**정보보안 개요**

1.보안에서 중요한 3가지 측면(CIA)  
1.Confidentiality(기밀성) : prevent unauthorized reading of information  
(정보를 허가 받지 않는 사람이 읽는 걸 막아야함)

2.Integrity(무결성) : prevent unauthorized writing of information  
(허가받지 않은 사람이 정보를 쓰는 걸 막아야함)

3.Availability(가용성) : Data is available in a timely manner when needed

(항상 서비스가 필요할때 마다 항상 이용가능해야함)

2.그외 보안과 관련된 측면  
①.Acess Controll(접근 제어)

①-1.Authentication(인증)

-네트워크가 수반되지 않는 경우 -Bob의 컴퓨터에서 Bob인지 Trudy인지 구분  
-Passwords

-Biometrics and other

①-2. Authorization(인가)

-Access Control Lists and Capabilities

-Bob이 AOB에 인증햇을때 밥의 행동들을 제한해야함

※Access Control은 authentication과 authorization 둘 다 포함

②non-repudiation(부인봉쇄)  
-Bob이 Charile에게 자동이체를 했는데 언제 자동이체를 했냐고 발뺌을 하는것을 방지  
-나중에 계약을 발뺌하는 것을 방지  
-offline에선 인감을 통해 부인봉쇄함  
-★online에선 전자서명을 통해 부인봉쇄야해줘야함

3.SoftWare Issue  
①secure coding  
-위의 보안관련된 측면들은 소프트웨어 속에서 구현됨  
-대부분 소프트웨어는 복잡해지고 버그가 많아지고 있음  
-그럴 경우 소프트웨어 상의 결함은 보안상 결함으로 발전하게 됨  
-어떻게 소프트웨어 상의 결함을 줄이느냐가 관건

②malware  
-몇몇 소프트웨어들은 인위적으로 사악한 소프트웨어가 있을 수 있음(Malware: viruses, worms 등등)  
-aclice와 bob입장에선 어떻게 malware로 부터 보호해야하는가? -trudy입장에선 어떻게 malware를 효과적으로 사용할 수 있는가?

③OS  
-OS도 점차 복잡해지는 이에 대한 보안을 강화해줘야함

**정보보안의 전반적인 구성요소**

1.Cryptography(암호학)  
-"secret codes"  
①symmetric key crytograhphy(대칭키암호)  
AES 암호알고리즘  
②public key crytograhphy(공개키암호)  
RSA 암호알고리즘  
③hash function(해쉬암호)  
SHA1 hash function

2.Access control  
①Authentication  
-passwords  
-biometrices

②Authorization  
-Access Control List 자원(file)을 중심으로 해서 누가 접근할 수 있느냐  
사람이 중심으로 해서 누가 접근할 수 있느냐  
-Firewall

3.Protocols  
-인증과 관련된 부분에서 다양한 프로토콜사용  
-프로토콜의 특성으로 조금만 변경만해도 security 측면에선 치명적효과를 가져올 수 있음(butterfly effect)  
-프로토콜에서 암호학이 사용됨  
-★SSL, IPSec, Kerberos, GSM security

4.Software  
-security-critical flaws : 소프트웨어의 결함이 보안적측면에서 치명적일지  
ex)Buffer overflow

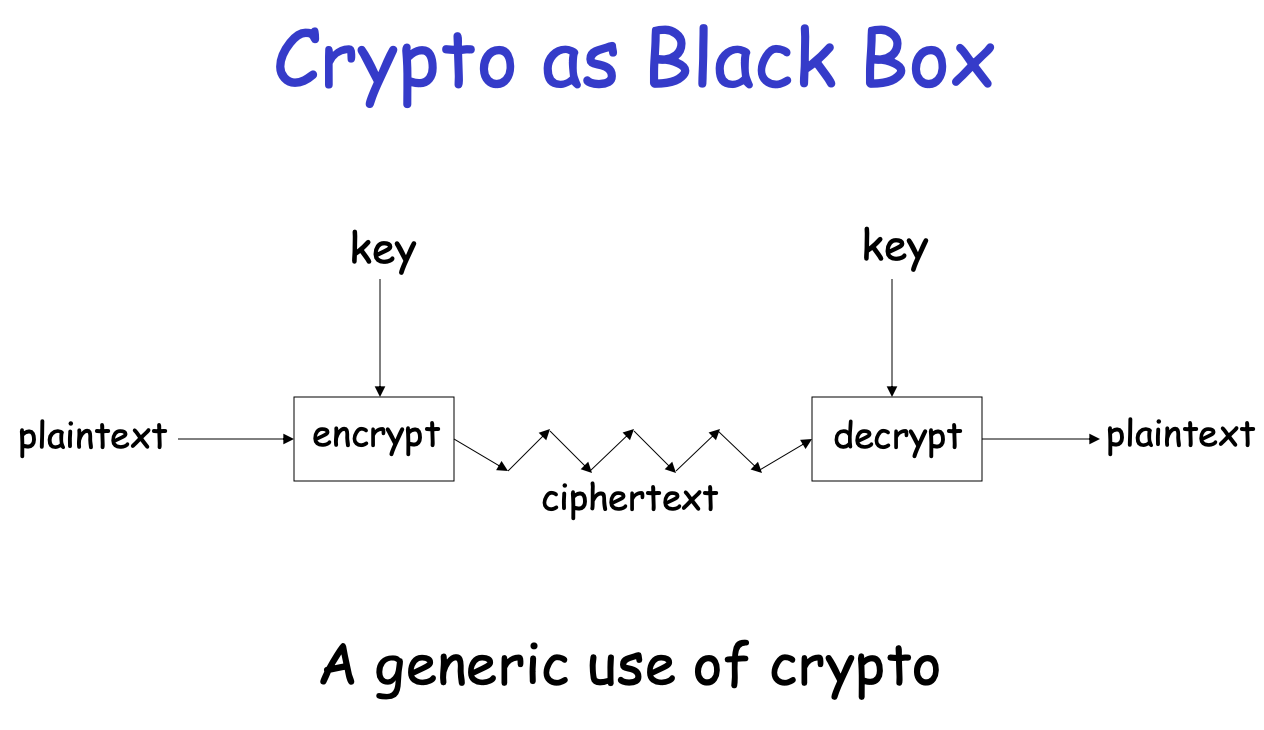
-Malware : viruses 나 worms들을 어떻게 방지하고 탐지해낼 것 인가  
-SRE(soft ware reverse engineering) : binary -> source  
-DRM(digital rights management) :mp3 , e-book , 완벽하게는 어려움이 있음 mp3를 틀고 핸드폰으로 녹음할수 있기에  
-OS

**<2-1-Crypto-대칭키암호>**

- 암호의 두 가지 방식

1)Symmetric Key Cryptography(대칭키암호) – DES, 3DES, AES  
-암호할때 사용하는 키랑 복호할때 사용하는 키가 같음

2)Public Key Cryptography(공개키암호,비대칭키암호)  
-암호할때 사용하는 키랑 복호할때 사용하는 키랑 다름 (publick-key : encrypt, private-key : decrypt)



-사용자가 보낸 plaintext데이터를 key를 사용하여 encrypt(암호화)하여 보내면 ciphertext형식으로 네트워크를 통해 전달되서 수신자 측에서 key를 사용하여 decrypt(복호화)하여 데이터를 얻어냄

**1)Symmetric Key Cryptography(대칭키암호) – DES, 3DES, AES**

-Feistel Cipher암호알고리즘: 특정한 암호알고리즘이 아닌 block cipher를 설계하는 방식

-DES (DataEncrypt Standard) (key size : 56bit)

·6bit를 4비트로 subtitution하는 것에 보안적 성능이 있다.

-2^56번의 전수조사로 키를 알아내어 이를 깰 수 있는데 이를 대체하기 위해 두 가지 등장

1)3DES ( DES와 호환성 유지하여 만듬 )

2)AES ( 아예 새로 갈아 엎어서 만듬 )

-3DES

(key size : 112bit, key 두 개 사용, EDE 세번의 과정을 진행하므로 속도가 3배정도 느려질 수 있다.)

· C = E(D(E(P,K1),K2),K1)

· P = D(E(D(C,K1),K2),K1)

·Why use Encrypt-Decrypt-Encrypt (EDE) with 2 keys?

-> Key를 하나 사용해서는 위의 EDE식이 성립 불가능하다.

-> E(D(E(P,K),K),K) = E(P,K)

·Why not C = E(E(P,K),K) ?

->똑같이 2^56번만 전수 조사하면됨

·Why not C = E(E(P,K1),K2) ?

->known plaintext attack이 존재하게 된다. C를 얻으면 P를 얻을 수 있기에 공격에 용이하기 쉽다.

->알려진 평문과 그 평문에 대한 싸이퍼 텍스트를 알 수 있으면 여러 번의 반복작업을 통해 Key를 알 수 있다.

-> E(E(P,K1),K2) 를 통해서 다음과 같은 식이 성립하게 된다.

->E(P,K1) = D(C,K2)

->P와 C는 공격자가 알 수 있다.

-> 2^56개의 key에 대한 모든 가능한 조합을 가진 테이블을 만든 다음에 K1, K2에 대해서 위의 식이 성립하는지 하나씩 넣으면서 공격하게 되면 K1, K2에 대해 모두 알 수 있게 된다.

-> 암호화, 복호화 : 2^56 + 2^56 = 2^57 번만 하면 된다.

-> 따라서 약 최대 2^112번의 시도 후에 이를 깰 수 있게 된다.

-AES (Key Size : 128,192,256 bit)

-> 새로운 것으로 갈아엎음(실생활에서 가장 많이 사용됨!!!보안우수)

-> Round Function을 이용한 반복적인 block단위 암호화 (like DES)

-> Feistel Cipher가 아니다 (unlike DES)

-> Block size: 128, 192 or 256 bits

-> Key length: 128, 192 or 256 bits

-> 키 사이즈와 블럭 사이즈는 독립적!

->키의 길이에 반복하는 횟수가 결정된다.

->키의 길이가 128비트면 10round, 192면 12round, 256이면 14round

->각 라운드는 4가지의 function으로 구성됨 (4가지 모두 역함수가 존재 -> 복호화가 가능하다)

1. ByteSub (nonlinear layer)

·AES의 S-BOX

2. ShiftRow (linear mixing layer)

·행에 대해 byte단위로 왼쪽으로 이동

3. MixColumn (nonlinear layer)

· 열에 대해 행렬을 곱해서 새로운 열을 만들어냄

4. AddRoundKey (key addition layer)

·Key와 exclusive-or를 통해서 block을 새로 만들어냄

·Key는 스케줄알고리즘에 의해 매 라운드마다 생성 됨

-Block Cipher Modes

->plain text는 여러 개의 블록으로 구성 됨

->블록들을 순서대로 독립적으로 암호화하게 된다 치면 보안상 취약해짐

->여러 개의 블록들을 어떻게 암호화 할 것이냐?

