SWE3025: Computer Security
Lecture 0x06: Crypto and TLS II

Hojoon Lee

Systems Security Lab @ SKKU

Answering Your Voice

Answering Your Voice

박종원(2014****25)

화요일

안녕하세요, 교수님.

강의의 마지막 부분에 try all keys 가 best attack 인 경우 secure 하다고 하셨는데,

Caesar's cipher 의 Cryptanalysis 같은 경우도 try all keys 를 하면 되는데, 그러면 Caesar's cipher 도 secure 한 Cryptosystem 이라고 볼 수 있는 건가요?

key 의 총 개수가 적은 경우에도 해당되는 건지 의문이 들어 질문드립니다.

감사합니다.

댓글 작성...

Answering Your Voice

- I admit that the wording was not very clear
- As the **박종원** points out, the security of the cryptosystem also greatly depends on the size of the key
  - For instance, AES256 is much safer than AES128
- What I meant was "*the security of the cipher itself*" is robust when exhaustive key search is the only option

Answering Your Voice

- Caesar cipher, just like the substitution permutation cipher,
  - is vulnerable to statistics-based inference
  - limited keysize (n < 26)
- Overall, it's a terrible cipher

Answering Your Voice

강경운(2015**71)**

금요일

안녕하세요, 교수님

강의 마지막 부분 3줄에 대한 설명이 빠지셨는데

왜 short cut attack이 가능한 crptosystem이 break가 어려운지 궁금합니다!

댓글 작성...

Systems Security Lab

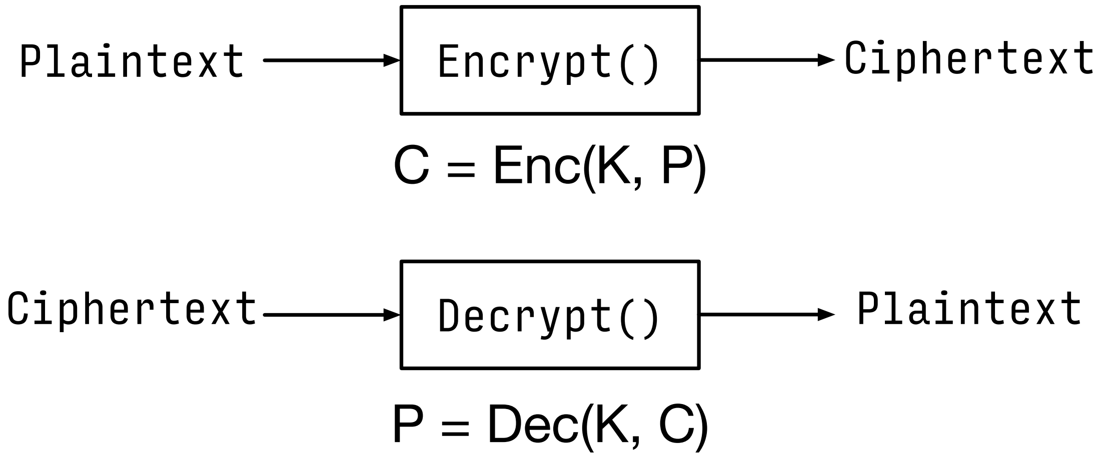
SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

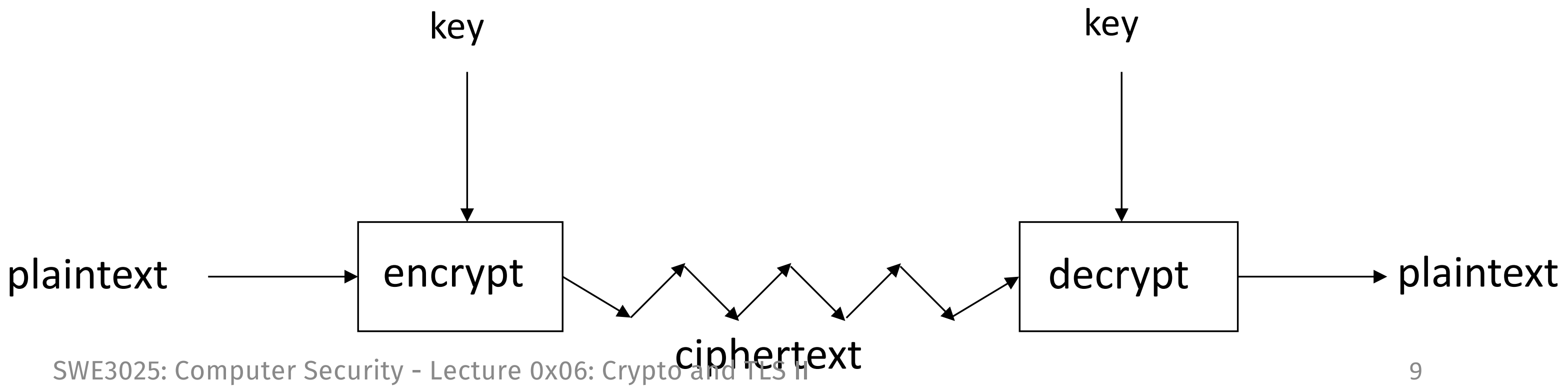
Answering Your Voice

Symmetric-Key Cryptography

Symmetric-Key Cryptography

- Uses the same key for encryption/decryption
- Assumption: Sender and Receiver already have a shared secret key





Symmetric Key Crypto

- Stream cipher — generalize one-time pad
  - {en/de}crypted 1 byte at a time
  - Good for {en/de}crypting unknown size of data
  - A different "key" is generated for each block depending on the previous blocks
  - Fast in SW and HW
- Block cipher — generalized codebook
  - Data is broken up into chunks of fixed size
  - Good for {en/de}crypting fixed size of data
  - The same "key" is used at each block
  - Fast in HW implementation (e.g., x86 AES-NI)

Symmetric Key Crypto

Block Cipher (AES) vs Stream Cipher (Chacha20)

- Two most widely used symmetric-key ciphers
- Google websites use Chacha20
- Performance
  - AES is fast when HW support is present (x86 AES-NI)
  - Chacha20 is faster when implemented in pure SW
- Chacha20 is more suitable for mobile devices without AES HW support

For example: decrypting a 1MB file on the Galaxy Nexus (OMAP 4460 chip):

