Network Security: 1
Introduction

관련 분야들

- 취약점 분석/공격
  - Hacking, Exploitation
    - "HACKING: the art of exploitation"
    - Web, System, Device (Mobile & IoT)
- 디지털 포렌식
  - 디지털 정보 수집 및 분석
    - Windows OS 및 FAT/NTFS 파일시스템 정보
    - Linux OS 및 EXT 파일시스템 정보
    - Memory 정보
    - 애플리케이션 사용 이력, 시스템 로그, 레지스트리, ...
- 보안 컨설팅
  - 보안 기본 지식, 최신 보안 트랜드
- Cryptography
  - Encryption, decryption, authentication, hashing

정보보안 관련 내용들: **attacks**

- Passive attacks
  - Network
    - Wire-tapping/Fiber-tapping, port scan, ...
    - Attack for each layer/protocol
  - Host
    - Keystroke logging, ...
- Active attacks
  - DOS(denial-of-service) attack / D-DOS(distributed dos)
  - Spoofing, Masquerade, Repudiation, Replay
  - Network
    - Man-in-the-middle, ARP poisoning, ping flood, ping of death, smurf attack, ...
  - Host
    - Buffer overflow, heap overflow, stack overflow
    - Format string attack
    - ...

정보보안 관련 내용들: **malicious sw**

- Viruses
    - Malicious software program loaded onto a user's computer without the user's knowledge and performs malicious actions.
    - Activated with their own specific target (ex. .docx macro)
    - Triggered by the activation of their host
- Worms
    - Stand-alone malicious programs that can self-replicate and propagate independently as soon as they have breached the system.
    - Activated without their own specific target
- Trojan horses
    - A program downloaded and installed on a computer that appears harmless, but is, in fact, malicious.
- Backdoor
    - Covert method of bypassing normal authentication or encryption in a computer, product, embedded device

- Adware
  - Unwanted software designed to throw advertisements up on your screen.
- Ransomware
  - A form of malware designed to encrypt files on a device, then malicious actors then demand ransom in exchange for decryption.
- Malicious Bot
  - A malicious bot is an automated malware program that can infect a system, steal data, or commit other fraudulent activities.
- Exploit
  - A piece of software, a chunk of data, or a sequence of commands that takes advantage of a bug or vulnerability to cause unintended or unanticipated behavior to occur on computer SW or HW.

정보보안 관련 내용들: cryptography

- Cryptography
  - Secret key cryptography, public key cryptography
  - Symmetric key cryptography, asymmetric key cryptography
  - Message authentication
  - Hashing
  - Certificate, Key management
- Authentication
  - Who you are?

Contents

- Introduction
- Basic encryption and ciphers
    - Substitution encryption
        - Character level encryption
        - Bit level encryption
    - One time pad (OTP)
    - Block ciphers
    - Stream ciphers
- Cryptography tutorial
    - Secret key cryptography (symmetric key algorithm)
        - DES, AES, IDEA, ...
    - Public key cryptography (asymmetric key algorithm)
        - RSA, Diffie_Hellman, ECC, ...
    - Hash and Message digests
        - MD5, SHA-2/3, ...

Security Requirements

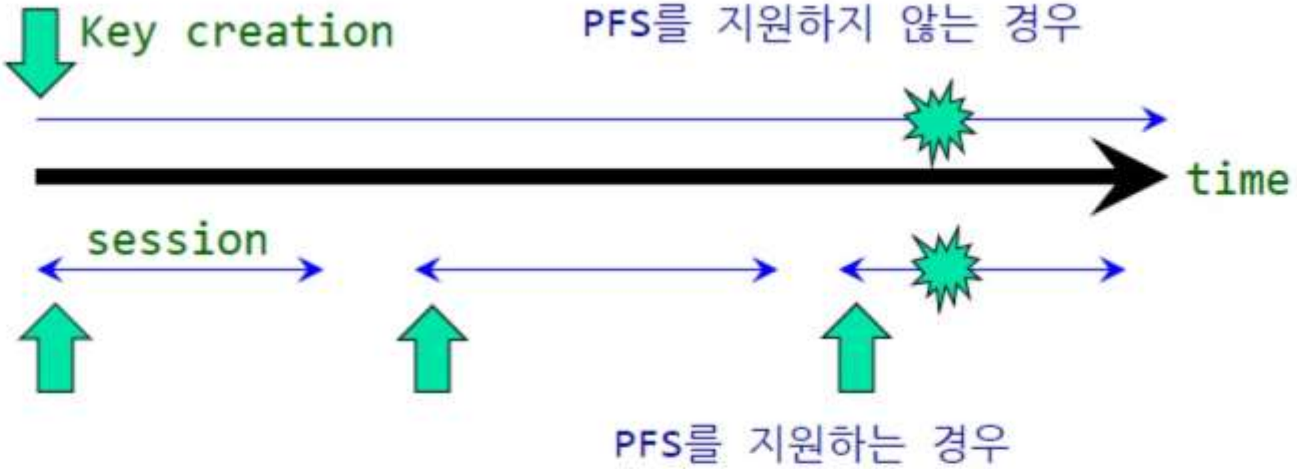
- **Confidentiality** (no **eavesdropping**) (기밀성)
  - Protection from disclosure to unauthorized persons
  - only sender, intended receiver should "understand" message contents
- (message) **Integrity** (무결성)
  - Maintaining data consistency (no modification)
  - sender, receiver want to ensure message not altered (in transit, or afterwards)
- **Authentication** (인증)
  - Proves who you are (login/password)
  - Mutual authentication
    - sender, receiver want to confirm identity of each other
  - ***Message authentication***
    - Message integrity (integrity) &
    - Verify the source of the message (authenticity)
- **Non-repudiation** (부인방지)
  - Originator of communications can't deny it later

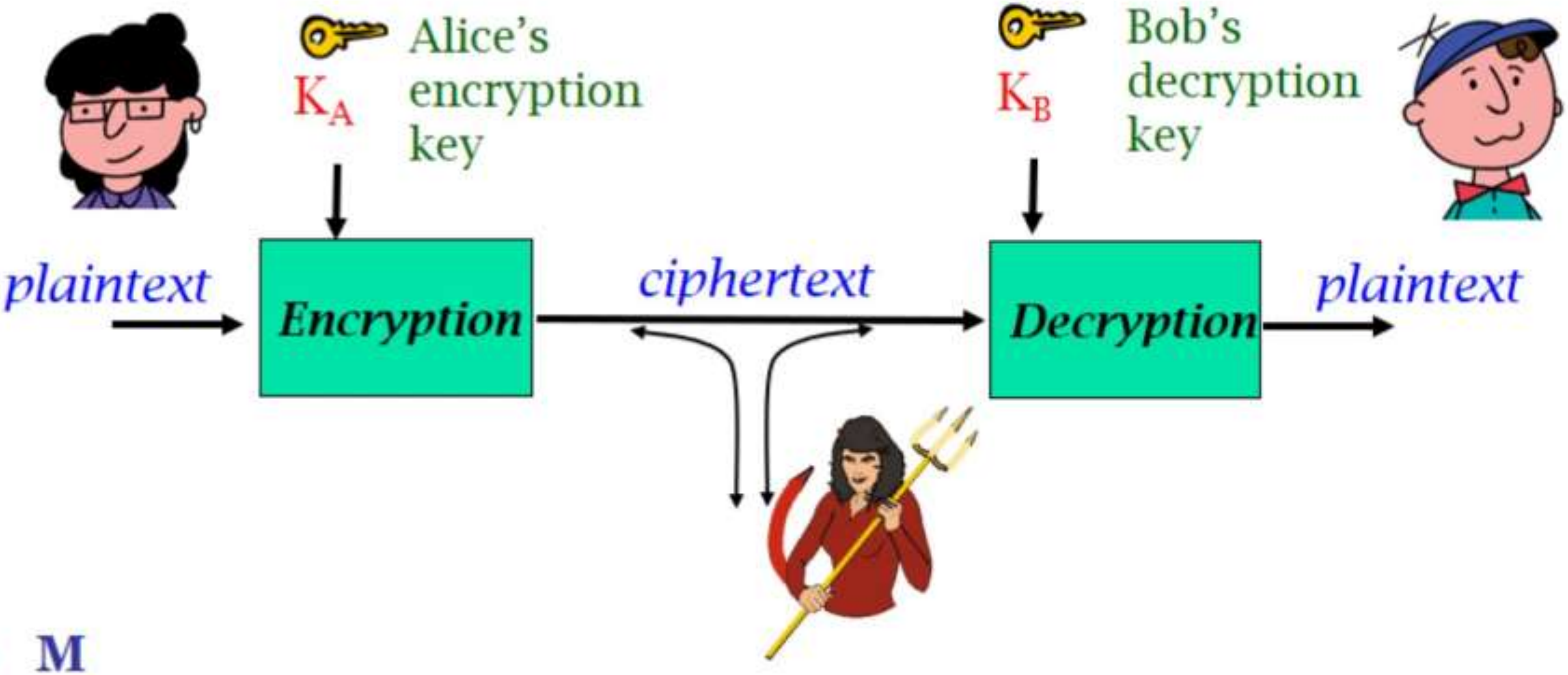
- Authorization (권한 부여)
  - Defines the user’s rights and permissions on a system (permission of “ls -l”)
  - *Check that the user can access the service he is trying to*
  - RFC 3552
    - “**Authentication** simply identifies a party, **authorization** defines whether they can perform certain action”
- Availability (가용성)
  - Legitimate users have access when they need it
- Access control (접근제어)
  - The ability to permit or deny the use of an object by a subject.
  - It provides 3 essential services
    - Authentication (who can login)
    - Authorization (what authorized users can do)
    - Accountability (can be traced what a user did)
- These are often combined
  - User authentication used for access control purposes
  - Non-repudiation combined with authentication

Terminologies

- Terminologies
  - **plaintext** (평문) - the original message
  - **ciphertext** (암호문) - the coded message
  - **cipher** - algorithm for transforming plaintext to ciphertext
  - **key** - information used in cipher known only to sender/receiver
  - **encipher** (**encrypt**) - converting plaintext to ciphertext
  - **decipher** (**decrypt**) - recovering ciphertext from plaintext
  - **cryptography** - study of encryption principles/methods
  - **cryptanalysis** (codebreaking) - the study of principles/ methods of deciphering ciphertext without knowing key
  - **cryptology** - the field of both cryptography and cryptanalysis

- **PFS**(Perfect Forward Secrecy)/**FS**(Forward Secrecy)
  - A feature of specific key agreement protocols that gives assurances that session keys will not be compromised even if the private key of the server is compromised. (Wikipedia)
    - Forward secrecy protects past sessions against future compromises of secret keys or passwords.
    - By generating a unique session key for every session a user initiates
  - Example: Diffie Hellman key agreement





K_A Alice's encryption key

K_B Bob's decryption key

plaintext

Encryption

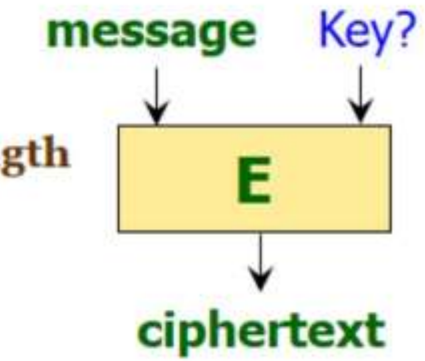
ciphertext

Decryption

plaintext

- **M**
  - plaintext message
- **C** = Enc (K_A, M) = Enc-K_A (M) = Enc_{KA} (M)
  - ciphertext, encrypted with key K_A
- **M** = Dec (K_B, **C**) = Dec (K_B, Enc (K_A, M))

Brute Force Search/Attack



- Always possible to simply try every key
  - *E* (message, key) → Ciphertext
    - Assume, we know **E**, (message, ciphertext) **pair**, **key length**
    - Ex.
      - E = message + key (ex. 'A' + 1 = 'B')
      - (message, ciphertext) pair = ('A', 'C')
      - Key length = 2 bits
  - key?

