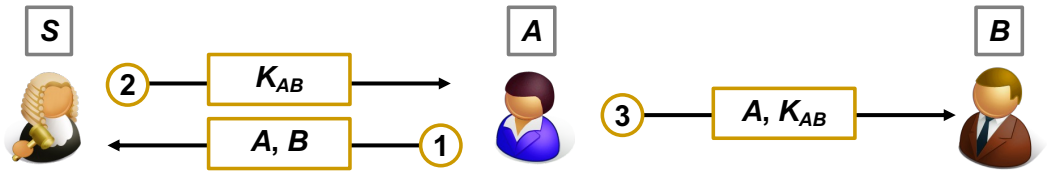
- 두 사용자가 대칭 암호알고리즘을 사용하여 원격에서 메시지를 비밀스럽게 교환하기 위한 전제조건은? 동일 대칭키의 공유
  - 가장 안전한 방법. 오프라인(직접 만나서)
  - 대칭키의 교환도 네트워크로 하고 싶다. How? 키 확립 프로토콜

- 키 확립 프로토콜(key establishment protocol): 둘 이상의 참여자 간의 비밀키를 공유할 수 있도록 해주는 암호프로토콜
- 키 확립 프로토콜을 통해 얻어지는 비밀키는 단일 세션에 사용하기 위한 세션키(session key)임
- 세션키를 사용하는 이유
  - 이유 1. 같은 키로 암호화된 암호문을 제한 (암호해독공격에 대한 방어)
  - 이유 2. 키가 노출되었을 때 누설되는 정보의 양을 제한
  - 이유 3. 키 저장에 관한 안전성 문제를 해결 (세션키는 메모리에 유지함)
  - 이유 4. 세션 간 또는 응용프로그램 간의 독립성 제공

키 확립 프로토콜의 요구사항

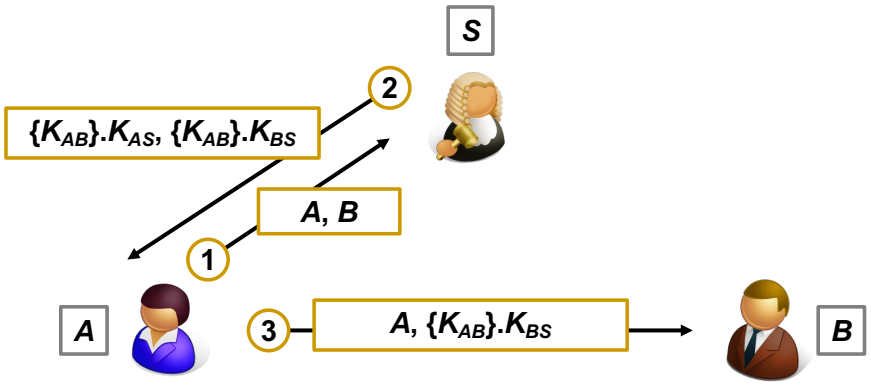
- 요구사항 1. 참여자는 의도한 다른 참여자들과 사용할 수 있는 새로운 키를 얻어야 함
  - 의도한 참여자
    - 키의 용도(누구와 사용하기 위한 키인지)를 확인할 수 있어야 함
    - 참여자의 인증(entity authentication)이 필요할 수 있음
      - 키 생성자 인증: 키를 누가 생성하였는지, 누가 보냈는지 확인할 수 있어야 함
  - 키 확인: 다른 참여자도 나와 같은 키를 가지고 있다는 것을 확인할 수 있어야 함
    - 참여 상대(키 생성 주체와 다른 경우)에 대한 인증이 필요할 수 있음
  - 키의 최근성(key freshness): 키가 최근에 생성(이 요청을 위해 생성)되었다는 것을 확인할 수 있어야 함
- 요구사항 2. 새로운 키는 의도된 참여자 외에 다른 참여자들은 얻을 수 없어야 함 (키의 비밀성)