- AES-128-GCM: 41.6ms
- ChaCha20-Poly1305: 13.2ms

One-Time Pad (Simplest Form of Stream Cipher)

`e=000   h=001   i=010   k=011   l=100   r=101   s=110   t=111`

Encryption: Plaintext ⊕ Key = Ciphertext

	h	e	i	l	h	i	t	l	e	r
Plaintext:	001	000	010	100	001	010	111	100	000	101
Key:	111	101	110	101	111	100	000	101	110	000
Ciphertext:	110	101	100	001	110	110	111	001	110	101
	s	r	l	h	s	s	t	h	s	r

One-Time Pad (Simplest Form of Stream Cipher)

`e=000   h=001   i=010   k=011   l=100   r=101   s=110   t=111`

Decryption: Ciphertext ⊕ Key = Plaintext

	s	r	l	h	s	s	t	h	s	r
Ciphertext:	110	101	100	001	110	110	111	001	110	101
Key:	111	101	110	101	111	100	000	101	110	000
Plaintext:	001	000	010	100	001	010	111	100	000	101
	h	e	i	l	h	i	t	l	e	r

One-Time Pad

- Provably secure
  - Ciphertext gives no useful info about plaintext
  - All plaintexts are *equally likely*
- BUT, only when be used correctly
  - Pad must be random, used only once
  - Pad is known only to sender and receiver
- Note: pad (key) is same size as message

One-time Pad

- Modern stream ciphers have the same concept except the key size is fixed (it's not equal to msg size)
- The symmetric key s is used for modern stream ciphers (e.g., chacha20, RC6,RC7) are used as a seed
- The seed is used to generate a stream of bits that seems random but deterministically computed from s

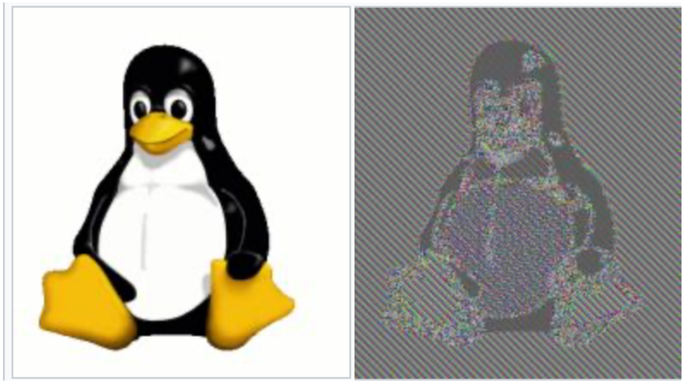
Codebook Cipher (Simplest form of Block Cipher)

- Literally, a book filled with "codewords"
- Zimmerman Telegram encrypted via codebook

Februar	13605
fest	13732
finanzielle	13850
folgender	13918
Frieden	17142
Friedenschluss	17149
:	:

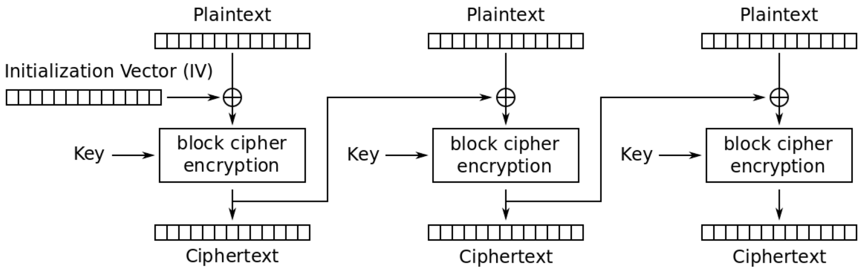
Codebook Cipher (Simplest form of Block Cipher)

- Modern block ciphers are codebooks



- Electronic Code Book
  - No diffusion, does not hide data patterns very well

- Cipher Blocker Chaining
  - Each block of plaintext if XOR'd with the previous ciphertext block
  - Ciphertext block depends on all previous blocks
  - Note: This is not Blockchain, we will cover Blockchain separately in this course if we have time ☺…



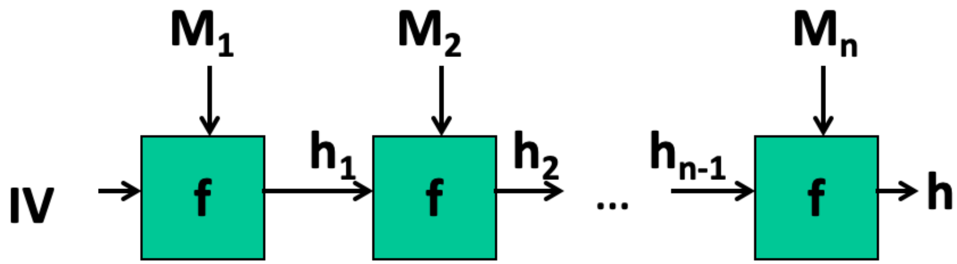
Symmetric-Key Cryptography

- Secure enough symmetric-key Ciphers today
  - CHACHA20
  - AES128
- Insecure symmetric keys today
  - DES
  - RC1,RC2,RC3,RC4
- Losers of the game (The "why not just use AES?" category)
  - Blowfish
  - RC5,RC6
  - Triple DES

Hash-based Message Authentication Code (HMAC)

