Network Security

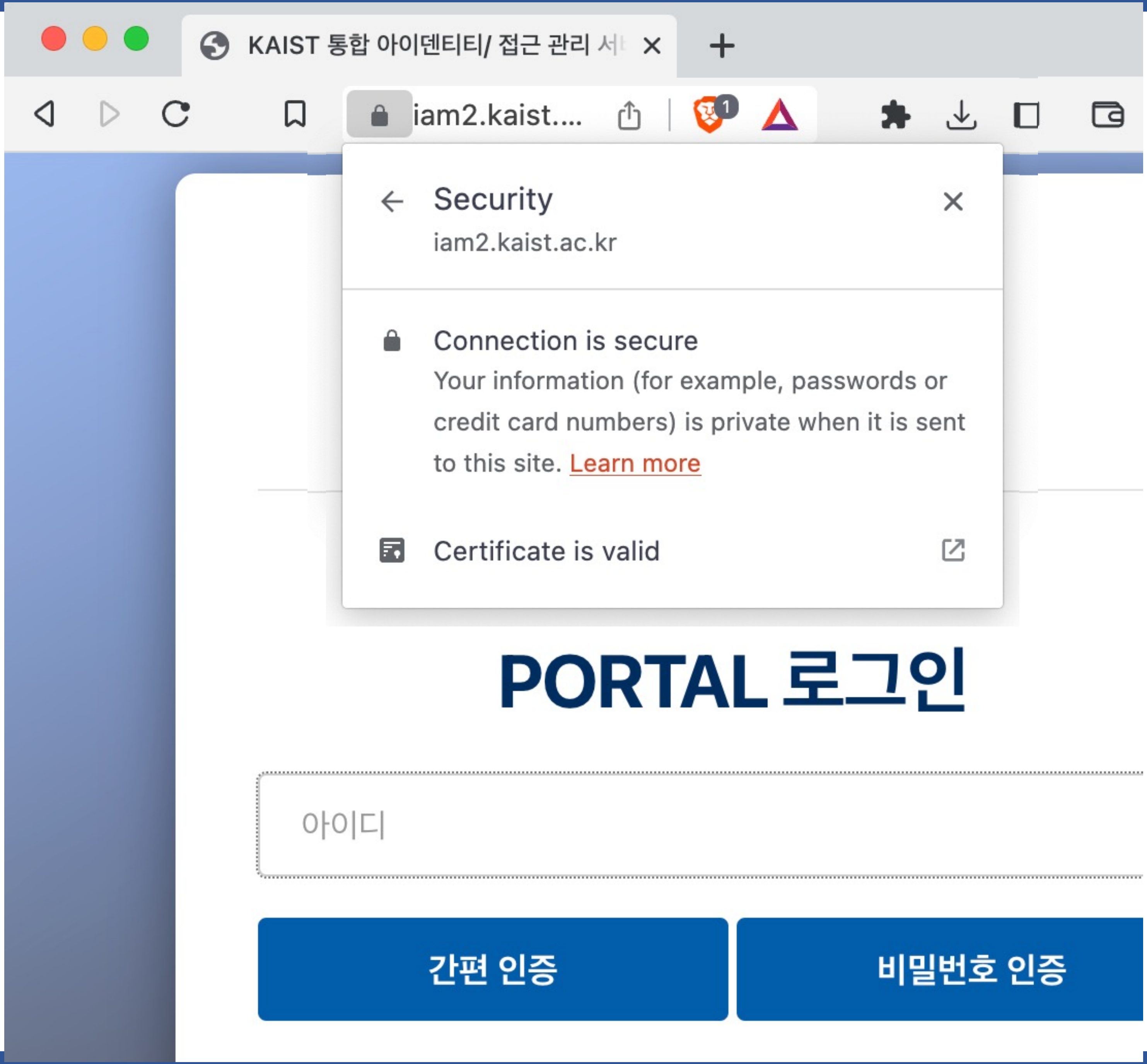
Dec 5, 2024

Min Suk Kang
Associate Professor
School of Computing/Graduate School of Information Security

KAIST

Chapter 8 outline

- What is network security?
- Principles of cryptography
- Message integrity, authentication
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS


KAIST 통합 아이덴티티/ 접근 관리 서
iam2.kaist....
Security
iam2.kaist.ac.kr
Connection is secure
Your information (for example, passwords or credit card numbers) is private when it is sent to this site. Learn more
Certificate is valid
PORTAL 로그인
아이디
간편 인증
비밀번호 인증

Security: overview

Chapter goals:

- understand principles of network security:
  - cryptography and its *many* uses beyond "confidentiality"
  - authentication
  - message integrity
- security in practice:
  - firewalls and intrusion detection systems
  - security in application, transport, network, link layers

What is network security?

기밀성

confidentiality: only sender, intended receiver should "understand" message contents

전송자와 의도한 수신자만 이해 가능 (암호화)

- sender encrypts message
- receiver decrypts message

authentication: sender, receiver want to confirm identity of each other → identity 확인으로 정체 인증

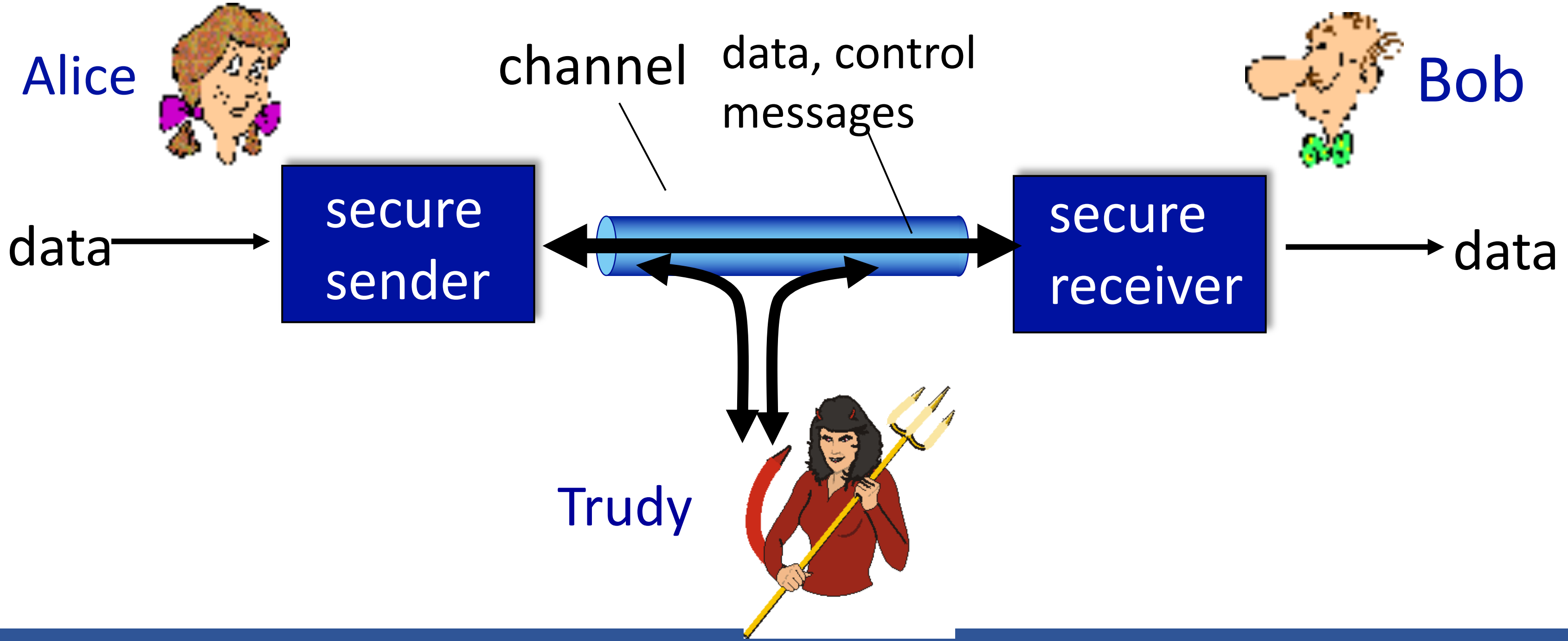
무결성

message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection → 메세지 조작 방지

access and availability: services must be accessible and available to users → 사용 편리

Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages



Friends and enemies: Alice, Bob, Trudy

Who might Bob and Alice be?