첫 번째 시도



- 중재 서버를 사용하는 키 전송 프로토콜
- 문제점 1. 공격자는 도청하여 세션키를 얻을 수 있음
- 문제점 2. Alice와 Bob은 K_{AB}가 둘 간의 공유하기 위해 생성된 키인지 알 수 없음 (키 용도)
- 문제점 3. Alice와 Bob은 K_{AB}의 최근성을 알 수 없음
- 문제점 4. Alice와 Bob은 K_{AB}를 서버가 생성하였다는 것을 확인할 수 없음
  - 생성 주체에 대한 인증 기능이 없음
- 문제점 5. 서로 같은 키를 가지고 있는지 확인 할 수 없음

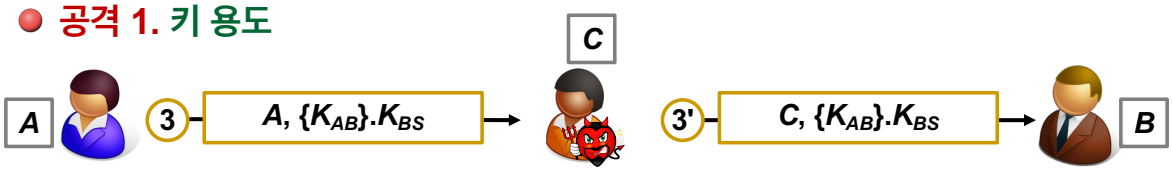
두 번째 시도



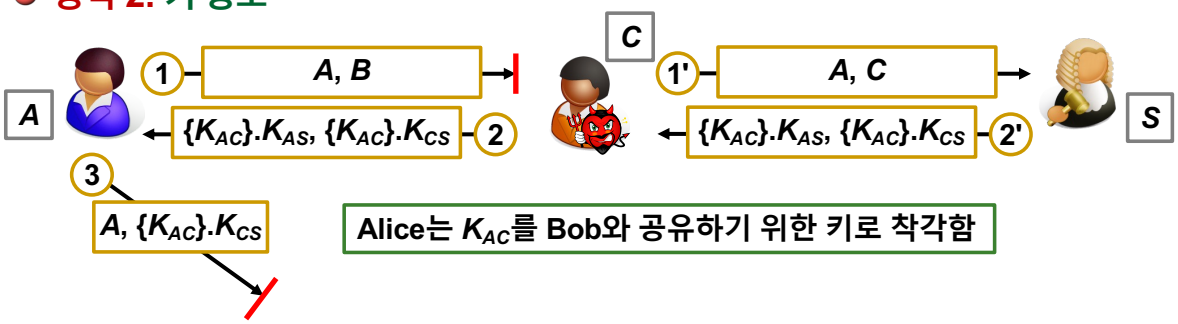
- K_{AS}: Alice와 서버 간에 공유하고 있는 장기간 대칭키
  - 이 키는 최초에 어떻게 확립?
- K_{BS}: Bob와 서버 간에 공유하고 있는 장기간 대칭키
- 첫 번째 시도 프로토콜에서 문제점 1만 해결됨 (키의 비밀성만 보장)

두 번째 시도에 대한 공격 (1/2)

- 공격 1. 키 용도

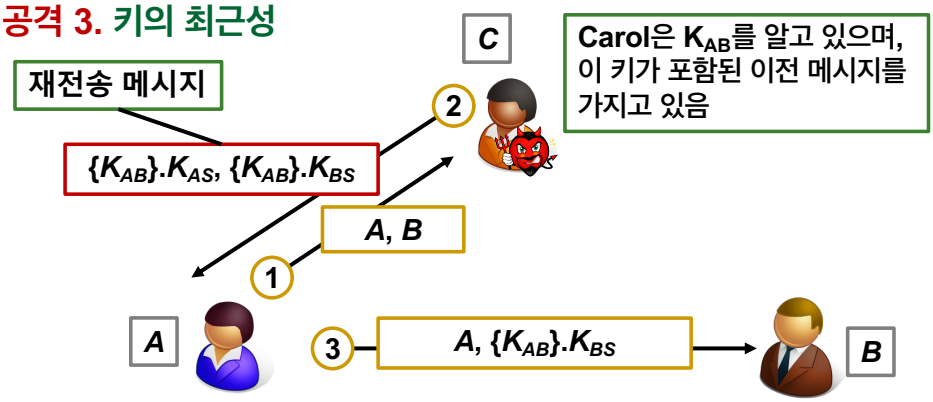


  - Bob은 K_{AB}가 Carol과 공유하기 위한 키로 착각함
- 공격 2. 키 용도



두 번째 시도에 대한 공격 (2/2)