Hash Functions

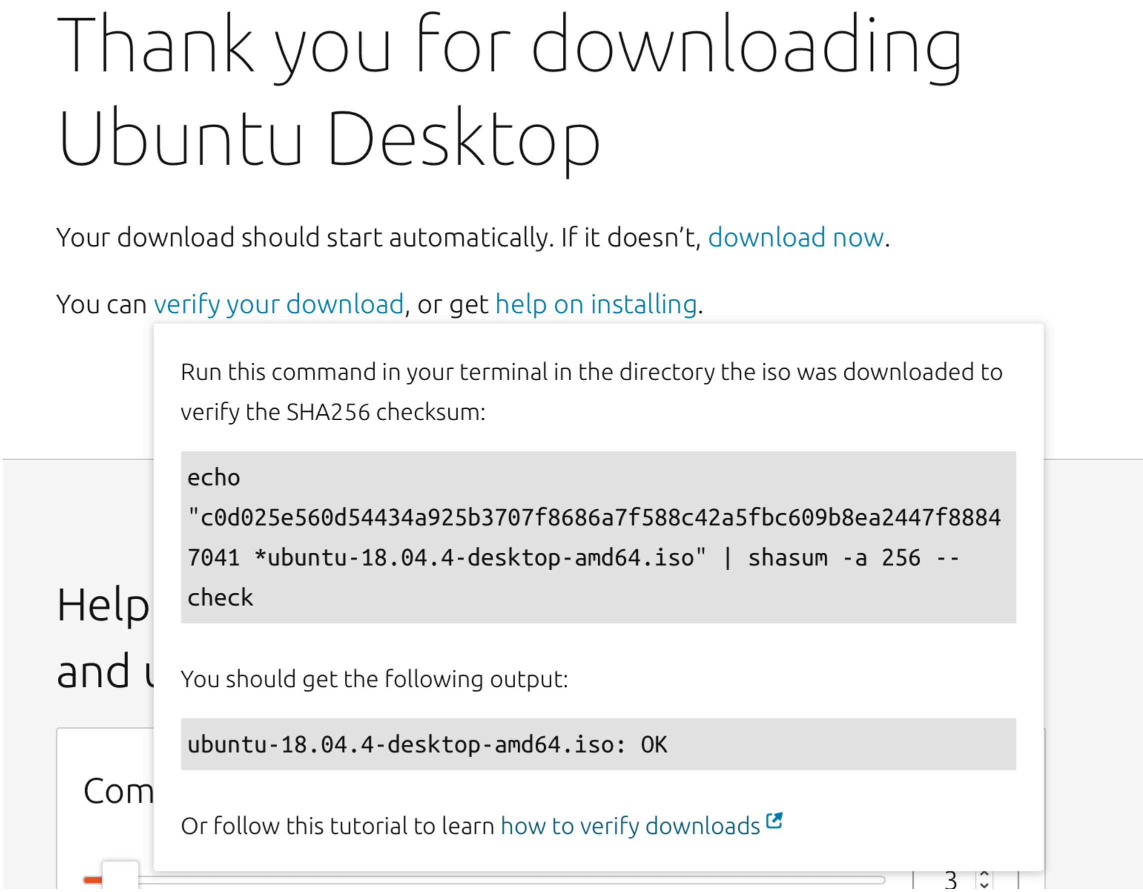
- Hashing is a one-way only encryption
  - No such thing as unhashing or dehashing
- There is no key used in hashing
  - $H(m) = h$ vs. $Enc(key_{enc}, m) = c$
- Fast computation time



Hash Functions

- Purpose: produce a fixed-size "fingerprint" or digest of arbitrarily long input data
- Hash passwords such that password plaintext need not be saved on the service or server
- To guarantee integrity

Hash Functions



MAC

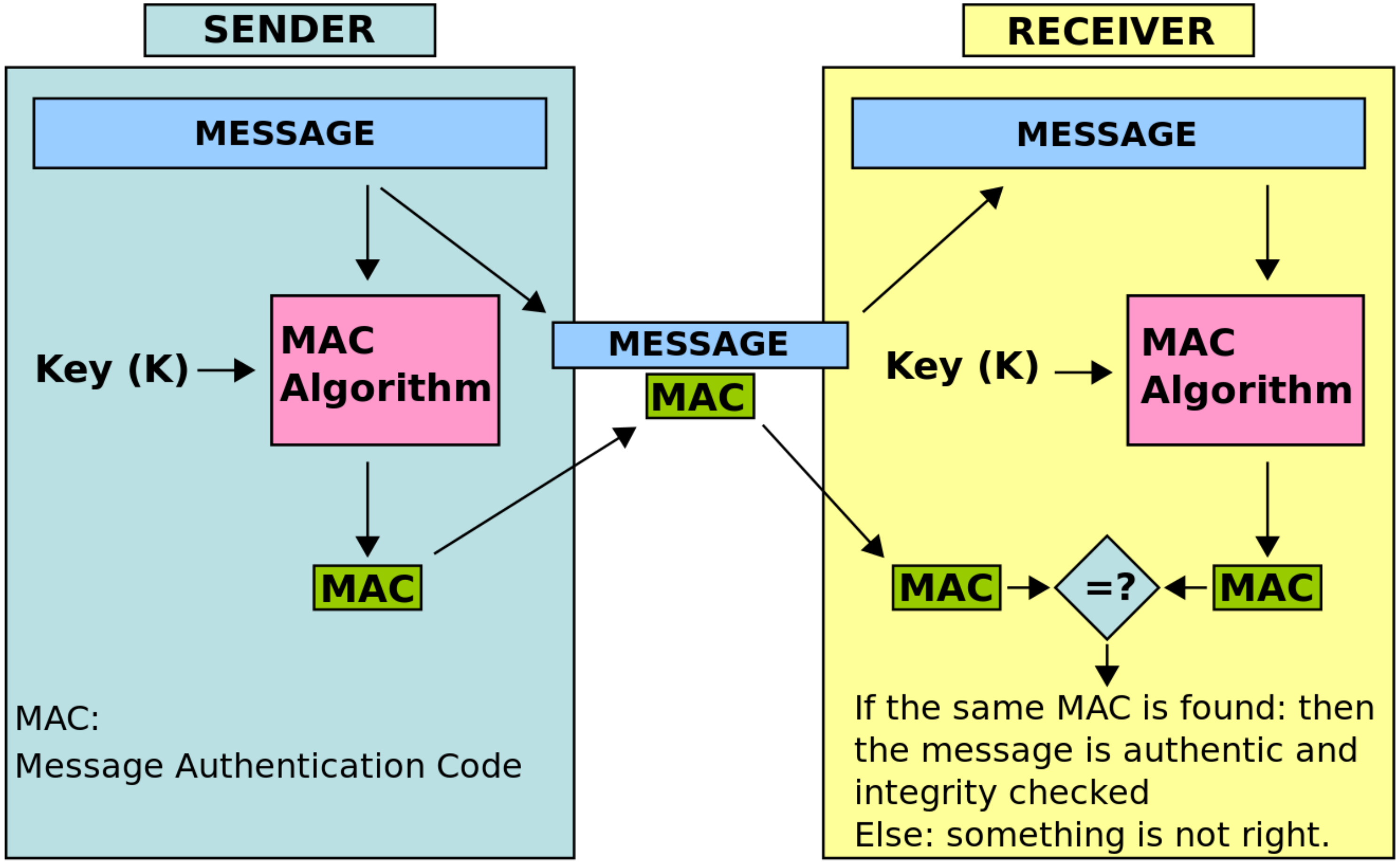
- Message Authentication Code (MAC)
- One-way Function (Basically a Hash function with a key) that creates a message *digest*
  - e,g, MAC(k,m) = d
- A digest is appended at the end of the message, so that the receiver can verify it

MAC vs Hash

- Key is used during computation
- Ensures integrity and authenticity of the message
- A shared key is need to verify a MAC

- Key is not used during computation
- Ensures only integrity
- Everyone can verify a hash

MAC


SENDER
MESSAGE
Key (K)
MAC Algorithm
MAC
MAC:
Message Authentication Code
MESSAGE
MAC
RECEIVER
MESSAGE
Key (K)
MAC Algorithm
MAC
=?
MAC
If the same MAC is found: then the message is authentic and integrity checked
Else: something is not right.

