Lecture 5: Network Security in Practice

-COMP 6712 Advanced Security and Privacy

Haiyang Xue

haiyang.xue@polyu.edu.hk

2023/2/14

Network Security in Practice

- Elliptic curve-based encryption and signature
- Recall AKE, PKI, and CA
- SSL/TLS
- HTTPS
- Last about 1 hour for tutorial

Elliptic Curve Group

- Let $p = 2 \cdot q + 1$, where p, q are primes
- $\boldsymbol{Z}_p^*$ has a subgroup $G = \langle g \rangle$ of order q

- Ex. $\boldsymbol{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ has a subgroup $\langle 3 \rangle = \langle 4 \rangle = \langle 5 \rangle = \langle 9 \rangle = \{1, 3, 4, 5, 9\}$ of order 5

$G = \langle g \rangle$

$\xrightarrow{A = g^a}$

$\xleftarrow{B = g^b}$

$a \xleftarrow{\$} \{1, \dots, |G|\}$

$\mathrm{K} \leftarrow B^a = g^{ab}$

$b \xleftarrow{\$} \{1, \dots, |G|\}$

$\mathrm{K} \leftarrow A^b = g^{ab}$

- We can build Diffie-Hellman and ElGamal encryption based on the group $G = \langle g \rangle$

$\mathrm{ElGamal.Enc} : G \times G \to \quad G \times \mathcal{C}$

$\mathrm{ElGamal.Dec} : \boldsymbol{Z}_p \times G \times G \to \quad G$

$G = \langle g \rangle$

$\xleftarrow{B}$

$\xrightarrow{A, C}$

KeyGen
1. $sk = \ b \xleftarrow{\$} \{1, \dots, |G|\}$
2. $pk = \ B \leftarrow g^b$
3. **return** (sk, pk)

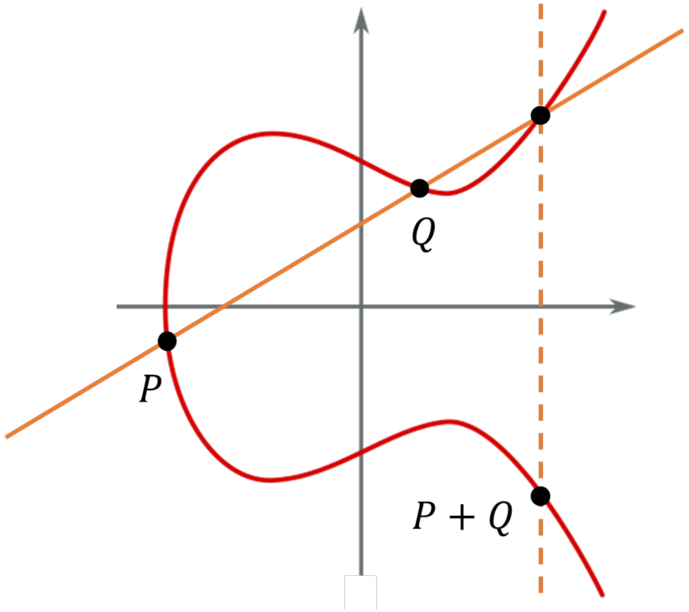
Enc(pk, M)
1. $a \xleftarrow{\$} \{1, \dots, |G|\}$
2. $A \leftarrow g^a$
3. $K \leftarrow B^a = g^{ab}$
4. $C \leftarrow K \cdot M$
5. **return** (A, C)

Dec(sk, C)
1. $Z \leftarrow A^b = g^{ab}$
2. $M \leftarrow C / K$
3. **return** M

Elliptic curves

$$y^2 = ax^3 + bx + c$$

a, b, c, x, y
$\in \boldsymbol{Z}_p$



- There is elliptic curve defined over $\boldsymbol{Z}_p$
- Such that the points on an elliptic curve (+ a infinite point) form a group of order $\sim p^2$
- Denoted by $(E(\boldsymbol{Z}_p\ ), +)$

Elliptic curve-based group

- Let $p = 2 \cdot q + 1$, where p, q are primes
- $\boldsymbol{Z}_p^*$ has a subgroup $G = \langle g \rangle$ of order q
- Ex. $\boldsymbol{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ has a subgroup $\langle 3 \rangle = \langle 4 \rangle = \langle 5 \rangle = \langle 9 \rangle = \{1, 3, 4, 5, 9\}$ of order 5

- elliptic curve defined over $\boldsymbol{Z}_p$
- $y^2 = x^3 + ax + b$, where $a, b, x, y \in \boldsymbol{Z}_p$
- Solutions of the form (x, y) can generates a group of order $q \sim p^2$
- Ex. Secp256k1, where $p = 2^{256} - 2^{32} - 977$,

$a = 0, b = 7$,

$q = 11579208923731619542357098500868790785283756427907490438260516314151816149433 7$

$G = \langle g \rangle, g \coloneqq (g_x, g_y)$

Elliptic curve-based DH and ElGamal Encryption

$G = \langle g \rangle$

$A = g^a \longrightarrow$

$\longleftarrow B = g^b$

$a \xleftarrow{\$} \{1, \dots, |G|\}$

$K \leftarrow B^a = g^{ab}$

$b \xleftarrow{\$} \{1, \dots, |G|\}$

$K \leftarrow A^b = g^{ab}$

$\text{ElGamal.Enc}: G \times G \to G \times C$

$\text{ElGamal.Dec}: \mathbb{Z}_p \times G \times G \to G$

$G = \langle g \rangle$

$\longleftarrow B$

$A, C \longrightarrow$

KeyGen
1. $sk = b \xleftarrow{\$} \{1, \dots, |G|\}$
2. $pk = B \leftarrow g^b$
3. **return** (sk, pk)

Enc(pk, M)
1. $a \xleftarrow{\$} \{1, \dots, |G|\}$
2. $A \leftarrow g^a$
3. $K \leftarrow B^a = g^{ab}$
4. $C \leftarrow K \cdot M$
5. **return** (A, C)

Dec(sk, C)
1. $Z \leftarrow A^b = g^{ab}$
2. $M \leftarrow C/K$
3. **return** M

We can also define the Diffie-Hellman and ElGamal encryption
The only difference is the underline group.

Understand the algorithm via code

Hash-then sign paradigm of RSA digital signature

$$\text{RSA.Sign: } \overbrace{\boldsymbol{Z}^{+}\times\boldsymbol{Z}^{*}_{\phi(n)}}^{\mathcal{SK}}\times\overbrace{\{0,1\}^{*}}^{\mathcal{M}} \rightarrow \overbrace{\boldsymbol{Z}^{*}_{n}}^{\mathcal{S}}$$

$$\text{RSA.Vrfy: } \underbrace{\boldsymbol{Z}^{+}\times\boldsymbol{Z}^{*}_{\phi(n)}}_{\mathcal{PK}}\times\underbrace{\{0,1\}^{*}}_{\mathcal{M}}\times\underbrace{\boldsymbol{Z}^{*}_{n}}_{\mathcal{S}} \rightarrow \{1,0\}$$

KeyGen

```
p, q <-$- two random prime numbers
n <- p · q
ϕ(n) = (p − 1)(q − 1)
choose e such that gcd(e, ϕ(n)) = 1
d <- e^{-1} mod ϕ(n)
sk <- (n, d)      vk <- (n, e)
return (sk, vk)
```

Vrfy$(vk = (n, e), M \in \boldsymbol{Z}^{*}_{n}, \sigma)$

```
if σ^e = H(M) mod n then
    return 1
else
    return 0
```

vk

M, σ

$H : \{0,1\}^{*} \rightarrow \boldsymbol{Z}^{*}_{n}$

Sign$(sk = (n, d), M \in \boldsymbol{Z}^{*}_{n})$

```
σ <- H(M)^d mod n
return σ
```

Elliptic curve-based Digital Signature (ECDSA)

Public parameters: $G =< P >$ with prime order q,
H is the hash function

Secret signing key: $d \leftarrow Z_q$

Public key: $Q = d \cdot P$

Signature Algorithm

- $R = k \cdot P$ where $k \leftarrow Z_q$
- $r = r_x$ where $R = (r_x, r_y)$
- $s = k^{-1}(H(m) + d \cdot r) \bmod q$
- Output (r, s)