1.Electronic Codebook (ECB) mode

->순서대로 블럭을 독립적으로 암호화하는 방법

->치명적인 약점을 가지고 있음

->Cut and Patste 공격 방법이 존재함!

->각 블록을 암호화하여 C0,1,2,3을 네트워크를 통해 전달하는 과정에서 트루디가 C1하고 C3의 순서를 바꿔 전달하게 되면 원래의 plain text값이 변조 되게 됨

->독립적으로 암호화하기 때문에 똑같은 블록이면 똑같은 암호문이 나오게 됨!

->트루디가 암호문을 도청하여 이를 통해 평문도 같겠구나 생각할 수 있게 됨

->예를 들어 사진을 네트워크를 통해 전송시 사진의 윤곽선을 도출 해 낼 수 있음

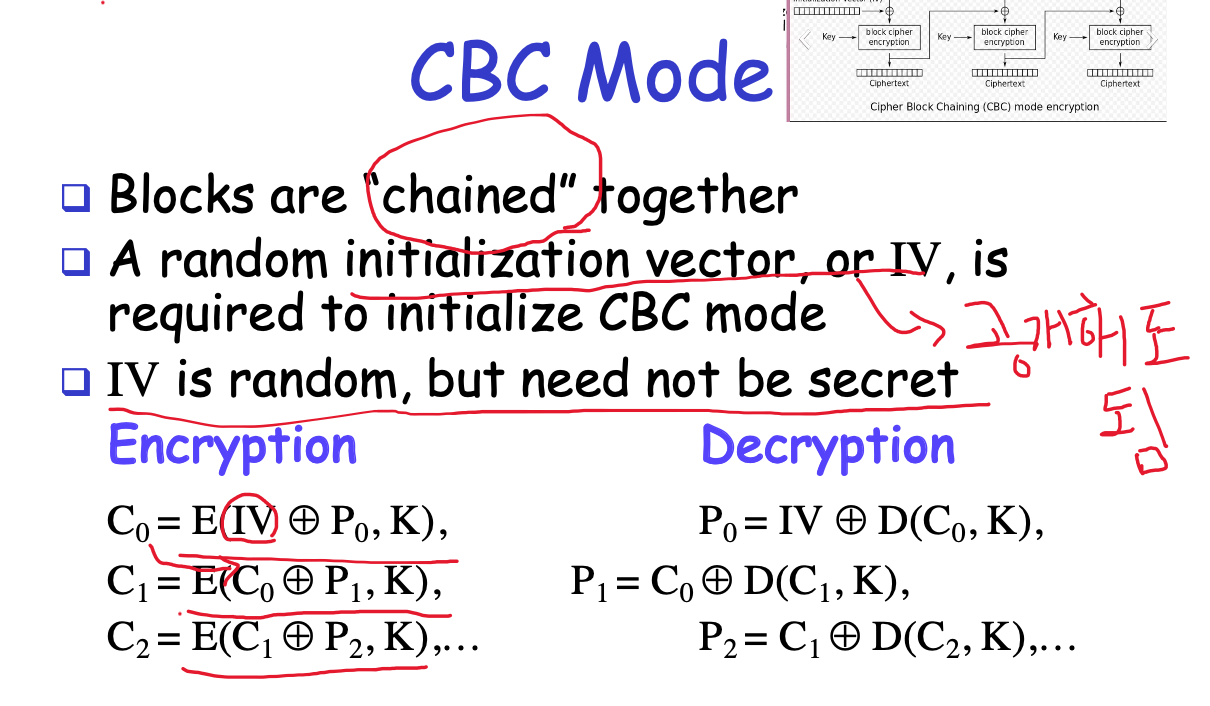
2.Cipher Block Chaining(CBC) mode - ★random access 가능, 에러 전파 없음 ★

-> ECB보다 더 안전하고 추가적인 작업이 들어가지 않는다.

-> 블록들을 Chain형태로 서로 의존시켜 암호화하는 방법

->초기벡터 존재(initialization vector), random값이며 기밀스럽게 유지할 필요없다

-> 초기벡터와 첫번째plain text를 exclusive-or 후 key를 넣어서 실제로 암호화해서 블록을 만든 후 이를 두번째 plain text의 초기벡터역할을 하여 동일한 작업을 반복하여 수행하게 된다. 이를 통해 체인을 형성하는 식의 블록 암호화가 이뤄진다.



->이전 암호문과 exclusive-or작업을 거치기에 동일한 plain text도 다른 암호문이 나온다.

->C1이 G로 바뀌어도 복호화할때 Error Propagation(에러 전파)이 없다. (딱 두개만 영향을 주게 됨)

->같은 평문이어도 다른 암호문이 나오게 된다!

-> random access : 암호문에 대해서 랜덤하게 접근해도 이후부터 쭉 복호화를 수행하여 얻을 수 있다.

-> C0,1이 있으면 P1을 구할 수 있기에 C1을 랜덤하게 접근해도 C0를 알 수 있기에 Random access가 가능하다!

-> 체인으로 연결되어 있어도 임의 지점 부터 복호화가 가능하다.

3.Counter Mode (CTR) mode - ★ random access 가능 ★

-> stream cipher와 비슷하며 random access에 많이 활용됨

-> 1씩 증가하는 Counter(Nonce값도 넣을수 있음)와 Key를 암호화한 후에 평문과 exclusive-or를 수행

-> C0,C1,C2,C3 가 있을때 IV는 공개되있기에 C2로 랜덤하게 접근해도 평문이 얻을 수 잇어 random access가 가능하다.

**<2-1-Crypto-공개키암호>**

**2)Public Key Cryptography(공개키암호, 비대칭키암호) - RSA**

-> 인터넷 뱅킹, 인터넷 주식 거래 등이 가능(공인인증서)

-> 수학적 연산에 많은 기반을 두고 있음 ( 주로 나머지 연산, modulus)

-> 두 개의 키가 존재! (공개키, 개인키)

-> 송신자가 암호화하기 위해 수신자의 공개키를 사용

-> 수신자가 복호화하기 위해 자신의 개인키를 사용

-> 단 방향으로는 계산하기 쉽지만 다른 방향으로는 계산하기 어렵다 (trap door, one-way-function에 기초를 하고 있다)

->대표적 one-way-function : 소인수 분해

->소수 p, q에 대해서 -> p\*q = n을 구할 수 있지만 반대로 구하기엔 너무 어렵다

->용도 : 암호 및 전자서명

1. 암호

- 밥의 공개키를 사용해서 암호화하고 밥의 개인키를 풀어서 복호화

2. 전자서명(digital signature)

- 자기 자신만의 개인키로 서명을 하게 된다. ( 앨리스가 서명한다하면 앨리스의 개인키로!)

- 검증 할 때는 앨리스의 공개키로 검증하게 된다. (공개키는 공개되있기에 서명 검증은 누구나 할 수 있다.)

- 인터넷 뱅킹, 주식거래를 할 때는 비밀번호를 친다. 이는 개인키를 얻어내어 전자서명을 하게 된다

- ★ 전자서명하는 이유는 부인봉쇄기능을 제공하기 위해서 (서명을 해서 보냈기 때문에 서명 값을 은행이나 증권사가 보관하고 있어 발뺌을 못하게 한다.)★

- 공개키 암호 알고리즘

1) RSA

2) Diffie-Hellman

**1. RSA**

1. 키 생성 ( Key – pair generation )하는 법 (개인키, 공개키)

- p와 q는 아주 큰 소수를 선택

- p와 q를 곱하여 N을 구한다. 이 N은 modulus라 한다. ( 24 mod 5 = 4 , 여기서 5가 모듈러스 )

- 그 다음 (p-1)(q-1)과 서로소인 수 중에 하나를 e로 선택

- 그 다음 ed = 1 mod (p-1)(q-1)을 만족하는 d를 찾는다.

- ed mod (p-1)(q-1) = 1 이렇게 바뀔수가 있다

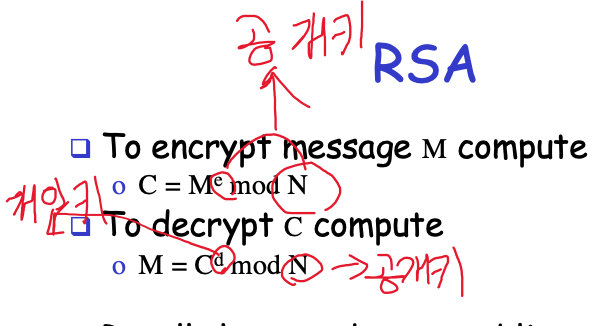
- 12 = 2 mod 5 (12를 5로 나눈 나머지와 2로 나눈 나머지는 같다)

- 곱해서 항등원1이 나오기에 d는 결국 e의 곱셈상의 역원이다!!!

- 여기서 (N,e)는 공개키

- d는 개인키

2. 암호(Encryption)

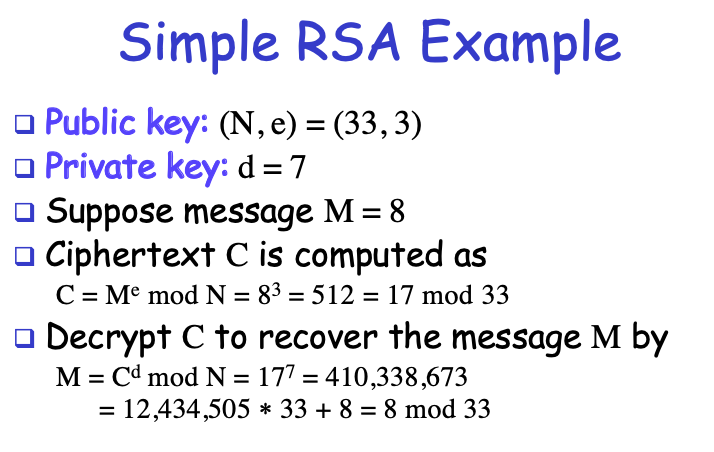


- 공개키인 N과 e를 사용하여 암호문 Cipher Text를 구한다

- 개인키인 d와 N을 사용하여 평문을 구한다

- 인터넷 쇼핑몰에 결제하려면 쇼핑몰의 공인인증서(공개키)를 가져와야 암호화해서 인터넷쇼핑몰에 보낼 수 있다

- 예시)



- 보통 M,d는 2048bit, e는 24bit

- 따라서 암호화 할 때 훨씬 더 속도가 빠르다

- e와 N은 공개키이므로 공격자도 접근이 가능하다. 만약 N을 소인수 분해 할 수 있다면 p와 q를 얻을 수 있어서 d를 얻을 수 있게 된다. 하지만 실제로 소인수 분해는 굉장히 어렵기 때문에 RSA는 소인수분해의 어려움에 보안적 성능을 기반하고 있다

- RSA깨는 방법이 아직까진 소인수 분해 밖에 없다

**- Diffie-Hellman 알고리즘 ( 대칭 키 교환 알고리즘!!!)**

★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★

1. 암호 알고리즘 비교

- 대칭키 암호: 속도가 빠르다, (파일, 이메일, 카톡 등등..) 키 공유가 필요하다

※송신자 ⬄ 수신자 (암복호화 하기 전에 사전에 키 공유가 필요하다)

- 공개키 암호: 속도가 느리다, 키 공유가 필요 없다

※각자가 생성하면 되기 때문에

1) 대칭키 암호 예 :

1. 전화(음성통화) : 가입자 <-> 이통사,

->대칭키는 USIM에 심어져 있고 이통사에서도 가지고 있다. 그러므로 음성 통화시 통화내용이 암호화되서 네트워크를 통해 전달되어진다.