HMAC

- Hash-based Message Authentication Code (HMAC)
- Most widely used form of MAC today
- Builds a MAC out of hash functions (e.g., SHA-256)

$$\mathrm{HMAC}(K, m) = \mathrm{H}\left(\left(K' \oplus opad\right) \parallel \mathrm{H}\left(\left(K' \oplus ipad\right) \parallel m\right)\right)$$

$$K' = \begin{cases} \mathrm{H}(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

H is a cryptographic hash function
m is the message to be authenticated
K is the secret key
K' is a block-sized key derived from the secret key

Summary

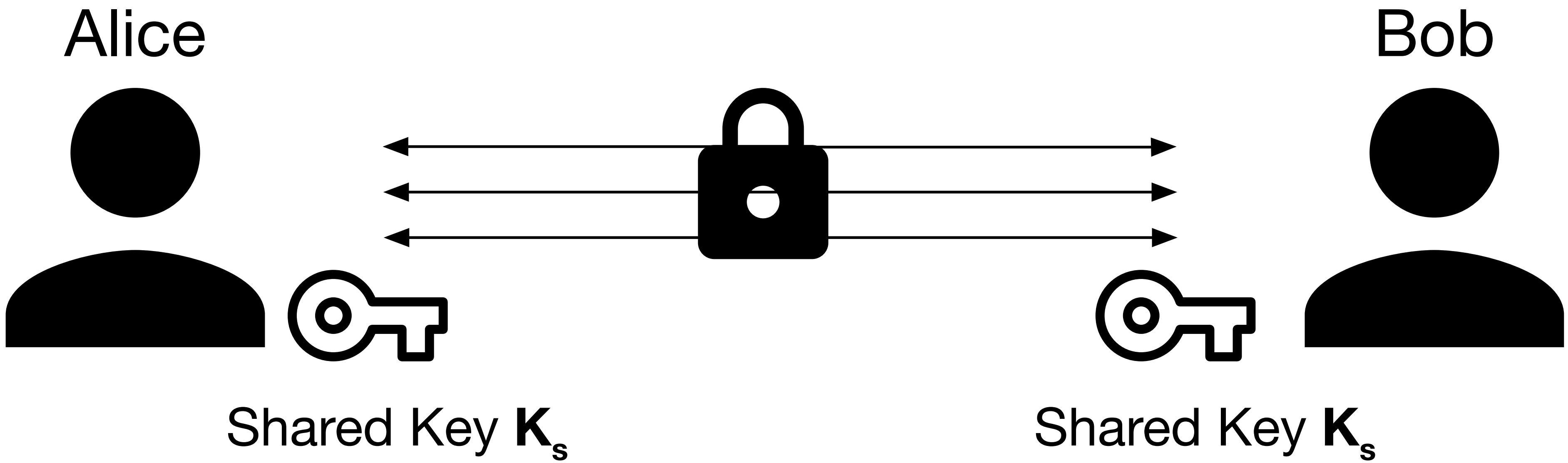
- MACs are One way functions that takes a key and a message and creates a message digest
  - Integrity
  - Authenticity
- The digest is usually appended at the end of the message so that the receiver can verify it
- HMAC turns hash functions into MACs and widely used today

Public-Key Cryptosystems
a.k.a
Asymmetric Cryptosystems

History of Public-Key Cryptosystems

- Before the mid 1970s all cipher systems were symmetric key algorithms.
- Symmetric keys are still widely used today
- known to 2-3 magnitudes faster than asymmetric (a.k.a public-key) algorithms.
- Why was public-key cryptosystems were such a breakthrough?