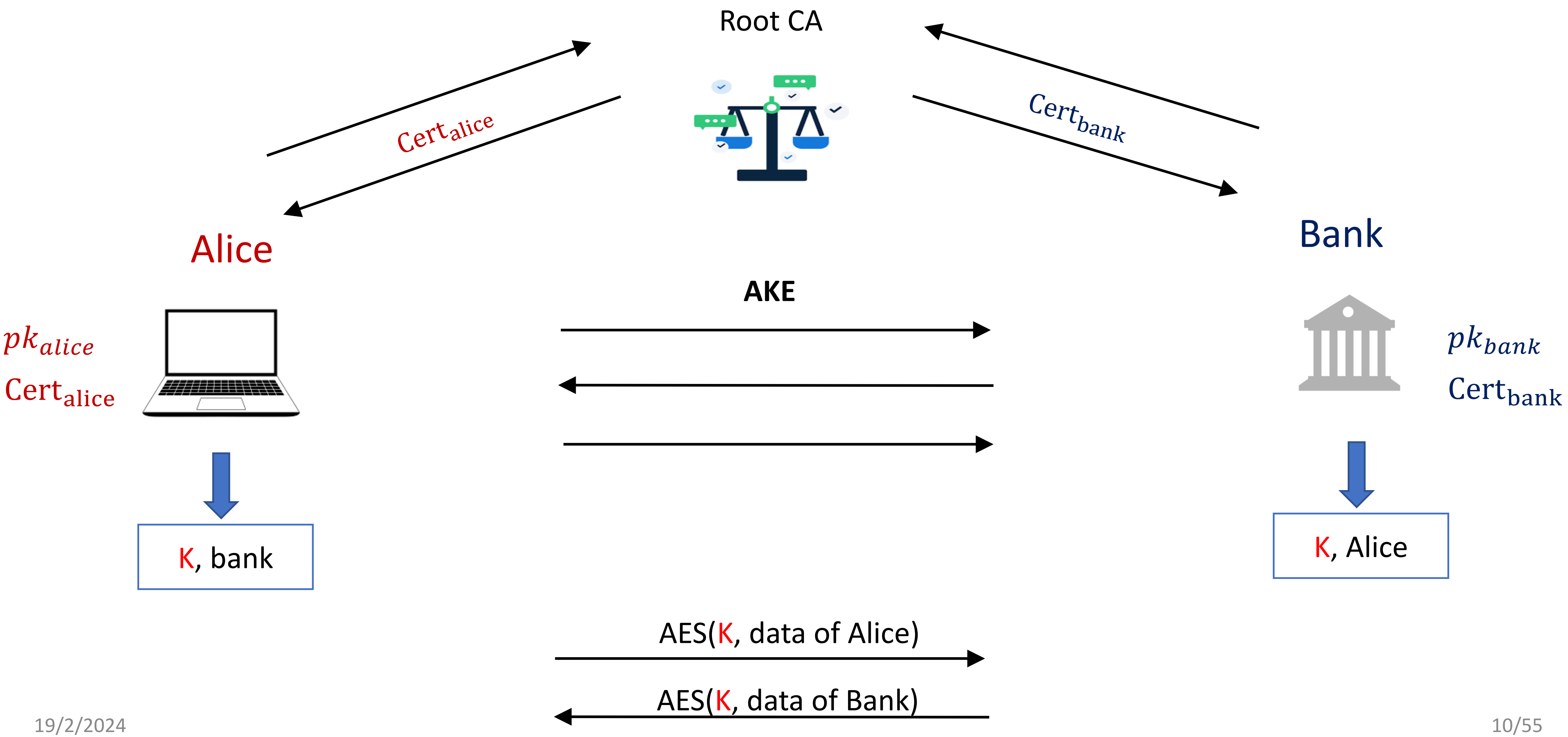
Verification Algorithm

- $s^{-1} = k \ \ (H(m) + x \cdot r)^{-1}$
- $R \coloneqq s^{-1}H(m) \cdot P + s^{-1}r \cdot Q$
- $r' = r_x$ where $R = (r_x, r_y)$
- Output $[\, r = r' \,]$

Network Security in Practice

- Elliptic curve-based encryption and signature
- Recall AKE, PKI, and CA
- SSL/TLS
- HTTPS
- Last about 0.5-1 hour for tutorial

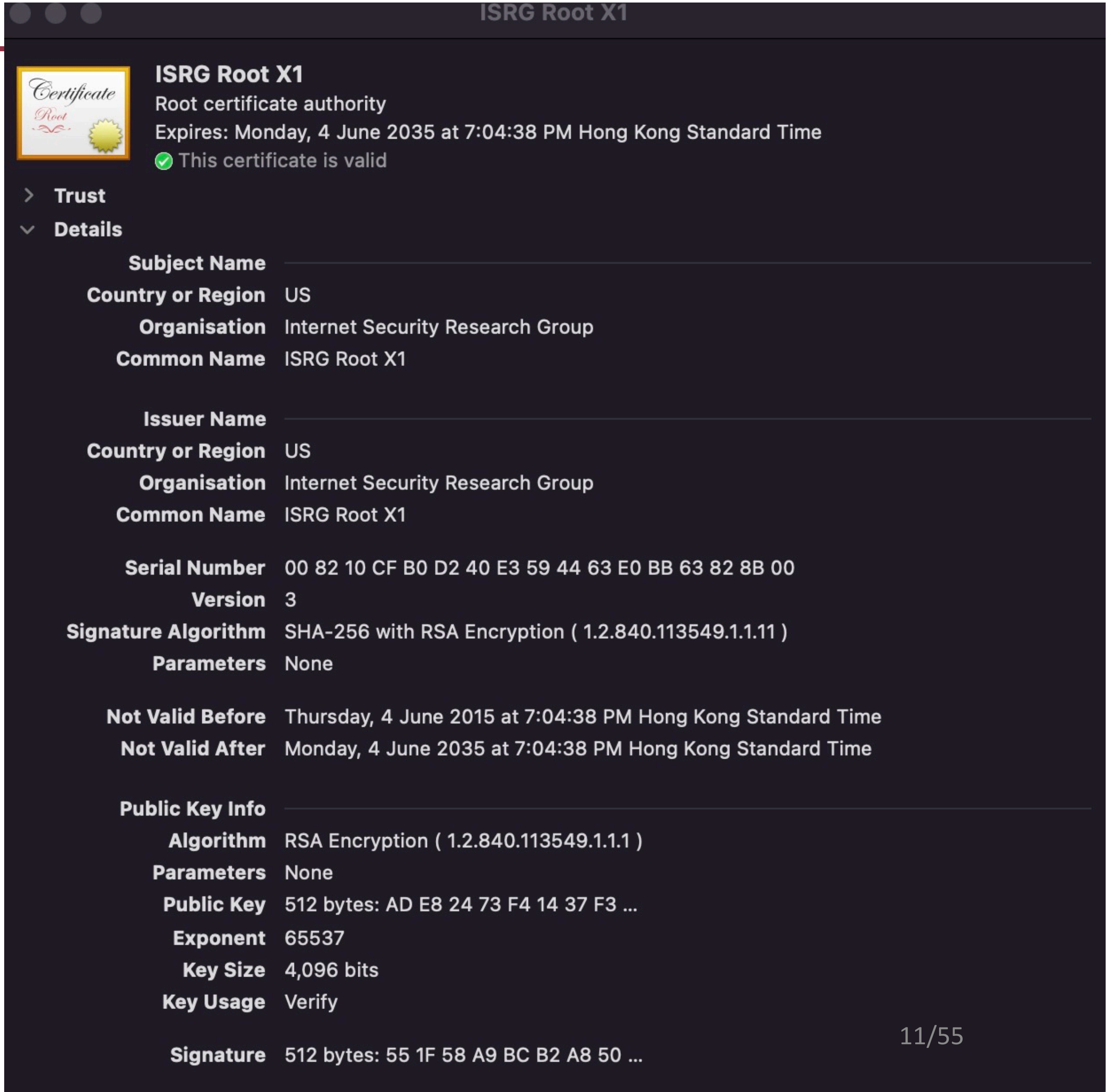
AKE-syntax



Root CA
Cert_{alice}
Cert_{bank}
Alice
Bank
AKE
pk_{alice}
Cert_{alice}
pk_{bank}
Cert_{bank}
K, bank
K, Alice
AES(K, data of Alice)
AES(K, data of Bank)

Certification Authorities

- Subject Name
  - Who's CA
- Issuer Name
  - Who gives this CA
  - Sign name
  - Valid
- PK information
  - pk
  - What is the pk is used
  - Key size

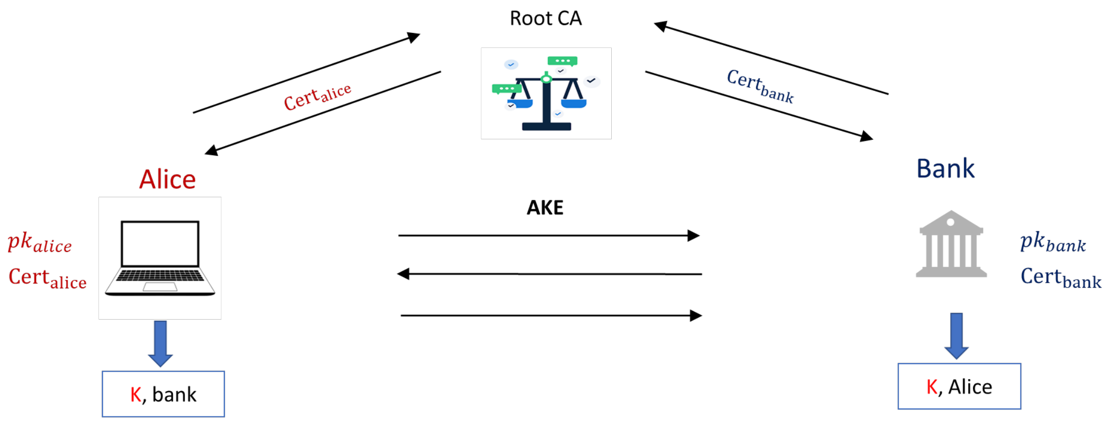


Problem: public key infrastructure (PKI)

- A single Root CA

- Single point of failure
  - What if Root CA is corrupted?

- How should we deploy the trust of certification?