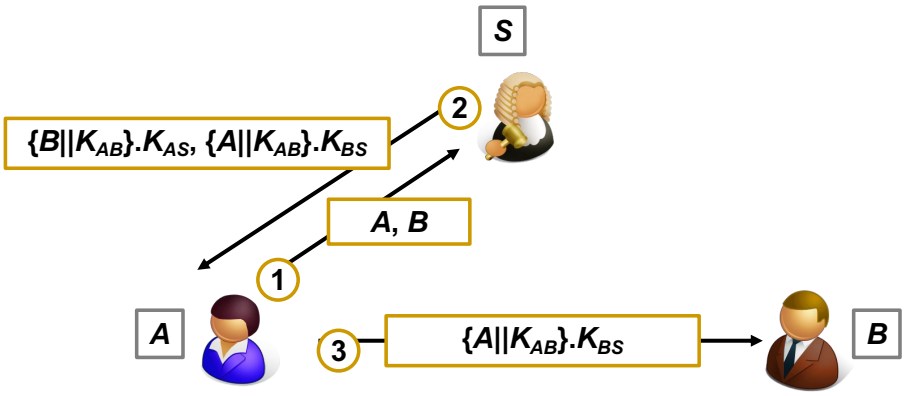
- 공격 3. 키의 최근성



이처럼 이전에 사용되었던 메시지를 이용하는 공격을 재전송 공격(replay attack)이라 하며, 노출된 암호키를 이용하여 공격하는 것을 기지키 공격(known key attack)이라 함

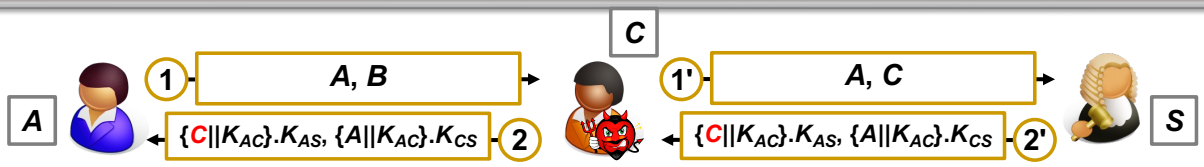
공격 결과: Alice와 Bob은 이미 노출된 이전 세션키를 다시 사용하게 됨

세 번째 시도

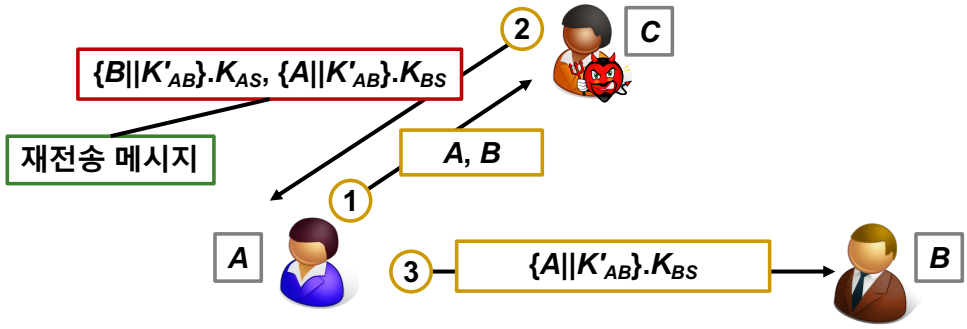


- 문제점 1 뿐만 아니라 문제점 2와 4까지 해결함
- Alice와 Bob은 *S*를 신뢰하므로 수신한 암호문을 복호화하여 포함되어 있는 식별자를 확인함으로써 키의 용도를 알 수 있음 (Naming 기법)
- 복호화하여 식별자를 확인함으로써 암호문을 누가 생성하였는지 확신할 수 있음 (여분 정보)
- 여전히 키의 최근성은 보장하지 못함