- … well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- BGP routers exchanging routing table updates
- other examples?

There are bad guys (and girls) out there!

Q: What can a "bad guy" do?

A: A lot! (recall section 1.6)

도청

- eavesdrop: intercept messages
- actively insert messages into connection

사칭

- impersonation: can fake (spoof) source address in packet (or any field in packet)
- hijacking: "take over" ongoing connection by removing sender or receiver, inserting himself in place

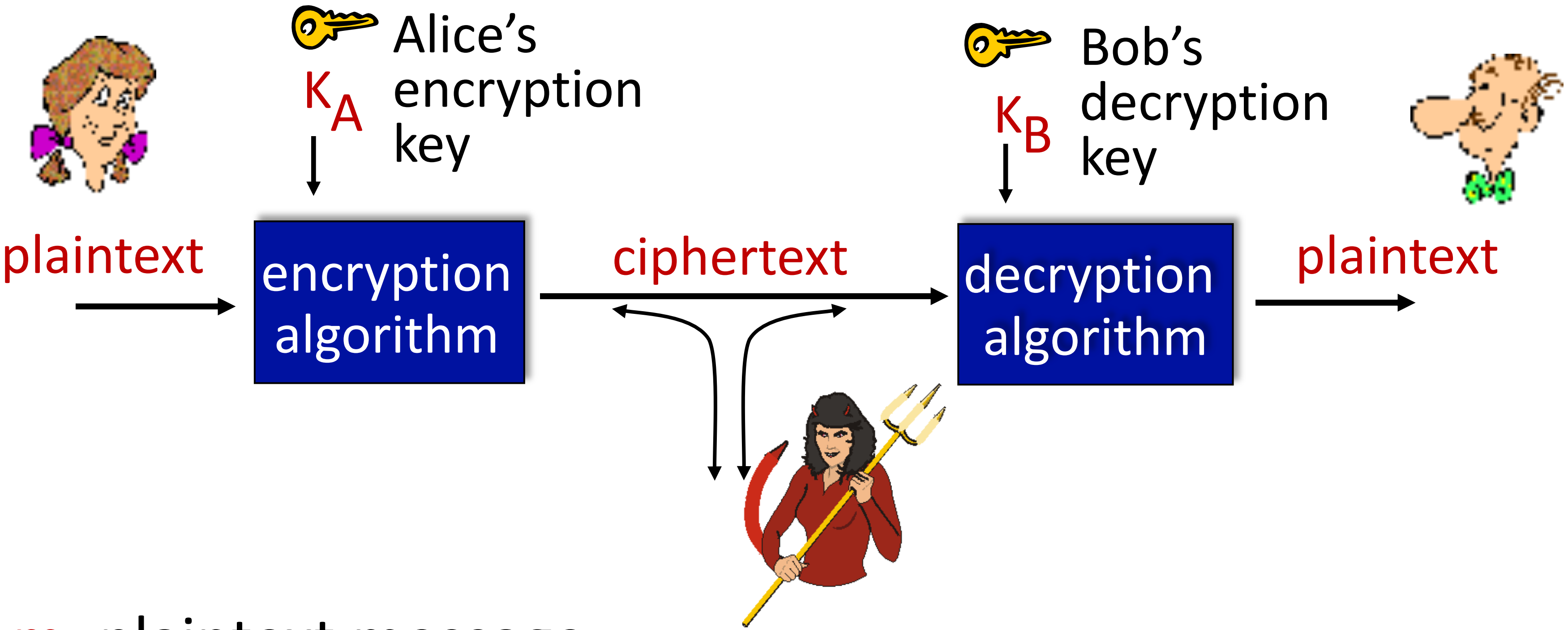
DoS

- denial of service: prevent service from being used by others (e.g., by overloading resources)

Chapter 8 outline

- What is network security?
- Principles of cryptography
- Message integrity, authentication
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

The language of cryptography



m: plaintext message

$K_A(m)$: ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

Breaking an encryption scheme

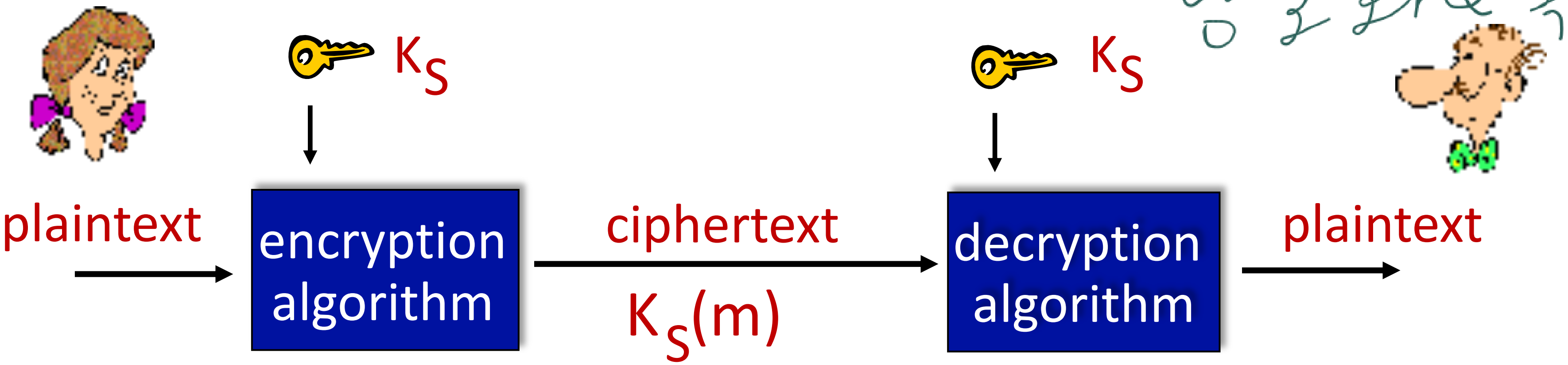
- **cipher-text only attack:** Trudy has ciphertext she can analyze → 암호문에만 접근 가능
- **two approaches:**
  - brute force: search through all keys → 무차별 대입
  - statistical analysis

알고 있는 plaintext와 암호문을 비교해 암호화 알고리즘 분석

- **known-plaintext attack:** Trudy has plaintext corresponding to ciphertext
  - *e.g.*, in monoalphabetic → 알파벳 치환 cipher, Trudy determines pairings for a,l,i,c,e,b,o,
- **chosen-plaintext attack:** Trudy can get ciphertext for chosen plaintext

원하는 평문에 대한 암호문 획득 가능 ⇒ data↑

Symmetric key cryptography → 같은 키로 암호화 & 복호화



symmetric key crypto: Bob and Alice share same (symmetric) key: K

- *e.g.,* key is knowing substitution pattern in mono alphabetic substitution cipher

Q: how do Bob and Alice agree on key value?

Simple encryption scheme

→ 다른 문자로 치환하여 암호화

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

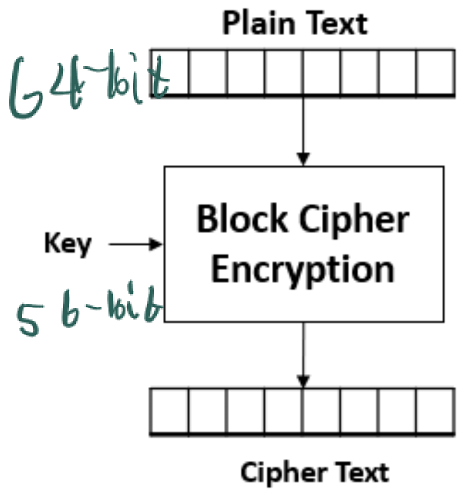
```
plaintext:   abcdefghijklmnopqrstuvwxyz

ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

e.g.:

```
Plaintext:  bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Encryption key: mapping from set of 26 letters to set of 26 letters

Symmetric key crypto: DES



DES: Data Encryption Standard

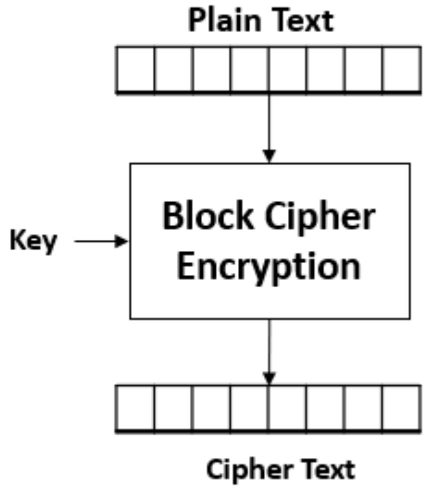
- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- block cipher with cipher block chaining
- how secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day → 무차별 대입으로 하루도 지나지 않아 해독
  - no known good analytic attack
- making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys