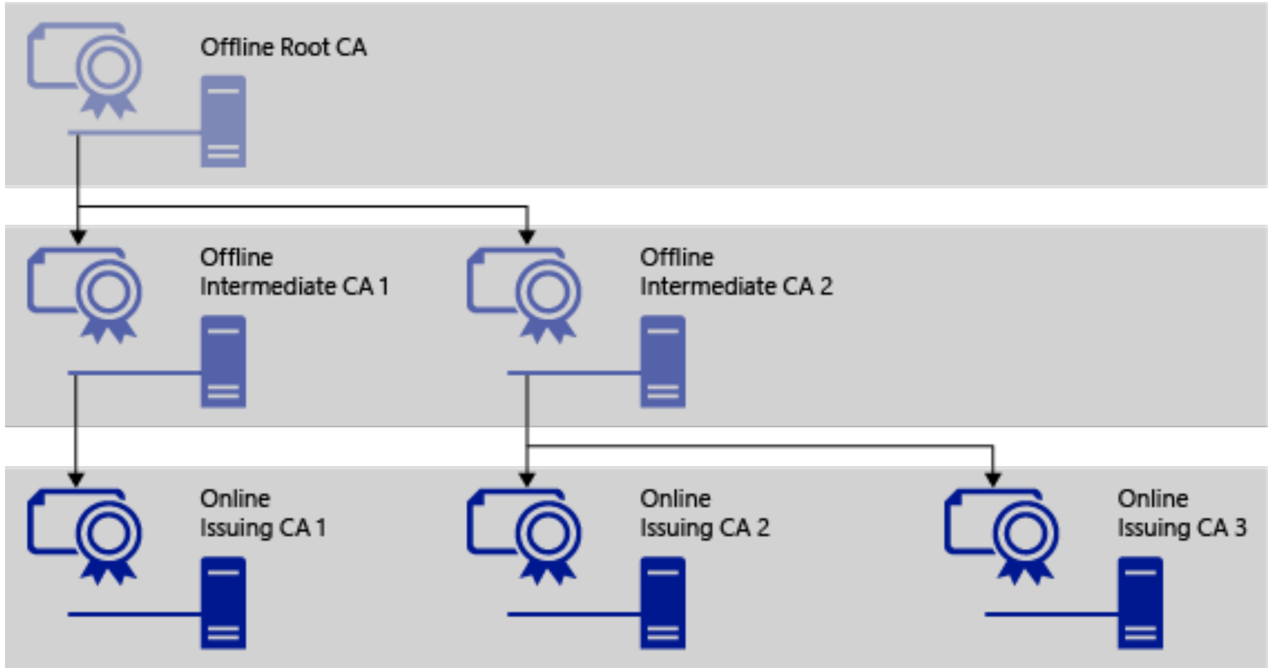
Authentication Chain

Root CAs ≈ 60

- 53 in windows

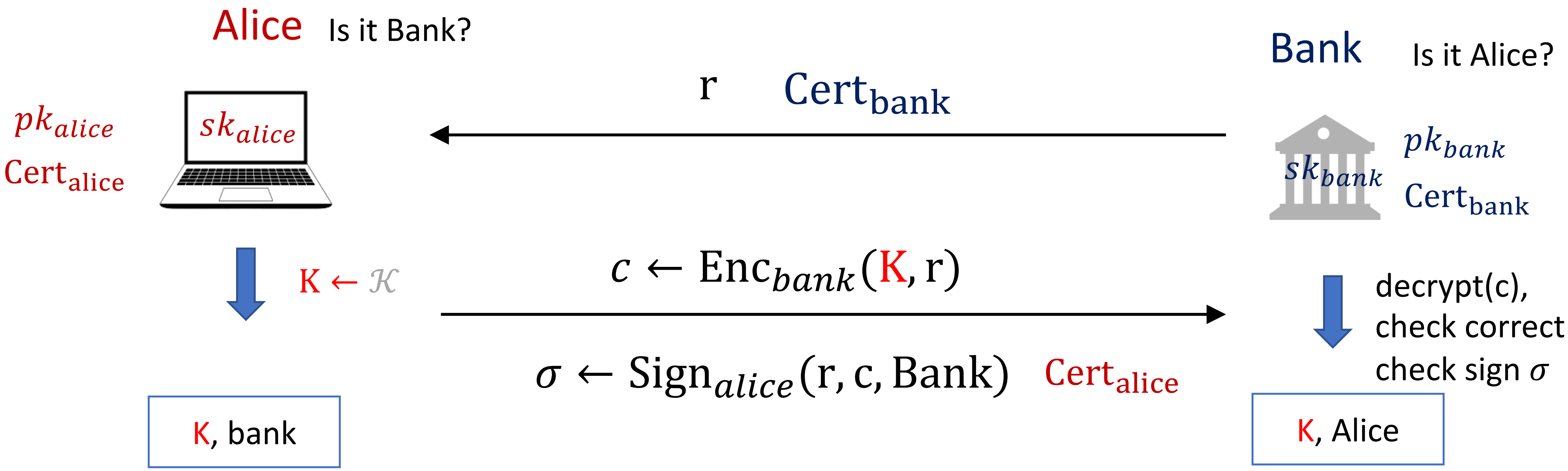
Intermediate CAs ≈ 1200

Many and many CAs



Protocol #1

AKE1 of section 21.2 in A Graduate Course in Applied Cryptography



- Theorem: Protocol #1 is a statically secure AKE
- Informally: if Alice and Bank are not corrupt then we have
  (1) secrecy for Alice\Bank and (2) authenticity for Alice\Bank

Protocol #1

AKE1 of section 21.2 in A Graduate Course in Applied Cryptography

Alice Is it Bank?

Bank Is it Alice?

$r \quad \mathrm{Cert}_{\mathrm{bank}}$ ←

sk_{bank}

pk_{bank}

$\mathrm{Cert}_{\mathrm{bank}}$

$\mathrm{K} \leftarrow \mathcal{K}$

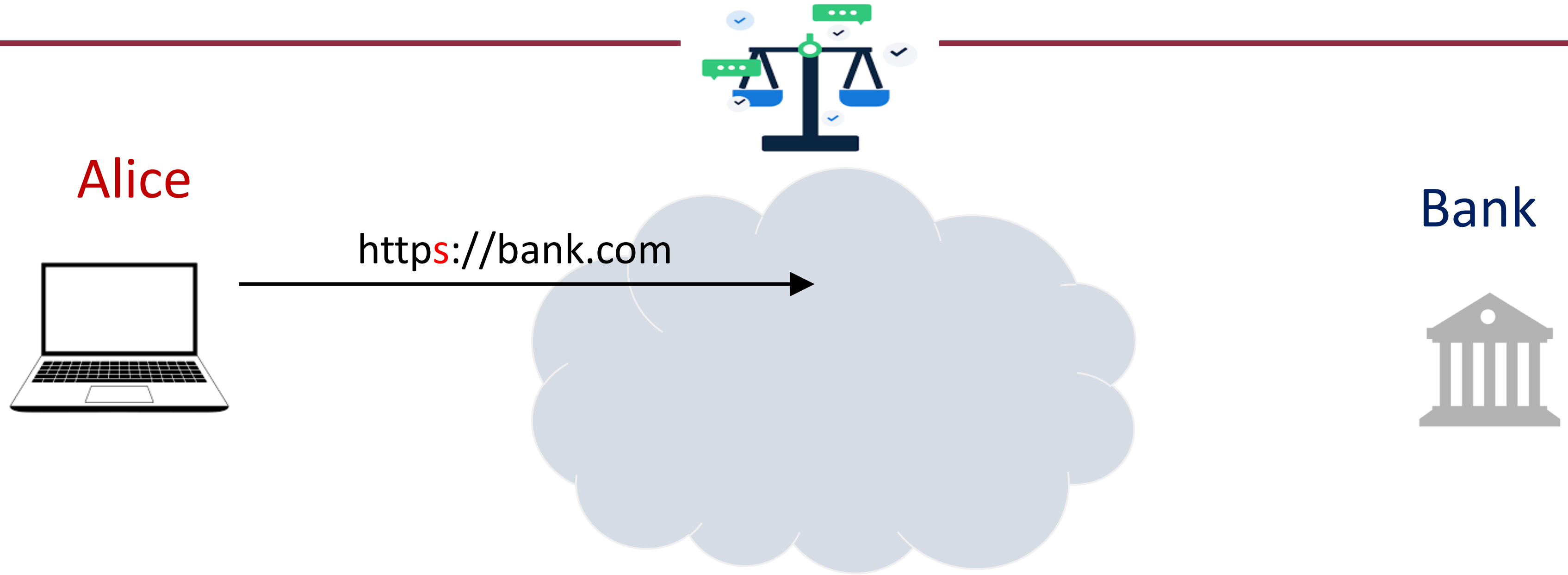
$c \leftarrow \mathrm{Enc}_{bank}(\mathrm{K}, \mathrm{r})$ →

decrypt(c),
check correct

K, bank

K, Alice

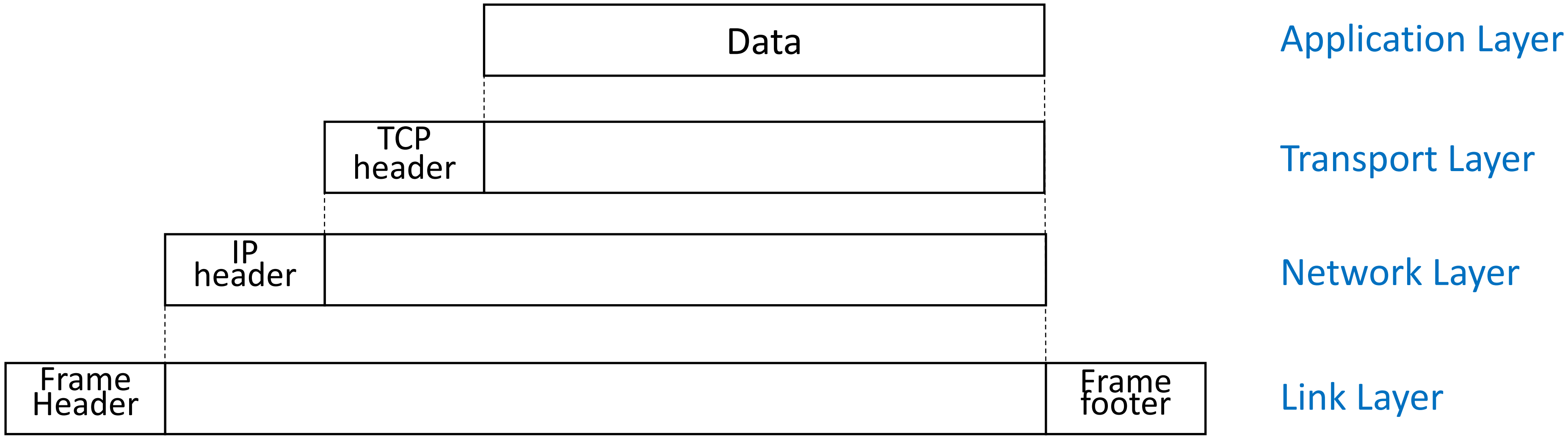
In practice



Transport Layer Security (TLS)
and
Secure Socket Layer (SSL)

TCP/IP

- TCP/IP (Transmission Control Protocol/Internet Protocol)
- introduced in the mid-1970s
- This protocol consists of four layers (other separations exist)