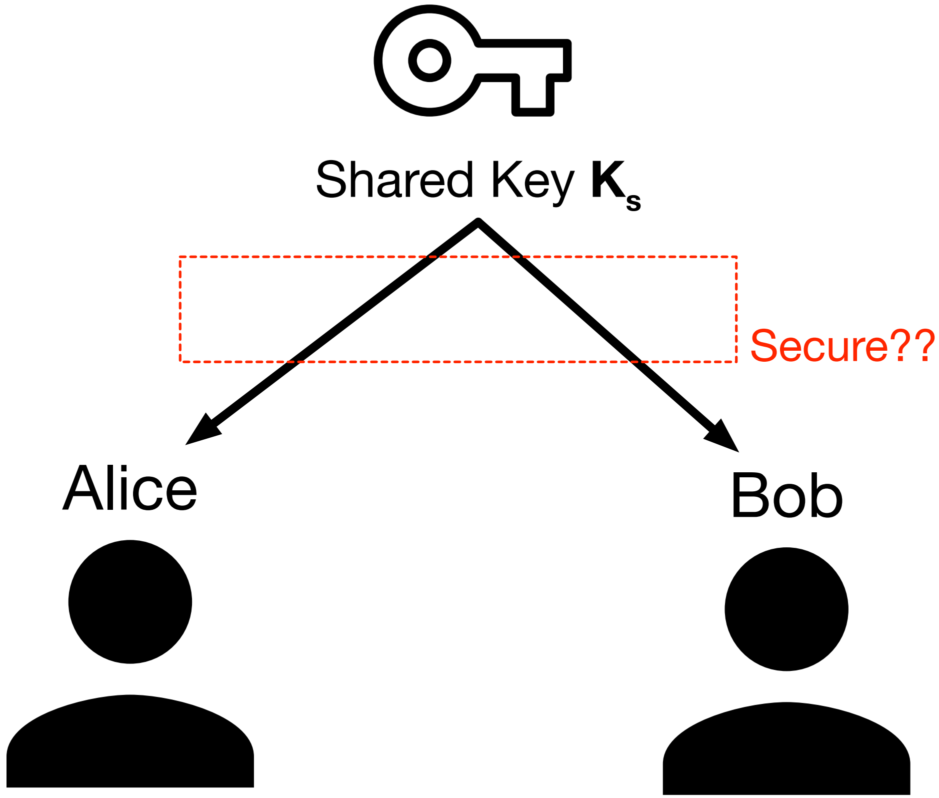
The Key Exchange Problem



- What is the problem here?
- Look at the title of the slide ☺

The Key Exchange Problem

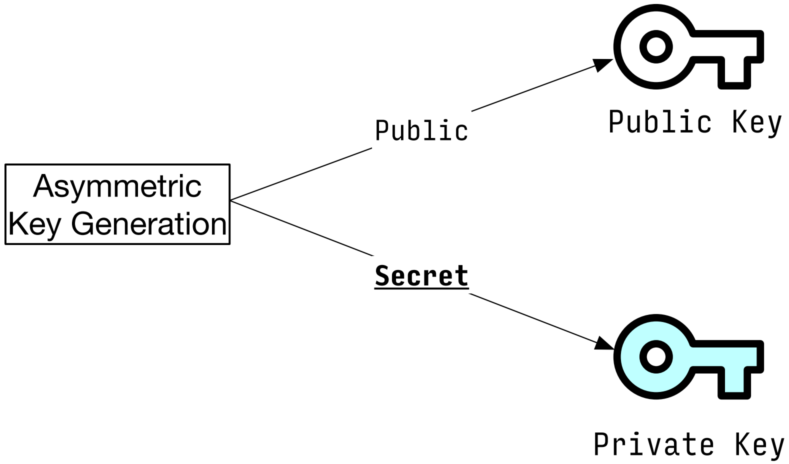
- Both Alice and Bob must be given the shared symmetric key K_s
- A secure channel is necessary for key distribution
- What if we have n participants and need to distribute n keys ?
  - (Key Distribution Problem)



History of Public-Key Cryptosystems

- Whitfield Diffie and Martin Hellman from Stanford published the asymmetric cryptosystem in 1976
  - Which is known today as *Diffie-Hellman Key Exchange*
- Ron Rivest, Adi Shamir, and Leonard Adleman from MIT published their Public-Key Cryptosystem in 1978
  - Which is known today as the RSA algorithm
- Clifford Cocks from GCHQ (British Intelligence Agency) concurrently implemented a form of PKC in 1973
  - Which was very similar to RSA

Public-Key (Asymmetric) Cryptosystem



In Public-Key Cryptosystems such as RSA, key generation gives you

- Public Key
  - Used for encrypting data
  - Not a secret
- Private Key
  - Used for decrypting data
  - A secret

Creating RSA Private/Public Key Pair

- From Github Help Page:

Generating a new SSH key

1. Open Terminalthe terminal.
2. Paste the text below, substituting in your GitHub Enterprise email address.

```
$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

This creates a new ssh key, using the provided email as a label.

```
> Generating public/private rsa key pair.
```

3. When you're prompted to "Enter a file in which to save the key," press Enter. This accepts the default file location.

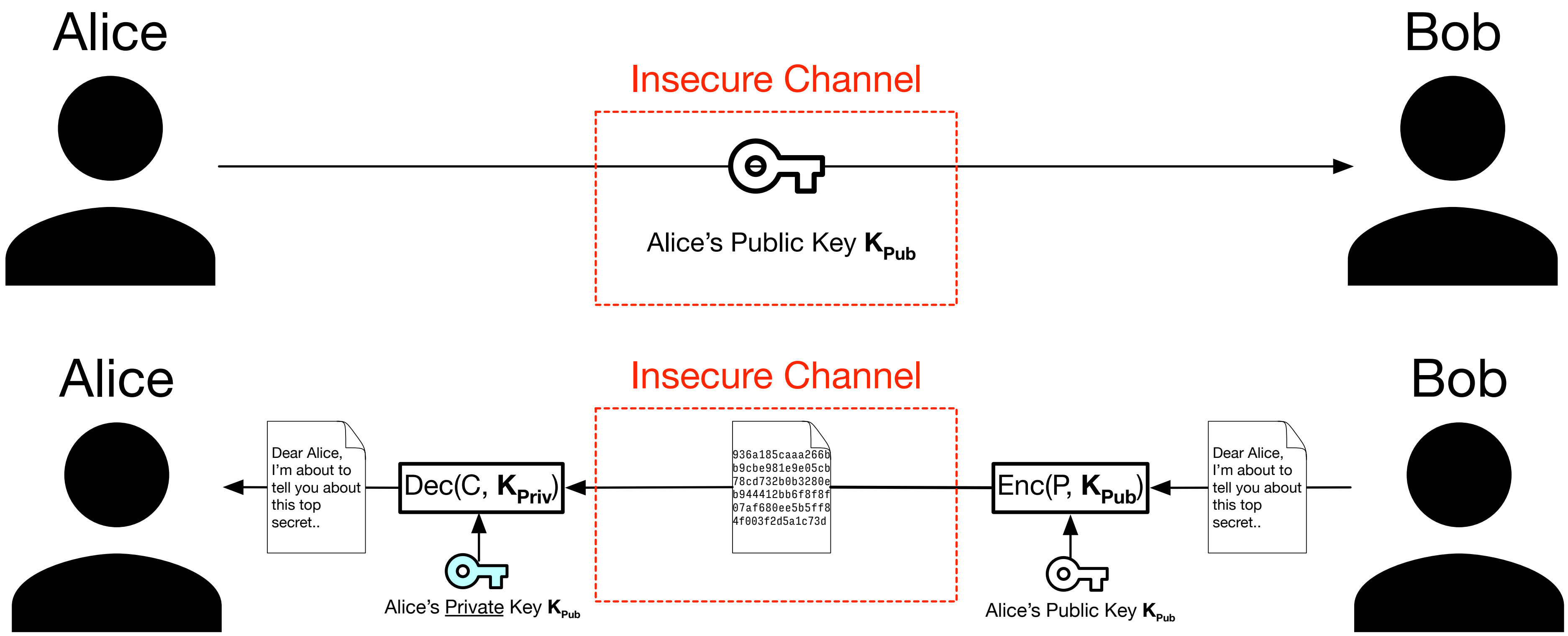
```
> Enter a file in which to save the key (/Users/you/.ssh/id_rsa): [Press en
```