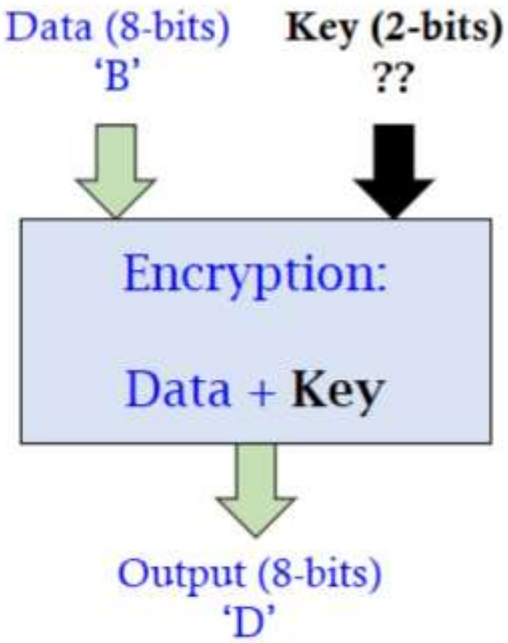
```
for (key=$0000_b$; key $\le$ $11_b$; key++)
    if (key + 'A' == 'C')
        this is the key!! (key=2)
```

- General (*n* bits key)

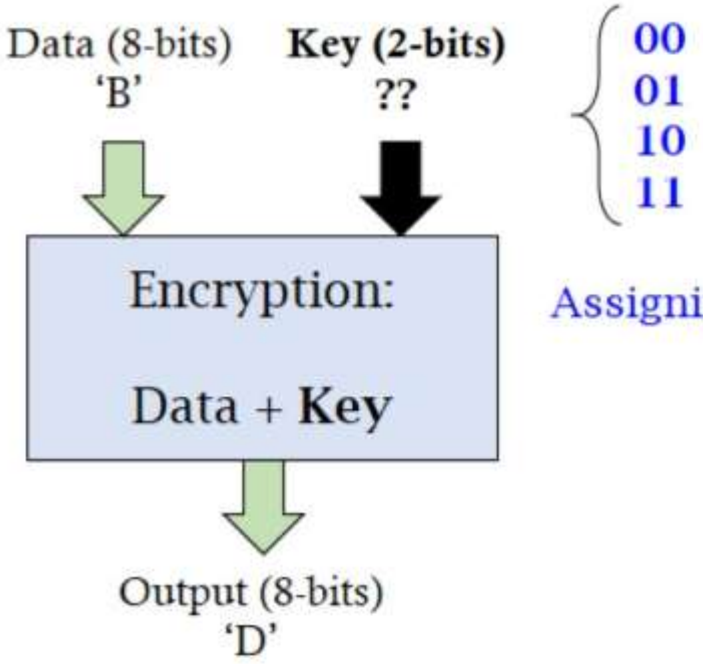
```
for (key=0; key < $2^n$; key++)    // assume key is n bits, E is known
                                   // with message/ciphertext sample
    if (E(message, key) == Ciphertext)
        this is the key!
```

- Always possible to simply try every key
- Most basic attack, proportional to key size

Publically known



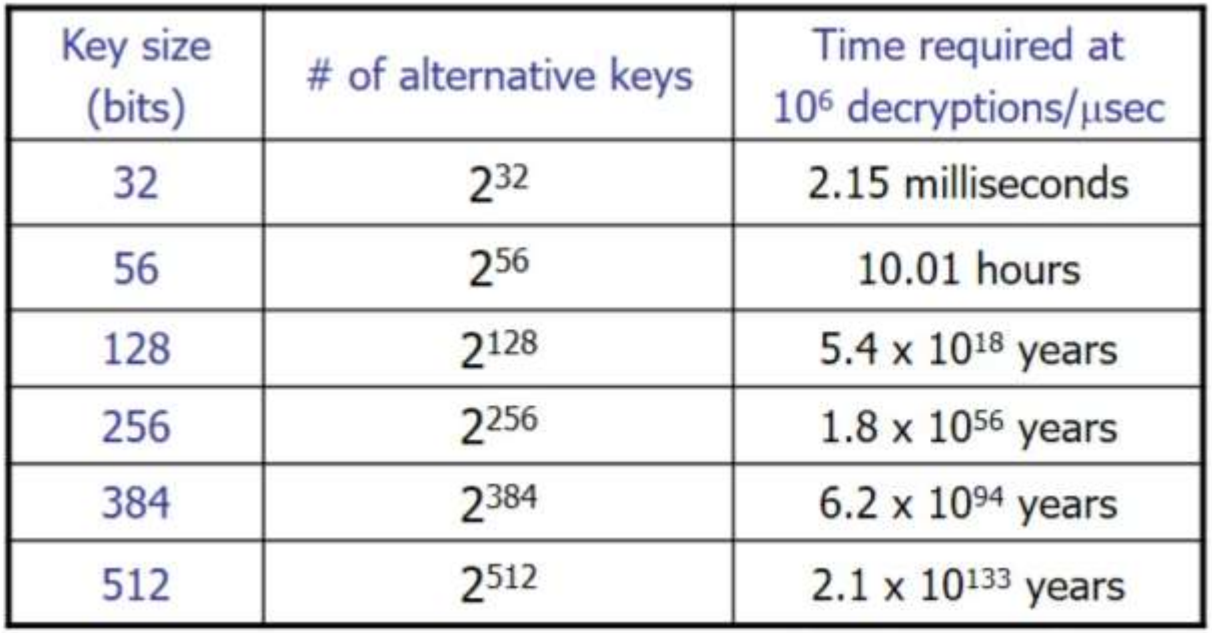
How to find key?



Assigning each value

Simple! But,
if 128 bits key? → 256/512 bits

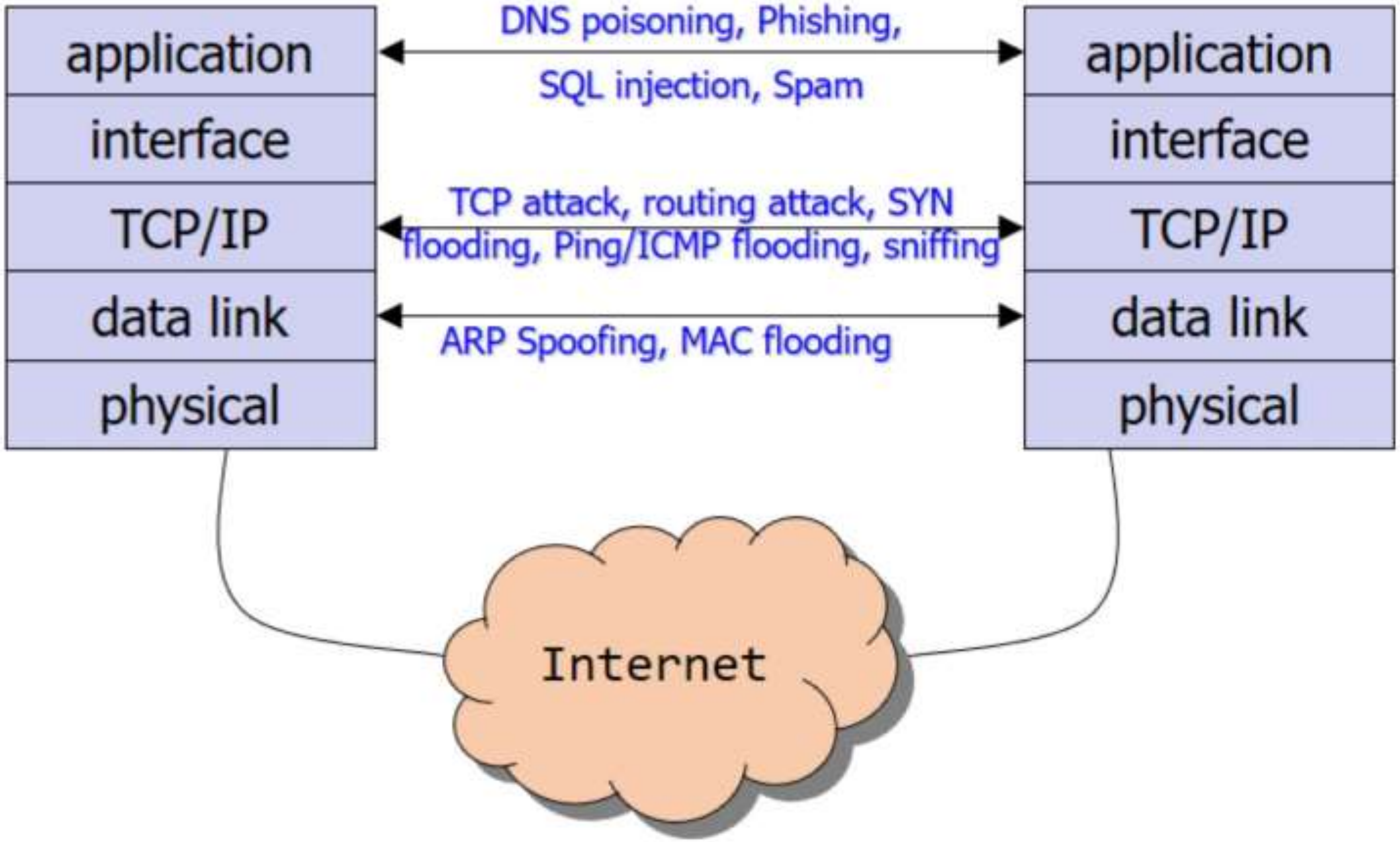
- Average time required for exhaustive key search



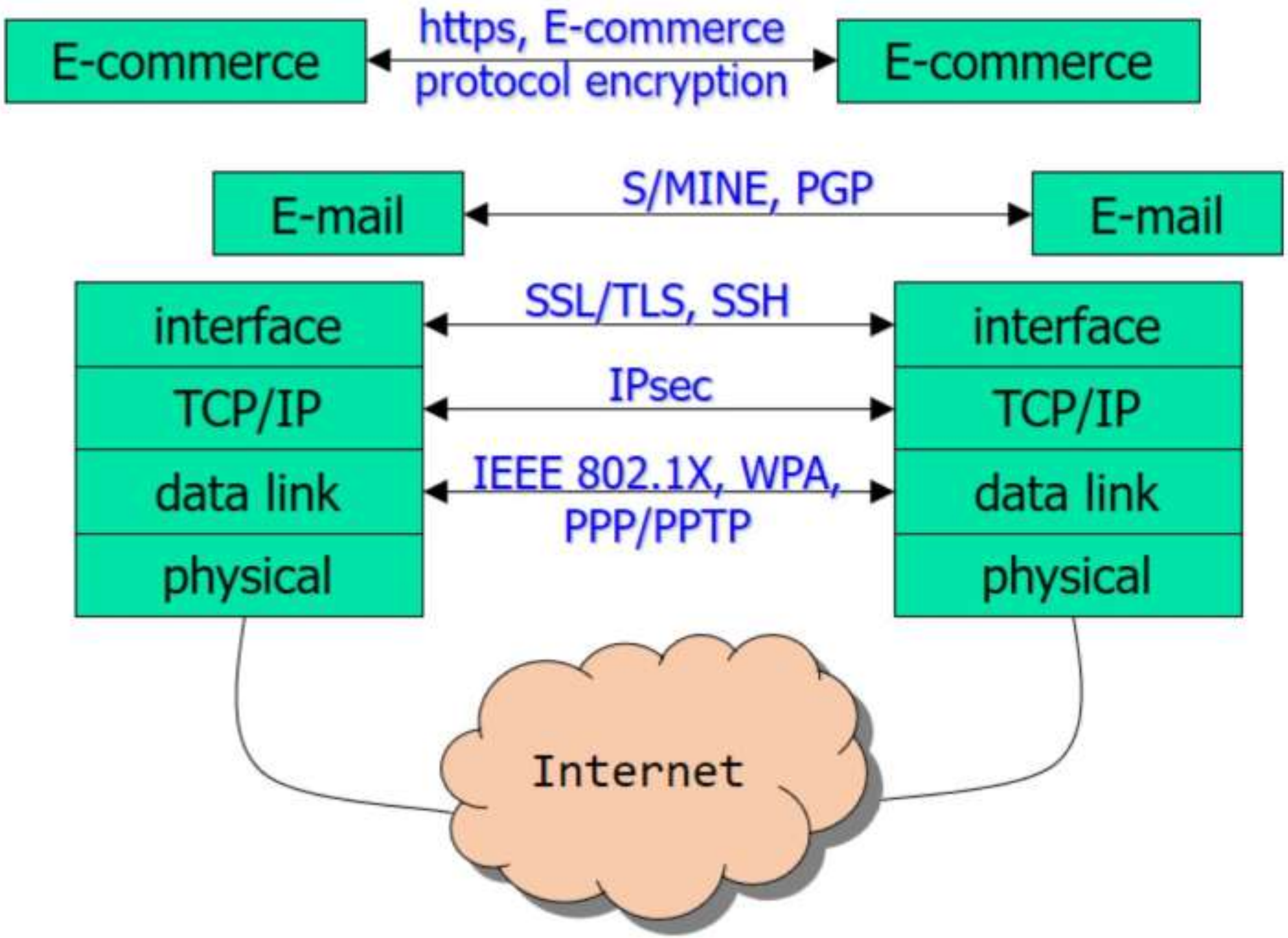
Key size (bits)	# of alternative keys	Time required at 10^6 decryptions/µsec
32	2^{32}	2.15 milliseconds
56	2^{56}	10.01 hours
128	2^{128}	5.4 x 10^{18} years
256	2^{256}	1.8 x 10^{56} years
384	2^{384}	6.2 x 10^{94} years
512	2^{512}	2.1 x 10^{133} years