AES: Advanced Encryption Standard

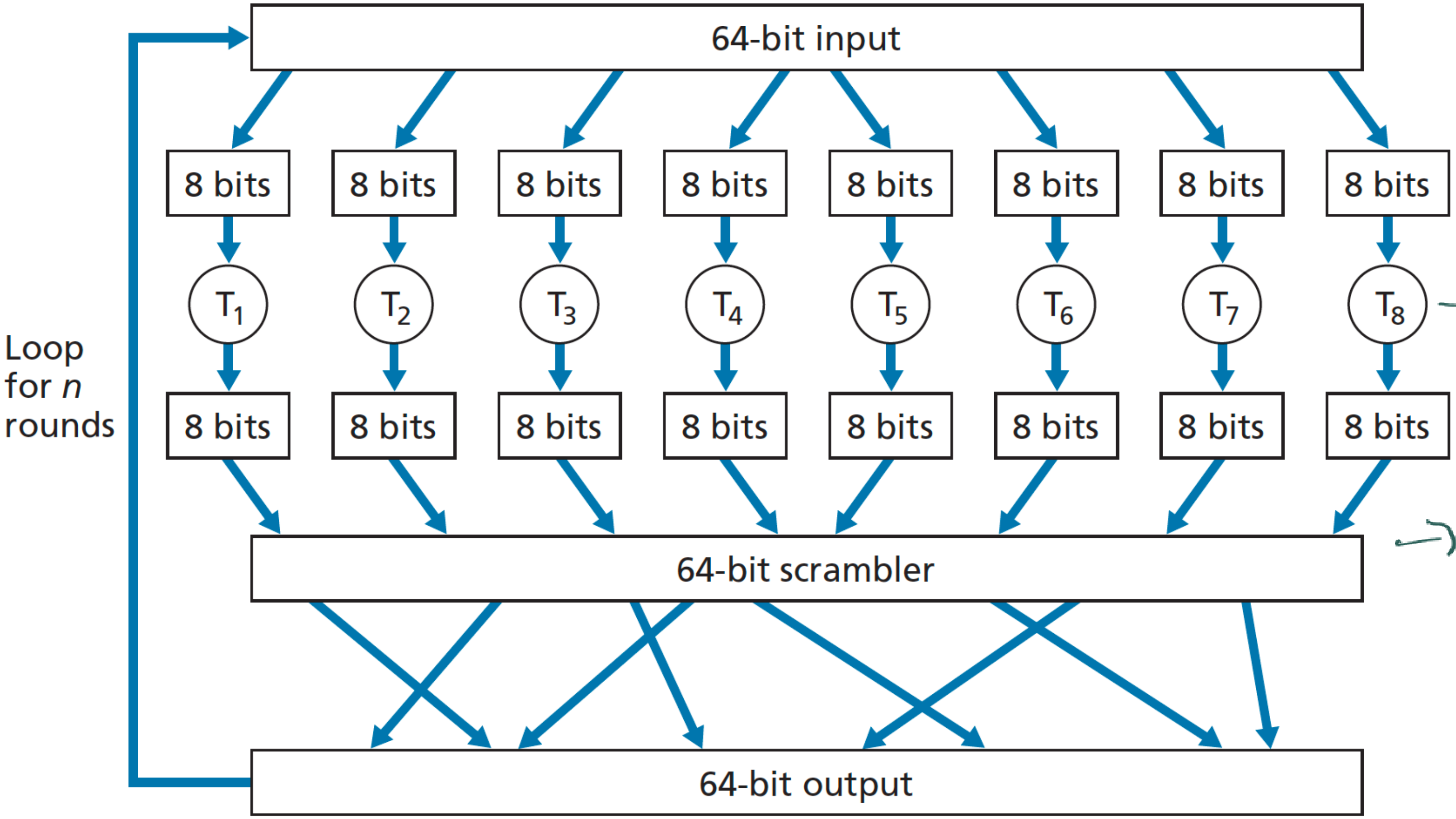
- symmetric-key NIST standard, replaced DES (Nov 2001)
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys > bit↑
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

