Computer Networks
COL 334/672

Network Security

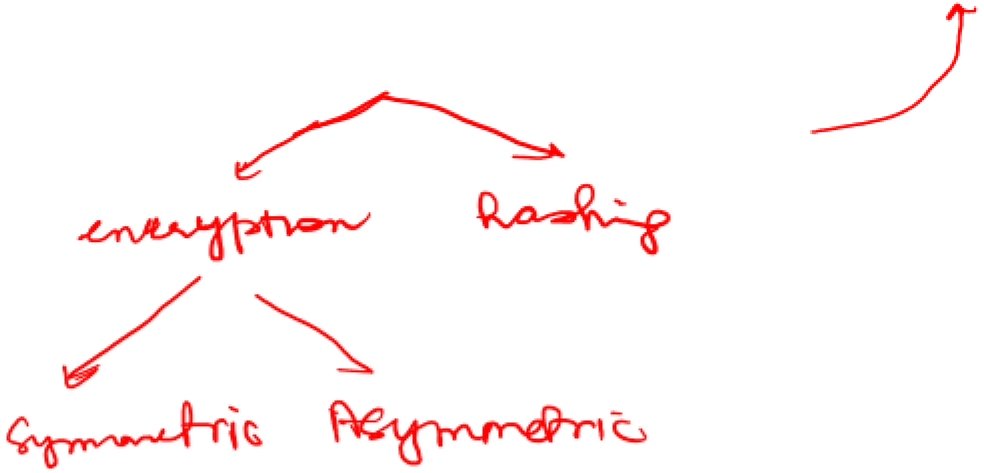
Slides adapted from KR

Sem 1, 2025-26

Recap

Techniques for secure communication

- confidentiality
- end-point authentication
- message integrity



encryption

hashing

Symmetric

Asymmetric

Network Security

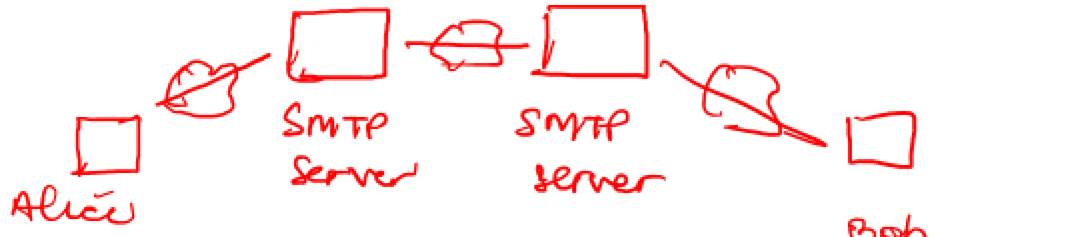
- Securing Protocols
  - Email → Appn layer
  - TCP → Transpor
  - Network-layer → N/w layer
- Operational security: firewall and IDS

Securing Email

- **Goal:** Alice wants to send confidential email, m, to Bob

$\boxed{K_B^+(K_S')}$

$m', H(m' + K_S')$



K_A^+, K_A^-

$(K_B^+)\ K_B^-$

m

Encryption : $K_B^+(m)$

computationally expensive

↳ Email headers

(META)