2. 카톡: 가입자 <-> 카톡, 대칭키는 분배 정책에 의해 할당

2) 암호 알고리즘의 키 갯수 비교

※대칭키에서 필요한 키의 개수

Alick, Bob, Charlie : (2\*3) /2 = 3

N명이 존재할 경우 (n-1)n/2 = O(n^2) ( y=x^2의 그래프라 생각하면 됨)

※ 공개키에서 필요한 키의 개수

공개키 암호에서 필요한 키의 개수

N명이 존재할 경우 2n = O(n) , 대칭키보다 훨씬 적다

★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★

- 키를 분배하는 알고리즘

- 암호하거나 전자서명하기 위한 용도가 아님!!!

- discrete log problem에 보안성을 기초(의존)하고 있다!!!

- g, p, g^k mod p가 주어졌을때 k를 찾는건 굉장히 어려운 걸로 알려져 있다

※완전 잉여계

Z6 = {0,1,2,3,4,5}

Z11 = {0,1,2,3,4,5,6,7,8,9,10}

※기약 잉여계 {0과 modulus와 서로소가 아닌 수 제외)

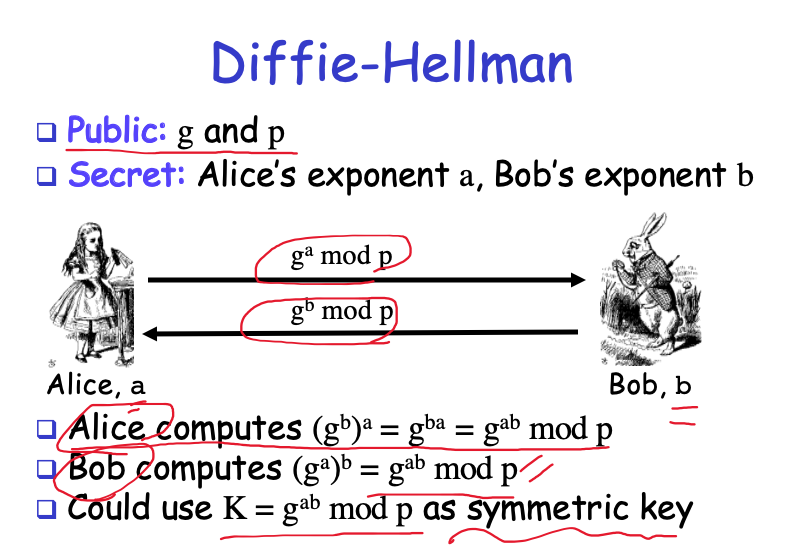
Z6\* = {1,5}

Z11\* = {1,2,3,4,5,6,7,8,9,10}

\* 모든 수를 generated하는 그러한 수를 생성자(generator)라 한다. 지수승을 했을 때 나머지들이 기약 잉여계를 나타낸다.

- 소수인 p와 생성자인 g를 공개를 한다.

- 앨리스는 a를 개인키로 가지고 밥은 b를 개인키를 가지고 있다.



- 트루디가 도청을 해서 g^a 와 g^b 를 곱해도 g ^ a+b가 나오기에 알 수가 없다

- 만약 트루디가 a와 b를 알아버리면 시스템이 깨져버린다

- discrete log problem을 풀면 a, b를 알 수 있지만 이 문제를 푸는 건 굉장히 어렵다

- man-in-the-middle (MiM) 중간자 공격을 받을 가능성이 존재한다

텍스트이(가) 표시된 사진

자동 생성된 설명

- 앨리스와 트루디 사이에는 g^at가 공유되고 밥과 트루디는 g^bt가 공유된다

- 따라서 앨리스가 암호화해서 전달하면 트루디가 다 볼 수 있게 된다

- 앨리스와 밥은 트루디가 중간에 있다는 걸 모른다

- 중간자 공격은 방지할 수 있는 법? -> 없다!

- 반드시 중간자 공격이 존재 할 수 있다는 걸 충분히 인지 해야한다

**- 공개키 암호의 용도**

1. Confidentiality (기밀성 유지를 위해)

-> 인터넷은 같은 공용망은 누구나 접근 할 수 있기에 이를 통해 데이터를 전송할 때 공개키 암호가 필요함

-> 디스크와 같은 비안전한 media에 안전하게 저장하고 싶을 때 공개키 암호가 필요함

2. Authentication (인증을 위해)

-> 인증과 관련된 프로토콜에서 사용됨 추후 설명 예정

3. Digital signature (무결성 및 부인봉쇄를 위한 전자서명을 위해)

-> 대칭키 암호를 사용하게 되면 부인 봉쇄를 제공 할 수 없다

- Notation

텍스트이(가) 표시된 사진

자동 생성된 설명

-> M을 개인키로 암호화하고 공개키로 암호화하거나 공개키로 암호화하고 개인키로 암호화해도 똑같은 M이 나온다

**- Public Key Infrastructure**

- 공개키 암호를 사용하려면 PK라는 기반구조가 필요하다

★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★

인증기관CA

(Certificate Authority)

1. 인증서 발급 받기

Your Computer (공인, 사설) 인증기관

1. 공개키 {N,e} -> 인증기관에 보내게 되있음 -> 인증기관에서 서명 후 인증서형태로 보내줌 (여러분의 컴퓨터에 인증서가 저장되게 되어있음)
2. 개인키 {d} (encrypted by 공인인증서 비밀번호)

2. 왜 인증서가 필요하나? -> 공개키는 기밀성은 필요 없으나 무결성이 요구된다. 따라서 공개키를 인증기관에서 서명을 해서 인증서에 기록 ->만약에 누가 공개키를 수정했을 경우 인증기관에서 한 서명값이 다르게 나오기에 변조가 됐음을 파악할 수 있도록 한다

★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★

- 인증서 – 주체(발급자), 공개키(N,e), 발급자에 의한 서명(발급자의 공개키에 의해 검증이 이뤄질 수 있다)

- CA : trusted 3rd party (신뢰 할 수 있는 제3의 업체)

★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★

Yessign(금융결제원)에서 alice의 인증서에 서명했다는 것은 alice의 신원을 검증했다는 것이다

Alice의 인증서를 소유하고 있는 사람은 Alice인걸로 신원을 파악 할 수 있는가?

No, Bob이 Alice의 인증서를 소유할 수 있으므로

(인증서는 누구에게나 공개하는게 목적이기에 누구에나 공개되므로)

★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★

- PKI : 공개키 암호를 사용하기 위한 모든 필요한 기반 구조(구성요소)!!! 기반 구조 안에 CA가 포함이 된다

- 대칭키 암호는 우리끼리쓰자 이런식으로 가능하지만 공개키암호는 국가 또는 기관에서 기반 구조가 필요하다

역할 3가지 : 키 생성 및 관리, 인증서 서명, 인증서 폐기 관리

**- Confidentiality in the Real World (실생활에서의 기밀성)**

- 대칭키의 장점 : 암복호화 속도가 빠르다, PKI가 필요 없음

- 공개키의 장점 : 전자서명이 가능하여 부인봉쇄 기능제공, 사전에 키를 공유할 필요가 없다

- 실생활에서의 기밀성 제공

-

- hybrid 형태로 제공 (공개키와 대칭키를 혼용해서 사용)

- 공개키로 대칭키를 분배 후 대칭키로 실제 데이터를 암호

- 과정

1. 먼저 Alice가 대칭키 K를 결정하여 Bob의 공개키로 암호화여 Bob에게 전달한다.

2. Bob은 개인키로 풀어서 대칭키 K를 얻어 키 공유가 이뤄진다.

(비록 여기서 트루디가 본다하더라도 밥의 개인키가 있어야하기에 대칭키 K를 얻을 수 없다)

3. 서로 공유 된 대칭키로 메시지를 암복호화하여 커뮤니케이션 한다

- Bob의 입장에서는 Alice인 걸 확인 할 수 없다!!! (트루디도 밥의 공개키로 암호화하여 던질 수 있기 때문에) <- 문제가 될 수 있다

-반대로 Alice입장에선 Bob인지 확신 할 수 없다 (K를 풀 수 잇는 사람은 Bob의 개인키를 가진 사람만 풀 수 있기에)

- 보내는 사람의 신원 확인이 안되는 문제가 발생!!!

**-Hash Function**

-어디다 쓰냐? (전자 서명 할 때 M에 대해 직접 서명 하면 시간이 너무 오래 걸리기에 이 시간을 단축시키기 위해)

· 앨리스가 주식을 매매,매도 할 때 메시지를 전자서명해서 보낸다 가정하자

· 앨리스는 M과 M을 전자 서명한 S를 밥에게 보낸다

· 밥은 S를 앨리스의 공개키로 풀어 나오는 값과 앨리스에게 받았던 M과 일치하는지 비교하여 앨리스가 제대로 전자 서명을 했는지 검증할 수 가 있다

· 하지만 M이 크다 하면 개인키로 암호화하는 과정은 굉장히 시간이 많이 걸린다

· 해결책? -> 대신은 M에 전자 서명하는게 아니라 h(M)에 대해 서명을 한다

· h(M)은 M보다 훨씬 작은 숫자다 (훨씬 시간이 적게 걸린다)

· 밥은 S를 앨리스의 공개키로 풀어 얻은 값과 받았던 M에 해쉬 함수를 적용하여 일치하는지 파악하여 검증!!!!