↳ 365·24·60·60sec

bit 수 증가로 경우의 수를 늘려 brute force 방지



Block Ciphers

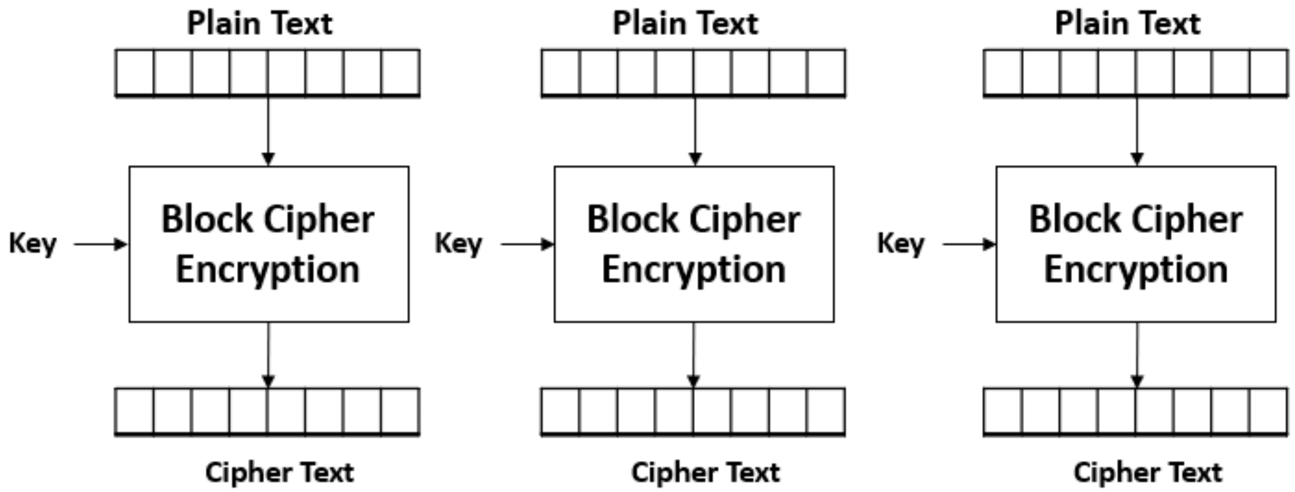


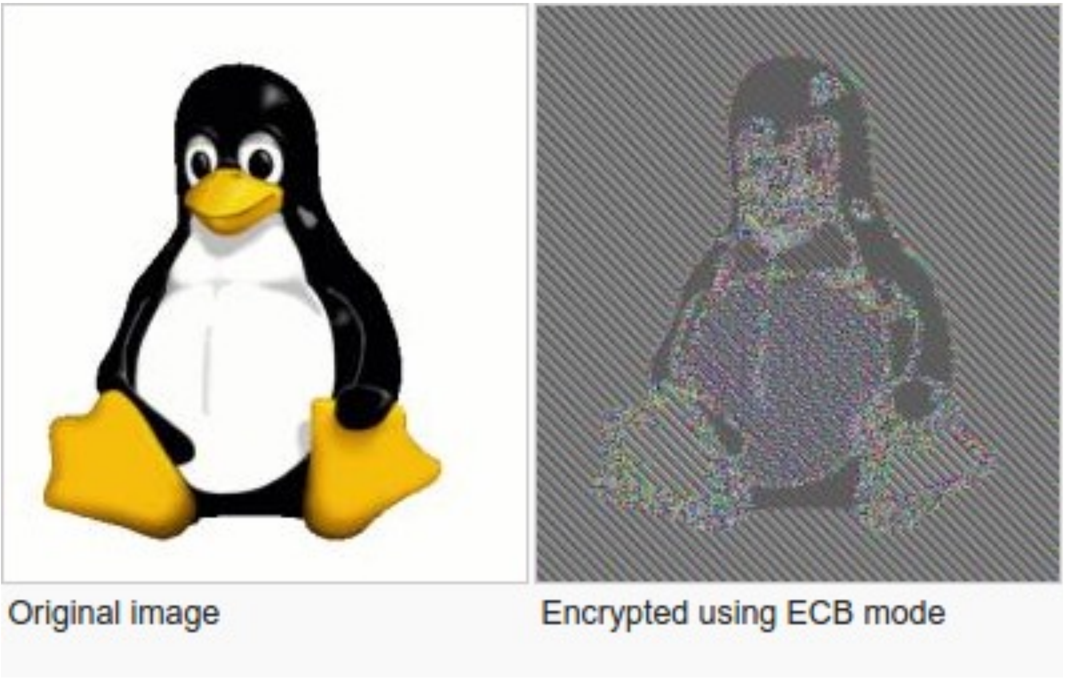
→ chunk 분리

→ 암호화

→ chunks 순서 섞기

Using Block Ciphers for Encryption



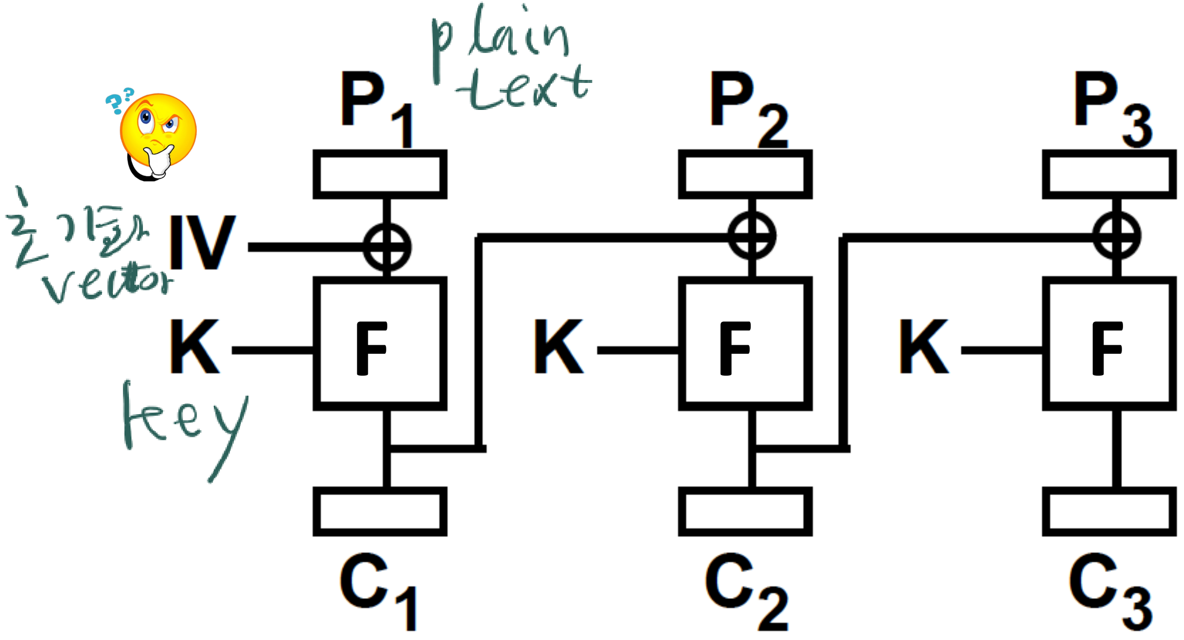


We should use a “secure” ***mode of encryption*** when utilizing block ciphers

Cipher Block Chaining (CBC)

- $C_j = F_K(P_j \oplus C_{j-1})$
- $P_j = F^{-1}_K(C_j) \oplus C_{j-1}$
- C_0 = IV called

  initialization vector



- Idea 1: chaining ciphertext with plaintext
- Idea 2: additional input IV to make encryption probabilistic

↳ 암호화에 random성 부여

What happens if we ignore IV?

- What about IV == 0?
  - This happens frequently (by software engineers with no security education)
  - Not a ***probabilistic*** encryption!

↳ 확률에 의한 변조 X

- In principle, a random IV is chosen each time

매번 random한 IV 선택

Public Key Cryptography

symmetric key crypto:

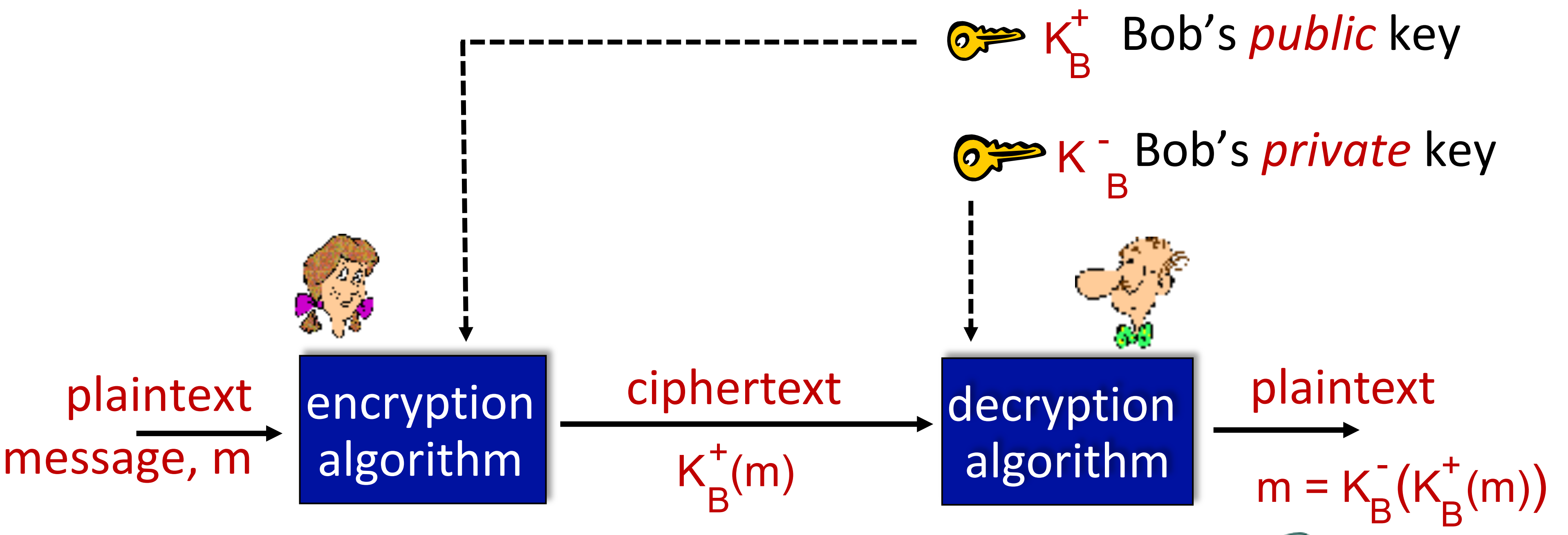
- requires sender, receiver know shared secret key 같은 키 공유
- Q: how to agree on key in first place (particularly if never "met")? (암호화/복호화키 동일)

ㄴ) 처음에 공유할 방법은?

public key crypto

- *radically* different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all* → 공개 키는 모두가 안다
- *private* decryption key known only to receiver → 비밀키는 수신자만

Public Key Cryptography



Wow - public key cryptography revolutionized 2000-year-old (previously only symmetric key) cryptography!