https://www.researchgate.net/

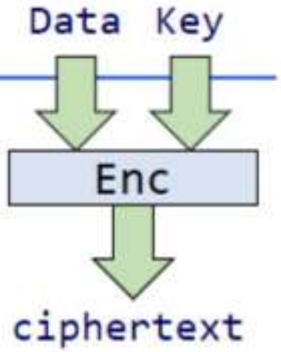
Attacks on protocol layers



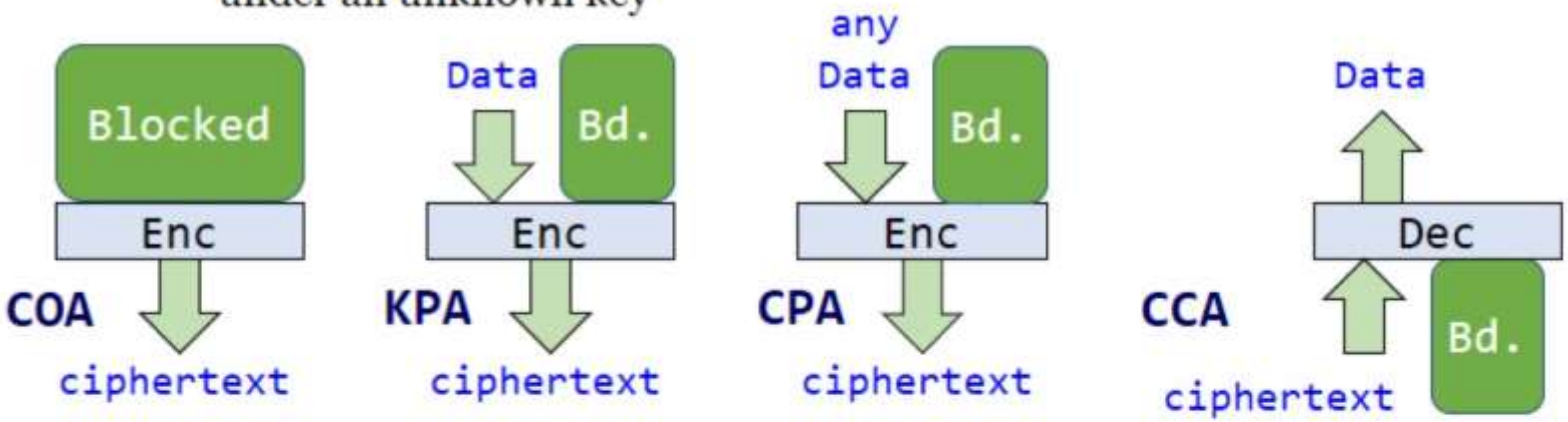
Security supporting protocol layers



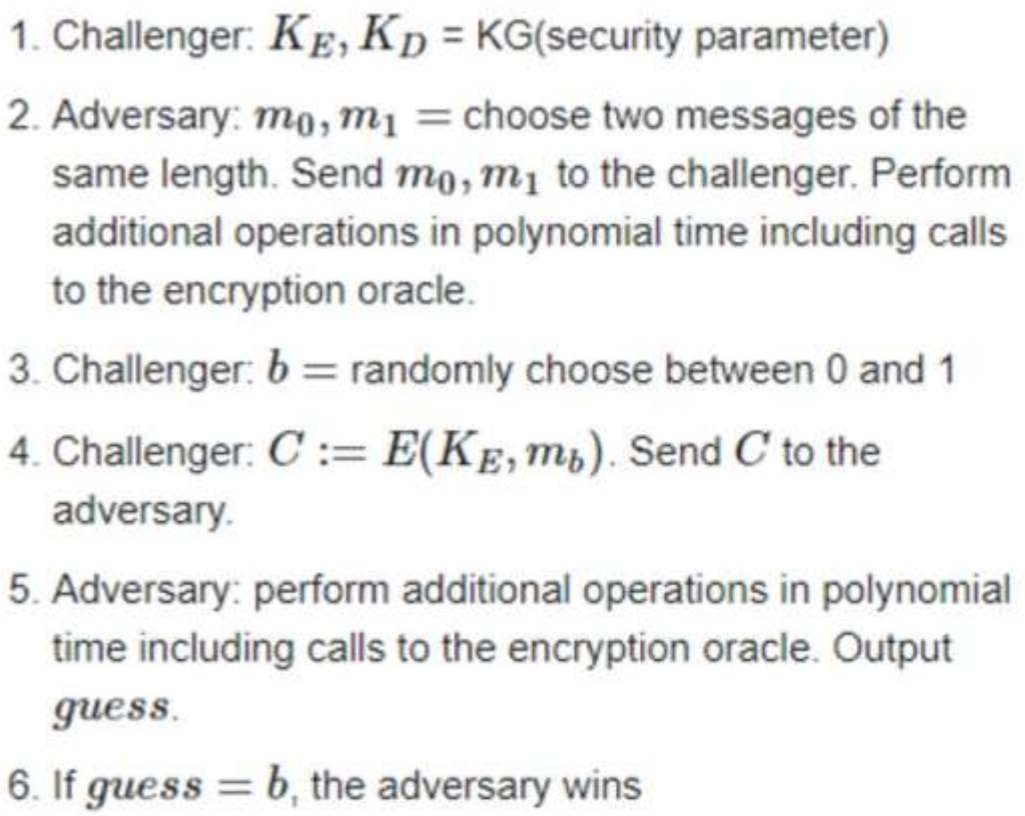
Types of cryptographic attacks



- Attack Models
  - Ciphertext-Only Attack (**COA**)
    - Attackers have access only to a set of ciphertexts
  - Known-Plaintext Attack (**KPA**)
    - Attackers have samples of both the plaintext, and its encrypted version (ciphertext)
  - Chosen-Plaintext Attack (**CPA**)
    - Attackers have the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts
  - Chosen-Ciphertext Attack (**CCA**)
    - Attackers have the capability to choose a ciphertext and obtaining its decryption under an unknown key

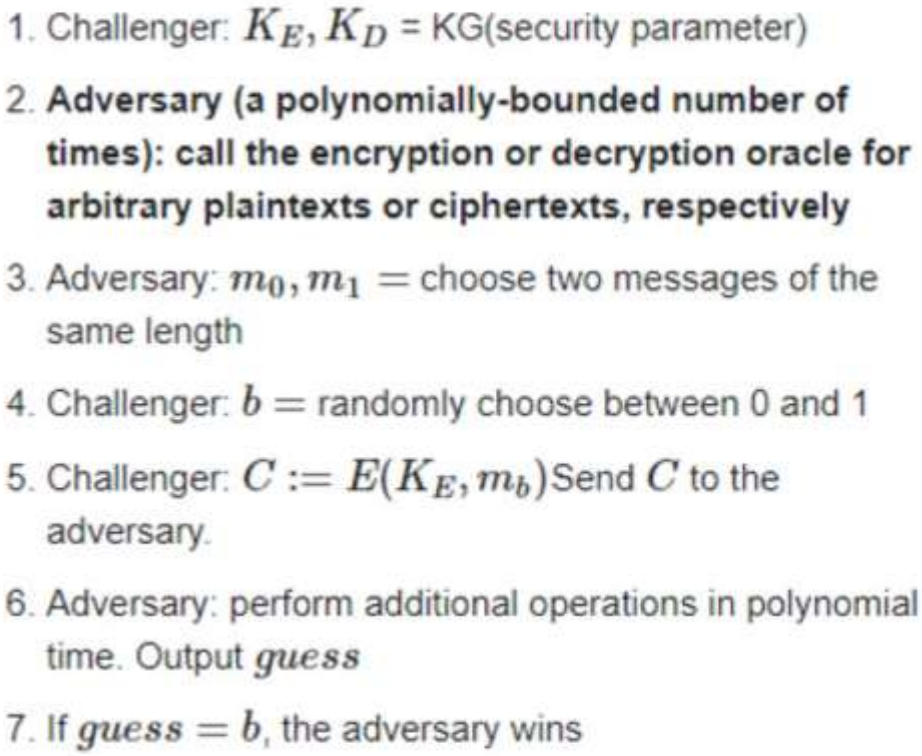


- IND-CPA: **IND**istinguishability under CPA
  - In words
    - the adversary generates two messages of equal length.
    - The challenger decides, randomly, to encrypt one of them.
    - The adversary tries to guess which of the messages was encrypted.
  - Algorithm



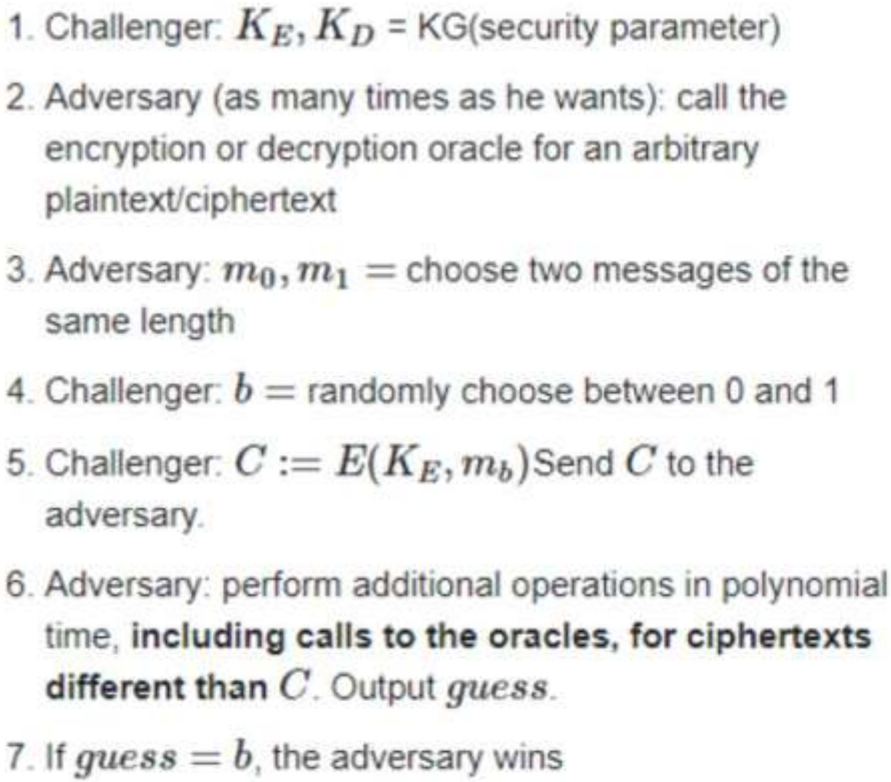
1. Challenger: K_E, K_D = KG(security parameter)
2. Adversary: $m_0, m_1 =$ choose two messages of the same length. Send m_0, m_1 to the challenger. Perform additional operations in polynomial time including calls to the encryption oracle.
3. Challenger: $b =$ randomly choose between 0 and 1
4. Challenger: $C := E(K_E, m_b)$. Send C to the adversary.
5. Adversary: perform additional operations in polynomial time including calls to the encryption oracle. Output $guess$.
6. If $guess = b$, the adversary wins

- IND-CCA1: **IND**istinguishability under CCA1
  - In words: the target of the game is the same as in IND-CPA.
    - The adversary has an additional capability:
      - to call an encryption or decryption oracle.
      - That means: the adversary can encrypt or decrypt arbitrary messages before obtaining the challenge ciphertext.
  - Algorithm:



1. Challenger: K_E, K_D = KG(security parameter)
2. **Adversary (a polynomially-bounded number of times): call the encryption or decryption oracle for arbitrary plaintexts or ciphertexts, respectively**
3. Adversary: $m_0, m_1 =$ choose two messages of the same length
4. Challenger: $b =$ randomly choose between 0 and 1
5. Challenger: $C := E(K_E, m_b)$Send C to the adversary.
6. Adversary: perform additional operations in polynomial time. Output $guess$
7. If $guess = b$, the adversary wins

- IND-CCA2: **IND**istinguishability under CCA2
  - In words: In addition to its capabilities under IND-CCA1,
    - the adversary is now given access to the oracles after receiving C, but cannot send C to the decryption oracle.
  - Algorithm:



1. Challenger: K_E, K_D = KG(security parameter)
2. Adversary (as many times as he wants): call the encryption or decryption oracle for an arbitrary plaintext/ciphertext
3. Adversary: $m_0, m_1 =$ choose two messages of the same length
4. Challenger: $b =$ randomly choose between 0 and 1
5. Challenger: $C := E(K_E, m_b)$Send C to the adversary.
6. Adversary: perform additional operations in polynomial time, **including calls to the oracles, for ciphertexts different than** C. Output $guess$.
7. If $guess = b$, the adversary wins

- Using the decryption oracle after knowing the ciphertext

Where we are ?