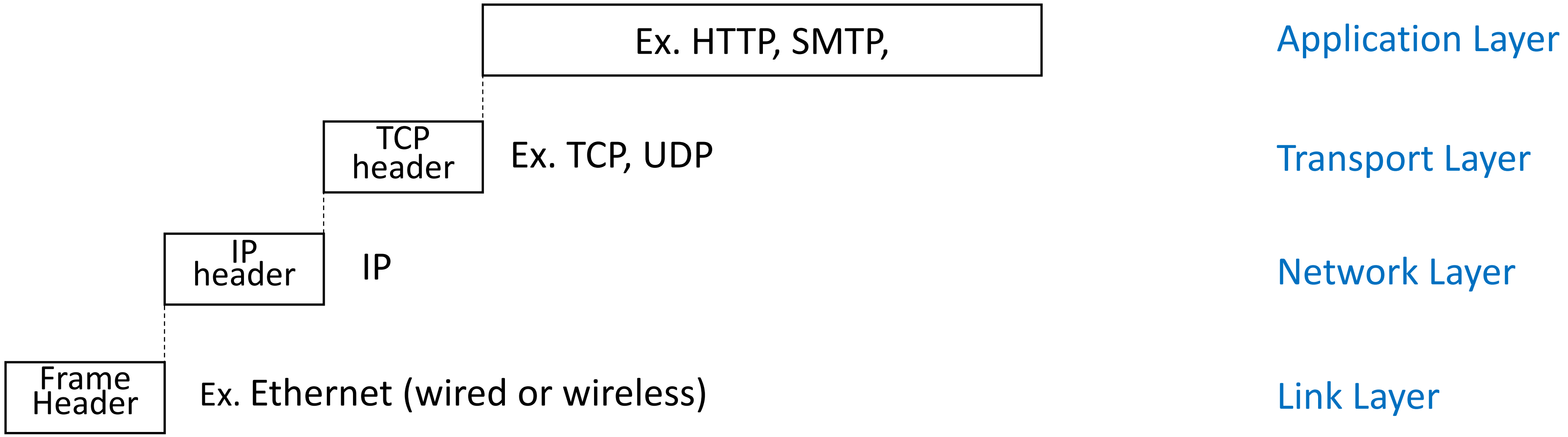


Headers of higher layer becomes lower data in the package

Basic Network Layers

	visit http://www.google.com/"	Application Layer
TCP header	connect to 74.125.136.105 create a channel for a specific application	Transport Layer
IP header	send this small packet of bytes to 74.125.136.105 which is some where else	Network Layer
Frame Header	send this small packet of bytes to this other computer that I am directly connected to	Link Layer

Basic Network Layers



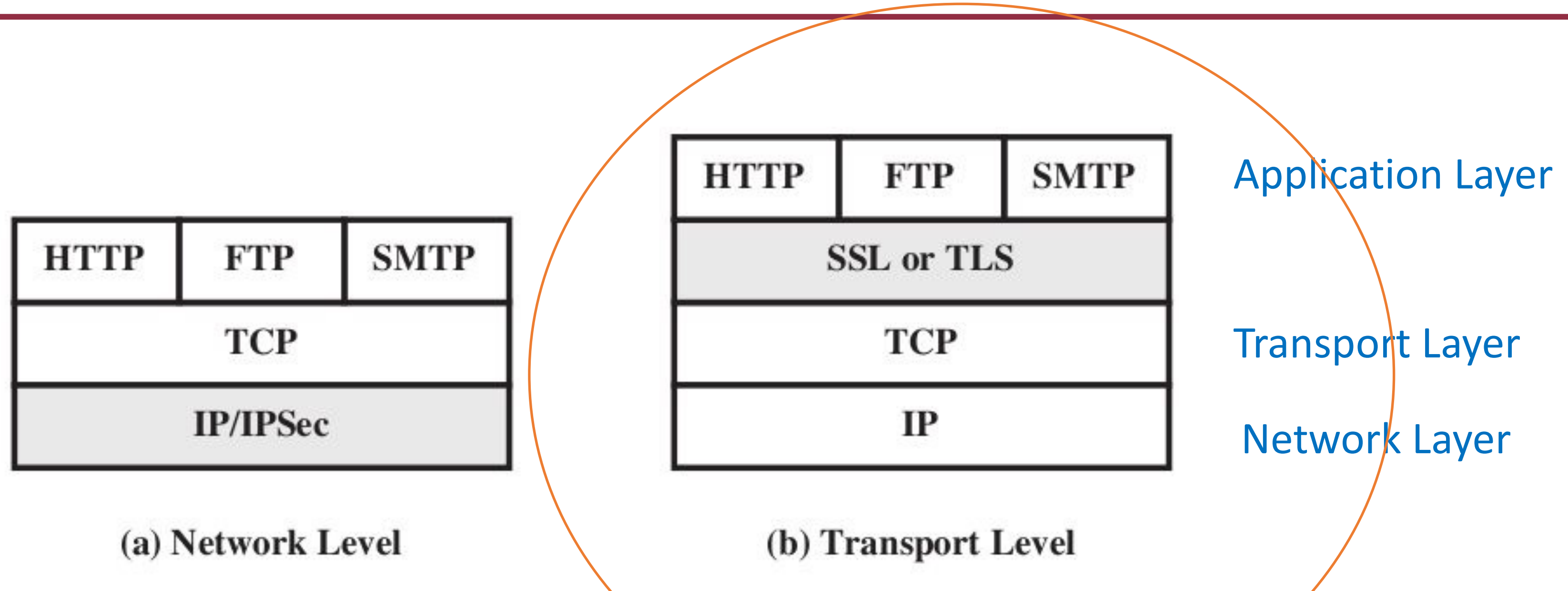


Figure 17.1 Relative Location of Security Facilities in the TCP/IP Protocol Stack

- Advantage of (a): Can protect all traffic (TCP, UDP, …)
  - Particularly good for VPNs
- Advantage of (b): Understands “connections”
  - Particularly good for protecting connections to specific application

TLS/SSL

- Transport Layer Security (TLS)/Secure Socket Layer(SSL)protocol
- are the protocols used by your browser any time you connect to a website using https rather than http

- It consists of two parts:
  - a handshake protocol that performs authenticated key exchange to establish the shared keys,
  - and a record-layer protocol that uses those shared keys to encrypt/authenticate the parties’ communication.

SSL/TLS History

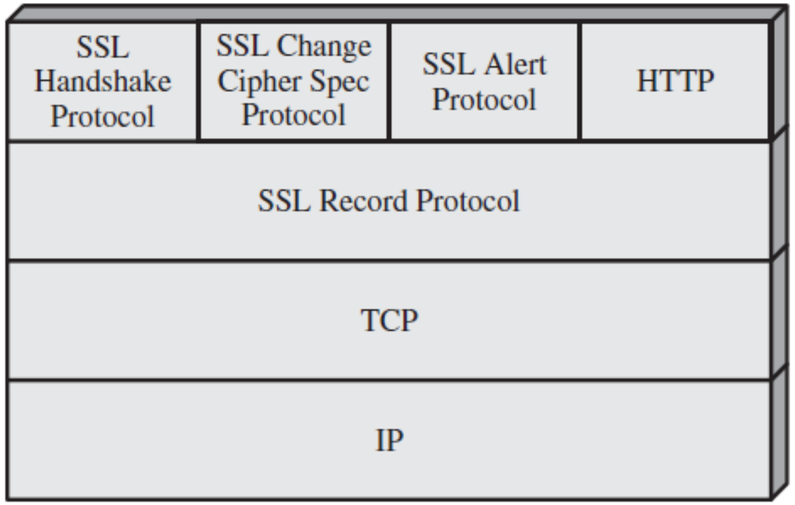
- SSL - “Secure Sockets Layer”
  - Invented by Netscape to enable secure web browsing/e-commerce
  - Fundamental to Netscape’s business model
  - First release version was “Version 2.0” - released in 1995
  - Quickly followed by security-fixes in version 3.0 (1996)

- TLS - “Transport Layer Security”: IETF standardization
  - TLS 1.0 is SSL 3.1 (released 1999)
  - TLS 1.2 in 2008
  - TLS 1.3 in use since 2018

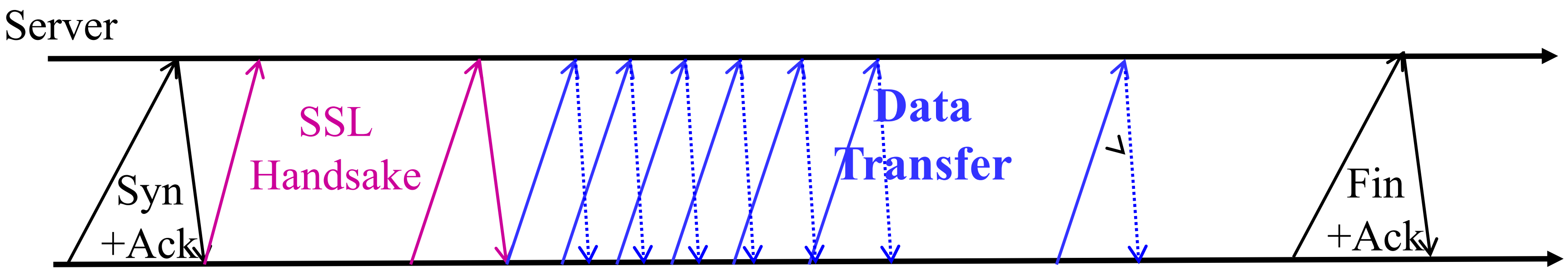
SSL Architecture

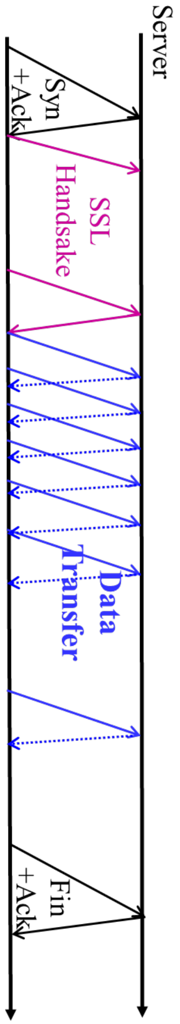
- handshake protocol: server[+client] authenticated key exchange, cipher suite negotiation, etc. to establish a shared key