- similar ideas emerged at roughly same time, independently in US and UK (classified)

Public key encryption algorithms

requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adelson algorithm

소인수분해 사용

Notion of public-key cryptography

- Diffie and Hellman introduced the notion of *public-key crypto*
  - a party generates a ***pair*** of keys (***K***$^{+}$, ***K***$^{-}$):
    - ***K***$^{+}$: public key. widely disseminated
    - ***K***$^{-}$: private key. kept secretly

private-key crypto

Alice
key

Bob
key

- ***key*** can be used for **encryption/decryption** or **MAC/verification**
- *symmetric* key or *private* key
- assumption: **only** two parties shares ***key***

public-key crypto

Alice
K_A^-

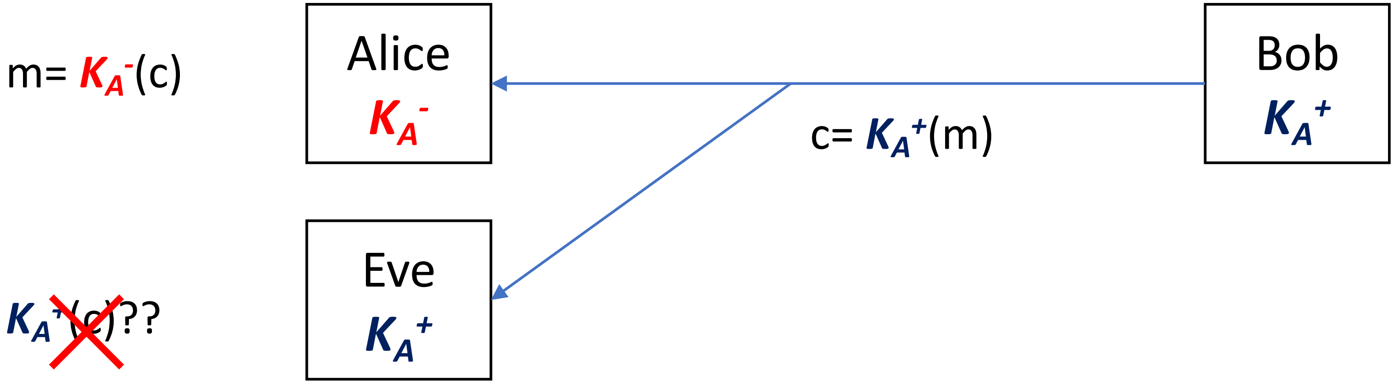
K_A^+

Bob
K_A^+

K_A^+

- K_A^+ for **encryption** and K_A^- for **decryption** or K_A^- for **signature** and K_A^+ for **verification**
- *asymmetric* key or *public* key
- assumption: K_A^+ is disseminated via **authenticated channel**

Public-key encryption



K_A^+ 를 가진 누구나 암호화 가능

- anyone with K_A^+ can encrypt message
  - assume that intended parties receive K_A^+ via *authenticated channels* (what if there's no such channel? We'll learn PKI)
- only the one with K_A^- can decrypt message

→ K_A^- 는 수신자만 지님, K_A^- 로만 복호화 가능

  - cannot decrypt with K_A^+ (Eve may encrypt but can't decrypt)
- Receiver (Alice) generates (K_A^+, K_A^-) and sends K_A^+ to Bob
  - or publicize her K_A^+ (e.g., her webpage, central database)

RSA: Creating public/private key pair

1. choose two large prime numbers *p, q.* (e.g., 1024 bits each)

매우 큰 소수 매우 큰 소수 → $\phi(n)$

2. compute $n = pq$, $z = (p-1)(q-1)$

3. choose e (with $e<n$) that has no common factors with z (e, z are "relatively prime"). → 공통인 소인수 X

4. choose d such that $ed-1$ is exactly divisible by z. (in other words: $ed \bmod z = 1$).

$e \cdot d = z \cdot k + 1$

5. *public* key is $\underbrace{(n,e)}_{K_B^+}$. *private* key is $\underbrace{(n,d)}_{K_B^-}$.

RSA: encryption, decryption

0. given (n,e) and (n,d) as computed above

1. to encrypt message m $(<n)$, compute
$$c = m^e \bmod n$$

2. to decrypt received bit pattern, c, compute
$$m = c^d \bmod n$$

magic happens! $m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$

RSA example:

Bob chooses *p=5, q=7*. Then *n=35, z=24*.

e=5 (so *e, z* relatively prime).

d=29 (so *ed-1* exactly divisible by z).

encrypting 8-bit messages.

encrypt:	bit pattern	m	m^e	$c = m^e \bmod n$
	0000l000	12	24832	17
decrypt:	c	c^d		$m = c^d \bmod n$
	17	481968572106750915091411825223071697		12

Why does RSA work?

- must show that $c^d \bmod n = m$, where $c = m^e \bmod n$
- fact: for any x and y: $x^y \bmod n = x^{(y \bmod z)} \bmod n$
  - where $n = pq$ and $z = (p-1)(q-1)$

오일러 정리

- thus,

$$
\begin{aligned}
c^d \bmod n &= (m^e \bmod n)^d \bmod n \\
&= m^{ed} \bmod n \\
&= m^{(ed \bmod z)} \bmod n \\
&= m^1 \bmod n \\
&= m
\end{aligned}
$$

n이 충분히 클때

RSA: another important property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

result is the same!

Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$?

follows directly from modular arithmetic:

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$
$$= m^{de} \bmod n$$
$$= (m^d \bmod n)^e \bmod n$$

Why is RSA secure?

- suppose you know Bob's public key (n,e). How hard is it to determine d?
- essentially need to find factors of n without knowing the two factors p and q
  - fact: factoring a big number is hard

소인수분해는 소수의 크기가 커질수록
계산이 어려워진다

RSA in practice: session keys

- exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

session key, K_S

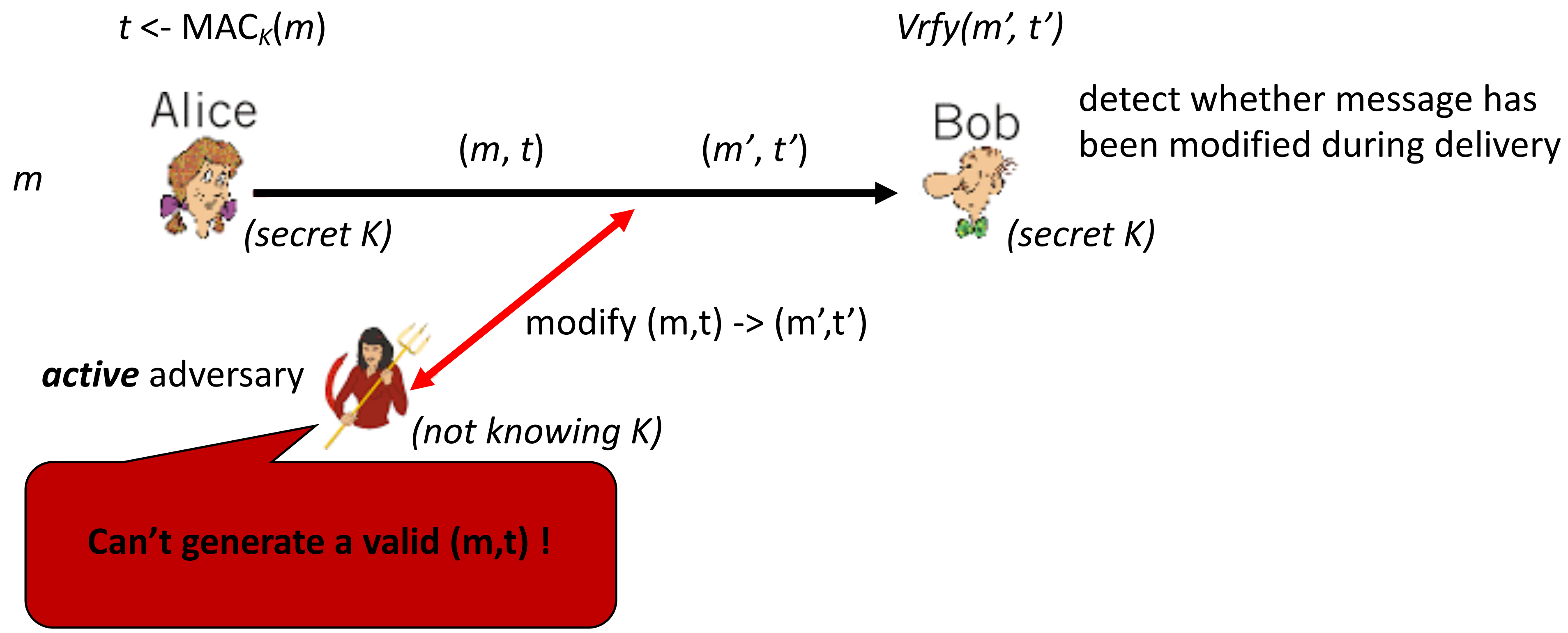
- Bob and Alice use RSA to exchange a symmetric session key K_S
- once both have K_S, they use symmetric key cryptography