- Introduction
- Basic Encryption and ciphers
  - Substitution encryption
    - Character level encryption
    - Bit level encryption
  - One time pad (OTP)
  - Block ciphers
  - Stream ciphers
- Cryptography tutorial
  - Secret key cryptography
    - DES, IDEA, AES, …
  - Public key cryptography
    - RSA, Diffie_Hellman, ECC, …
  - Hash and Message digests
    - MD5, SHA-1, …

Substitution encryption (character level)

- Caesar Cipher
  - Julius Caesar
  - $C = \boldsymbol{E_k}(p) = (p + k) \bmod (26)$
    $p = \boldsymbol{D_k}(C) = (C - k) \bmod (26)$
  - Example
    - Letter + 3 (k=3)
      - DEAR (plaintext) → GHDU (cipher text)

- Mono-alphabetic (simple) substitution
  - Rather than just shifting the alphabet in Caesar cipher
    - Shuffle the letters arbitrarily
  - Each plaintext letter maps to a different random ciphertext letter

```
Plain:  abcdefghijklmnopqrstuvwxyz
Cipher: DKVQFIBJWPESCXHTMYAUOLRGZN

Plaintext:  ifwewishtoreplaceletters
Ciphertext: WIRFRWAJUHYFTSDVFSFUUFYA
```

- Cryptanalysis for Mono-alphabetic Cipher
  - Human languages are redundant
    - Letters are not equally commonly used
      - in English **e** is by far the most common letter
        - then T, R, N, I, O, A, S
      - other letters are fairly rare (cf. Z, J, K, Q, X)
  - Example
    - Given ciphertext:

```
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
```

    - Count relative letter frequencies, guess P & Z are e & t
    - Consider word
    - Finally

```
it was disclosed yesterday that …
```

- Playfair Key Matrix
  - Used in world war I, II
  - Create a 5×5 matrix of letters
    - Fill the matrix in letters of keyword
      - Keyword is important factor
    - Fill remaining field in alphabetic other
  - Example:
    - Keyword = “charles”, plaintext = “the scheme really works”
    - The plaintext is divided into two-letter groups.
      - If there are double letters occurring, in the message,
        - either an 'x' will be used to separate the double letters or
        - an 'x' will be added to make a two-letter group combination.
      - Modified text: “th es ch em er ea ***lx*** ***ly*** wo rk ***sx***”
    - Continued in next page…

c	**h**	**a**	**r**	**l**
e	**s**	b	d	f
g	i/j	k	m	n
o	p	q	t	u
v	w	x	y	z

- Playfair Key Matrix (cont.)
  - Example: (cont.)
    - Keyword = “*charles*”
    - Plaintext = “*the scheme really works*”
    - Modified text: “th es ch em er ea lx ly wo rk sx”

c	**h**	a	**r**	l
e	**s**	**b**	d	f
g	i/j	k	m	n
o	**p**	q	**t**	u
v	w	x	y	z

    - If the two letters are in the same column(↓) of the matrix, (1)
      - use the letter ***below*** it as the cipher text (columns are cyclical).
    - If the two letters are in the same row(←→) of the matrix, (2)
      - use the letter to the ***right*** as the cipher text (rows are cyclical)
    - If neither the same column or row, (3)
      - than each are exchanged with the letter at the intersection of its own row and the other column.
    - Finally

Plain text: **th**(3) **es**(2) **ch** **em** er ea lx ly wo rk sx

Cipher text:**pr**(3) **sb**(2) **ha** **dg** dc bc az rz vp am bw

- Poly-alphabetic substitution cipher
  - Mono-alphabetic substitution with letter position
    - Example
      - 1. a = p, b = m, c = f, …, 2. a = l, b = t, c = a, …, 3. a = f, b = x, c = p, …
        - aaa (plaintext) → plf (ciphertext)
    - Break by decomposing into individual alphabets, then solve as simple substitution
  - Vigenère Cipher (key: deceptive)

plaintext:	wearediscoveredsaveyourself
key:	deceptivedeceptivedeceptive
ciphertext:	SICVTWQNGRZGVTWAVZHCQYGLMGJ

	abcdefghijklmnopqrstuvwxyz
a	abcdefghijklmnopqrstuvwxyz
b	bcdefghijklmnopqrstuvwxyza
c	cdefghijklmnopqrstuvwxyzab
d	defghijklmnopqrstuvwxyzabc
e	efghijklmnopqrstuvwxyzabcd
...	

- Transpositional substitution
  - Rail Fence
    - Write message letters out diagonally over a number of rows, then read off cipher row by row
    - Example: plaintext = “meet me after the toga party”

```
m e m a t r h t g p r y
 e t e f e t e o a a t
ciphertext = "MEMATRHTGPRYETEFETEOAAT"
```

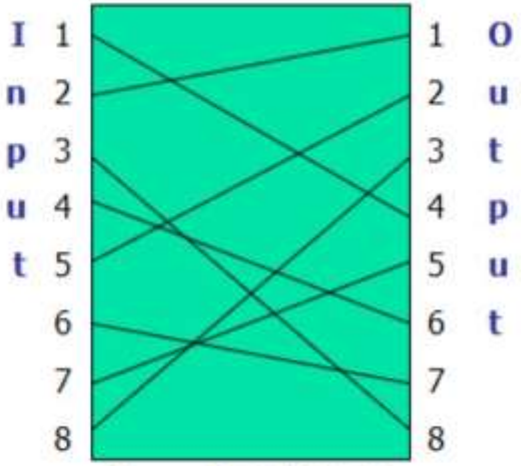
  - Row Transposition Ciphers
    - A more complex scheme
    - Write letters of message out in rows over a specified number of columns,
      - then reorder the columns according to some key before reading off the rows
      - “attack postponed until two...”

```
Key:        3 4 2 1 5 6 7            3 4 2 1 5 6 7
Plaintext:  a t t a c k p            t a t a c k p
            o s t p o n e            t p s o o n e
            d u n t i l t            n t u d i l t
            w o a m x y z            a m o w x y z
Ciphertext: TTNAAPTMTSUOAODWCOIXKNLYPETZ          or
             tatackp…
```

Bit level encryption

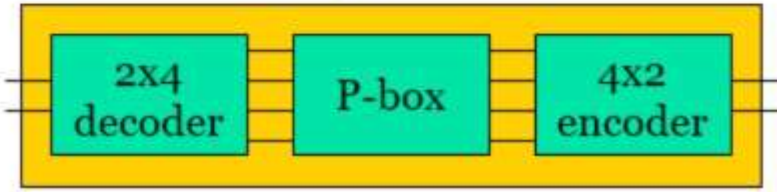
- Encoding/decoding (ex. 8 × 3 encoder, 3 × 8 decoder)
  - Input (n bits) → output (m bits)
- Permutation (치환)
  - Bit level transpositional method
    - Bit shuffle
    - Straight (right figure)
    - Compressed
    - Expanded permutation
  - Called ***P-box***
- Substitution (대치)
  - Bit level substitution (Called ***S-Box***)
  - 000→011, 001→101, ...
- Product
  - Chained combination of P-box and S-box
- Exclusive OR: ⊕
  - **Sender**: 0101(data) ⊕ 0111(key) = 0010, **Receiver**: 0010 ⊕ 0111 = 0101
- Rotation (right, left)

P-Box



Key=25817463

product



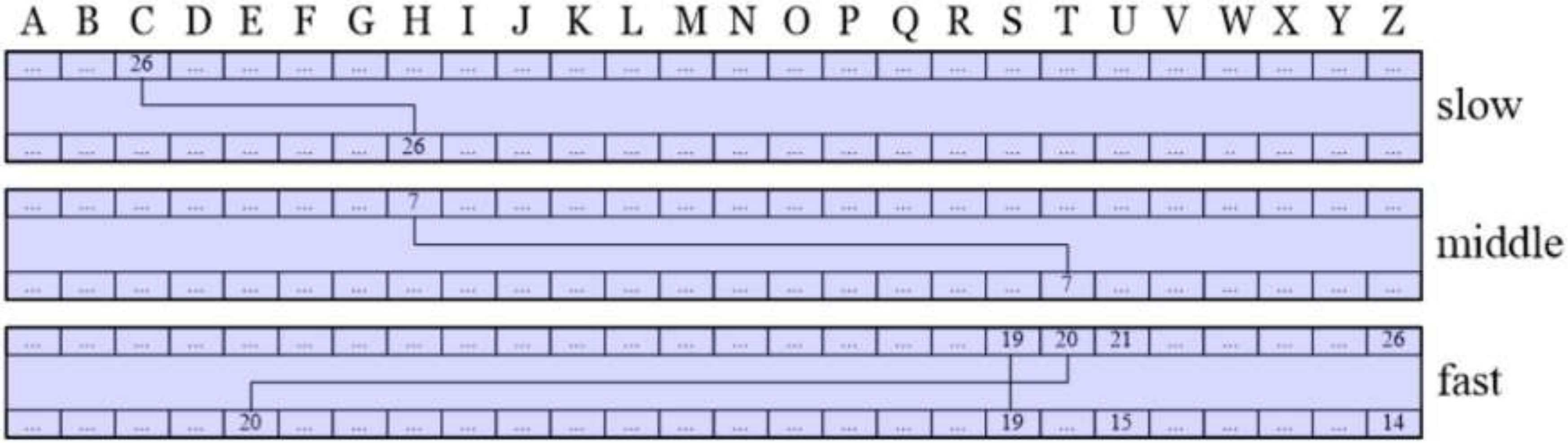
One Time Pad (OTP)

- In 1917
- Sometimes called ***Vernam*** cipher
- Basically you have your random OTP, which both you and your intended recipient have
  - Sender: Ciphertext = plaintext ⊕ OTP
  - Receiver: plaintext = Ciphertext ⊕ OTP = plaintext ⊕ OTP ⊕ OTP
    = plaintext ⊕ 0 = plaintext
  - Example
    - Plaintext: "ATTACK"
    - OTP: 4, 8, 20, 10, 16, 1
    - Ciphertext: "EAMKSL"
      - A ⊕ 4 (100 0001 ⊕ 000 0100) → E (100 0101)
      - E ⊕ 4 (100 0101 ⊕ 000 0100) → A (100 0001)
  - Instead of ⊕, the vigenère Cipher may be used.
- You must *never* re-use the OTP, other wise it wouldn't be a "One-Time" pad anymore

Cipher machines

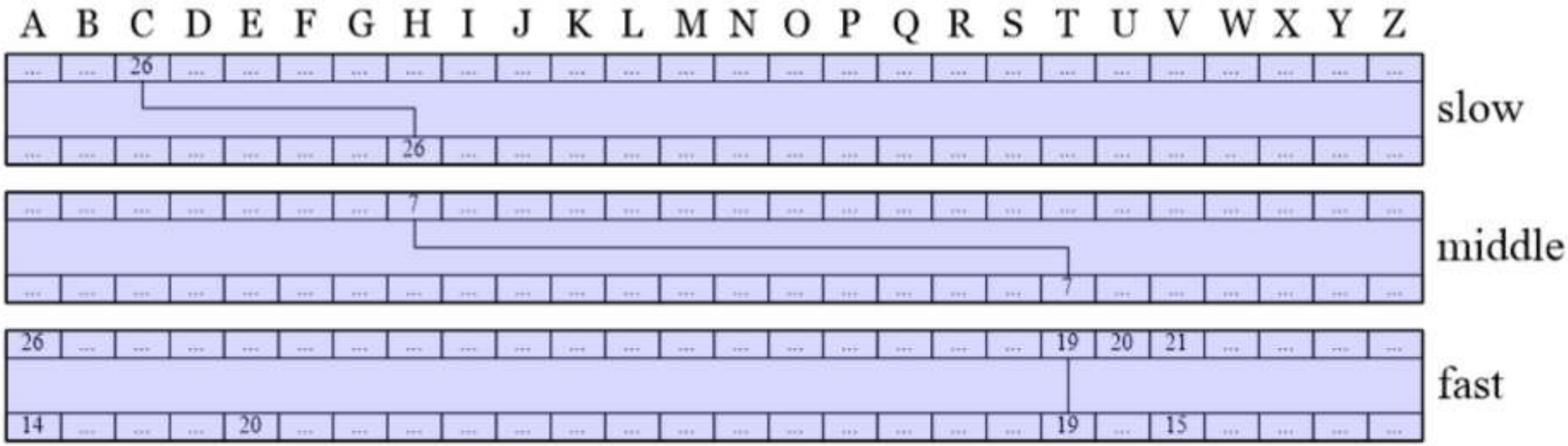
- The Enigma machine (Rotor)
  - The basic Enigma was invented in 1918
    - Used by both Germany and Japan in World War II
  - Improved rotor machines were used into the 70's and 80's

- Initial setting



  - "C" is encrypted to "E"

- After encryption of 1 character
  - Cylinder for ‘fast’ is one character shifted right (‘C’ → ‘T’)



- Cylinder Shift
  - After encryption of 1 character
    - Cylinder for ‘fast” is shifted right one
  - After encryption of 26 ‘fast’ shift
    - Cylinder for ‘middle” is shifted right one
  - After encryption of 26 ‘middle’ shift
    - Cylinder for ‘slow” is shifted right one

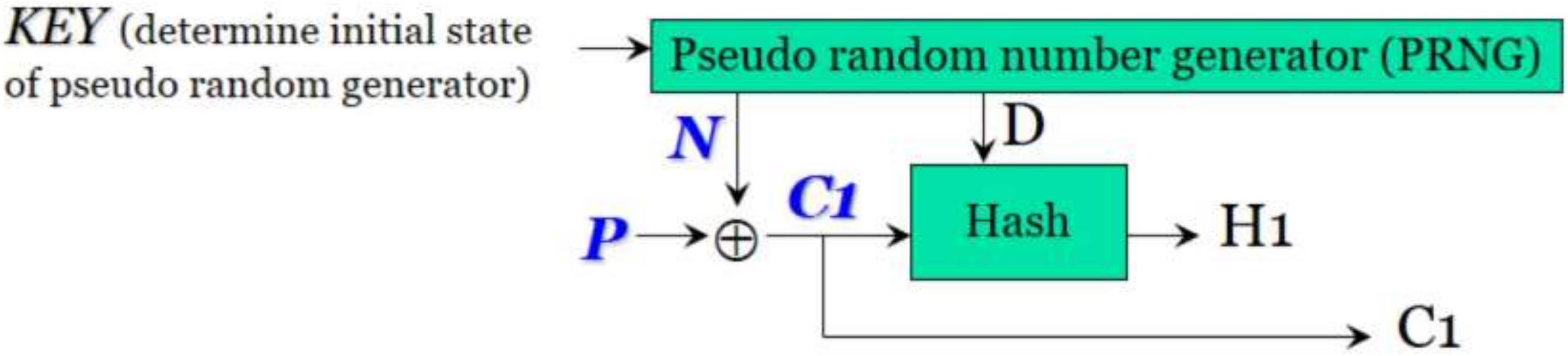
Two types of symmetric ciphers

- Block cipher
  - Operation
    - Breaking up data into fixed-size blocks (64, 128, ... bits)
    - Encrypt each block individually
    - Slower than stream cipher
  - Example
    - DES, 3DES, **AES**, Skipjack, Blowfish, IDEA, ...