```
> Enter a file in which to save the key (/home/you/.ssh/id_rsa): [Press ent
```

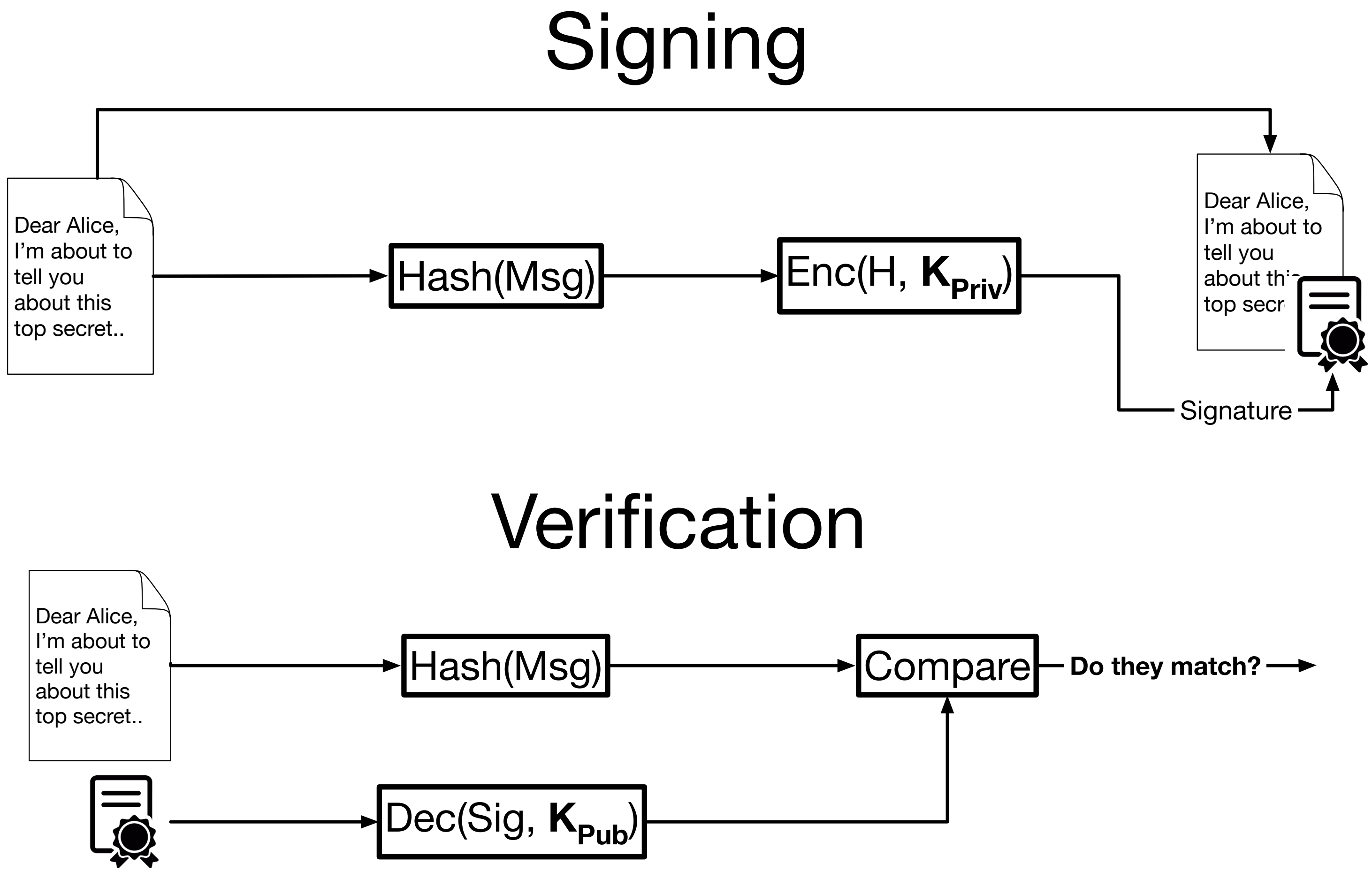
Public-Key Applications

- Encryption/Decryption (Confidentiality)
- Digital Signatures (Authentication)
- Key Exchange

Encryption and Decryption



Digital Signatures

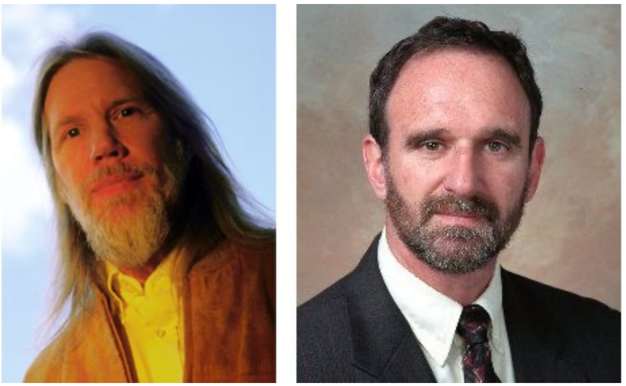


Diffie-Hellman Key Exchange

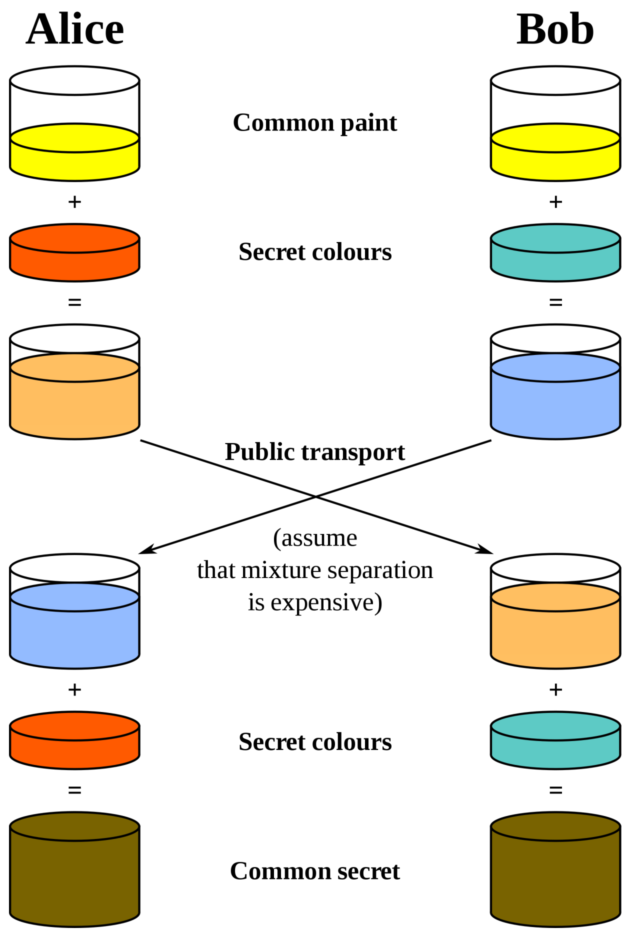
- Problem: PKEY systems are much slower than Symmetric-key systems
- Diffie-Hellman uses the Public-key cryptosystem concept to implement symmetric key exchange
- This is often referred to as *hybrid cryptosystem*:
  - Key Encapsulation using public-key system
  - Data Encapsulation using symmetric key

Diffie-Hellman Key Exchange

- Diffie-Hellman is a public-key based key exchange algorithm
- DH enables two parties to "create a shared key together" only exposing public key components of the cryptographic calculation



Whitfield Diffie and Martin Hellman



Diffie-Hellman Key Exchange

1. Alice and Bob publicly agree to use a modulus p = 23 and base g = 5 (which is a primitive root modulo 23).
2. Alice chooses a secret integer S_Alice = 4, then sends Bob A = g^{S_Alice} mod p
   - A = 5^4 mod 23 = 4
3. Bob chooses a secret integer S_Bob = 3, then sends Alice B = g^{S_Bob} mod p
   - B = 5^3 mod 23 = 10
4. Alice computes S_{Shared} = B^{S_Alice} mod p
   - S_{shared} = 10^4 mod 23 = 18
5. Bob computes Key_{shared} = A^{S_Bob} mod p
   - S_{shared} = 4^3 mod 23 = 18
6. Alice and Bob now share a secret (the number 18).