Chapter 8 outline

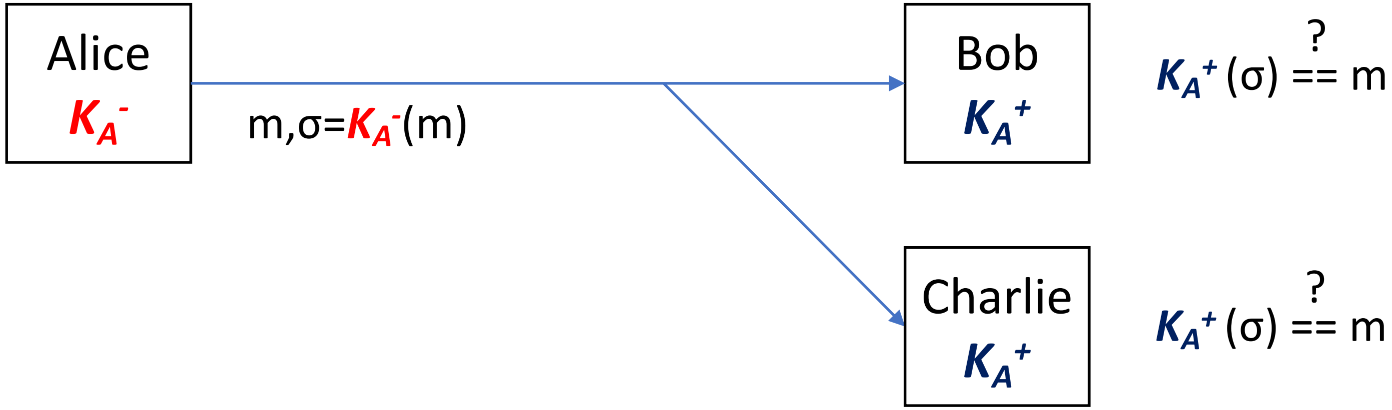
- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

Message integrity (in symmetric key setting)

Message Authentication Code



Digital signature

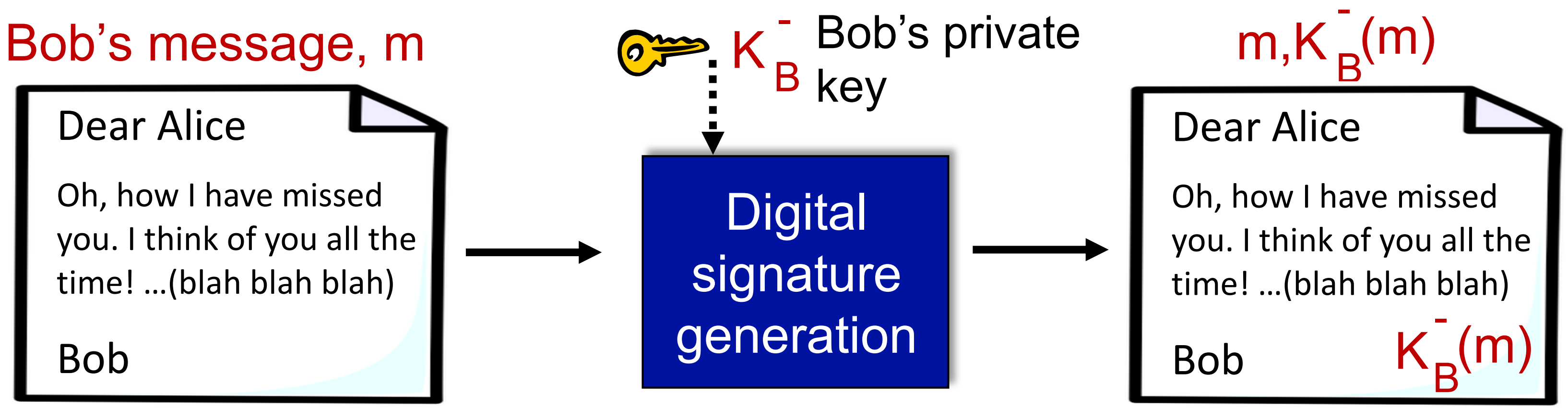


- only the one with K_A^- can generates signature σ for message m
- anyone with K_A^+ can verify the signature
  - assume that intended parties receive K_A^+ via *authenticated channels*
- **non-repudiation**: signed document becomes proof that Alice indeed signed the document

Digital signatures

cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document: he is document owner/creator.
- *verifiable, nonforgeable:* recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
- simple digital signature for message m:
  - Bob signs m with his private key K_B, creating "signed" message, $K_B^-(m)$



Digital signatures

- suppose Alice receives msg m, with signature: m, $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key

Alice thus verifies that:

- Bob signed m
- no one else signed m
- Bob signed m and not m'

non-repudiation:

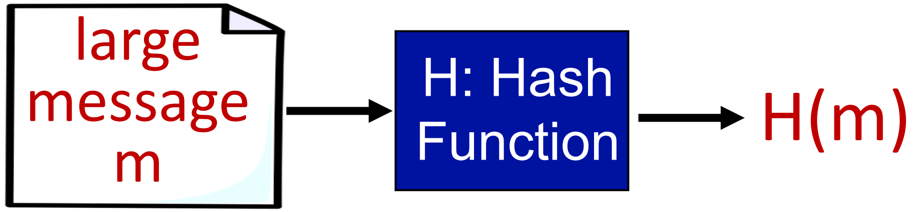
✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m

Message digests

computationally expensive to public-key-encrypt long messages

goal: fixed-length, easy- to-compute digital “fingerprint”

- apply hash function H to *m*, get fixed size message digest, *H(m)*



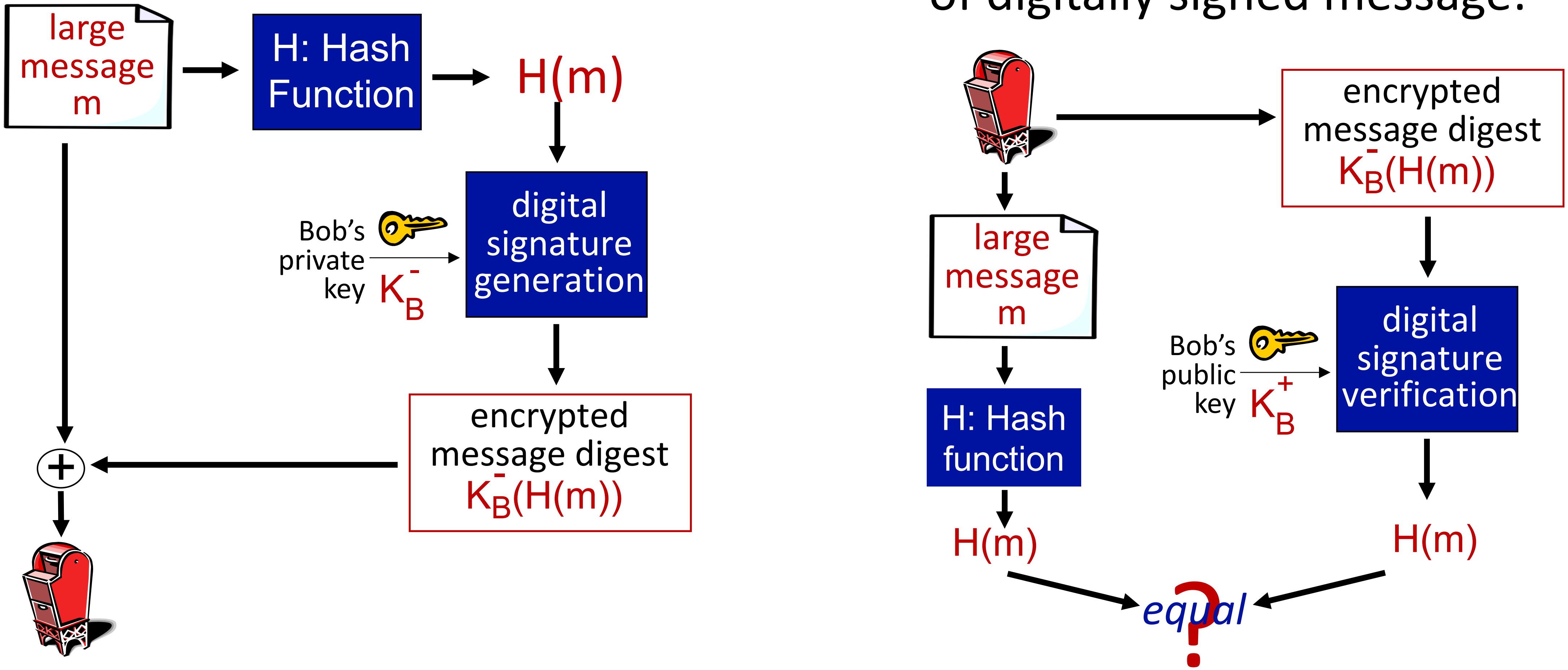
“Cryptographic” (or secure) hash function properties:

- produces fixed-size msg digest (fingerprint)
- given message *x*, computationally infeasible to find *x’=/=x* such that *H(x)=H(x’)*
- given message digest *y*, computationally infeasible to find *x* such that *y = H(x)*

Digital signature = signed message digest

Bob sends digitally signed message:

Alice verifies signature, integrity of digitally signed message:



Hash function algorithms

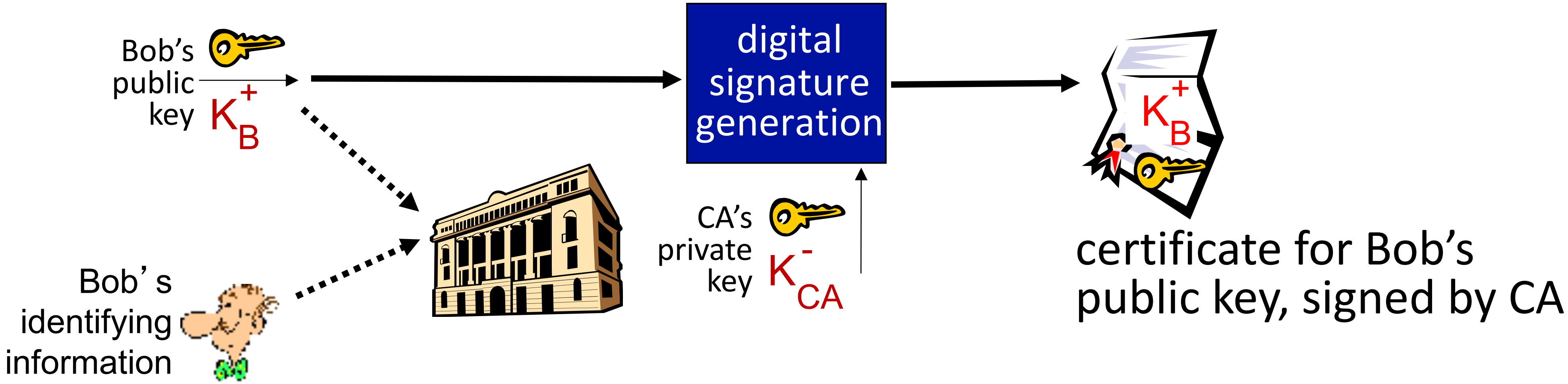
- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string x, appears difficult to construct msg m whose MD5 hash is equal to x
- But now SHA-1, SHA-2, SHA-3 are used in newer systems
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit, 256-bit, … message digest

Need for certified public keys

- motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:
    Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it's Bob's public key
  - Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
  - Bob doesn't even like pepperoni

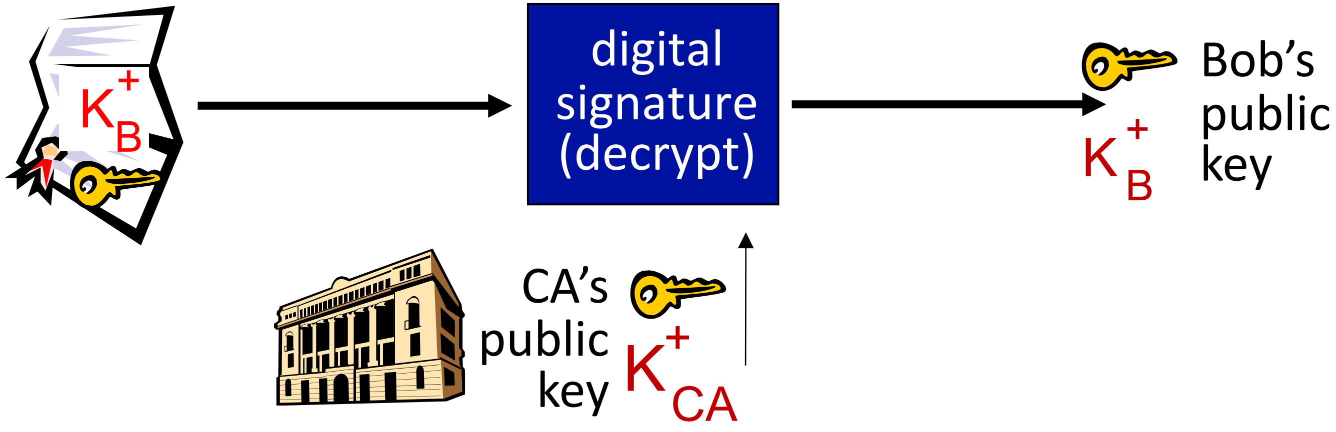
Public key Certification Authorities (CA)

- certification authority (CA): binds public key to particular entity, E
- entity (person, website, router) registers its public key with CE provides “proof of identity” to CA
  - CA creates certificate binding identity E to E’s public key
  - certificate containing E’s public key digitally signed by CA: CA says “this is E’s public key”



Public key Certification Authorities (CA)

- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere)
  - apply CA's public key to Bob's certificate, get Bob's public key



Chapter 8 outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- **Securing TCP connections: TLS**
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

Transport-layer security (TLS)

- widely deployed security protocol above the transport layer
  - supported by almost all browsers, web servers: https (port 443)
- provides:
  - confidentiality: via *symmetric encryption*
  - integrity: via *cryptographic hashing*
  - authentication: via *public key cryptography*

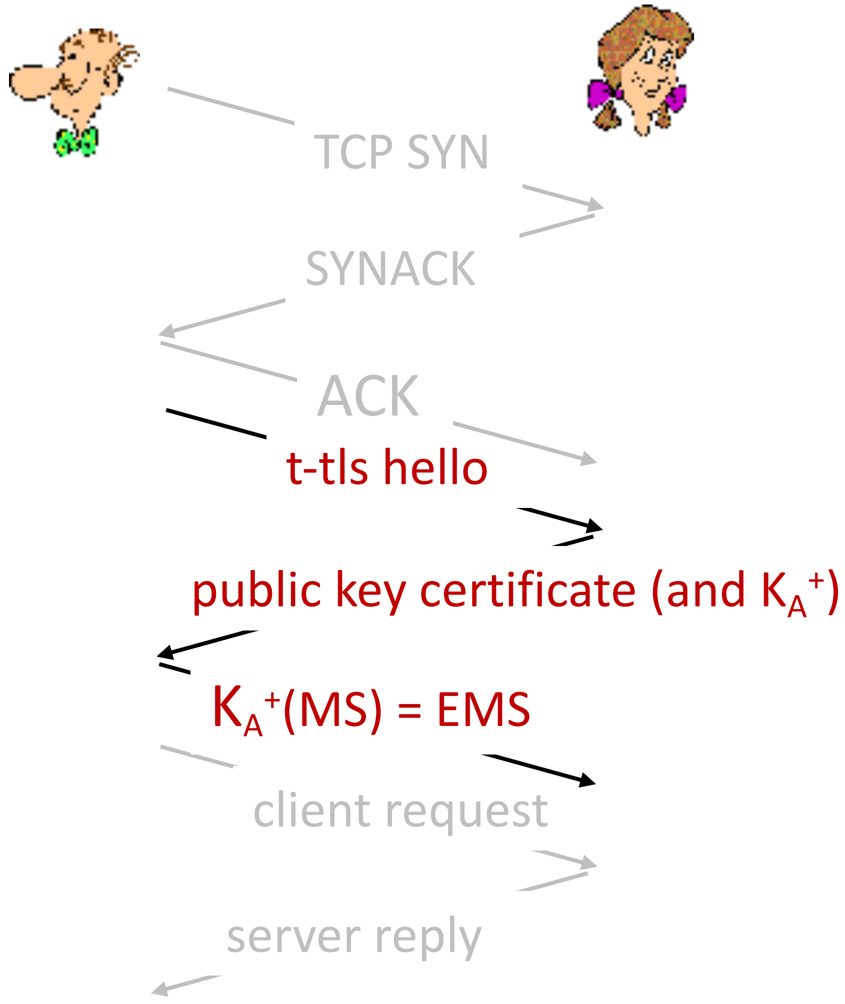
all techniques we have studied!