텍스트이(가) 표시된 사진

자동 생성된 설명

- 해쉬 함수의 조건!

1. Compression - 해쉬 함수의 결과 값은 길이가 작아야 한다

2. Efficiency – 어떤 x여도 상관 없이 계산하기 쉬워야 한다

3. One-way – y라는 값이 주어져 있을 경우에 h(x)=y라 할 때 거꾸로 x를 찾는 건 매우 어려워야 한다

\*collision? -> 다른 input이 해쉬함수를 통해 하나의 output을 가지게 되는 현상

(해쉬 함수에서 무조건 고려해야하는 조건!)

-> 이러한 충돌이 발생하기 마련인데 이를 찾는 건 매우 어려워야만 한다!!!

4. weak collision resistance (보안에서 필요한 조건) – x와 h(x)가 주어졌을 때 x와 동일한 해쉬 값을 가지는 다른 y값을 찾는 건 불가능해야 한다

ex) 트루디가 M과 S를 알 수 있고 앨리스의 공개키로 S를 풀게 되면 h(M)도 알 수 있다 이를 가로채서 M과 동일한 해쉬 값을 갖는 M’을 찾아서 밥에게 던지면 밥은 h(M’)과 h(M)을 일치한다 판단하여 보안 시스템이 깨질 수가 있다

->Pre-Birthday Problem 발생!

· 방에 몇 명 이상 있어야지 나와 동일한 생일을 가진 사람이 있을 확률이 50%가 넘겠느냐?

· Solve: 1/2 = 1  (364/365)N for N

· Find N = 253명!

5. Strong collision resistance (보안에서 필요한 조건) – 동일한 해쉬 값을 가지는 임의의 x와 y를 찾는 건 불가능해야 한다

ex)

원본 메시지 : M1, M2, M3, ….. Mn

위조 메시지 : M’1, M’2, M’3, …. M’n

원본 메시지의 각 M들의 해쉬 값들과 위조 메시지의 해쉬 값들 중 일치하는 걸 찾아서 앨리스가 M1과 [h(M1)]alice를 보낼 때 M’n과 [h(M’n)]alice를 보내면 보안이 깨지게 된다

-> Birthday Problem 발생!

· 방에 같은 생일을 가진 사람이 존재할 확률 50%가 넘으려면 몇명이 있어야하는지?

· 1 - 365/365 \* 364/365 · · · (365-N+1)/365

· Set equal to 1/2 and solve: N = 23명!

- 2^N/2 만큼만 해쉬 함수를 수행하게 되면 충돌을 찾게 된다!!!



- 대칭키는 결국 2^N-1정도하면 키를 찾을 수 이잇는데 해쉬는 2^N정도하면 찾게 될수 있다 따라서 동일한 정도의 보안 강도를 가진다

-AES 128 bit 2^128/2 = 2^127

-Hash 256bit 2^(256/2) = 2^128

\* 대칭키 키길이의 두배정도의 길이를 가져야 비슷한 보안 강도를 가진다

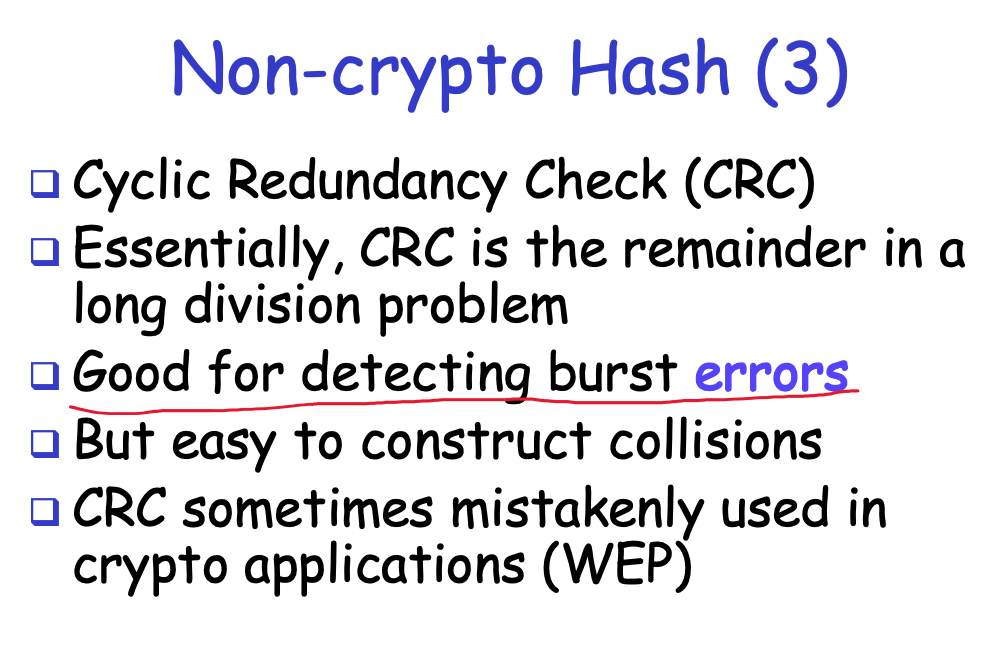
- 보안에서 사용할 수 없는 해쉬 함수의 사례

조류이(가) 표시된 사진

자동 생성된 설명

-> 단순히 메시지들을 더하면 교환법칙에 의해 충돌을 찾기가 쉽다





\* CRC (Cyclic Redundancy Check )

http://blog.skby.net/tag/%EC%88%9C%ED%99%98%EC%A4%91%EB%B3%B5%EA%B2%80%EC%82%AC/

-> 에러발견을 위해 전송. Data뒤에 CRC검출 정보를 붙여 전송하는 기법

- 보안에서 사용할 수 있는 해쉬 함수의 종류

1. MD5

– 128bit output

– 2^64정도의 보안 강도 (DES가 2^56) =>그리 강하지 않음

2. SHA-1

· 160bit output

· 2^80정도의 보안 강도

\* 메시지를 블록 단위로 쪼개서 해쉬 함수가 이뤄지게 되있음

- 해쉬 함수를 설계할 때 고려해야 할 사항

-> Avalanche effect(눈사태 효과) : input에서 한 비트만 변경을 해도 output의 반 정도 영향을 줘야 한다 (Block ciphers설계와 유사)

- 해쉬 함수의 용도

1. Message fingerprint : 세금계산서를 보낼 때 수정이 됐는지 판단 용도

2. Digital signature

3. 그외에 대칭키 암호로 할 수 있는 무언가

4. Online Auction : 온라인으로 제시 금액을 해쉬 값으로 보낸 뒤 나중에 선정할 때 제시한 금액과 해쉬 값이 일치하는지 검증하여 진행

5. Span Reduction : 스팸을 어떻게 하면 줄일 수 있을까

- 암호학 총 정리!!!

조류이(가) 표시된 사진

자동 생성된 설명

**<4-1.AuthenticationProtocol>**

- 인증 프로토콜

1. e-commerce (쿠팡)

2. 이동통신사(SKT)

3. 카카오톡

4. AWS EC2

5. 자체적인 서비스 구축 시

- 보안에서의 프로토콜 ( SSL(웹 보안), IPSec(가상 사설망), Kerberos(윈도우의 보안), etc…)

-> 프로토콜의 결함은 매우 미묘 할 수 있다

-> 잘 알려진 보안 프로토콜들도 심각한 결함을 가지고 있다

-> 구현 상의 오류를 쉽게 찾을 수 있고 그만큼 결함 없는 프로토콜이 되기 위해선 어렵다

-> 그렇다면 이상적인 프로토콜?

1. 보안 관점에서 명확하게 정의된 요구사항을 만족시켜야 한다

2. 효율적이어야 한다 (연산이 너무 오래 걸리면 안된다. 특히 공개키 암호에서 개인 키를 사용하는 연산은 시간이 많이 걸린다, 딜레이 대역폭을 줄여야 한다)

3. 공격자에 의해 깨지기 쉬워서 안된다 (환경이 변화될지라도 잘 동작해야 한다 공격자가 깨려 할 때 잘 막아야한다)

4. 구현하기 쉬워야 하고 유연해야 한다

\* 그러나 이 모든 것을 만족하기엔 쉽지 않다

- 인증

-> 앨리스가 밥에게 그녀의 정체성을 인증해야 한다

-> 반대로 밥이 자기 자신이 밥이라고 증명 해야 할 필요도 있다 (상호인증 mutual authentication)

\* 시스템에 따라 상호 인증이 필요한 경우나 단방향인증만 있으면 되느냐 결정 된다

-> 인증 된 후 세션 키 분배가 이뤄져야 한다

-> 그 외 다양한 요구사항이 존재 할수있음 (공개키, 대칭키, 해쉬함수등 ..)

-> 네트워크를 통해 인증하는 경우에는 공격이 가능하기에 훨씬 복잡하다

-> 어떻게 트루디가 앨리스와 밥사이의 인증과정을 공격할 수 있느냐?

1. 수동적으로 주고 받는 메시지를 들여다 볼 수 있다(도청)

2. 메시지를 가로채서 재활용(replay)할 수 있다

3. 또한 메시지를 새로 삽입, 삭제, 변경 할 수 있다

- Stand Alone system에서 인증 예시)



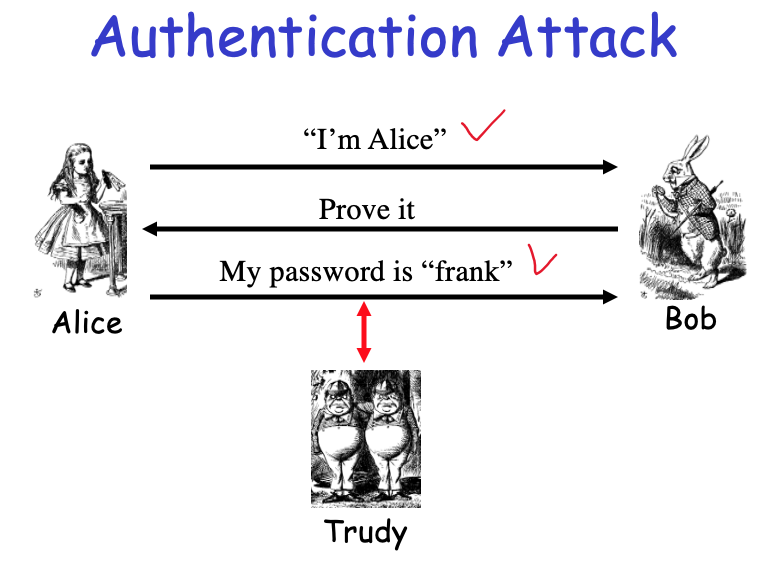
-> 네트워크 시스템에선 안전하지 않다 두 가지 문제가 존재

1. 공격자 입장에서 재생 공격이 가능

2. 밥은 앨리스의 패스워드를 알아야만 한다 (악용할지 모른다)

- stand alone 인증의 공격방법 예시

1.