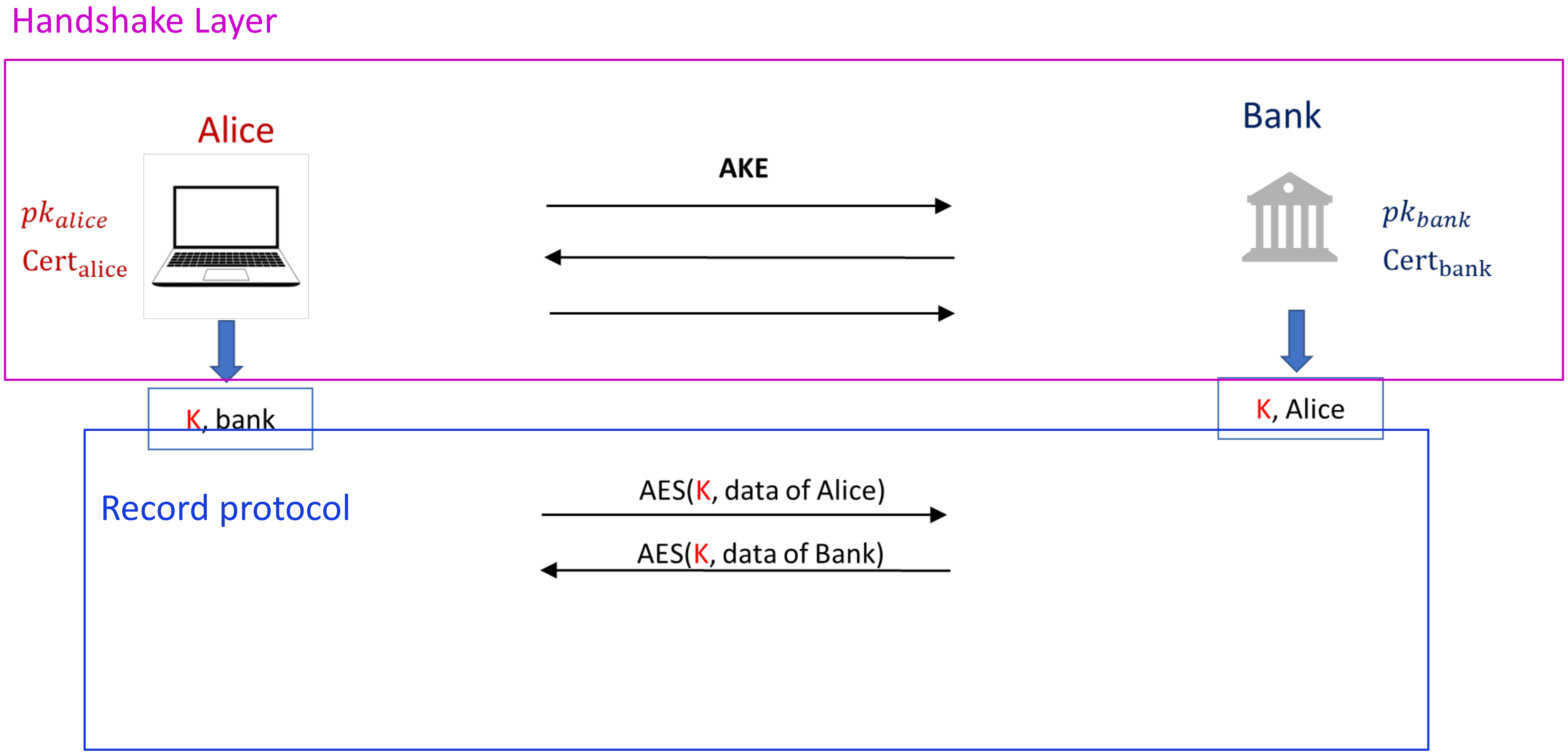
- a record protocol: secure communication between client and server using exchanged session keys



- TCP Connection setup (Syn+Ack)
- Handshake (key establishment)
  - Negotiate (agree on) algorithms, methods
  - Authenticate server and optionally client, establish keys
- Data transfer
- TCP connection closure (Fin+Ack)




Server
Syn
+Ack
SSL
Handsake
Data
Transfer
Fin
+Ack

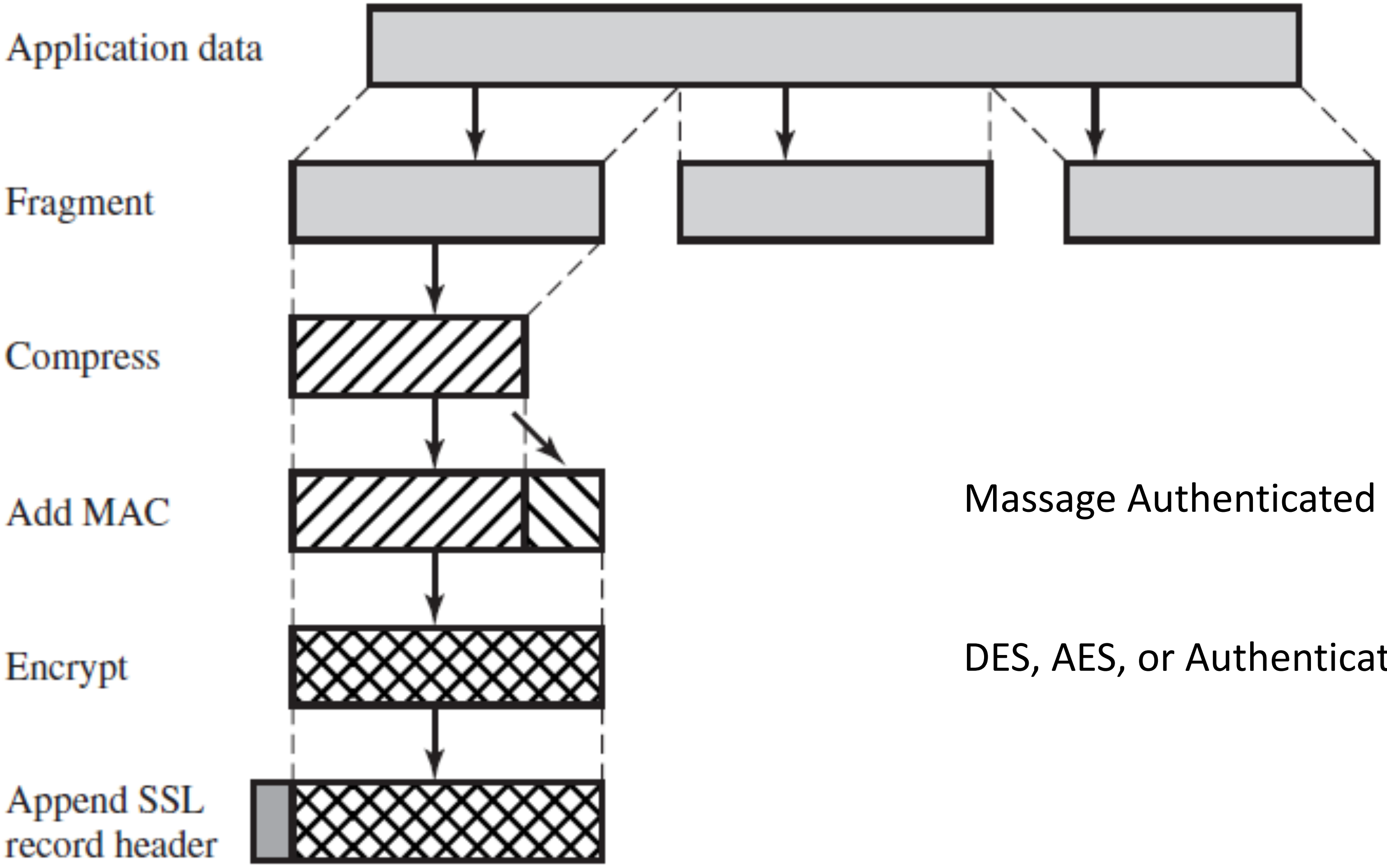

Handshake Layer
Alice
pk_{alice}
$Cert_{alice}$
AKE
Bank
pk_{bank}
$Cert_{bank}$
K, bank
K, Alice
Record protocol
AES(K, data of Alice)
AES(K, data of Bank)

The record-layer protocol

- Assume underlying reliable communication (TCP)
- Assume a session key is established by Handshake

- Four services (in order):
  - Fragment: break TCP stream into fragments (<16KB)
  - Compress (lossless) each fragment
    - Reduce processing, communication time
    - Ciphertext cannot be compressed – must compress before
  - Authenticate: [seq#||type||version||length||comp_fragment]
  - Encrypt
    - After padding (if necessary)

Record Protocol

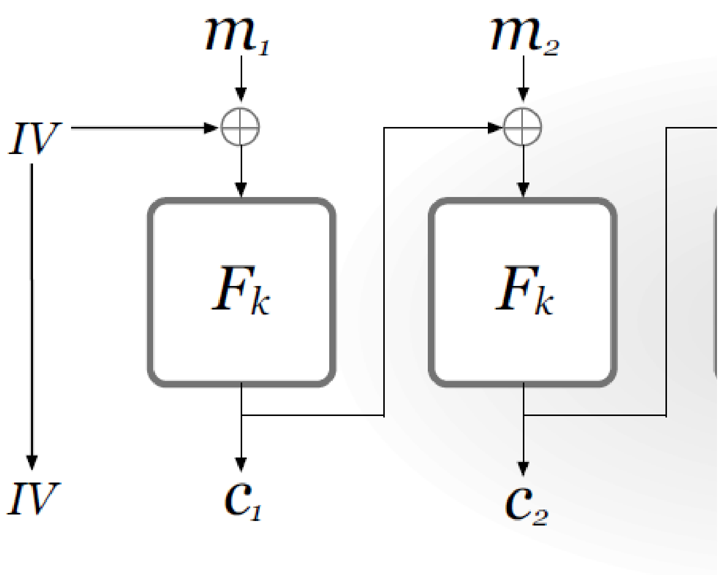


Massage Authenticated Code: HMAC MD5/SHA256(k, m)