세 번째 시도에 대한 공격



키의 용도를 확인할 수 있으므로 두 번째 시도에 대한 공격 1과 2와 같은 형태의 공격은 가능하지 않음



키의 최근성이 보장되지 않으므로 두 번째 시도에 대한 공격 3과 동일한 공격이 가능함. 즉, 재전송 공격, 기지 키 공격이 가능함

여기서 잠깐. 블록 암호의 특징

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

B	K_{AB}

⇩

C_1	C_2

⇩

B	K_{AB}

```
struct sessionKeyMsg{
    char id[16];
    byte key[16];
};
```

- 가정 1. 128bit(16byte) 블록 암호 방식 사용
- 가정 2. ECB 모드 사용 (각 블록을 독립적으로 암호화)
- 가정 3. 채우기는 사용하지 않음

블록 암호의 특성
- 엉뚱한 키로 복호화하면 그 결과는 예측할 수 없음
- 평문의 일부만 깨질 수 없음

B_L		B_R	

$\{C||K_{AC}\}.K_{AS}$

C	K_{AC}

⇩

C'_1	C'_2

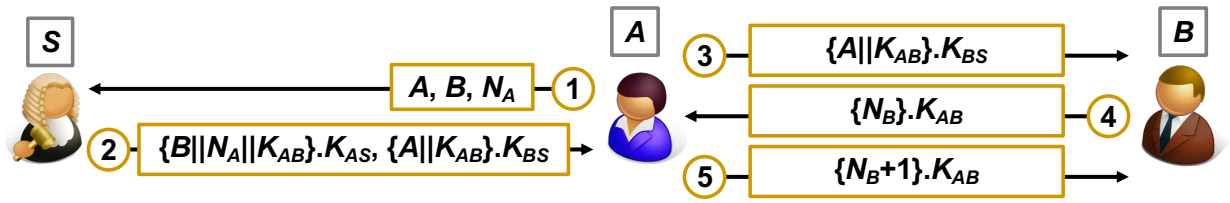
- C_1을 복호화한 결과 B를 얻으면 이 암호문은 K_{AS}로 암호화한 암호문임을 확신할 수 있음
- C_2는 랜덤한 값을 암호화한 것이기 때문에 같은 확신을 할 수 없음
- C_2는 독립적인 블록이기 때문에 C_1을 통해 얻은 확신을 C_2에 적용할 수 없음
- 공격자는 C_2를 임의 랜덤 블록으로 바꾸더라도 C_1을 C'_1로 바꾸더라도 그것을 A가 알아챌 수 없음
- B와 K_{AB}가 함께 암호화되어 있기 때문에 K_{AB}의 용도를 알 수 있다는 것은 정확한 설명이 아님
- 이 때문에 인증 암호화의 사용이 필요함

C_1	C'_2

C'_1	C_2

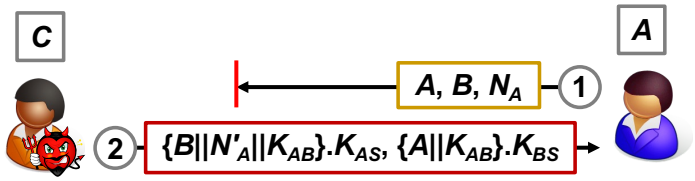
C_1	R

네 번째 시도

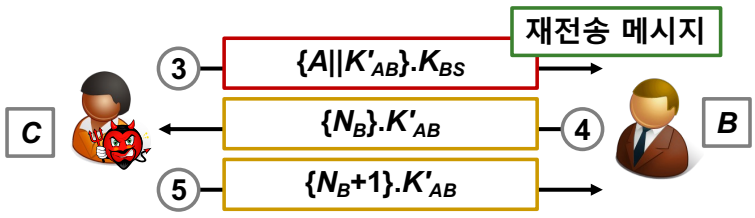


- 여기서 N_A는 난스로 Alice가 처음 사용한 랜덤 수임
- Alice는 메시지 2의 첫 암호문을 복호화하여 이 값을 확인하면 이 암호문이 이번 요청을 위해 생성한 메시지(메시지의 최근성)임을 확인할 수 있음
  - 서버를 신뢰하므로 추가로 K_{AB}의 최근성을 확신할 수 있음
- 하지만 Bob은 K_{AB}의 최근성을 확인할 수 없음
- 지금까지 시도와 달리 키 확인 과정을 포함하고 있음 (메시지 4,5)
  - 하지만 완벽하지 못함. Bob은 Alice가 자신과 동일한 키를 가지고 있다고 확인할 수 있지만 Alice는 그렇지 못함
- 메시지 5에서 "+1"은 메시지 5와 메시지 4를 구별하기 위한 수단임