1. 트루디가 패스워드를 도청한다

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

2. 도청했던 패스워드를 그대로 활용 (재생 공격!!!)

- 좀 더 나은 인증

스크린샷이(가) 표시된 사진

자동 생성된 설명

-> 여전히 재생 공격에 취약!!!

-> 일반 유저이름과 패스워드 방식은 재생공격에 취약!!! 이를 막는 방법이 필요!!! -> challenge-response-protocol

**- Challenge-Response Protocol** (이통사에서 주로 사용하는 프로토콜)

예) 신용카드 분실, 신용카드 회사에서 여러분의 신원을 어떻게 확인할까?

->신용카드회사측에서 고객에게 질문 (성명, 주소, 전화번호, 최근 사용 거래)함

\* 반드시 그 고객만이 알 수 있는 질문을 던져줘야함!!!

->고객의 답변을 통해 신용카드 측에서 고객의 신원을 확인할 수 있게 됨

- challenge-response protocol구현 방식

1. 대칭키 암호

2. 공개키 암호

- 예시)

1. 밥이 앨리스를 인증하길 원한다

2. 밥이 앨리스에게 앨리스만이 정확하게 대답할 수 있는 질문(challenge)를 던진다

3. 이 질문은 재생공격이 불가능하도록 선택되어야만 한다

4. 앨리스가 답을 한다

5. 질문(challenge)으로는 앨리스만이 알수 있는 패스워드 같은게 될수 있고 Fresh하게 진행되기 위해 한 번 사용되는 nonce값을 사용한다

스크린샷이(가) 표시된 사진

자동 생성된 설명

-> 밥은 받았던 해쉬값과 자신이 계산한 해쉬값을 계산하여 앨리스의 패스워드를 알고 있구나하고 인증할 수 있게 된다

-> Nonce가 challenge, h(Alice’s, password,Nonce)는 response가 된다

-> 항상 다른 nonce값을 던지면서 fresh하게 재생 공격을 막을 수 있다

-> 물론 밥은 앨리스의 패스워드를 알고 있어야만 한다

-> 여기서 패스워드 대신에 앨리스로 부터만 나올수 있는 값과 밥은 이 값만 미리 알고 있으면 된다

-> 가능한 암호학을 기초하면 훨씬 더 낫다!!!

**1. 대칭키 암호를 이용한 Challenge-response protocol**

- 앨리스와 밥은 대칭키(K )를 미리 공유한다

- 대칭키는 앨리스와 밥 오직 단 둘만 알고 있어야 한다

- 공유된 대칭키를 알고 있다는 사실을 검증하면서 인증이 이뤄진다

- 키가 노출되거나 재생공격이 허용되선 안된다

텍스트이(가) 표시된 사진

자동 생성된 설명

-> 밥이 response로 받은 값을 자신의 키로 풀어 맞는지 비교하여 검증!!!

(이통사의 인증 방식 – 스마트폰의 유심에 대칭키저장, 이통사도 대칭키를 가지고 있음)

-> 내 스마트폰 유심칩에 대칭키가 저장 되있으므로 이통사에 로그인이 된다

이통사에서 랜덤 넘버를 보내주면 유심칩에 있는 대칭키로 암호화하여 던져 주게 되면 인증이 이루어진다

-> 앨리스는 밥을 인증하지 않는다

-> 상호 인증을 위해선???

스크린샷이(가) 표시된 사진

자동 생성된 설명

-> 앨리스가 R을 던지면(challenge) 밥이 R을 키로 암호화하여(response) 전송

-> 밥이 보냈던 R과 키로 복호화하여 response로 받은 값과 비교하여 밥을 검증!

-> 그 다음 앨리스가 동일하게 R을 키로 암호화하여 밥에게 전달하면 밥이 앨리스를 인증하게 됨

-> 근데 밥이 보낸 response를 그대로 던지기에 트루디가 던진 것일 수도 있기에 안전하지 않다

-> 그렇다면 명백한 방법은? 해결책은?

\* 이 두가지 관점을 다 만족시켜야함

1.상호인증이 되느냐

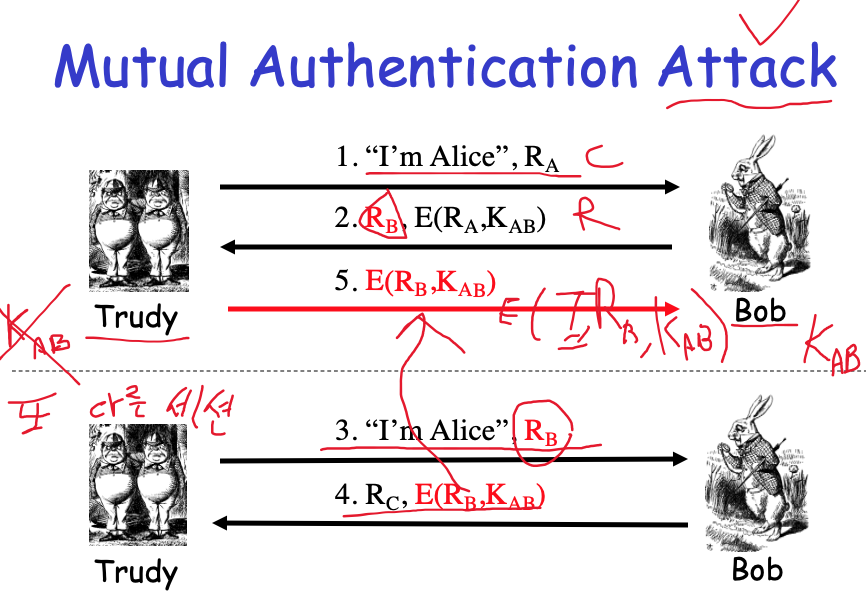
2.안전하냐

스크린샷이(가) 표시된 사진

자동 생성된 설명

-> 상호 인증은 되지만 공격을 받을 수 있다

-> 어떻게 공격이 가능하냐???



-> 트루디는 키를 모르기에 2번에서 밥이 던진 값에 답을 할 수 없다. 그러기에 또 다른 세션을 만들어서 밥이 R(B)를 3번에서 던지면 응답으로 오면 4번 답을 그대로 5번으로 던져주게 되면 밥이 트루디를 인증하게 되어 공격 가능성이 존재하게 된다

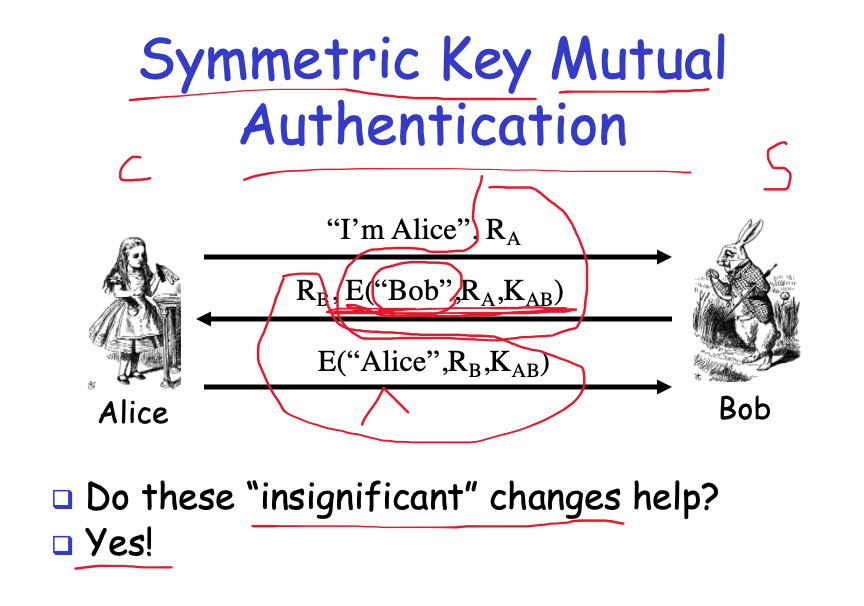
-> one-way 인증 프로토콜은 상호 인증에 안전하지 않다!!!!

-> 그만큼 프로토콜은 미묘하고 명백한것도 안전하지 않을지 모른다

-> 또한 환경이 변한다면 또 어떻게 될지 모른다

-> 그렇다면 어떻게 대칭키 상호인증 프로토콜이 구현 되야하냐???

**-> 해결책**



-> 밥은 보내는 사람의 이름과 받았던 R(A)와 키로 암호화하여 앨리스에게 응답한다 그러므로 앨리스는 밥았던 값과 자신이 푼값을 검증하여 밥을 인증하게 된다 한 편 앨리스는 자신의 이름과 받았던 R(B)를 키로 암호화하여 밥에게 보내 되어 인증이 이루어지게 된다

-> 이 방법이 도움이 되냐? Yes! (상호 인증도 되고 안전하다)

-> 트루디가 세션을 두개 만들어 4번 답을 얻어 5번으로 던질 수가 없다

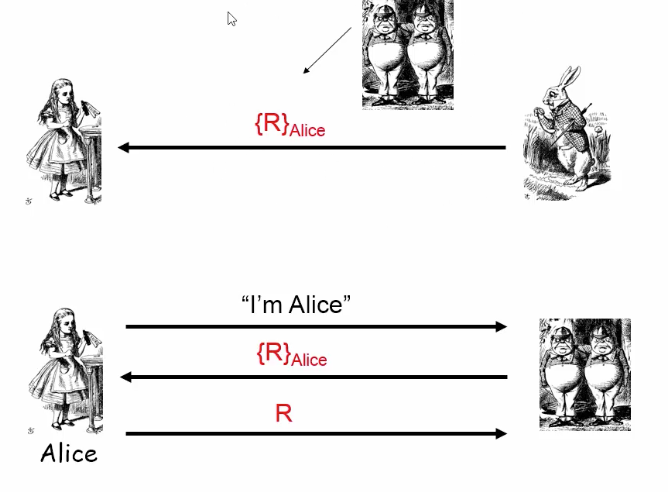
**2. 공개키 암호를 이용한 Challenge-response protocol**

텍스트이(가) 표시된 사진

자동 생성된 설명

-> 앨리스의 개인키로 풀어서 R을 던졌기에 앨리스임을 인증할 수 있다

-> 하지만 안전하지 않다!!!