- Stream cipher
  - Plaintext의 각 비트가 bit stream (key stream) bit와 exclusive OR
    - ciphertext(i) bit = plaintext(i) ⊕ key_stream(i)
  - Example
    - Salsa20, ...

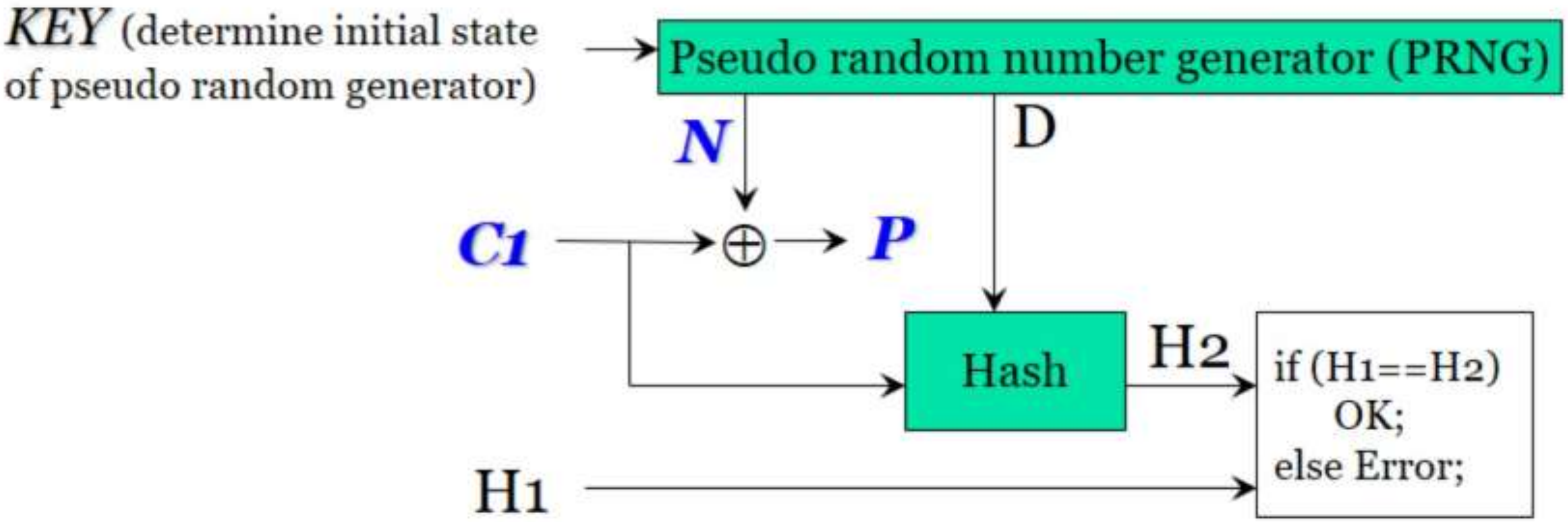
Stream Ciphers

- Generate pseudo random #, XOR with data
- Sender
  - Starts with P, the plaintext. It is *n* bits long.
  - Generates N, a string of *n* bits, using the ***pseudorandom generator***.
  - Computes $C_1 = P \oplus N$.
  - Generates D, a string of bits the size of a key, using the pseudorandom generator.
  - Now computes $H_1 = \text{hash}(C_1, D)$.
  - Transmits $C_1 + H_1$ as the ciphertext for P.
    - If (transmit only C1) then anyone can modify C1 (bit flipping attack)

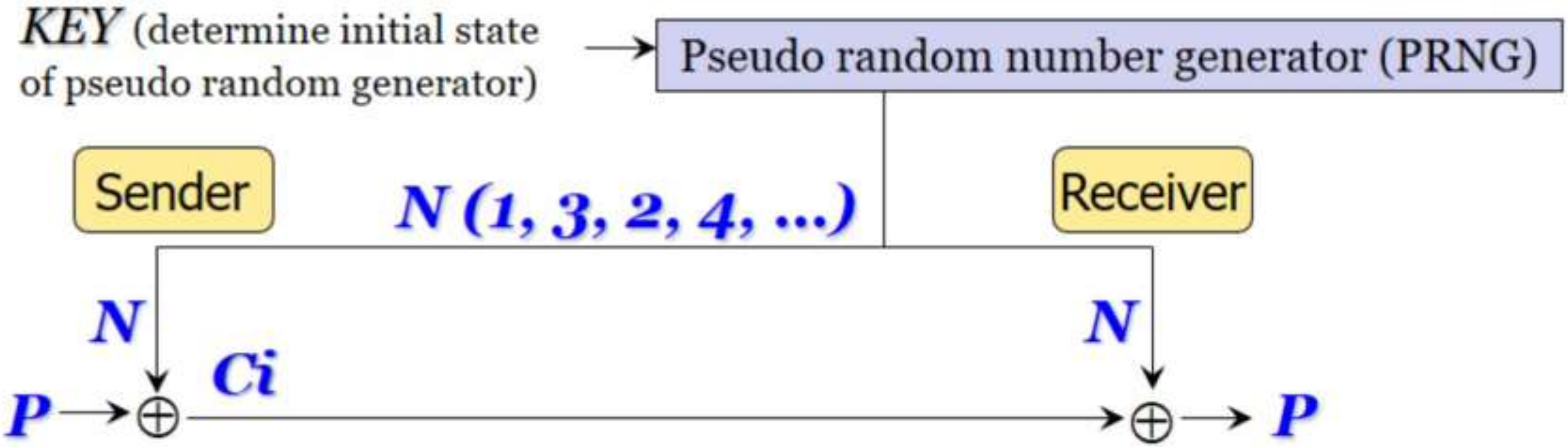


KEY (determine initial state of pseudo random generator)
Pseudo random number generator (PRNG)
N
D
P
⊕
C1
Hash
H1
C1

- Receiver
  - Receives $C_1 + H_1$
  - Generates N, a string of n bits, using the pseudorandom generator.
  - Computes $P = N \oplus C_1$, so he has the plaintext.
  - Generates D, using the pseudorandom generator.
  - Calculates $H_2 = \text{hash}(C_1, D)$ and checks to make sure it matches H_1.


KEY (determine initial state of pseudo random generator)
Pseudo random number generator (PRNG)
N
D
C1
⊕
P
Hash
H2
if (H1==H2)
OK;
else Error;
H1

- Overall example



KEY (determine initial state of pseudo random generator) → Pseudo random number generator (PRNG)

Sender

N (1, 3, 2, 4, …)

Receiver

N

Ci

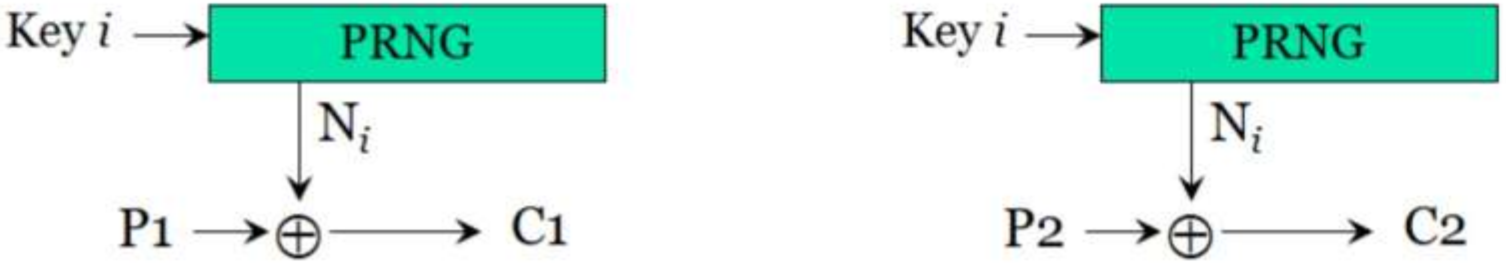
N

P → ⊕ → ⊕ → ***P***

	A⊕1	A⊕1 ⊕1
“Attack”	t⊕3	t⊕3 ⊕3
	t⊕2	t⊕2 ⊕2
	a⊕4	a⊕4 ⊕4

Stream Ciphers: problems

- If you use the *same key* for plaintext P1 and plaintext P2, then anyone can do



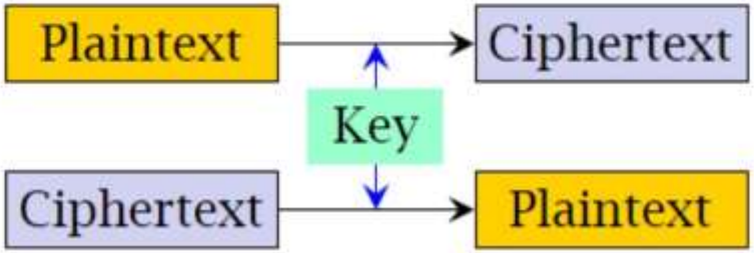
  - $C1 \oplus C2 = (P1 \oplus N_1) \oplus (P2 \oplus N_1)$
    $= (P1 \oplus P2) \oplus (N_1 \oplus N_1) = (P1 \oplus P2) \oplus 0$
    $= (P1 \oplus P2)$
  - It's a pretty easy to tease apart into two separate messages.
    - If messages are separated, the original plaintext can be obtained easily.
  - If attacker know P1 then → $C1 \oplus C2 \oplus P1 = P1 \oplus P2 \oplus P1 = P2$
- All pseudorandom generators are periodic
  - If pseudorandom generator loops one cycle, same effect on security as using the same key twice.

Where we are ?

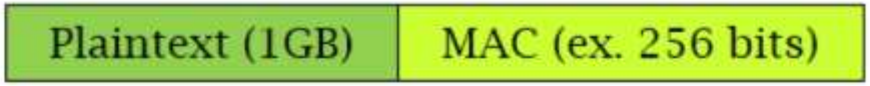
- Introduction
- Basic Encryption and ciphers
  - Substitutional encryption (Character Level)
  - Bit level encryption
  - One Time Pad (OTP)
  - Cipher Machines
  - Block Cipher, Stream Ciphers
- Cryptography tutorial
  - **Secret key cryptography**
    - DES, AES, IDEA, ...
  - **Public key cryptography**
    - RSA, Diffie_Hellman, ECC, ...
  - **Hash and Message digests**
    - SHA-2/3, MD5, ...

Secret key cryptography

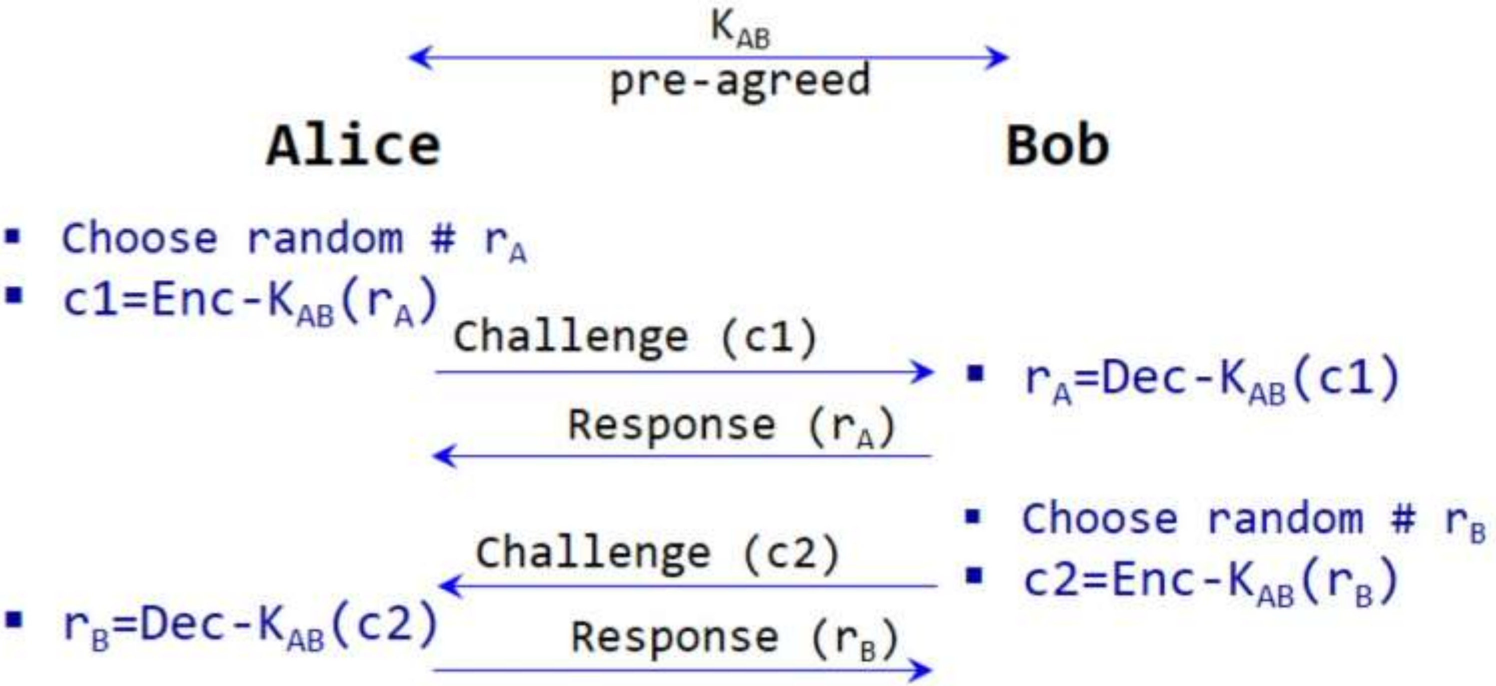
- Sometimes called as ***conventional cryptography*** or ***symmetric cryptography***
  - 송수신 측에서 같은 key를 공유
  - Usual key length of 128~256 bits
  - DES, AES, IDEA, RC2/4/5, Blowfish
  - Usually quite fast (공개키 방식에 비해)
    - Used for bulk data encryption
  - Key management is a problem



- Usage
  - Transmitting over an insecure channel
  - Secure storage on insecure media
  - Integrity check
    - Given *a key and a message*, the algorithm produce a fixed length ***message authentication code (MAC)***
      - MAC is often called a MIC (message integrity code), in PEM&S-MIME
    - If anyone were to modify the message, and they didn't know the key, they would have to guess a MAC.
  - Continued...