DES, AES, or Authenticated Enc

Record Layer Vulnerabilities

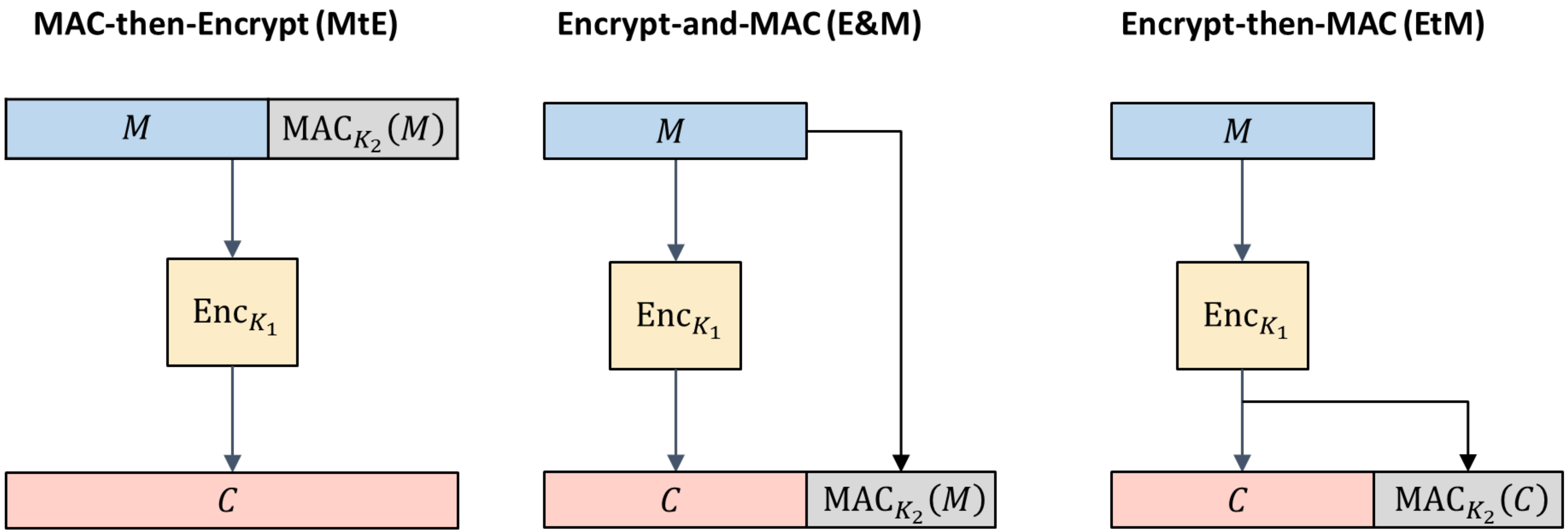
- Surprisingly many found, exploited!
- ➔ SSL, TLS1.0: vulnerable record protocol
  - Examples…
  - Attacks on RC4 → to be avoided
  - CBC IV reuse in session (BEAST)
  - `MAC-then-Encrypt': padding attacks



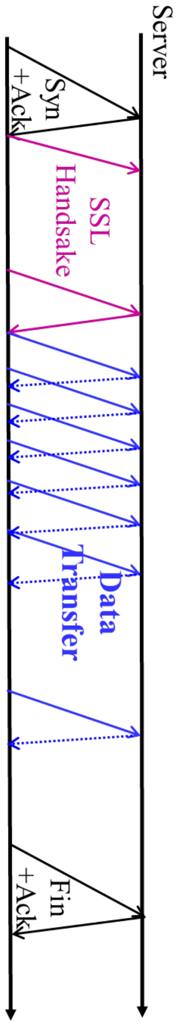
CBC IV

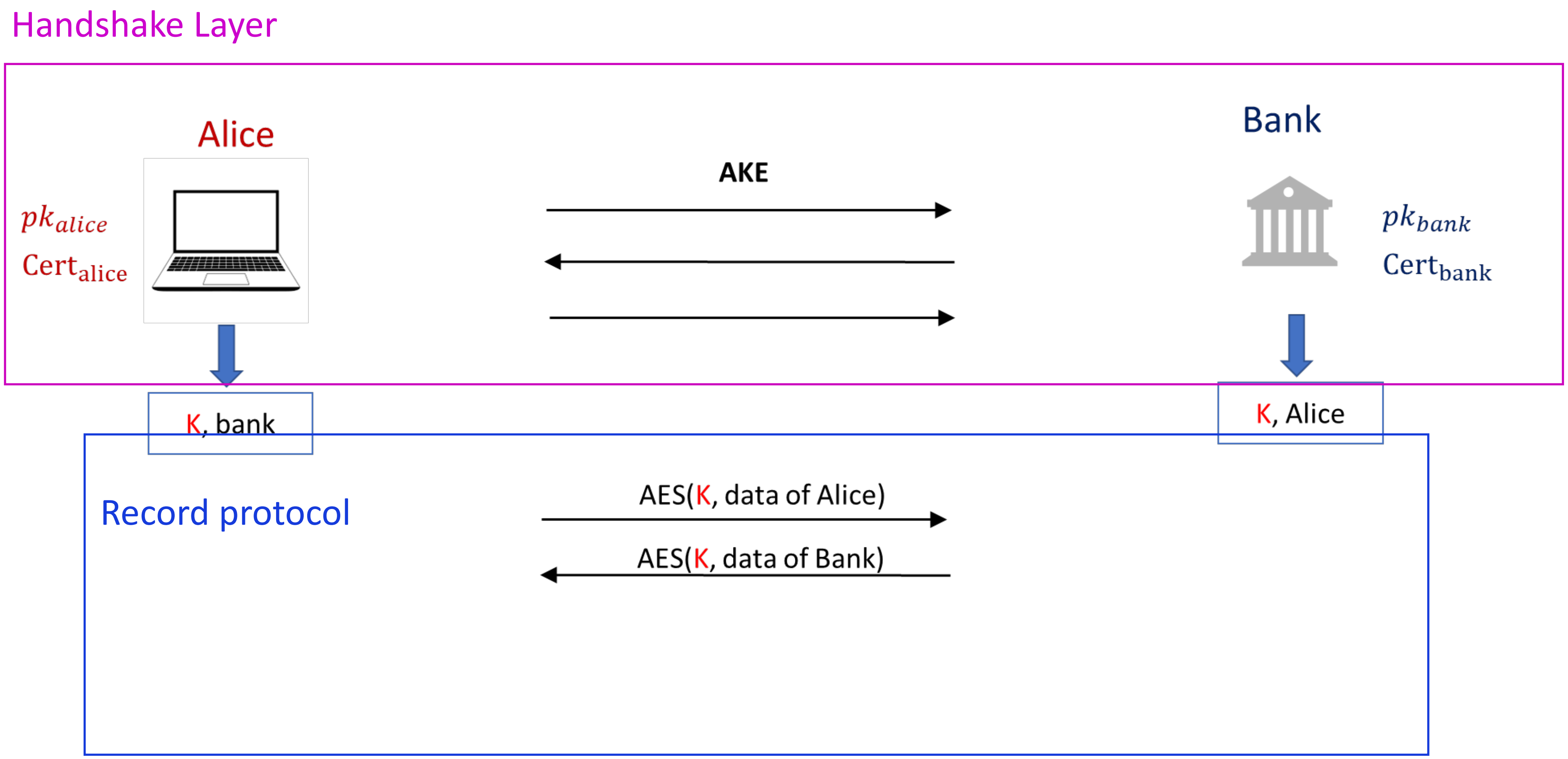
Record Layer Vulnerabilities

- ➔ SSL, TLS1.0: vulnerable record protocol
  - `MAC-then-Encrypt’: padding attacks