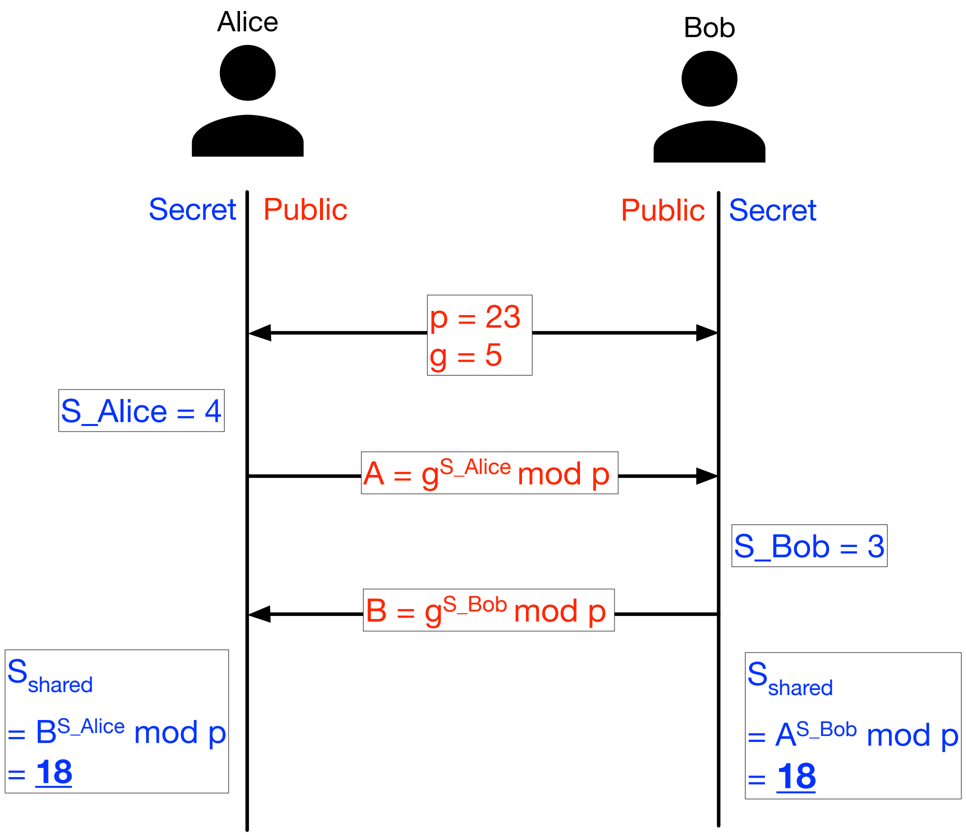
Both Alice and Bob have arrived at the same values because under mod p,

$$A^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p = B^a \bmod p$$

$$(g^a \bmod p)^b \bmod p = (g^b \bmod p)^a \bmod p$$

Diffie-Hellman Key Exchange

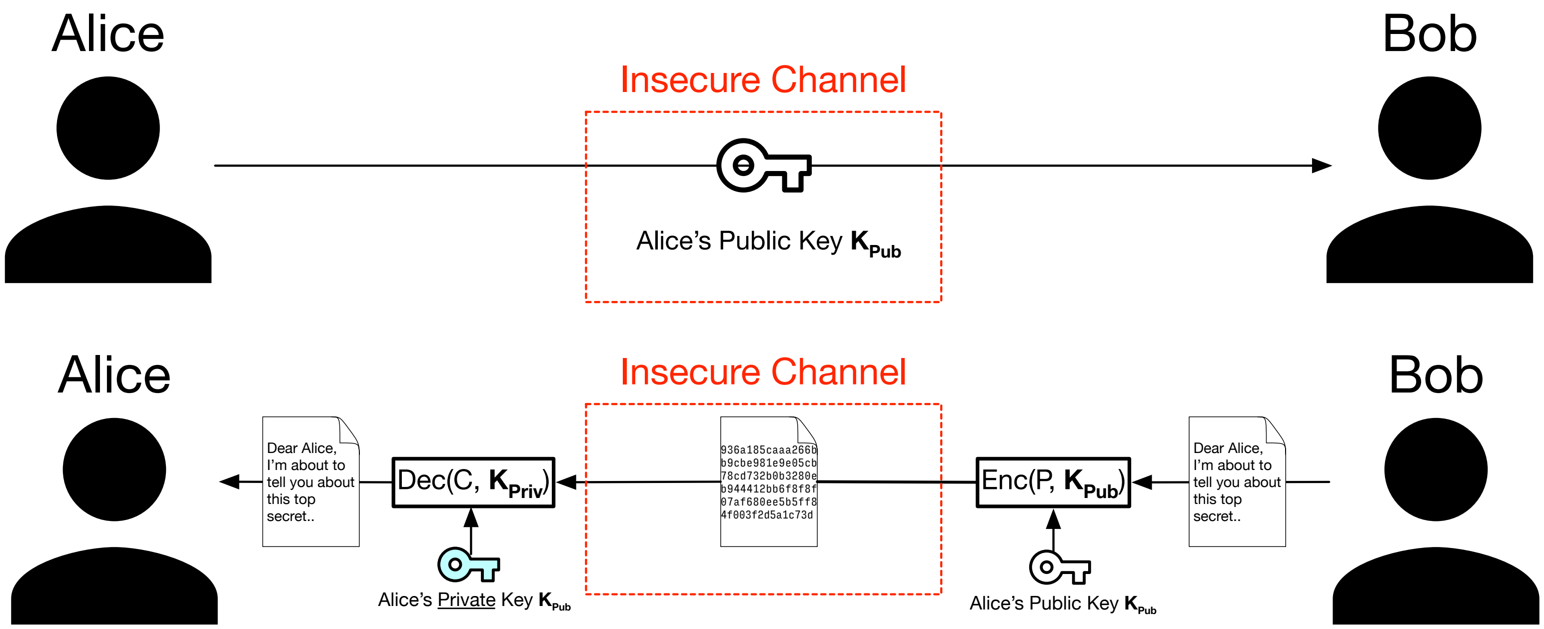
Terminal Here



Authenticated Diffie-Hellman Key Exchange

- Diffie-Hellman solves the problem of key exchange, but is it safe against man-in-the-middle attacks?
- e.g., you can create a shared key only shared with Bob, but how do you know you're actually talking to Bob?
- How do we authenticate each other in DH?
- Hint: Digital Signatures
- We will come back to this when we get to TLS

Forward Secrecy and Diffie-Hellman



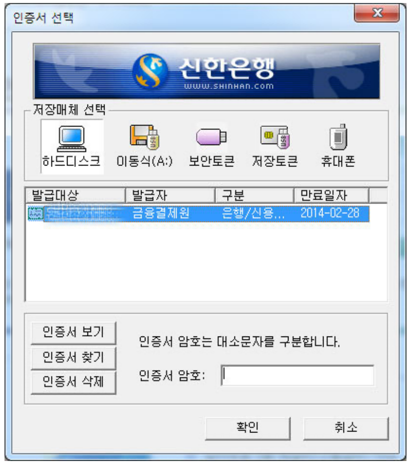
- RSA PKEY System can also be used to achieve secure key exchange?
- Yes. It has also been used along with DH
- But web browsers and web servers are by default prefer DH over RSA. why?

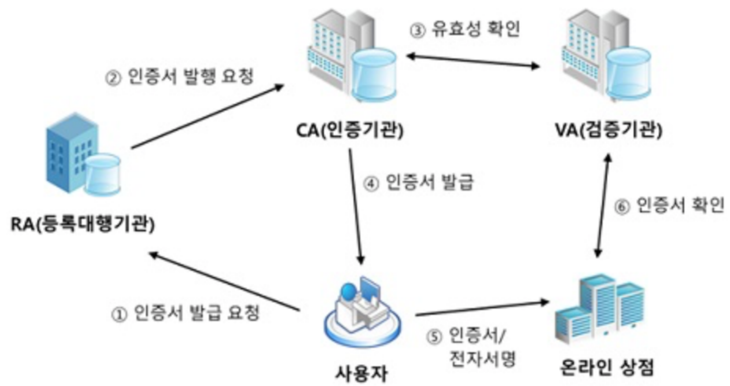
Forward Secrecy and Diffie-Hellman

- Forward Secrecy
  - Feature of key exchange protocols that give assurances that all future session is not compromised even when server's private key is leaked
- With RSA, private key exposure means all previous communication can be decrypted
- Solution: generate private keys (e.g., S_Alice and S_Bob) for each connections and discard them (Diffie-Hellman Ephemeral)
- <u>Generating Priv/Pub key pair is much faster with DHE than RSA and this is why DHE has been selected as the default KE algorithm in TLS 1.3</u>