-> 앨리스의 공개키로 암호한 암호문을 도청해서 트루디가 풀 순 없으나 이 값을 악용해서 R값을 얻을 수 있다 (AWS에서 사용하는 인증 방식)

-> 앨리스에게 도청한 값을 던져주면 자연스레 평문 값을 얻을수가 있다!!

-> 목적이 기밀성을 제공하기 위해 암복호화 하는건데 기밀성이 깨지게 된다

-> 한 목적을 위해서만 오직 한 키 페어를 사용해야 한다

- 인증을 위한 키페어 한 쌍만 사용한 예시)

스크린샷이(가) 표시된 사진

자동 생성된 설명

-> 밥은 앨리스를 인증 가능!

-> 문제는 앨리스에게 뭐든 던지면 다 서명을 해주게 된다!!!

-> 그러기에 두 개의 키 페어를 사용해야 한다!! (인증(서명)하는데 하나, 암호화하는데 하나!)

**- 세션키 (Session Key)**

※실생활

Step 1 : 인증 (Challenge-Response protocol)

Step 2 : 암호화 (음성, 문자) -> 대칭키 암호 (암복호화 속도가 빠르다, 하지만 키 공유가 필요)

-> 대칭키 공유를 위해서 세션키 분배가 필요하다!!!!

- 특정한 세션을 위해 사용되는 대칭키 (무결성과 기밀성을 제공할 수 있다)

- 단 방향 인증도 하고 세션키도 분배하는 예시)

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 앨리스가 자신의 이름과 R을 던진다

2. 받았던 R과 키를 하나 생성해서 앨리스의 공개키로 암호화하여 앨리스에게 보낸다 (이 단계 후에 세션키 분배가 이뤄짐)

3. 앨리스는 받았던 값을 풀어서 키를 풀고 R+1과 키를 밥의 공개키로 암호화하여 보내면 밥은 밥의 개인키로 풀어서 앨리스가 제대로 풀었구나 하고 앨리스를 검증할 수 있다 (앨리스를 인증할 수 있음)

-> 앨리스나 밥의 개인키를 알수 없으므로 안전하다

-> 밥이 앨리스를 인증 가능하나 앨리스가 밥임을 인증할수 없다(상호인증 안됨)

-> 상호인증하려면???

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 앨리스가 R을 던짐

2. 밥이 R과 키를 밥의 개인키로 서명해서 보냄 (키 공유 및 밥을 인증)

3. 앨리스가 밥의 공개로 푼 키와R+1을 앨리스의 개인키로 서명해서 보내면 밥이 앨리스의 공개키로 풀고 앨리스임을 인증할 수 있다 (앨리스를 인증)

-> 문제는 트루디가 도청해서 주고 받는 값들을 앨리스나 밥의 공개키로 풀 수 있다

**- 상호 인증도 하고 세션키도 분배하는 예시)**

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 앨리스가 R을 던짐

2. 밥이 R값과 키를 생성하여 밥의 개인키로 서명 후 앨리스의 공개키로 암호화하여 보냄 (키 분배 및 앨리스가 밥을 인증 가능)

3. 앨리스가 키와 R+1을 앨리스의 개인키로 서명후 밥의 공개키로 암호화하여 던진다 (밥이 앨리스를 인증 가능)

-> 상호 인증되고 세션키 분배도 된다!!!

-> 트루디는 개인키를 알 수 없으므로 안전!!

-> 순서를 기억! 서명 후 암호화!!!!

-> 만약 암호화 후 서명하게 되면스크린샷이(가) 표시된 사진

자동 생성된 설명

-> 문제는 없으나 밥과 앨리스의 공개키는 알 수 있기에 안에 있는 내용을 알 수 있어 그전보다 덜 안전하다!!!

※ 각 단계에서 인증과 세션키분배가 어떻게 이루어지는지 잘 파악할 것!

※ 요점은 개인키가 제대로 사용되었으면 그 사람을 인증 할 수가 있다

**- ZKP(Zero Knowledge Proofs 영지식 증명)**

- 영지식증명은 증명자가 자신이 알고 있는 정보를 공개하지 않으면서 정보를 알고 있다는 사실을 검증자에게 증명

Client(Prover) ----> Server (Verifler)

-사례

1.Alibaba’s Cave

2.Sudoku

3.Zcash

·블록체인 네트워크에는 수신자, 송신자 및 전송 금액 등의 거래 내역 정보가

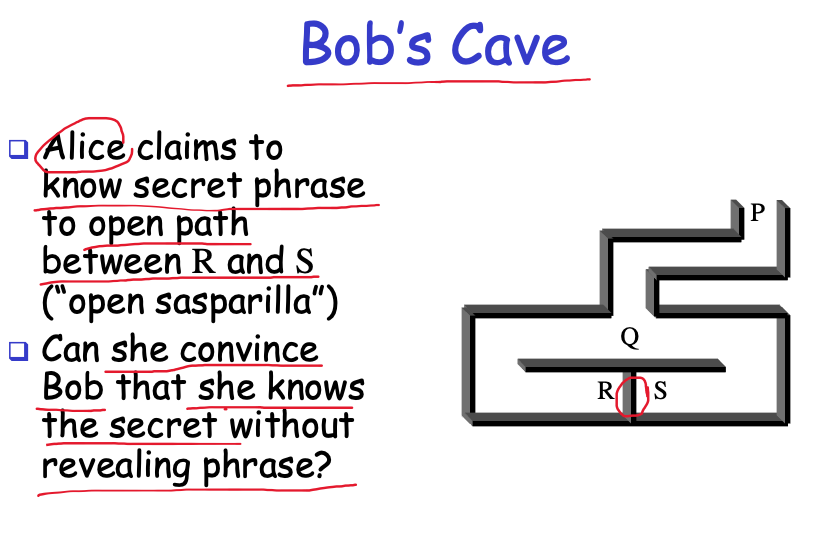
·공개 및 기록되어 있습니다.

·개인정보를 보호하고 보안을 강화해야 하는 필요성 대두

·익명성을 보장하는 거래

- 대화식 증명 시스템 (앨리스와 밥이 지속적으로 메시지 주고 받고 연산을 하면서 증명 하는 시스템)

- 영지식 증명의 예시1) 알리바바’s Cave



-> 앨리스는 R과 S를 열수 있는 비밀스런 문장을 알고 있다 주장

-> 앨리스는 자신이 알고 있는걸 노출하지 않으면서 어떻게 알고 있단 사실을 어떻게 증명할 수 있으냐?

스크린샷이(가) 표시된 사진

자동 생성된 설명

1. 밥 : 앨리스야 S쪽으로 나와봐 (현재 앨리스가 R에 있음)

2. 앨리스 : 조용히 열려라 참깨해서 문을 열고 나옴

-> 만약 비밀 정보를 모른 상태로 S사이드 나올 확률이 1/2

-> 밥이 n번 반복 했을 때 앨리스가 밥을 속일 확률은 (1/2)^n

(n이 커질수록 속일 확률이 0에 되므로 앨리스가 모르는 걸 알 수 가 있다)

- 영지식 증명의 예시2) 스도쿠 게임

스크린샷이(가) 표시된 사진

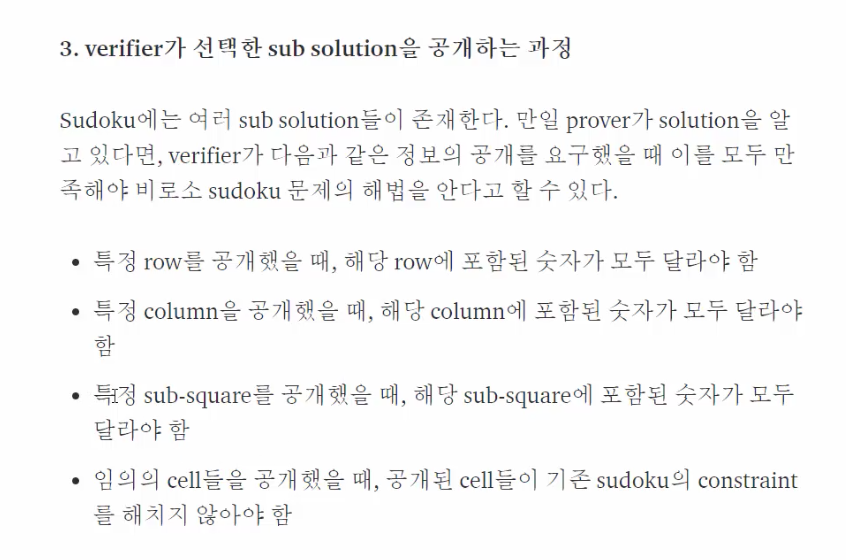
자동 생성된 설명

스크린샷이(가) 표시된 사진

자동 생성된 설명

스크린샷이(가) 표시된 사진

자동 생성된 설명



-> 이러한 작업들을 반복해서 실시 했는데도 불구하고 전부 스도쿠의 제약사항을 만족시키면 prover가 해답한다는 걸 검증할 수 있다.

※이때 주의해야 될 사항

-> Shuffling solution 수행 전에 매번 mapping table을 새로 구성해야 함

(만약 그러지 않을 시 결국 prover가 sub-problem 을 공개 할 때의 단서를 모아서 실제로 prover가 어떤 solution을 가지고 있는지 유추할 가능성이 생긴다)

**- Fiat-Shamir Protocol - 영지식 증명을 구현한 프로토콜**

-> 동굴 없이 동일한 효과를 얻기 위한 프로토콜

-> Modulo N의 제곱근을 구하는게 어렵다는 걸 기반으로 만들어진 프로토콜

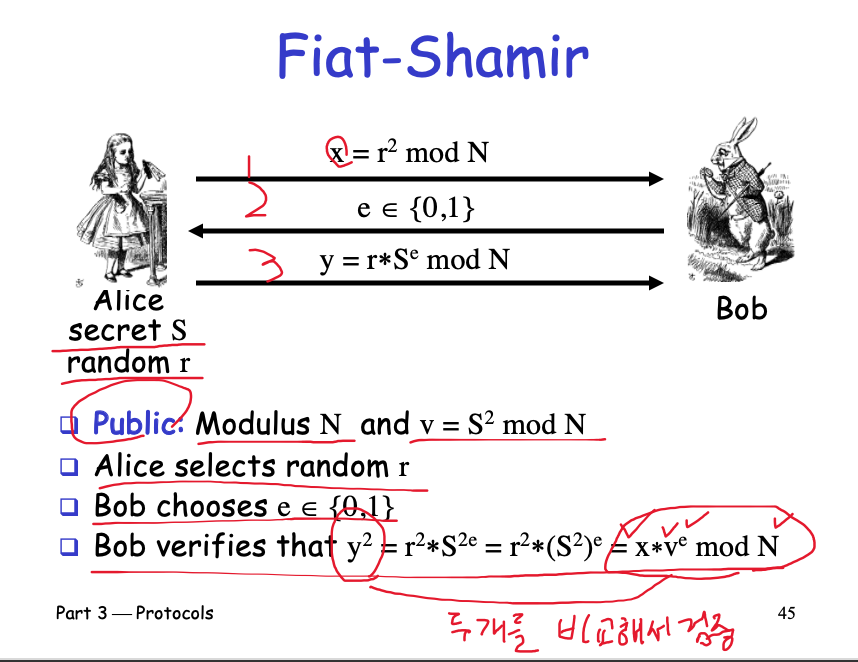
1. 소수 p와 q의 곱을 N이라 가정

2. 이때 앨리스는 시크릿S를 가지고 있다고 가정

3. N과 S^2 mod N을 한 v는 공개를 시키고S는 시크릿하게 유지 시킨다

(제곱근을 구하기 어렵기 때문에 v를 공개하더라도 사람들은 v를 통해서 S를 유추 해내기 어렵다)