- Usage (cont.)
  - Authentication
    - Challenge/Response
      - K_{AB}: Shared secret key



K_{AB}

pre-agreed

Alice

Bob

- Choose random # r_A
- $c1=Enc\text{-}K_{AB}(r_A)$

Challenge (c1)

- $r_A=Dec\text{-}K_{AB}(c1)$

Response (r_A)

- Choose random # r_B
- $c2=Enc\text{-}K_{AB}(r_B)$

Challenge (c2)

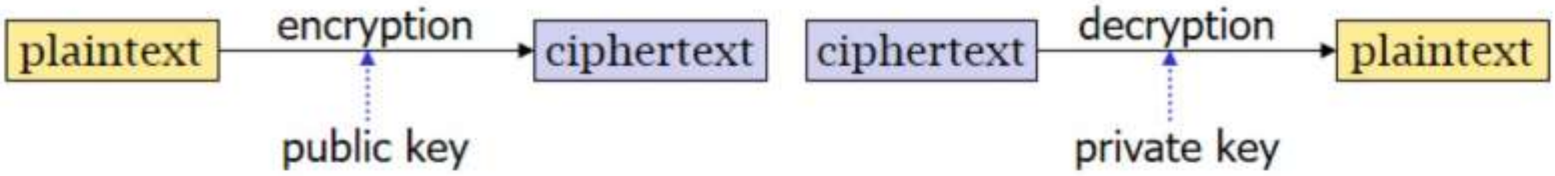
- $r_B=Dec\text{-}K_{AB}(c2)$

Response (r_B)

Mutual authentication

Public key cryptography

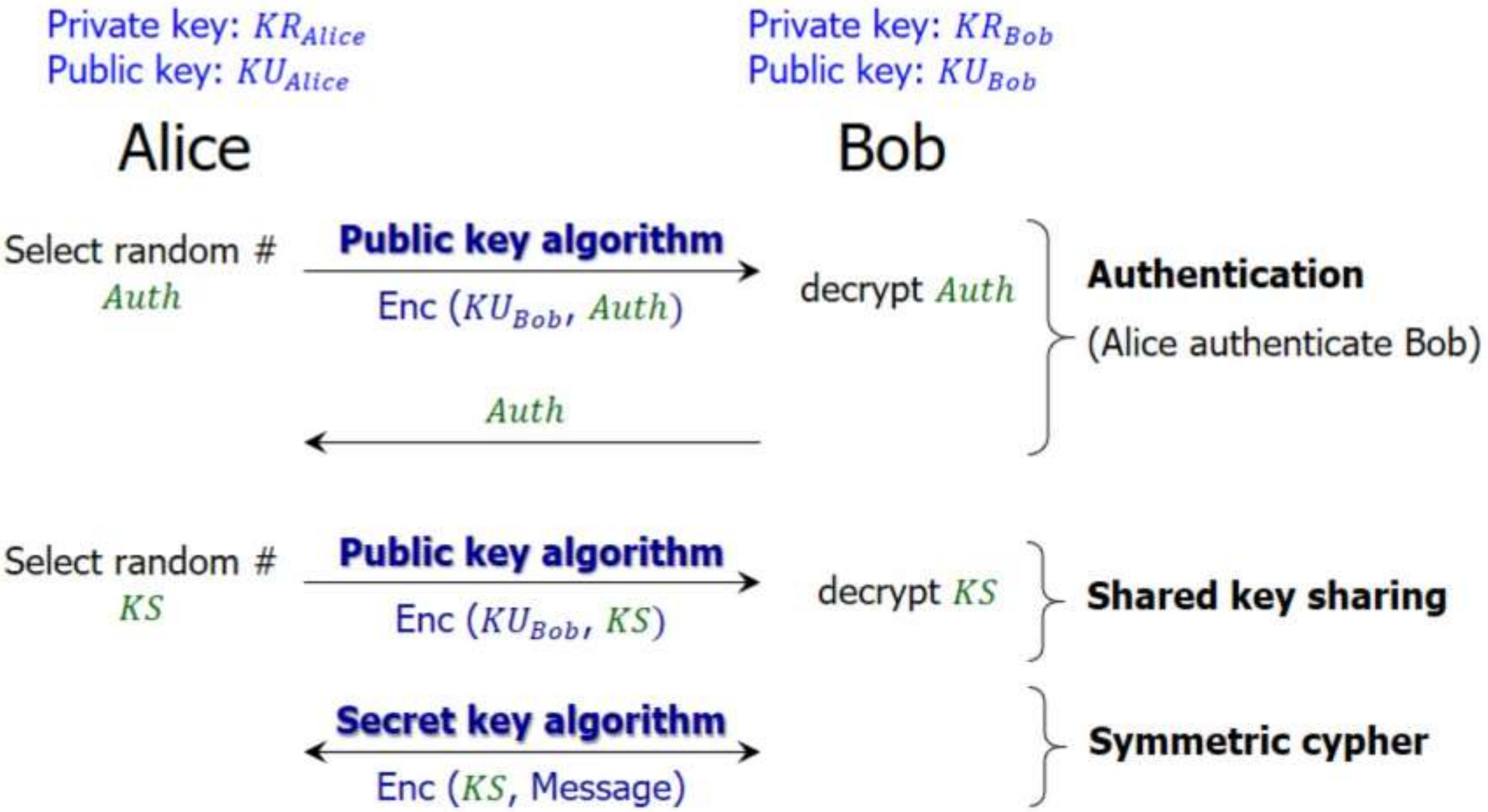
- Sometimes called as ***Asymmetric Cryptography***
- Keys are not shared
  - 2 pair keys
    - ***Private key***: saved securely
    - ***Public key***: known to entire world
  - Usual key length of 1024 ~ 4096 bits
  - RSA, Diffie-Hellman, Elgamal, Elliptic curves (256/384 bits key), …
  - Relatively slow
    - Used in low volume encryption service
  - Simple key management
- Mechanism
  - Anyone can encrypt message using public key
  - Decryption can be done only using a private key
  - 반대도 성립



- Asymmetric key 방식은
  - Symmetric key 방식으로 할 수 있는 모든 기능을 제공하고
    - 비밀키 암호로 지원할 수 없는 기능도 제공 가능 (예: non-repudiation)
  - Symmetric key 방식에 비하여 처리 속도가 느림
    - 키 길이: 비밀키 방식(128~512 bits), RSA (2048~4096 bits)

    → Elliptic curve public key algorithm: 256/384 bits

- Two algorithms are usually used together
  - Asymmetric key 방식은
    - 통신 초기에 인증과 임시로 사용할 대칭키(temporal secret key)를 생성하는데 사용
  - 이후에는 symmetric key 방식으로 데이터를 암호화
    - 생성된 temporal secret key를 이용



Private key: KR_{Alice}
Public key: KU_{Alice}

Alice

Private key: KR_{Bob}
Public key: KU_{Bob}

Bob

Select random # $Auth$

Public key algorithm

Enc (KU_{Bob}, $Auth$)

decrypt $Auth$

$Auth$

Authentication

(Alice authenticate Bob)

Select random # KS

Public key algorithm

Enc (KU_{Bob}, KS)

decrypt KS

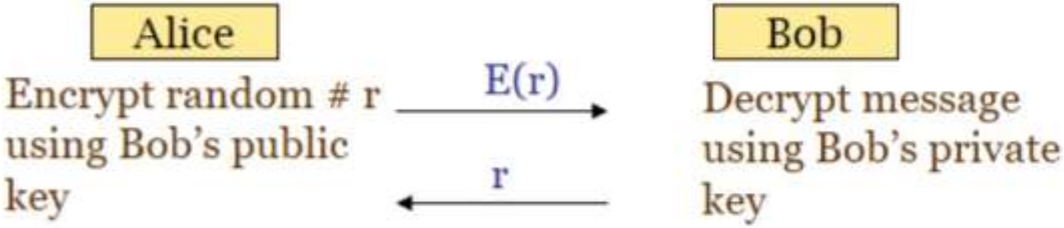
Shared key sharing

Secret key algorithm

Enc (KS, Message)

Symmetric cypher

- Usage
  - Transmitting over an insecure channel
  - Secure storage on insecure media
  - Authentication



Alice

Encrypt random # r using Bob's public key

E(r)

r

Bob

Decrypt message using Bob's private key

How can we acquire bob's public key? →

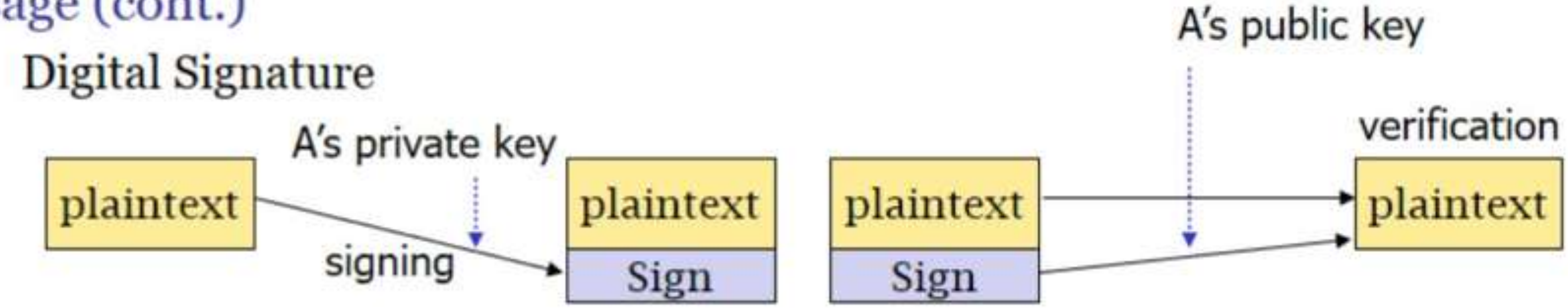
Someone give me a "Bob's public key" →

is really bob?

Is it really bob's public key? →

PKI (public key infrastructure), (public) certificate

- Usage (cont.)
  - Digital Signature



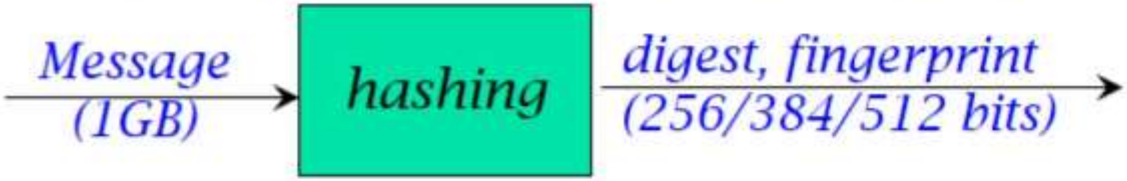
- Prove who generate the information
- Prove that the information has not been modified
- Offer an important advantage over secret key based cryptography MAC
  - ***Non-repudiation***
    - Bob이 Alice에게 물건을 파는 경우를 가정
    - 3자에 의한 거짓 주문의 가능성을 없애기 위하여, Alice와 Bob은 키를 공유하고, 주문서는 공유된 키를 이용하여 암호화
    - 두 가지 방법이 가능: secret key based MAC 또는 public key based signature
    - Secret key based MAC 방법: 나중에 Alice가 주문서를 보낸 사실을 부정하면, Bob이 이를 증명할 수 없음 (Bob와 Alice가 secret shared key를 공유하고 있으므로, Alice는 Bob이 주문서를 임의로 만들었다고 주장)
    - Public key based signature:

      - Alice는 주문서를 자신의 private key로 encryption 하여 전송

      - Alice가 자신이 보낸 주문을 부인할 수 없음

Hash Algorithm

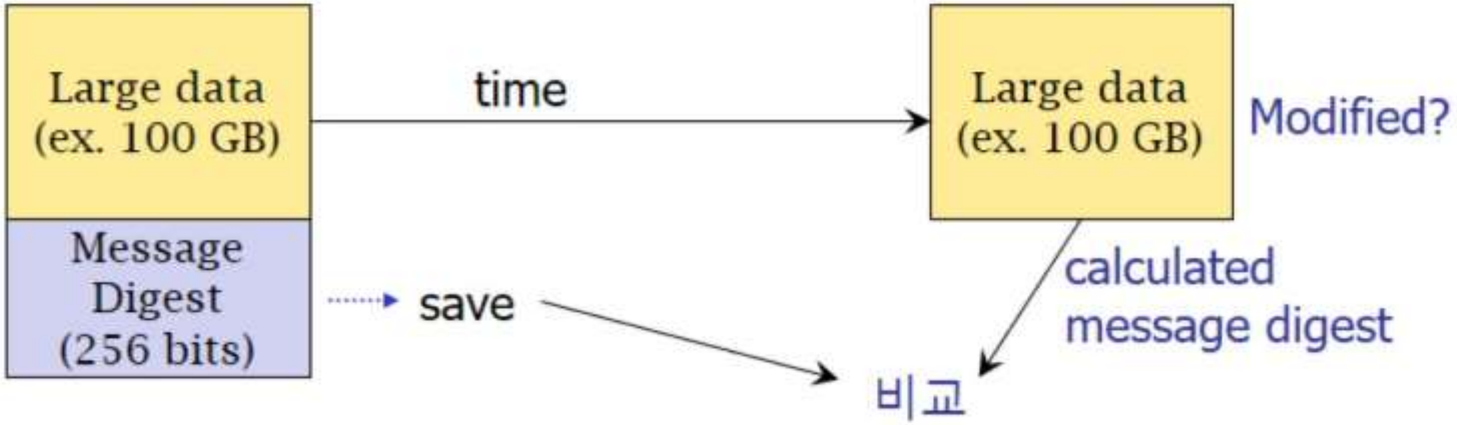
- Also known as ***message digests*** or ***one-way transformations***
  - A cryptographic hash function is a mathematical transformation that takes a message of arbitrary length and computes from it a fixed-length (short) number.
    - Fixed length short number is a ***message digest*** or ***fingerprint***
  - Simple example
    - Hash_Function (Message, 1GB) → message digest/fingerprint (256 bits)



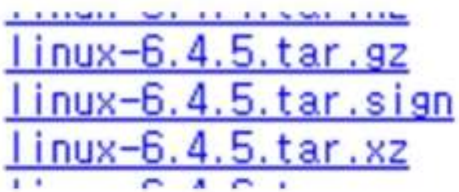
  - Example
    - **M:** 1 G bytes의 메시지
    - **M':** M에서 1비트 변경한 메시지
    - *h(M)* → ***h*** (256 bits)
    - *h(M')* → ***h'***
    - ***h***와 ***h'***은 완전히 다른 값

- Usages
  - Password hashing: “login_name, hash(password)”
  - Message fingerprint
    - If you want to know whether some large data structure has been modified
      - You can simply save the message digests of large data structure, and compare it with the calculated value.
        - If (saved message digest == calculated message digest)

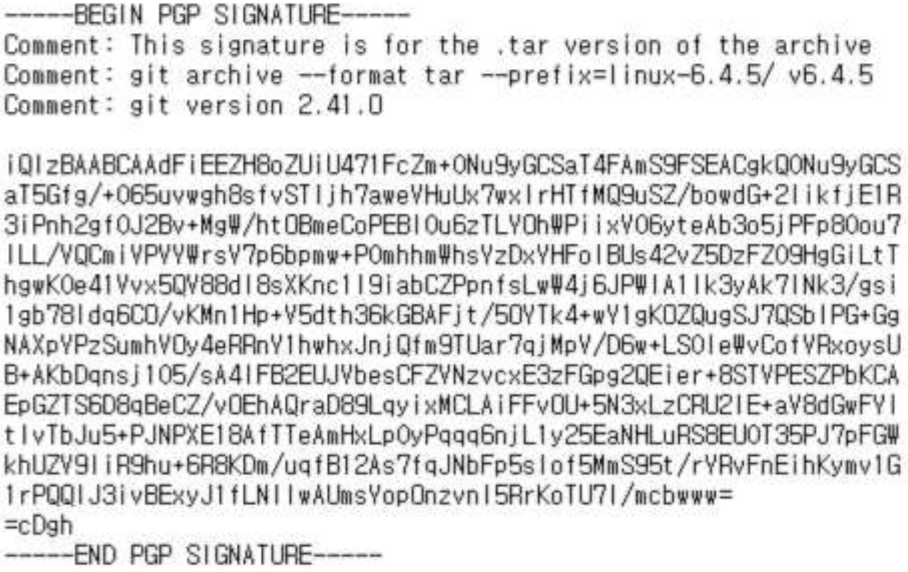
          large data structure is not changed



- Download security
  - Example: some ISO file download
    - "Decompress ISO file, insert virus file, remake ISO file, transfer"
    - Trojan horse attack
  - To verify the downloaded data is not corrupted/modified, use hash function.
    - hash value is publicly known

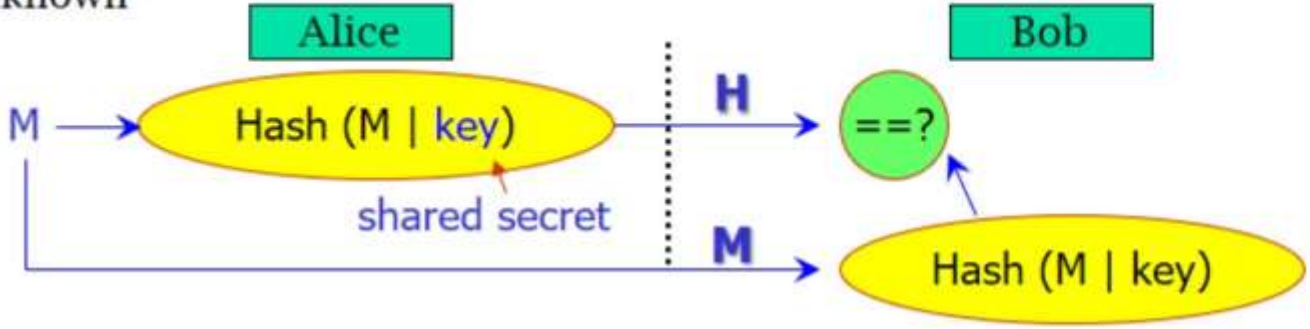


linux-6.4.5.tar.gz
linux-6.4.5.tar.sign
linux-6.4.5.tar.xz

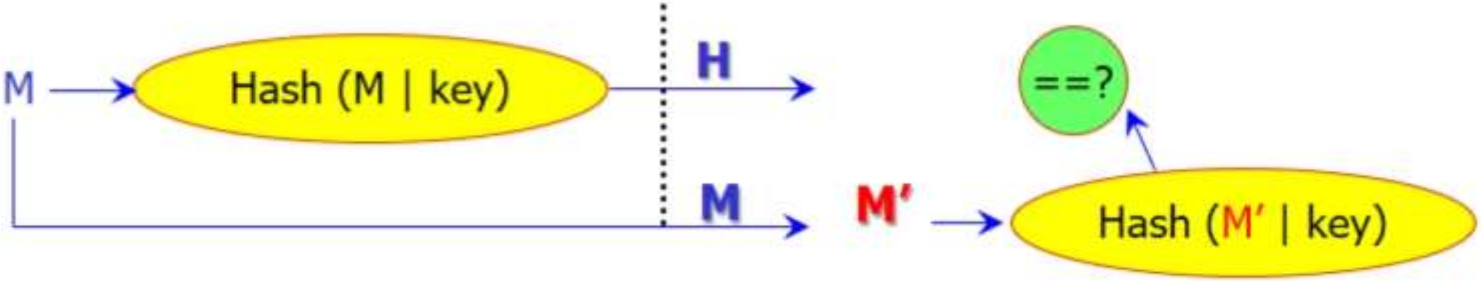


```
-----BEGIN PGP SIGNATURE-----
Comment: This signature is for the .tar version of the archive
Comment: git archive --format tar --prefix=linux-6.4.5/ v6.4.5
Comment: git version 2.41.0

iQIzBAABCAAdFiEEZH8oZUiU471FcZm+ONu9yGCSaT4FAmS9FSEACgkQONu9yGCS
aT5Gfg/+065uvwgh8sfvSTIjh7aweVHuUx7wxIrHTfMQ9uSZ/bowdG+2IikfjE1R
3iPnh2gf0J2Bv+MgW/htOBmeCoPEBIOu6zTLVOhWPiixV06yteAb3o5jPFp80ou7
ILL/VQCmIVPVYWrsV7p6bpmw+POmhhmWhsVzDxVHFoIBUs42vZ5DzFZ09HgGiLtT
hgwKOe41Vvx5QV88dI8sXKnc1I9iabCZPpnfsLwW4j6JPWIA1Ik3yAk7INk3/gsi
1gb78Idq6CO/vKMn1Hp+V5dth36kGBAFjt/50YTk4+wV1gKOZQugSJ7QSbIPG+Gg
NAXpVPzSumhVOy4eRRnY1hwhxJnjQfm9TUar7qjMpV/D6w+LSOIeWvCofVRxoysU
B+AKbDqnsj105/sA4IFB2EUJVbesCFZVNzvcxE3zFGpg2QEier+8STVPESZPbKCA
EpGZTS6D8qBeCZ/vOEhAQraD89LqyixMCLAiFFvOU+5N3xLzCRU2IE+aV8dGwFVI
tIvTbJu5+PJNPXE18AfTTeAmHxLpOyPqqq6njL1y25EaNHLuRS8EUOT35PJ7pFGW
khUZV9IiR9hu+6R8KDm/uqfB12As7fqJNbFp5sIof5MmS95t/rYRvFnEihKymv1G
1rPQQIJ3ivBExyJ1fLNIIwAUmsYopOnzvnI5RrKoTU7I/mcbwww=
=cDgh
-----END PGP SIGNATURE-----
```

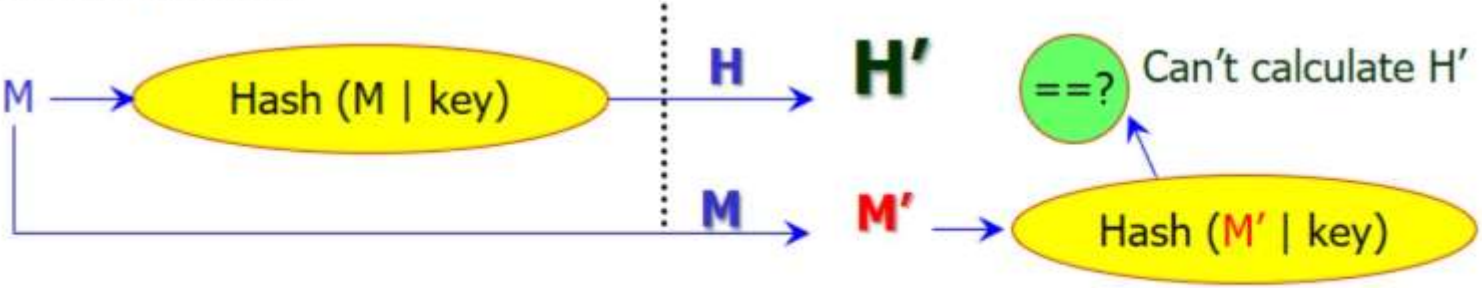
- Message integrity
  - Modify message & Calculate appropriate hash value is impossible
    - key is unknown



  - Attacker



  - For successful attack



기타

- Security related standards
  - NIST (National Institute of Standards and Technology)
    - Ex. FIPS PUB 186-5
  - ISO
    - Ex. ISO 27002
  - NSR (국가보안기술연구소)
- NSA (National Security Agency)
- Post quantum cryptography (PQC)
  - Standardization by NIST

Authentication

- Three factors
  - knowledge factors
    - Something the user knows (e.g., a password, pass phrase, ...)
  - ownership factors
    - Something the user has (e.g., ID card, security token, implanted device, cell phone, ...)
  - inference factors
    - Something the user is or does (e.g., fingerprint, DNA sequence, ...)
- Single-factor authentication
- Multi-factor authentication (MFA) / two-factor authentication (2FA)
  - a way of 'double checking'
  - When setting up 2FA, the service will ask you to provide a 'second factor', which is something that you (and only you) can access.
    - SMS
    - Authenticator Apps on your smart phone

Example attacks

Side channel attack

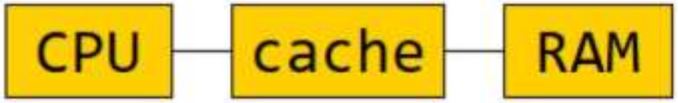
- Wikipedia
  - A side-channel attack is any attack based on *information gained from the implementation* of a computer system,
    - rather than weaknesses in the implemented algorithm itself (e.g. cryptanalysis and software bugs).
  - Timing information,
  - power consumption,
  - electromagnetic leaks or
  - even sound can provide an extra source of information, which can be exploited.