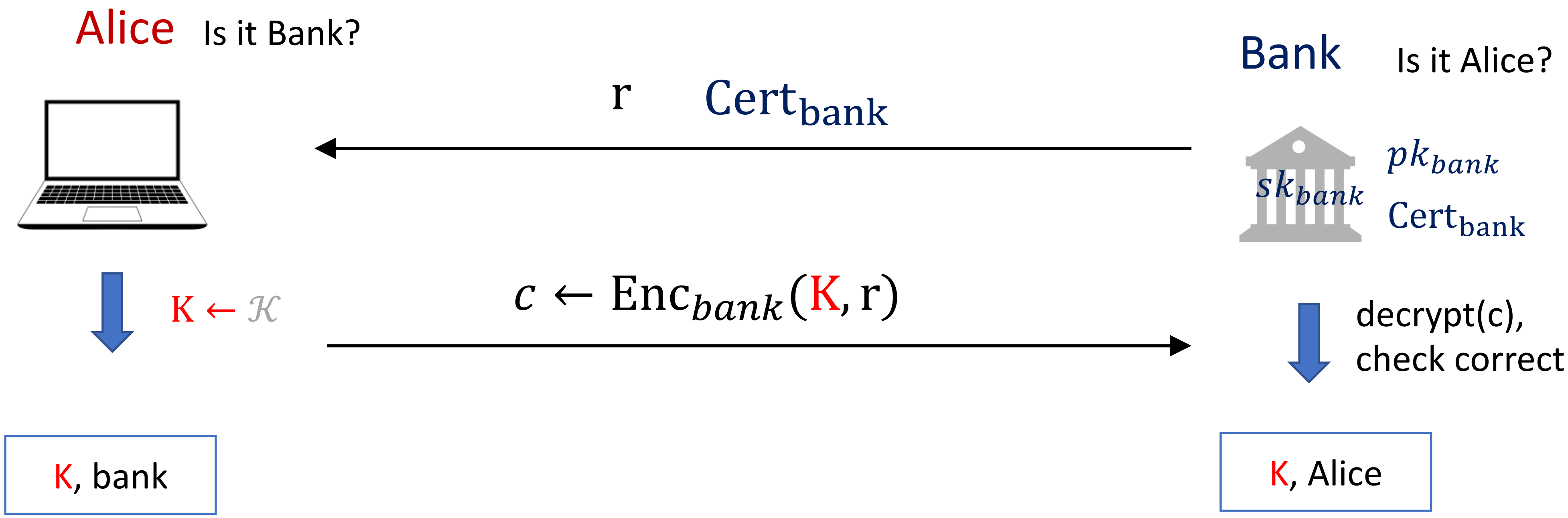
Handshake Layer



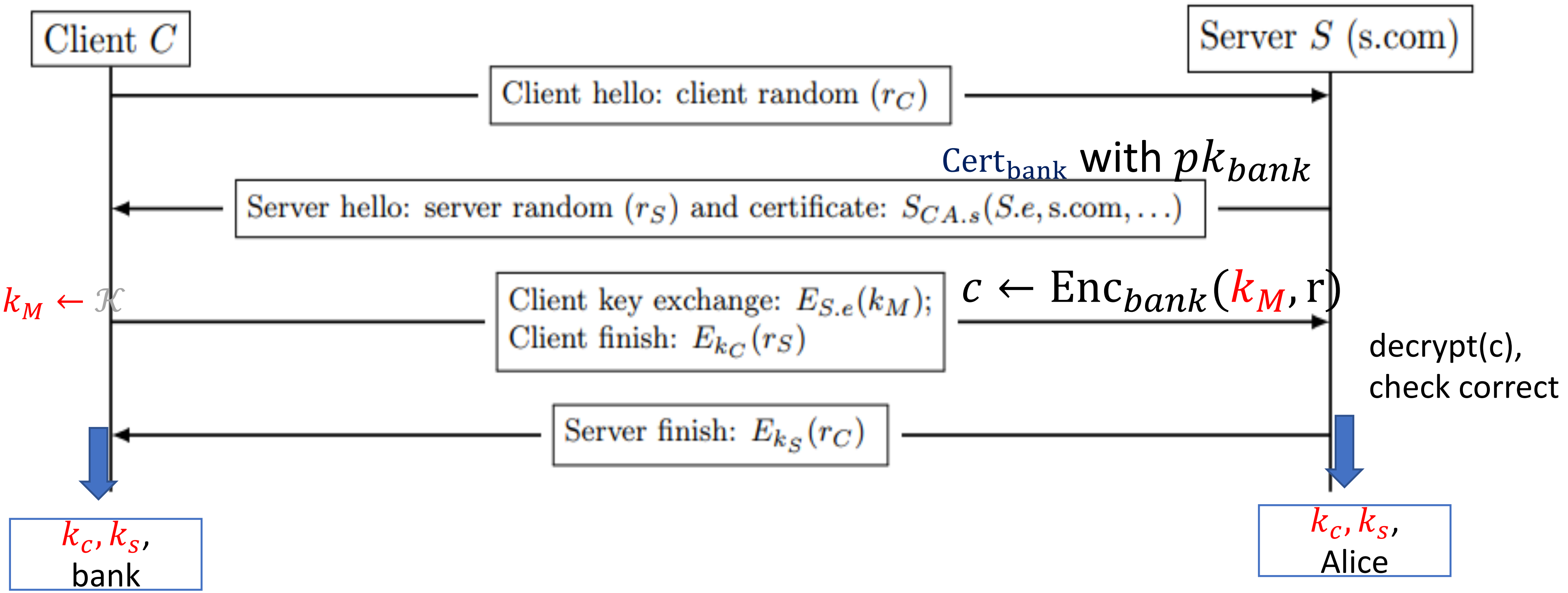


Protocol #1

AKE1 of section 21.2 in A Graduate Course in Applied Cryptography



Simplified SSLv2 Handshake



- Key derivation in SSLv2:
  - Client randomly selects k_M and sends to server
  - Client and server derive encryption keys: $K_c = K_s = KDF(k_M)$

Important concepts

- Key Derivation function, from master key K, two <u>separate</u> keys:
  - k_C, for protecting traffic from client to server
  - k_S, for protecting traffic from server to client

- Why we need a Key Derivation function here?

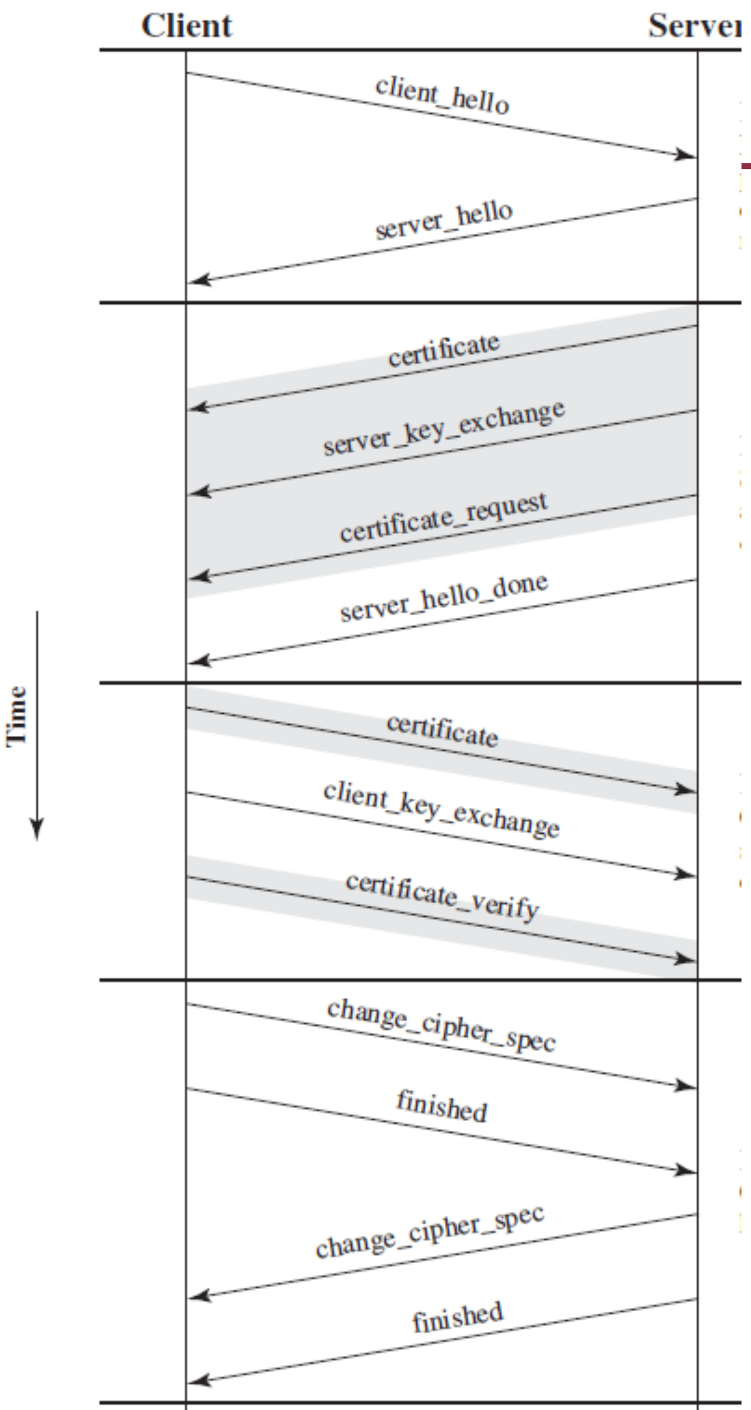
- DH overZ_p^* ? $K \in Z_p^*$
  - To encrypt a message Z_p^* by $K \cdot M \ mod \ p$
  - To encrypt a message using AES, the key should be bits? $\mathrm{K_c} = \mathrm{Hash}(K)$ etc
    - It is not secure to utilize $\mathrm{K}\ from\ Z_p^*$ as a bit string; NOT EVERY bits is random

More detail about handshake:

Phase 1: Establish security capabilities, including session ID, cipher suite, compression method, and initial random numbers.

Phase 2: Server may send certificate, key exchange, and request certificate

Phase 3: Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.



TLS 1.2 in 2008

- MD5/SHA-1----> SHA256
- Addition of support for Authenticated Encryption
  - authenticated encryption with additional data (AEAD)
- Added HMAC-SHA256 cipher suites
- Removed IDEA and DES cipher suites.

Message flow of TLS 1.2-RFC 5246

```
RFC 5246                        TLS                        August 2008

      Client                                               Server

      ClientHello                  -------->
                                                      ServerHello
                                                     Certificate*
                                               ServerKeyExchange*
                                              CertificateRequest*
                                   <--------      ServerHelloDone
      Certificate*
      ClientKeyExchange
      CertificateVerify*
      [ChangeCipherSpec]
      Finished                     -------->
                                               [ChangeCipherSpec]
                                   <--------             Finished
      Application Data             <------->     Application Data

             Figure 1.  Message flow for a full handshake
```

cipher suite, initial random numbers

cipher suite, initial random numbers

Cert_server

pk_server

Cert_client

$\mathrm{Enc}_{server}(K)$

Sign_c ()

TLS 1.2

- RSA encryption
  - We have talked before. It need to fix a public key
  - Diffie-Hellman Key exchange is better and provides forward security
- CBC model encryption
  - BEAST and Lucky 13 attack
- RC4 encryption: insecure
- SHA1: insecure

TLS 1.3-2018- RFC 8446

- Authenticated Encryption with Associated Data (AEAD)
- Static RSA and Diffie-Hellman (Enc) cipher suites have been removed
- All handshake messages is encrypted/after key is established
- Key derivation function is HMAC
- Etc.

Protocol #4 one side-use Diffie-Hellman instead of PKE

Alice Is it Bank?

Bank Is it Alice?

$\text{Cert}_{\text{bank}}$

g^a

g^b

$\text{K}||\text{k}' \leftarrow H(g^{ab})$

$\text{c}' = \text{AES}(\text{k}';\ \text{Cert}_{\text{bank}},)\ Sign_{bank}(pk,.)$

$\text{K}||\text{k}' \leftarrow H(g^{ab})$

Dec c′ with k′

Check Sign

K, bank

Delete a

K, Alice

Delete b

[variant of TLS 1.3]

TLS 1.3

- Another important feature is
- The supporting of “zero round-trip time” (0-RTT)
- If there is a pre-shared keys (PSK),
- then may be used to establish a new connection ("session resumption" or "resuming" with a PSK)

Message flow of TLS 1.3

g^a

```
       Client                                           Server

Key  ^ ClientHello
Exch | + key_share*
     | + signature_algorithms*
     | [redacted]*
     v                          -------->
                                                  ServerHello  ^ Key
                                                 + key_share*  | Exch
                                                   [redacted]  v
                                        {EncryptedExtensions}  ^  Server
                                        {CertificateRequest*}  v  Params
                                               {Certificate*}  ^
                                         {CertificateVerify*}  | Auth
                                                   {Finished}  v
                                <--------  [Application Data*]
     ^ {Certificate*}
Auth | {CertificateVerify*}
     v {Finished}               -------->
       [Application Data]       <------->  [Application Data]
```

g^b

Cert_server

$Sign_{bank}$

Cert_client

$Sign_{client}$

Brackets { } [ ] encrypted Data

Message flow of TLS 1.3

g^a

g^b

```
       Client                                            Server

Key  ^ ClientHello
Exch | + key_share*
     | + signature_algorithms*
     | + psk_key_exchange_modes*
     v + pre_shared_key*       -------->
                                                  ServerHello  ^ Key
                                                 + key_share*  | Exch
                                            + pre_shared_key*  v
                                        {EncryptedExtensions}  ^  Server
                                        {CertificateRequest*}  v  Params
                                               {Certificate*}  ^
                                         {CertificateVerify*}  | Auth
                                                   {Finished}  v
                               <--------  [Application Data*]
     ^ {Certificate*}
Auth | {CertificateVerify*}
     v {Finished}              -------->
       [Application Data]      <------->  [Application Data]
```