네 번째 시도에 대한 공격



- A에 대한 기지 키 공격: 난스 때문에 성공할 수 없음



- B에 대한 기지 키 공격
  - 공격 결과: Bob은 이전에 사용한 세션키를 Alice와 사용하기 위한 새 세션키로 착각하게 됨. 실제로 Bob은 Carol과 새 세션키를 확립하게 됨

다섯 번째 시도

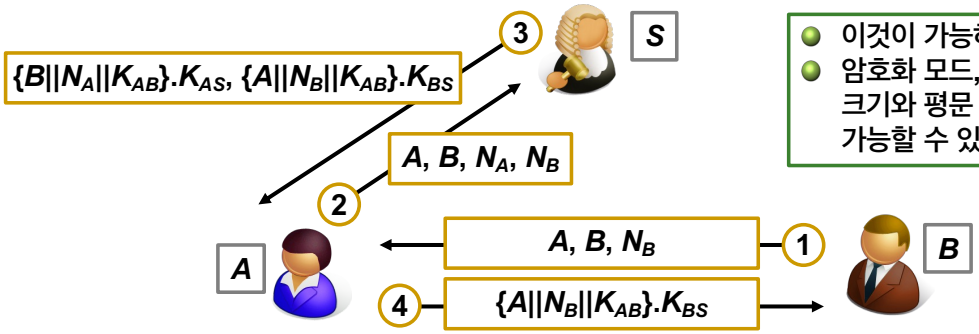
$\{B||N_C||K_{BC}\}.K_{CS}$, $\{C||N'_B||K_{BC}\}.K_{BS}$

$\{A||N'_C||K_{AC}\}.K_{CS}$, $\{C||N'_A||K_{AC}\}.K_{AS}$

⇩

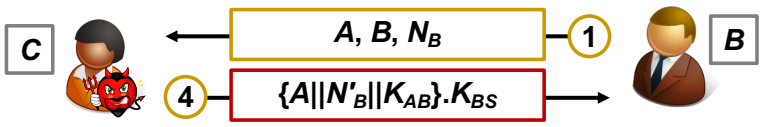
$\{B||N_A||K_{AC}\}.K_{AS}$, $\{A||N_B||K_{BC}\}.K_{BS}$

- 이것이 가능하면 공격이 가능함
- 암호화 모드, 평문 각 요소의 크기와 평문 내부 구조에 따라 가능할 수 있음

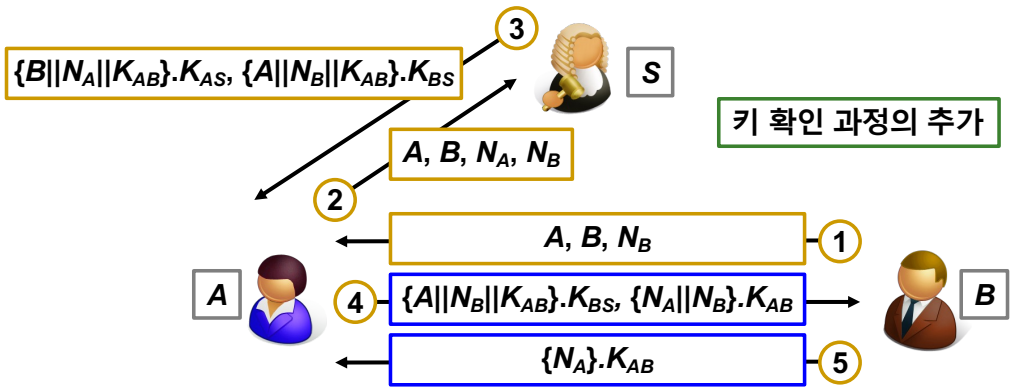


- 이 프로토콜은 지금까지 살펴본 모든 공격에 대해 강건함
- 네 번째 시도와 달리 키 확인 과정이 없음
  - 하지만 추가하는 것은 어렵지 않음
- 그러면 이 프로토콜은 안전한 프로토콜인가? Maybe/Maybe Not