-> 앨리스는 S에 관한 어떠한 정보도 노출하지 않으면서 S를 알고 있다는 사실을 증명해보자



1. 앨리스가 먼저 S와 random number r을 선정하여 r^2 mod N을 던진다

2. 밥은 0또는 1을 선택해서 보낸다

3. 앨리스는 자신이 알고 있는 r과 S^e mod N을 곱한 y를 계산해서 보내준다

-> 밥은 y값을 제곱하고 x \* v^e mod N을 계산하여 일치하는지 비교하여 앨리스가 S값을 정확히 알고 있는걸 검증

(x는 처음에 받은 값, v는 공개 되 있는 값, e는 자신이 던진 값)

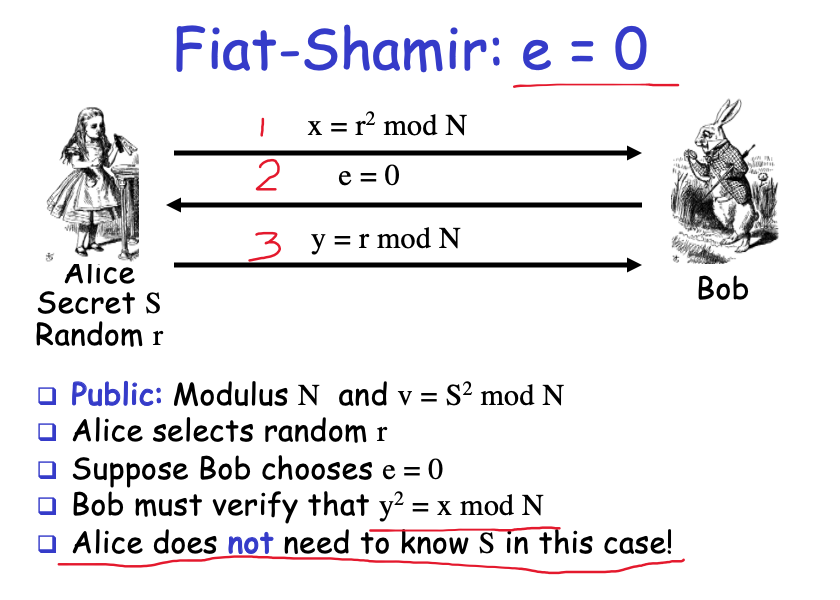
**- 만약 e가 1이 될때의 상황**

스크린샷이(가) 표시된 사진

자동 생성된 설명

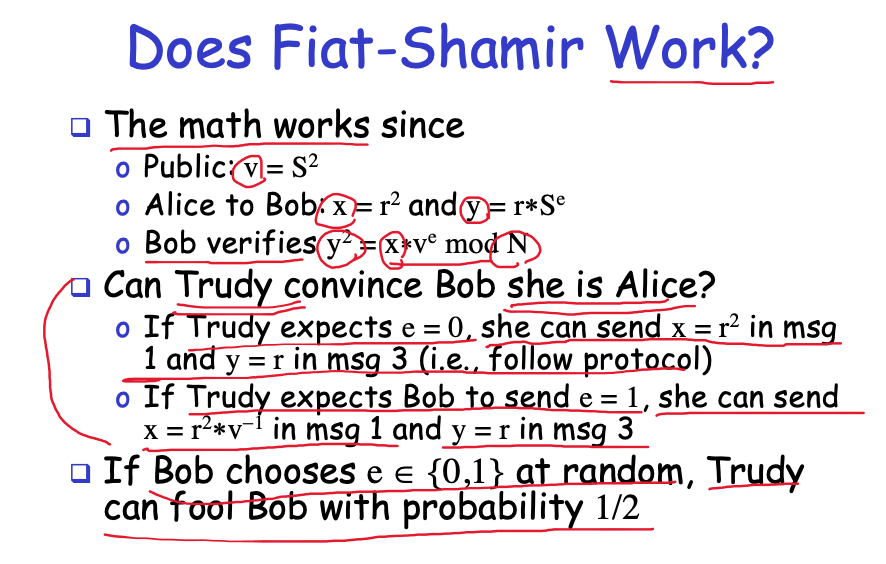
-> 밥은 x와 v를 알고 있기에 쉽게 검증 가능, 앨리스 S를 안다는걸 검증 가능

**- 만약 e가 0이 될때의 상황**



-> x를 알고있기에 쉽게 검증가능, 앨리스가 안다는 걸 밥이 검증 못함

- 수학적으로 Fiat-Shamir가 동작하느냐?



-> 수학적으로 검증가능하다 증명되있음

-> 트루디 입장에서 밥이 만약 0을 보낼거라 예상이 되면 세번째 메시지에서 r을 그냥 던지면 된다

-> 트루디가 밥이 1을 던질거라 예상이되면 S를 모르기에 첫번째 메시지엔r^2 \* v^-1을 x대신 보내고 세번째메시지엔 y를 r로 보내게 되면 밥 입장에선 검증이 이뤄진다

-> 하지만 트루디 입장에선 밥이 무엇을 보낼지 모르고 이 과정을 여러번 반복하게 되면 트루디는 밥에게 영지식 증명을 할 수가 없다

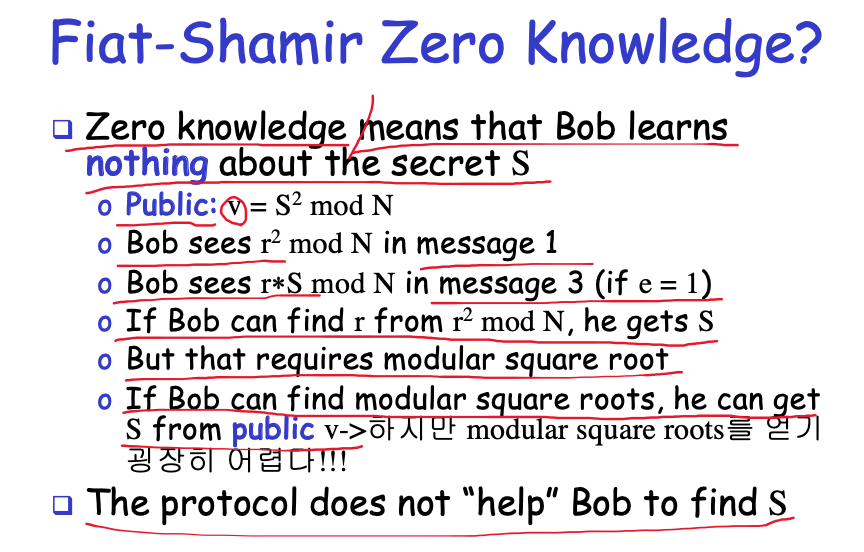
( (1/2)^n인데 n을 반복하게 되면 0에 수렴 )

- random number를 사용하지 않으면 S를 찾을 수 있게 된다!!!



-> 결론적으로 밥은 S에 대해서 아무 것도 얻어 낼 수 없다

-> 만약 밥이 modular square roots를 알면 S를 얻을 수 있지만 이를 알기는 굉장히 어렵다!



-> 실생활에서 ZKP

· 전자서명을 검증하는 과정에서 앨리스의 개인키를 알 수 없기에 영지식 증명과 비슷하긴 하나 공개키를 사용하면 익명성이 보장되지 않는다

· 사람의 신원을 노출하지 않으면서 인증하는 방법을 제공

· 마이크로소프트에서도 ZKP를 활용하고 있다

**<4-2.RealWorldProtocol>**

- 실생활에서 많이 사용되는 프로토콜

· SSL – 웹에서 보안을 제공하기 위해 사용

· IPSec – IP계층에서 보안

\* VPN(가상 사설망) : 인터넷은 공용망이기에 회사 중요문서를 보내기에 문제 우려가 있음, 그러기에 인터넷을 이용하되 회사 내의 전용망 같은 효과를 얻는 전용망

컴퓨터이(가) 표시된 사진

자동 생성된 설명

· Kerberos – 윈도우 운영체제에서 대칭키 인증 시스템

· GSM – 모바일 2G에서의 보안

**- SSL ( Secure Socket Layer)**

- socket layer란?

텍스트이(가) 표시된 사진

자동 생성된 설명

· Physical , data link layer -> 네트워크 카드에 디바이스 드라이버에 해당하는 부분

· network, transport layer -> OS차원에서 지원

· Application layer -> library차원에서 지원

· socket layer 는 트랜스포트와 네트워크층의 중간에 존재함

- SSL이란? 인터넷에서 안전한 트랜잭션을 위해 사용되는 프로토콜

- 아마존에서 책을 산다면?