- Side channel attack example: **Timing attack**
  - password check code (ex. 4 bytes password)

```
bool check_password (char input_pwd[], char actual_pwd[]) {
   for (int i=0; i<4; i++) {
       sleep(1);   // for simple
       if(input_pwd[i] != actual_pwd[i])
           return False;
   }
   return True;
}  // assume, real password is "cbdd"
```

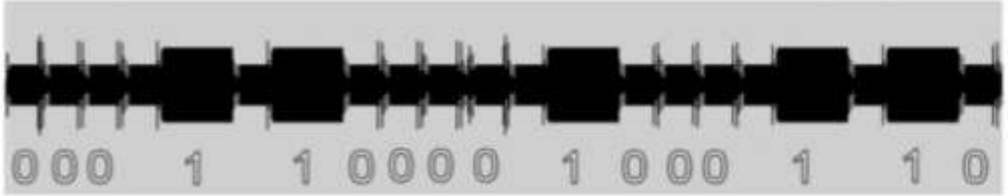
  - How to find first character of password?
    - "**a**aaa" → 1 sec   // return with i=0
    - "**b**aaa" → 1 sec   // return with i=0
    - "**c**aaa" → 2 sec   // return with i=1   ← hit !
    - "**d**aaa" → 1 sec   // return with i=0
    - Try with "**ca**aa", "**cb**aa", "**cc**aa", "**cd**aa", ...

- Side channel attack example: **cache attack on AES cipher**
  - All modern computers have a “cache” between main memory (RAM) and the CPU. → for speed up
  - If an area of memory has been accessed recently → it’s probably in the cache → so accessing it again will be quick.
    - If the area hasn’t been accessed in a long time → it has probably been evicted from the cache → so accessing it will be slow.
    - The difference in time can leak information about what a process is doing.
  - Fast implementations of AES use lookup tables, which are arrays used to quickly convert one value into another
    - The lookup table can be at cache or memory.

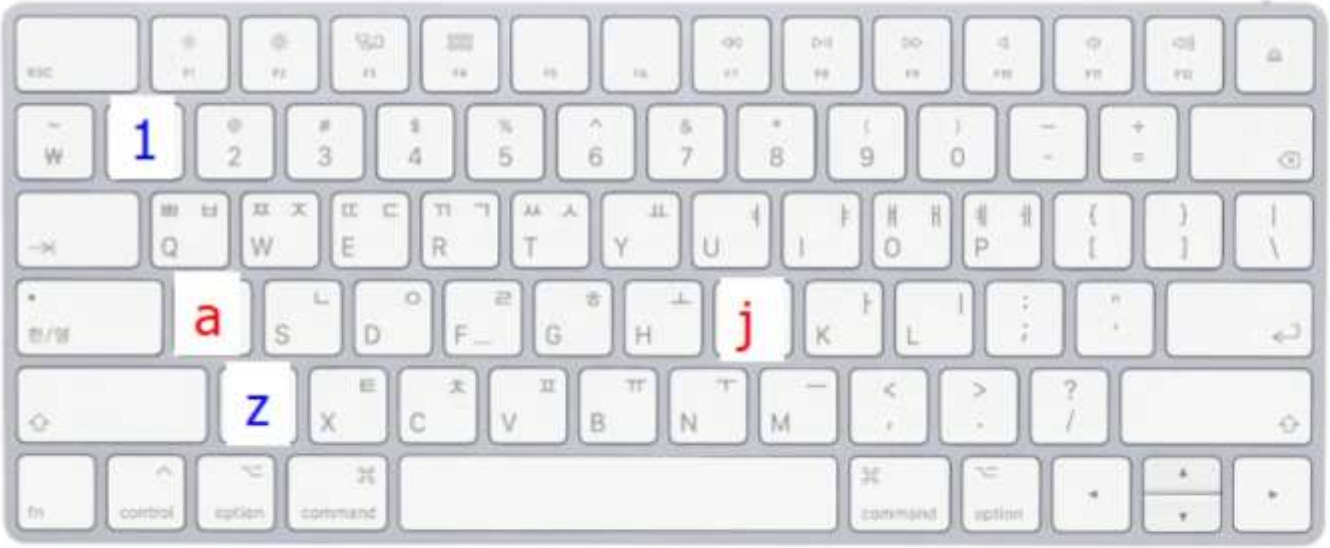


- Side channel attack example: **cache attack on AES cipher (cont.)**
  - <Cache attack of AES>
    - Completely fills the cache with the attacker's data.
    - AES encryption
    - The attacker accesses their data again (to see which parts have been evicted from the cache), and this tells them which table indexes were used by the encryption process, leaking information about the key.
  - <Cache timing attack on AES>
    - Make all of the AES lookup tables are in the cache except for one index in one table.
    - AES encryption
    - If the encryption accesses that index that has been evicted from the cache, it will run slower, otherwise it will run at a normal speed.
      - So, by timing how long the encryption takes, they can figure out if that table index was accessed or not.
      - Since the table index depends on the key, this leaks information about the key.

- Side channel attack example: **Power Analysis of RSA**
  - Encrypting a message involves "square and multiply."
  - Loop over all bits in the secret exponent, and for each bit,
    - if it is one → perform a multiply operation then a square operation.
    - If the bit is zero → do not perform the multiply, just go straight to the square operation.
  - Timing attack is possible
  - by watching the amount of power a device uses, you can determine whether the multiply routine executed or not.



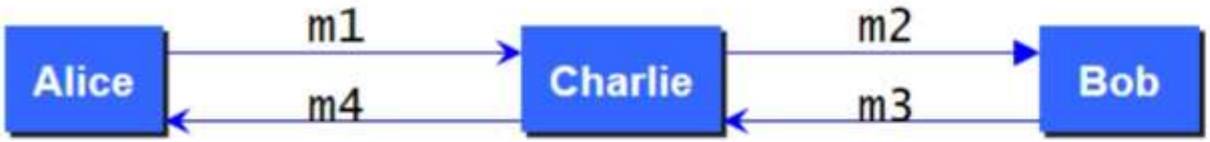
- Side channel attack example: **Timing Analysis of Keystrokes inside SSH**
  - For example, on a QWERTY keyboard, the time between pressing “a” then “j” is going to be a lot shorter than the time between pressing “z” and “1”



  - You can tell what someone is typing just by analyzing the sound of the fingers on the keyboard, from a certain distance.
    - Acoustic cryptanalysis

Attacks

- Man in the middle (MITM) attack



  - Alice wants to send secure messages to Bob.
  - Charlie talks to Bob and pretends to be Alice.
  - Charlie talks to Alice and pretends to be Bob.

- Brute Force Attacks
  - Described

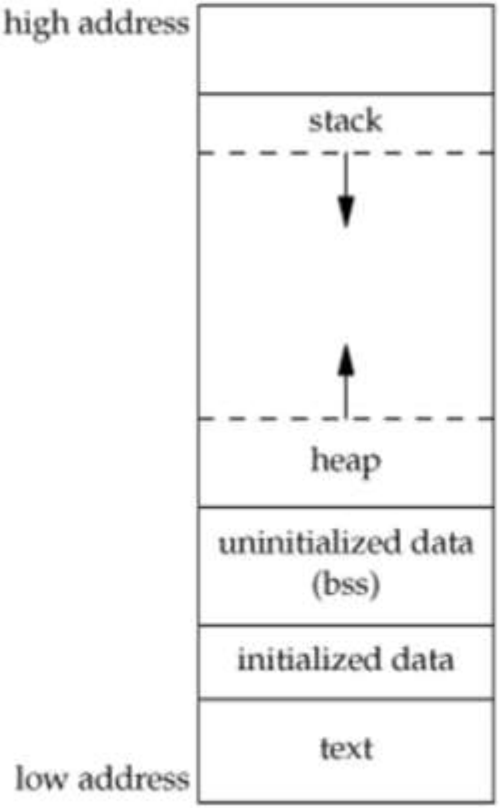
Buffer overflow attack

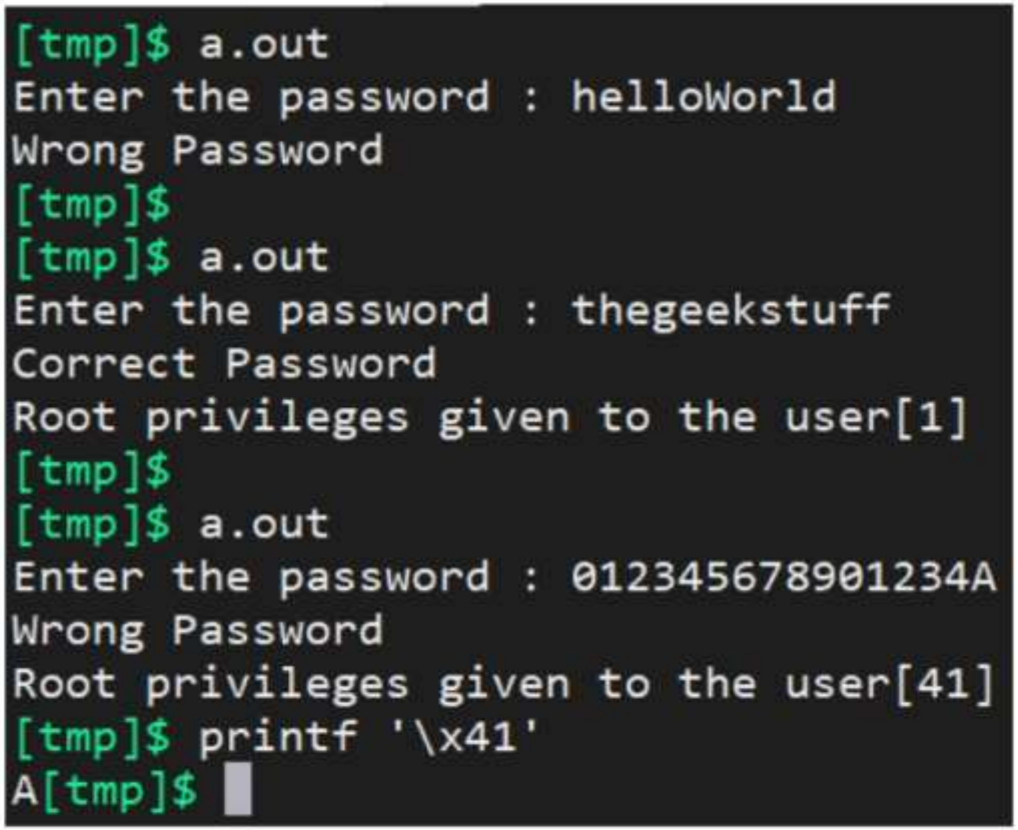
- Example

```
#include <stdio.h>
#include <string.h>

int main (void) {
    int pass;
    char buff[15];

    printf ("Enter the password : ");
    gets (buff);

    if (strcmp(buff, "thegeekstuff")!=0) {
        printf ("Wrong Password \n");
        pass = 0;
    }
    else {
        printf ("Correct Password \n");
        pass = 1;
    }

    if (pass) {
      /* Now Give root rights to user*/
        printf ("Root privileges...[%x]\n", pass);
    }
}
```

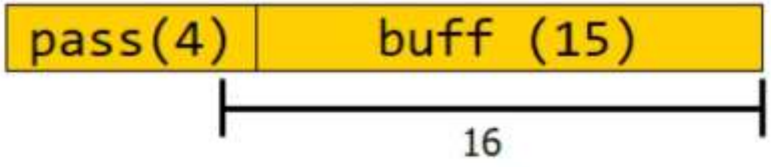
- Compile with "-fno-stack-protector"

```
% gcc   example.c  -fno-stack-protector
```



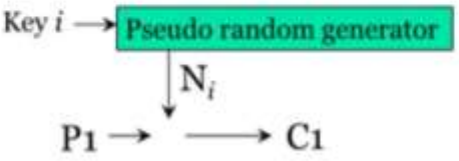


```
[tmp]$ a.out
Enter the password : helloWorld
Wrong Password
[tmp]$
[tmp]$ a.out
Enter the password : thegeekstuff
Correct Password
Root privileges given to the user[1]
[tmp]$
[tmp]$ a.out
Enter the password : 012345678901234A
Wrong Password
Root privileges given to the user[41]
[tmp]$ printf '\x41'
A[tmp]$ 
```



Backdoors

- Types
  - Self-contained malware program that reside on a victim's machine
  - Malicious properties that are built into an cryptographic algorithms
- Example: Dual_EC_DRBG
  - Dual-EC pseudo-random number generator
    - NSA & NIST (?)



  - If you know a certain property about the Dual_EC parameters, you can predict all subsequent outputs of PRNG.
  - This is a huge deal in the case of SSL/TLS, for example.
    - If I use the Dual-EC PRG to generate the "Client Random" nonce transmitted in the beginning of an SSL connection,
      - then the NSA will be able to predict the "Pre-Master" secret that I'm going to generate during the RSA handshake.
    - Given this information the connection is now a cleartext read.

Kali Linux/Tools

- KALI LINUX is a security distribution of Linux derived from Debian
  - and specifically designed for computer forensics and advanced penetration testing.
  - Kali Linux has over 600 preinstalled penetration-testing applications to discover.
- Tools
  - Nmap
  - Metasploit

CVE, CWE

- CVE (Common Vulnerabilities and Exposures)
  - A list of entries for publicly known cybersecurity vulnerabilities.
    - Each containing an identification number, a description, ...
  - Refers to a specific instance of a vulnerability within a product or system.
  - CVE Entries are used in numerous cybersecurity products and services from around the world
- CWE (Common Weakness Enumeration)
  - CWE refers to the types of software weaknesses, rather than specific instances of vulnerabilities within products or systems.
- NVD (National Vulnerability Database)
  - The U.S. government repository of standards-based vulnerability management data
  - The NVD actually uses CWEs to score CVEs.