Public Key Infrastructure (PKI)

- An infrastructure involving roles, policies, hardware, software, and procedures for digital certificates
- (Korea) Government-issued certificates that can be used for proving your identity





In Case You Haven't Noticed

Systems Security Lab

Research People Publications Courses SSLab-CTF

News

*** Open Positions at Systems Security Lab (SSLab) ***

시스템 보안 연구실 (SSLab)은 열정 있는 학생들을 기다리고 있습니다. 저희 연구실은 뭔가를 만들고, 고장내고, 고치는 걸 좋아하는 자신도 모르게 해커 기질을 가진 여러분들을 찾고 있습니다. SSLab은 소프트웨어 및 시스템 보안 분야 연구를 중점적으로 연구하지만, 학생 개인의 연구 분야 선택에 최대한 자유를 보장하고 있기도 합니다. 또한, 체계적인 신입생 교육 프로그램을 통해 자신의 연구 주제를 만들어 갈 수 있도록 지원합니다. SSLab의 철학은 구성원 개개인이 연구를 하는 가장 큰 동기부여가 재미가 되도록 하는 것입니다. 현재 저희 연구실은 아래와 같은 포지션을 모집하고 있습니다:

We are looking for passionate students who would be interested in doing research at SSLab. The most important qualification that I want from you is that you inherently enjoy hacking and fixing stuff. While SSLab focuses on software and systems security, the most important criteria in choosing a student research topic is that you have fun exploring the topic. So, do not hesitate to talk to us if you are interested:) Currently, I have the following positions open:

- Undergraduate Research Internship (학부연구생)
- Masters Students
- Doctoral Students
- Post-Doctoral Researchers

That's it for Today

- We learned
  - Symmetric-key cryptosystem
  - Public-key (asymmetric-key) cryptosystem
  - HMAC
- What we will try to finish next time
  - TLS in a nutshell
- Coming up: CTF challenge on TLS