Brackets { } [ ] encrypted Data

Message flow of TLS 1.3-RFC 8446

g^a

```
Client                                              Server

ClientHello
+ early_data
+ key_share*
+ psk_key_exchange_modes
+ pre_shared_key
(Application Data*)        -------->
                                                ServerHello
                                           + pre_shared_key
                                               + key_share*
                                      {EncryptedExtensions}
                                              + early_data*
                                                 {Finished}
                           <--------    [Application Data*]
(EndOfEarlyData)
{Finished}                 -------->
[Application Data]         <------->     [Application Data]
```

g^b

Brackets () { } [ ] encrypted Data

Find more about SSL

- Defined in RFC 2246, `http://www.ietf.org/rfc/rfc2246.txt`
- Open-source implementation at `http://www.openssl.org/`

Find more about TLS

- TLS is defined as a Proposed Internet Standard

- TLS v1.2 RFC 5246
- TLS v1.3 RFC 8446

HTTPS

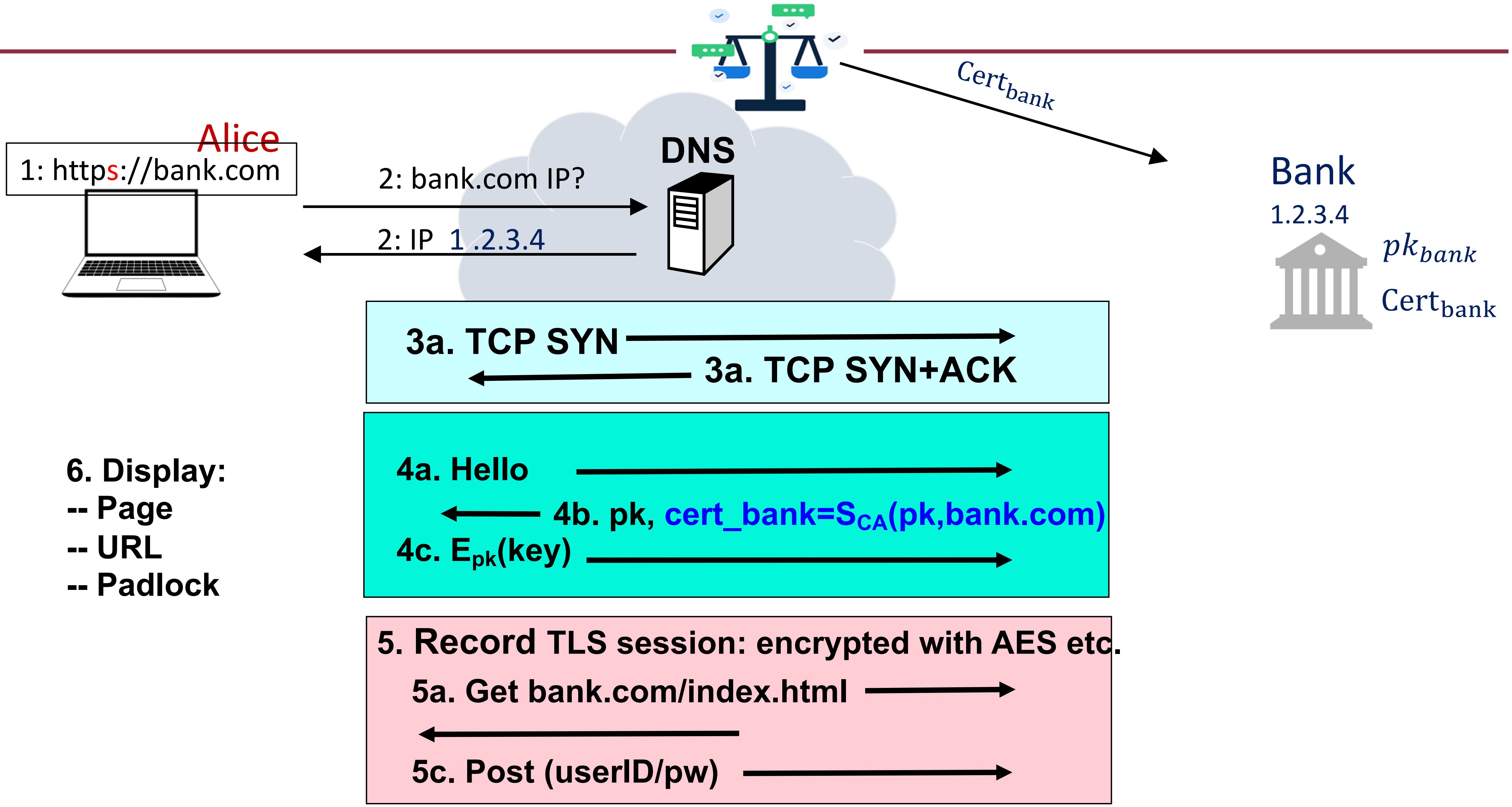
Put it all together

HTTPS

- HTTPS (HTTP over SSL) refers to the combination of HTTP and SSL to implement secure communication

- The principal difference seen by a user is that URL addresses begin with https:// rather than http://.
    - A normal HTTP connection uses port 80.
    - If HTTPS is specified, port 443 is used, which invokes TLS/SSL.

HTTPS



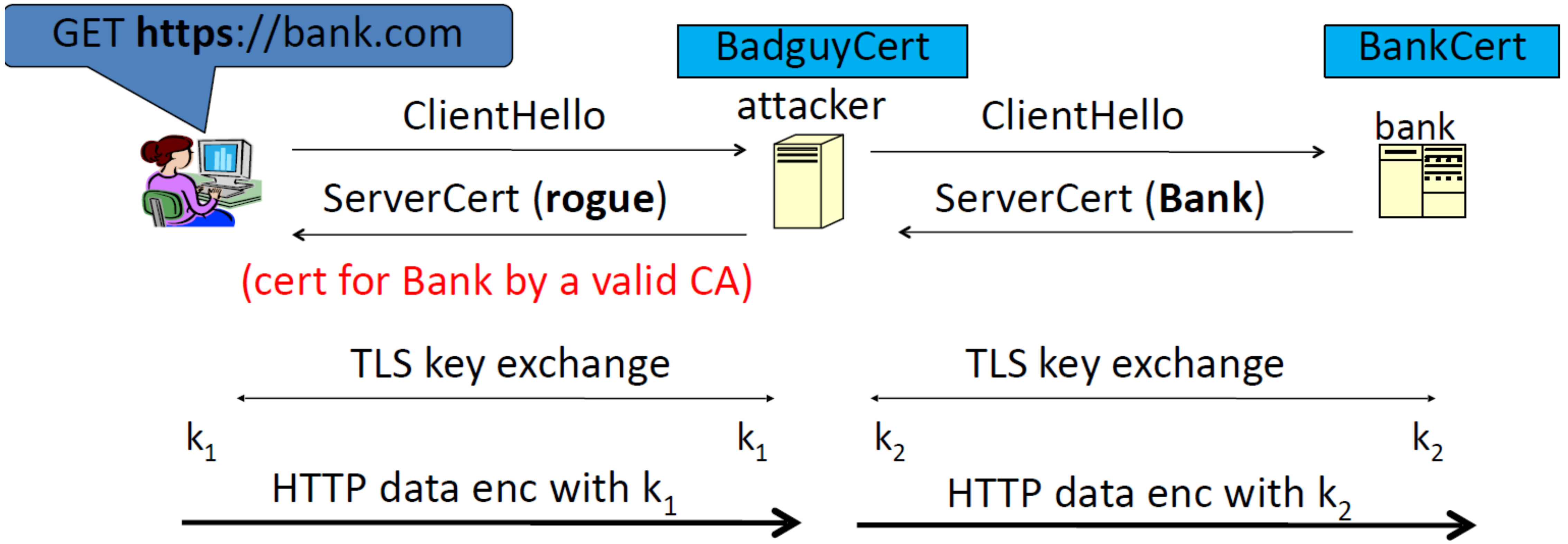
Cert$_{bank}$
Alice
1: https://bank.com
2: bank.com IP?
DNS
2: IP 1 .2.3.4
Bank
1.2.3.4
pk_{bank}
Cert$_{bank}$
3a. TCP SYN
3a. TCP SYN+ACK
4a. Hello
4b. pk, cert_bank=S_{CA}(pk,bank.com)
4c. E_{pk}(key)
5. Record TLS session: encrypted with AES etc.
5a. Get bank.com/index.html
5c. Post (userID/pw)
6. Display:
-- Page
-- URL
-- Padlock

HTTPS:Certificates: wrong issuance

- We know that all the security is based on that $\mathrm{Cert}_{\mathrm{bank}}$ is correct and safe

- 2011: **Comodo** and **DigiNotar** CAs hacked, issue certs for Gmail, Yahoo! Mail, …
- 2013: **TurkTrust** issued cert. for gmail.com
- 2016: **WoSign** (沃通) issues cert for GitHub domain (among other issues)
  Result: WoSign certs no longer trusted by Chrome, Firefox, and Apple

Man in the middle attack using rogue cert



Attacker knows data between user and bank.
Sees all traffic and can modify data at will.

Summary

- Recall AKE, PKI, and CA

- TLS/SSL
- HTTPS

- For your lecture notes, please refer to
- [Sta] Section 16
  [KPS] Section 13
  RFC 2246, 5246, 8446

Assignment 1 (Deadline 10 March)

- Implement the ElGamal Enc algorithm in Sage
  - submit the code
  - Provide “known answer-test” (KAT) values （i.e., example of pk, sk, m and c）

- Implement the Textbook RSA signature in Sage
  - submit the code
  - And show the attack that if $\sigma_1 = M_1^d, \sigma_2 = M_2^d$, then $\sigma_1\ \sigma_2$ is the Textbook RSA signature of $M_1\ M_2$
  - run the attack to Hash-then-sign of RSA and show it does not work for Hash-then-sign of RSA
  - Provide “known answer-test” (KAT) values (i.e., example of vk=(n, e), sk=d, m and σ)

- Write a report about the algorithms and implementation
- Assignment 1 will be available on the blackboard

Tutorial

- If you have any questions, I will be here
  - Assignment
  - Lecture notes
  - Previous lectures
    - Symmetric key cryptography
    - Public key cryptography
    - Etc.

Thank you