다섯 번째 시도에 대한 공격



재전송 메시지/기지 키 공격 난스 때문에 성공할 수 없음



키 확인 과정의 추가

암호화의 용도 (1/3)

- 비밀성을 보장하기 위해 사용
  - 누가 복호화할 수 있는지가 중요
  - 예) $\{M\}.+K_A$: Alice만 이것을 복호화할 수 있음
  - 예) 키 확립 프로토콜 예에서 $\{B||N_A||K_{AB}\}.K_{AS}$
- 인증을 제공하기 위해 사용
  - 누가 암호화할 수 있는지가 중요
  - 예) $\{M\}.-K_A$: Alice만 이것을 생성할 수 있음
  - 예) $\{M\}.K_{AB}$: K_{AB}가 Alice와 Bob만 알고 있는 비밀키라 하자. 그러면 $\{M\}.K_{AB}$는 비밀성뿐만 아니라 인증을 보장할 수 있음
    - Bob은 $\{M\}.K_{AB}$ 가 K_{AB} 로 암호화된 암호문이고, 이 암호문을 자신이 생성한 적이 없다고 확신할 수 있다면, 이 암호문을 Alice가 생성하였다고 확신할 수 있음
    - 키 확립 프로토콜 예에서 $\{B||N_A||K_{AB}\}.K_{AS}$
      - Alice는 이 메시지는 서버가 생성한 것임을 확신할 수 있음

암호화의 용도 (2/3)

- 메시지의 구성요소를 바인딩 하기 위해 사용
  - 바인딩. 비밀성이 필요하지 않는 것을 함께 암호화하여 비밀성이 필요한 요소에 추가적인 의미를 부여하는 것을 말함
  - $\{X||Y\}.K$를 수신하는 것과 $\{X\}.K$, $\{Y\}.K$를 수신하는 것은 의미가 다름
  - 예) $\{B||N_A||K_{AB}\}.K_{AS}$
    - 이 암호문에서 실제 비밀성이 요구되는 요소는 K_{AB} 뿐임
      - 다른 요소는 K_{AB}에게 어떤 의미를 부여하기 위해 함께 암호화하고 있
      - 이 암호문을 구성한 서버는 K_{AB}가 B, N_A와 연관되어 있다고 주장하고 있음
    - 예와 $\{B||N_A\}.K_{AS}$, $\{K_{AB}\}.K_{AS}$를 비교
    - 예와 $\{C||N_A||K_{AB}\}.K_{AS}$ 를 비교
  - 암호화 모드에 따라 함께 암호화되어 있는 것인지 확신을 못할 수 있음
  - 무결성이 제공되어야 바인딩 효과를 얻을 수 있음 (인증 암호화)
  - 바인딩은 암호화뿐만 아니라 해시함수, MAC 등을 통해서도 제공할 수 있음

암호화의 용도 (3/3)

- 예) $\{B||N_A||K_{AB}\}.K_{AS}$를 아래와 같이 바꾸면 동일한 효과를 얻을 수 있음

  - 암호문을 복호화한 다음에 MAC을 계산하여 K_{AB}의 무결성과 복호화의 정확성을 확인할 수 있음
  - $\{B||N_A||K_{AB}\}.K_{AS}$ 와 차이점
    - 비밀성이 요구되는 정보만 암호화하였음
    - 바인딩은 MAC을 통해 제공하고 있음 (암호화 모드와 무관함)
    - K_{AB}의 무결성을 확인할 수 있음
  - 키 추측 공격이 어려워진 것은 아님
  - 비밀성과 무결성을 동시 제공하고 싶으면 encrypt-then-mac 방식의 인증 암호화가 더 효과적인 방법임