-> 유저가 제대로 아마존에 제대로 접근했는지 인증과정이 필요 (서버에 대한 인증)

-> 신용카드 정보가 네트워크를 통해서 전달이 될 때 암호화를 통해 보호가 되야함 (무결성과 기밀성이 유지 되야함)

-> SSL에선 클라이언트는 인증할 필요가 없다 ( 상호인증 필요 없음 )

**- 실제 동작 원리!!!**

음식이(가) 표시된 사진

자동 생성된 설명

1. 나는 너와 보안상 안전하게 인증하고 싶어

2. 밥이 자신의 인증서를 보내주게 됨

3. 앨리스는 밥과 공유할 대칭키를 선정 후 밥의 공개키로 암호화해서 던져주게 되있음

4. 밥은 자신의 개인키로 풀면 대칭키 분배가 이뤄짐

-> 밥과 앨리스 서로 인증할 수 없다

( 후로 데이터를 주고 받는 과정에서 메시지가 제대로 오는지 확신할때 밥을 인증할 순 있게 된다)

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 클라이언트인 앨리스가 서버인 밥에게 자신이 처리할 수 있는 암호화알고리즘의 리스트인 cipher list와 R(A)를 던진다

(앨리스와 밥 사이에 암호화 알고리즘 (해쉬, 대칭키알고리즘, 공개키암호알고리즘)에 대한 협상이 필요함)

\* Random Number -> 재생 공격을 막기 위함

2. 밥이 앨리스에게 인증서와 서버가 암호화알고리즘 리스트 중 하나를 선택해서 랜덤숫자 R(B) 와 같이 앨리스에게 보냄

3. 앨리스는 내부적으로 가진 pre-master secret 값인 S를 선정해서 밥의 공개키로 암호화한 것과 S와 R(A), R(B)를 해슁을 통해서 나온 키 값으로 E(h(msgs,CLNT,K),K) 이렇게 보내게 됨 (이전 메시지가 변조 되지 않은 걸 여기서 확인 할 수가 있고 세션키 분배가 이뤄진다!!!)

\*Msgs -> 모든 이전 메시지들 , CLNT, SRVR 는 상수 값

4. 밥은 msgs와 SRVR라는 상수값과 키를 해슁해서 앨리스에게 보내게 된다

(밥이 S를 제대로 얻어서 키로 제대로 알고 있구나란 걸 알수 있다 그래서 밥이란 걸 인증할 수 있다)

5. 그 후로 키로 데이터를 암호화하여 네트워크를 통해 주고 받음

\* 인증서에 있는 정보

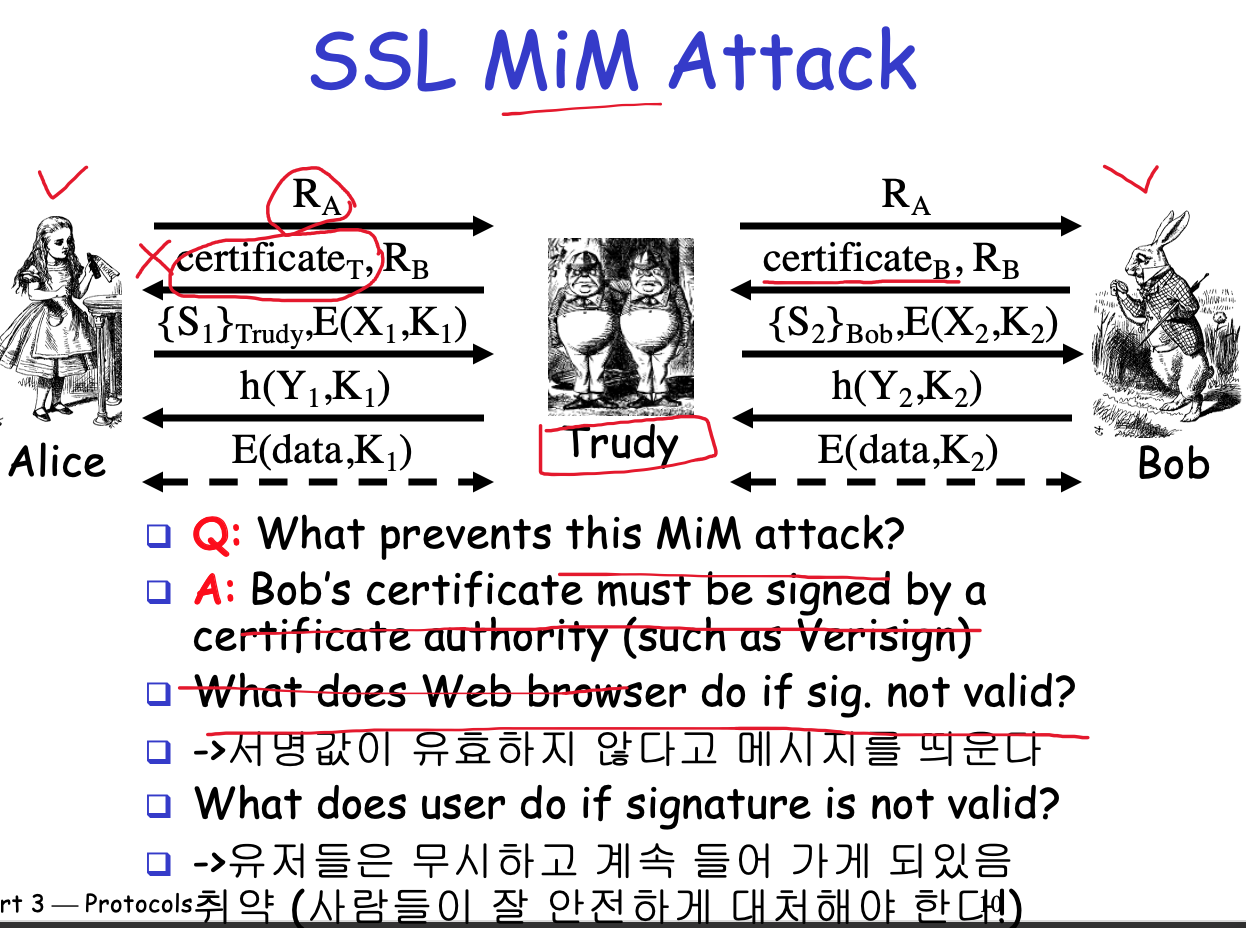
-> 인증기관, 발급자 (주체), 국민은행의 공개키(N,e), 유효기관, 발급자(CA)의 서명 (이로 인해 무결성이 이뤄짐)

- 키를 바탕으로 해서 6개의 키가 파생됨

- 클라이언트는 서버를 네번째 단계에서 인증하지만 서버는 클라이언트를 인증하지 않는다!!!! (네이버에 들어갈려면 공인인증서가 꼭 필요하다하면 너무 번거롭고 상호 인증이 굳이 필요없다)

- 상호 인증 하려면 두번째 메시지에서 인증서를 보내달라 요청해야 한다

- SSL에서의 중간자 공격 (MiM Attack)



-> 밥의 개인키를 모르기에 중간자 공격이 불가능 하다!!!!!

-> 인증서를 위조하면 CA의 서명 값이 달라지기에 불가능하다!!!

-> 브라우저는 서명값이 잘 못되어 있으면 경고창을 띄우게 된다

-> 유저가 여기서 무시하고 들어가면 문제가 발생할 수 있다 (프로토콜상에선 안전 해도 유저가 어떻게 하느냐에 따라 문제가 발생할 수 있다)

- SSL 세션 하나에 커넥션을 여러 개 만들어 줄 수 있다

(국민은행 홈페이지를 창 두개로 들어갈 수 있다)

- 세션 만드는 작업은 공개키 작업이 들어가기에 비용(시간)이 많이 들어감

- 이미 세션이 존재하는 경우에 커넥션을 만들때 좀 더 효율적인 프로토콜이 있다

- 세션이 존재한다는 가정하에 SSL Connection

스크린샷이(가) 표시된 사진

자동 생성된 설명

-> 따라서 공개키 연산 자체가 없기에 훨씬 빠르게 커넥션을 만들 수 가 있다 정도만 알고 있을 것!!

- 정리

\* SSL (IEEE스탠다드에서는 TLS라 불림)

- 소켓레이어에 존재

- encryption, integrity, authentication, etc

- 클라이언트는 서버를 네번째 단계에서 인증하지만 서버는 클라이언트를 인증하지 않는다!!!! (네이버에 접속할때 공인인증서가 꼭 필요하다하면 너무 번거롭고 상호 인증이 굳이 필요 없다) -> 상호 인증 하려면 두번째 메시지에서 인증서를 보내달라 요청해야 한다

- 중간자 공격 불가능하다!!!

- SSL은 라이브러리 레벨에서 지원하고 있고 IPSec은 복잡하고 OS차원에서 지원하고 있다

\* 인증서에 있는 정보

-> 인증기관, 발급자 (주체), 국민은행의 공개키(N,e), 유효기관, 발급자(CA)의 서명 (이로 인해 무결성이 이뤄짐)

**< Access Control >**

**1. Autentication**

1)Something you know

-패스워드, PIN, 생일 등 개인적으로 알고 있는 정보들을 이용한 인증

-> 패스워드에 대한 Dictionnary Attack가능함

2)Something you have (스마트카드)

-> 차키, 랩탑 컴퓨터(구체적 맥 주소), 패스워드 발생기, ATM카드, 스마트카드 등

3)Something you are

- 지문, 손 서명, 얼굴 인식, 홍채 인식을 비롯한 생체인식!!!

\*Fraud rate : A를 B로서 잘 못 인식할 확률(기만율)

\*insult rate : A를 A로서 인증하지 못할 확률 (모욕율)

**2.Authorization**

- Firewall은 시스템 내부로 트래픽이 들어오고 나가는 것을 결정 한다

ex)

보통 외부에서 들어오는 공격(트래픽)에 대해 차단한다

외부에서 나가는 것도 허용하고 차단할 수 있다

-Firewall의 종류

1. Packet filter -> works at network layer

2. Stateful packet filter -> transport layer

3. Application proxy -> application layer

4. Personal firewall -> for single user, home network, etc.

**< Web Security >**

- 웹 공격 방법

1.SQL injection – 공격자가 희생자에게 쿼리나 command의 부분으로서 데이터를 보내면서 발생하는 공격

ex) SELECT \* FROM users WHERE username=‘ roy' OR '1'='1 ' AND password=''

2.XSS - 악의적인 코드를 웹 페이지에 집어넣는 공격 방법, 유형으로는 Reflected XSS, Persistent XSS, Dom-based XSS 세가지 방법이 존재함

3.CSRF -클라이언트의 request를 위조하여 공격 하는 방법, spirng security는 이에 대한 방지를 지원

- 웹 공격에 대한 방지

1.SQL Injection 방지하는 법

-> Prepared Statement사용하여 query와 데이터를 분리 후 들어온 input데이터를 검증

2.XSS

-> Encoding – 브라우저가 유저의input을 코드로 해석하는게 아니라 데이터로 해석하도록 하는 방식

-> Validation – 유저의 input을 필터링해서 악의적인 command없이 코드로서 브라우저가 실행시키도록 하는 방식

XSS 방지하는 법

-> JSP에서 JSTL을 사용하여 <c:out> tag를 사용하면 escape시켜서 보내주기 때문에 브라우저에 디스플레이 하되 스크립트로 실행시키지 않는다

3.CSRF

-> Credentials이나 tokens을 사용하여 브라우저에서 submit할때 이를 자동적으로 서버 쪽으로 보내도록 설정