- history:
  - early research, implementation: secure network programming, secure sockets
  - secure socket layer (SSL) deprecated [2015]
  - TLS 1.3: RFC 8846 [2018]

Transport-layer security: what's needed?

- let's *build* a toy TLS protocol, *t-tls,* to see what's needed!
- we've seen the "pieces" already:
  - handshake: Alice, Bob use their certificates, private keys to authenticate each other, exchange or create shared secret
  - key derivation: Alice, Bob use shared secret to derive set of keys
  - data transfer: stream data transfer: data as a series of records
    - not just one-time transactions
  - connection closure: special messages to securely close connection

t-tls: initial handshake



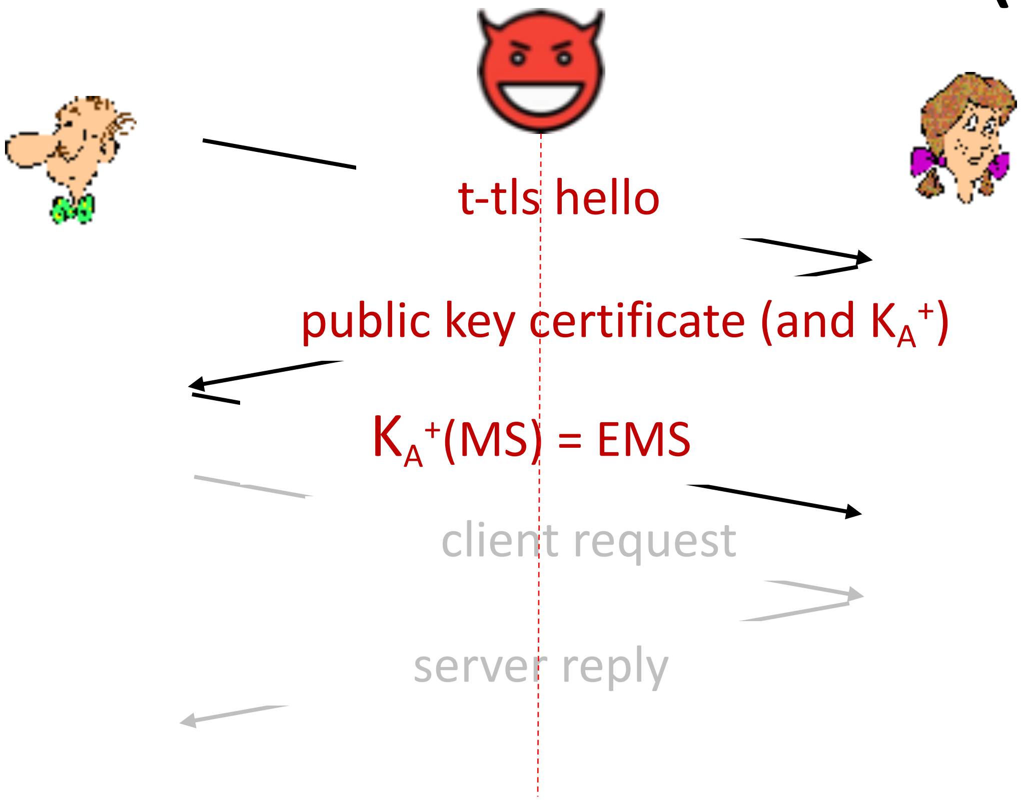
t-tls handshake phase:

- Bob establishes TCP connection with Alice
- Bob verifies that Alice is really Alice
- Bob sends Alice a master secret key (MS), used to generate all other keys for TLS session
- potential issues:
  - 3 RTT before client can start receiving data (including TCP handshake)

t-tls: cryptographic keys

- considered bad to use same key for more than one cryptographic function
  - different keys for message authentication code (MAC) and encryption
- four keys:
  - K_c : encryption key for data sent from client to server
  - M_c : MAC key for data sent from client to server
  - K_s : encryption key for data sent from server to client
  - M_s : MAC key for data sent from server to client
- keys derived from key derivation function (KDF)
  - takes master secret and (possibly) some additional random data to create new keys

Man-in-the-Middle (MITM) Attacks



- Can Mallory obtain encryption and authentication keys?

t-tls: encrypting data

- recall: TCP provides data *byte stream* abstraction
- Q: can we encrypt data in-stream as written into TCP socket?
  - *A:* where would MAC go? If at end, no message integrity until all data received and connection closed!
  - *solution:* break stream in series of "records"
    - each client-to-server record carries a MAC, created using M_c
    - receiver can act on each record as it arrives
- t-tls record encrypted using symmetric key, $K_{c,}$ passed to TCP:

$K_c($ | *length* | *data* | *MAC* | $)$

t-tls: encrypting data (more)

- possible attacks on data stream?
  - *re-ordering:* man-in middle intercepts TCP segments and reorders (manipulating sequence #s in unencrypted TCP header)
  - *replay*
- solutions:
  - use TLS sequence numbers (data, TLS-seq-# incorporated into MAC)
  - use nonce

t-tls: connection close

- truncation attack:
  - attacker forges TCP connection close segment
  - one or both sides thinks there is less data than there actually is
- solution: record types, with one type for closure
  - type 0 for data; type 1 for close
- MAC now computed using data, type, sequence #

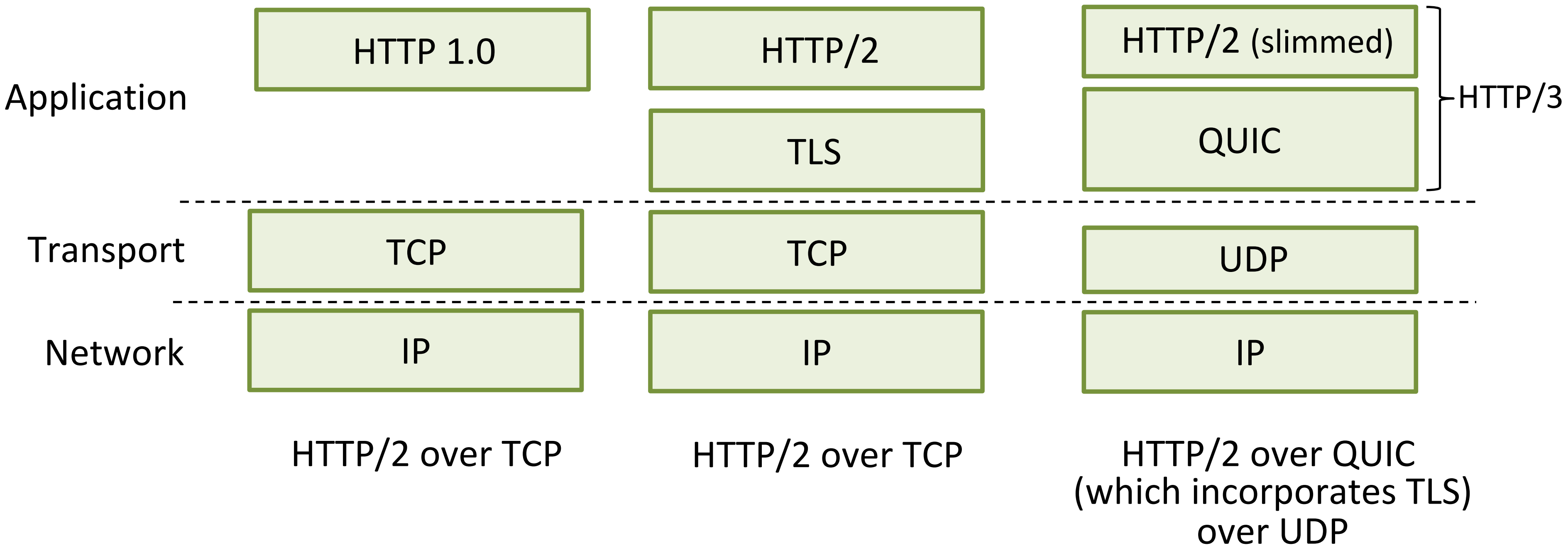
K_c(

length	*type*	*data*	*MAC*

)

Transport-layer security (TLS)

- TLS provides an API that *any* application can use
- an HTTP view of TLS:



Next…

- *Chapter 8.6 Securing TCP Connections: TLS*
- *Chapter 8.7 Network-Layer Security: IPsec and Virtual Private Networks*