$$C=\{B||N_A||K_{AB}\}.K_{AS}\ ,\ MAC.K'_{AS}(C)$$

인증 암호화

$C=\{B||N_A||K_{AB}\}.K_{AS}\ ,\ MAC.K'_{AS}(C)$

- encrypt-then-mac을 사용한 경우: 태그를 먼저 검증함
  - 수신자가 자신이 생성한 것이 아니라면 생성자가 태그를 생성한 것이라고 확신할 수 있음
  - 태그에 의해 암호문은 태그를 계산할 때 사용한 암호문과 다르지 않다는 것을 확신할 수 있음
    - 전송 과정에서 조작되지 않음
      - 참고. 송신자가 잘못된 평문을 암호화하여 전달할 수 있고, 잘못된 암호문에 대한 태그를 계산하여 전달할 수 있음
  - 상대방을 신뢰할 수 있다면 여분 정보가 없어도 암호문이 어떤 키로 암호화한 암호문인지 알 수 있음
  - 그러나 인증 암호화를 하여도 키 용도나 최근성 확인을 위해 함께 암호화한 식별자나 난스를 생략할 수 없음

대칭 암호알고리즘의 사용

- 대칭키로 암호화의 의미
  - 예) Alice와 Bob만이 공유하고 있는 대칭키 K_{AB}로 Alice가 메시지 M을 암호화하여 전달

    - Bob은 자신이 생성한 메시지가 아니면 $\{M\}.K_{AB}$는 Alice가 생성하였다고 확신할 수 있음 (인증)
    - 오직 Bob와 Alice만 이 암호문을 복호화하여 M을 얻을 수 있음 (비밀성)
- 대칭키 사용의 가장 큰 이슈
  - 대칭키를 어떻게 안전하게 공유할 것인가?
- 대칭/비대칭 암호알고리즘은 기본적으로 무결성을 제공하지 않음
  - 무결성까지 필요하면 인증 암호화

비대칭 암호알고리즘의 사용 (1/2)

- 공개키 암호알고리즘에서 개인키로 암호화
  - 예) Alice가 자신의 개인키 $-K_A$로 메시지 M을 암호화하여 전달

    - Alice의 공개키로 이 메시지를 복호화하는데 성공하면 Bob은 $\{M\}.-K_A$는 Alice가 생성하였다고 확신할 수 있음 (인증)
  - 모든 공개키 암호알고리즘은 개인키/공개키로 모두 암호화하는 기능을 제공하지 않음

비대칭 암호알고리즘의 사용 (2/2)

- 공개키 암호알고리즘에서 공개키로 암호화
  - 예) Alice가 Bob의 공개키 $+K_B$로 메시지 M을 암호화하여 전달

    - 오직 Bob만 이 암호문을 복호화하여 M을 얻을 수 있음 (비밀성)
    - Bob은 누가 이 메시지를 생성하였는지 확인할 수 없음 (인증)
- 두 경우 모두 대응되는 공개키가 확실히 Alice 또는 Bob 것임을 확신할 수 있어야 함 (공개키의 인증이 가장 중요한 이슈)

암호프로토콜의 안전성

- 암호프로토콜은 그것의 보안 요구사항을 모두 충족하였을 때 안전한 프로토콜이라 함
  - 암호프로토콜은 안전한 것만으로는 부족하고 사용되는 응용에서 요구하는 효율성도 충족해야 함
    - 보통 안전성과 효율성(편리성?)은 상반관계임. 그러나 안전성을 희생하는 것이 절대로 용인될 수 없는 경우가 많음
- 암호프로토콜의 안전성을 증명하기 위해 다양한 방법을 사용하고 있지만 쉽지 않음
  - 보통 지금까지 알려진 공격에 대해서만 안전한 경우가 많음
  - 여러 가정 하에서만 증명이 가능한 경우가 많으며, 특정 요구사항 충족에 대해서만 증명이 가능한 경우도 있음
- 암호프로토콜만 안전하다고 하여 전체 시스템이 안전한 것은 아님
  - 서버, 클라이언트 해킹, 서비스 거부 공격 등 프로토콜의 설계를 통해서 방지할 수 없는 위협이 존재함

안전성 증명 (1/2)

- 암호프로토콜에 대한 증명은 보통 프로토콜의 보안 요구사항을 고려하여 보안 모델을 세우고, 이 모델에서 증명함
- 보안 모델이란 프로토콜의 안전성을 논하기 위해 세우는 가정들의 집합을 말함
  - 이와 같은 가정 중에 가장 중요한 것은 공격자의 능력임
    - 다른 말로 공격자 모델(adversary model, threat model)이라 함
  - 공격자의 능력에 따라 특정 공격의 성공 가능성이 다름
    - 특정 공격을 논할 때 항상 가정하는 공격자의 능력도 있음
  - 공격자의 능력이 강력할수록 공격이 성공할 가능성이 커지며, 이와 같은 상태에서도 프로토콜이 안전하다고 증명할 수 있으면 프로토콜의 안전성 수준이 매우 높다는 것을 의미함
  - 공모 공격에 대한 검토가 있어야 함
    - 공모 공격: 여러 참여자가 협력하여 공격하는 경우

안전성 증명 (2/2)

- 공격자의 능력
  - 능력 1. 모든 참여자 간의 통신을 제어할 수 있음
    - 공격자는 교환되는 모든 메시지를 관찰할 수 있고, 메시지를 변경할 수 있고, 새 메시지를 삽입할 수 있고, 메시지를 차단할 수 있음
  - 능력 2. 지난 프로토콜 수행을 통해 확립된 세션키를 얻을 수 있음
  - 능력 3. 공격자가 참여자의 장기간 키를 알고 있음
- 공격자의 종류
  - 제3의 공격자와 내부 공격자(프로토콜에 정상적으로 참여할 수 있는 자)로 나눌 수 있고, 내부공격자에 대해서는 다음과 같은 가정을 할 수 있음
    - 가정 1. 악의적인 내부자의 존재: 프로토콜의 적법한 참여자가 악의적인 행동을 할 수 있음
    - 가정 2. 정직한 내부자: 적법한 참여자는 항상 정직하게 행동함
  - 보통 가정 1의 공격자를 가정하고 프로토콜의 안전성을 논함

프로토콜의 효율성 (1/3)

- 두 가지 측면에서 효율성을 고려함
  - 계산 효율성(computational efficiency): 프로토콜의 참여자가 프로토콜을 완료하기 위해 계산해야 하는 양에 의해 측정됨
  - 통신 효율성(communications efficiency): 메시지의 수와 각 메시지의 크기에 의해 측정됨
- 계산 효율성
  - 사용하는 암호기술에 의해 결정됨
    - 예) 공개키 암호알고리즘은 비밀키 암호알고리즘에 비해 상대적으로 많은 비용이 필요함
  - 공개키를 사용하는 프로토콜은 공개키 연산을 최소화하는 것이 필요함
  - 공개키 연산의 경우에는 개인키를 이용하는 것인지 공개키를 이용하는 것인지에 따라 비용 차이가 있음. 예) RSA: 개인키 > 공개키
  - 기술의 발달에 따라 효율성이 변할 수 있음

프로토콜의 효율성 (2/3)

- 통신 효율성
  - 필요한 라운드(round)의 수를 줄이는 것이 가장 효과적임
  - 정의. 프로토콜의 한 라운드는 한 시점에서 병렬로 전달할 수 있는 모든 메시지를 포함함
  - 메시지가 서로 독립적이지 않으면 서로 다른 라운드에 포함됨
    - 메시지가 이전 메시지의 구성 요소를 포함하면 이 두 메시지는 서로 의존하는 메시지임

프로토콜의 효율성 (3/3)

- 단말의 특성에 따라 효율성이 매우 강조되는 경우도 있음
  - 예) 이동 휴대 단말
- 무선 통신 기반 이동 휴대 단말은 전원이 매우 중요한 자원임
  - 전원 소모를 줄이는 것이 매우 중요함
  - 전원 소모에 가장 많은 영향을 주는 것
    - 무선 메시지 전송, 무선 메시지 수신, 내부 연산, (화면 처리)
- 내부 연산을 줄이는 방법
  - 공개키 연산과 같이 연산 비용이 높은 연산을 최소화함
  - 사전 계산(pre-computation)이 가능하면 사전 계산(온라인이 아니라 고정 전원을 사용할 때 미리 계산함)을 활용함
    - 사전 계산을 하면 저장 공간 요구가 높아짐
  - 모바일 단말에서 해야 하는 연산을 서버로 옮김