$[K_B^+(K_S)]$ ⊕ $K_S(m)$ → one email

MIME

encrypt the content

Key / crypto algos meta info

Content of Email

symmetric

$\boxed{K_B^+(K_S)}$

$m, H(m + \cancel{K_S})$ → won't work

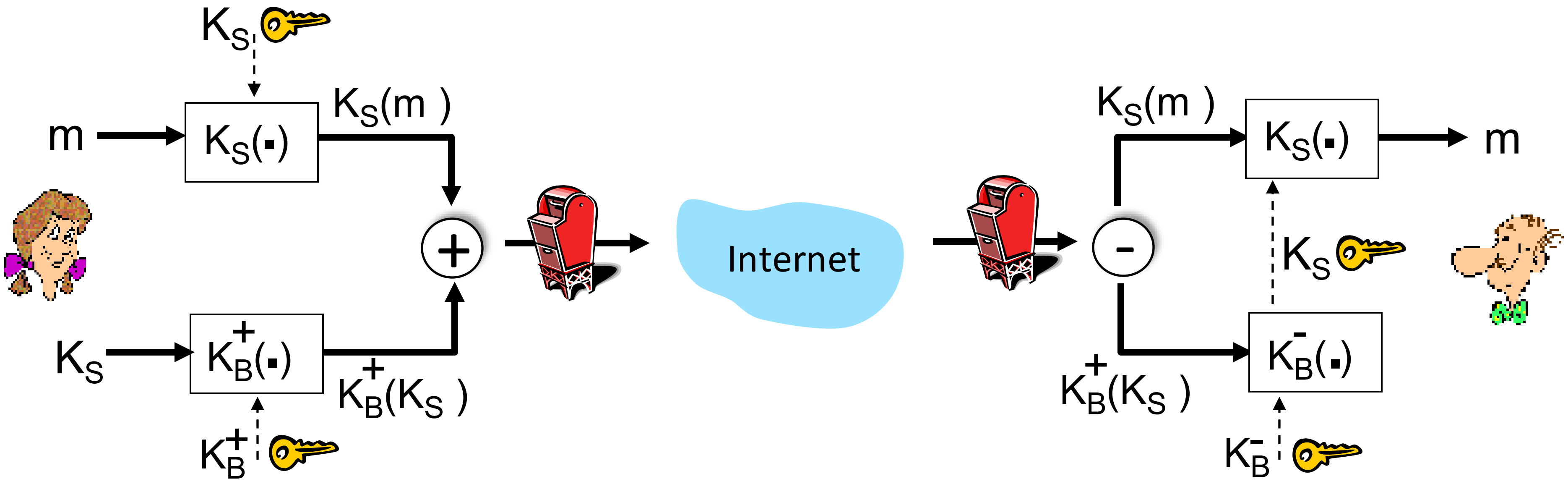
Asymmetric

$m, K_A^-(H(m))$

Bob: $H(m) \stackrel{?}{=} K_A^+(K_A^-(H(m)))$

Secure e-mail: confidentiality

Alice wants to send *confidential* e-mail, m, to Bob.



Alice:

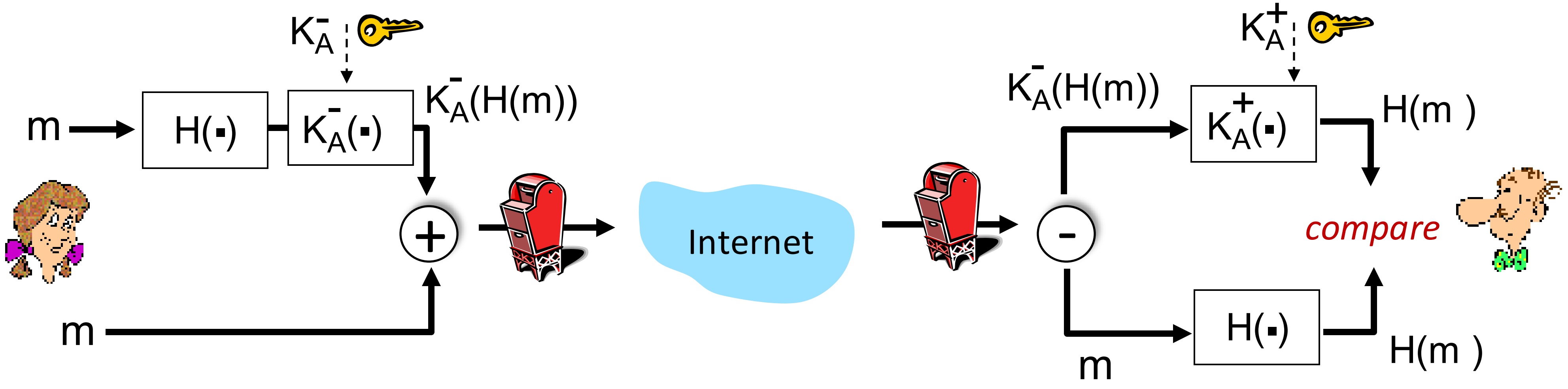
- generates random *symmetric* private key, K_S
- encrypts message with K_S (for efficiency)
- also encrypts K_S with Bob's public key
- sends both $K_S(m)$ and $K^+_B(K_S)$ to Bob

Bob:

- uses his private key to decrypt and recover K_S
- uses K_S to decrypt $K_S(m)$ to recover m

Secure e-mail: integrity, authentication

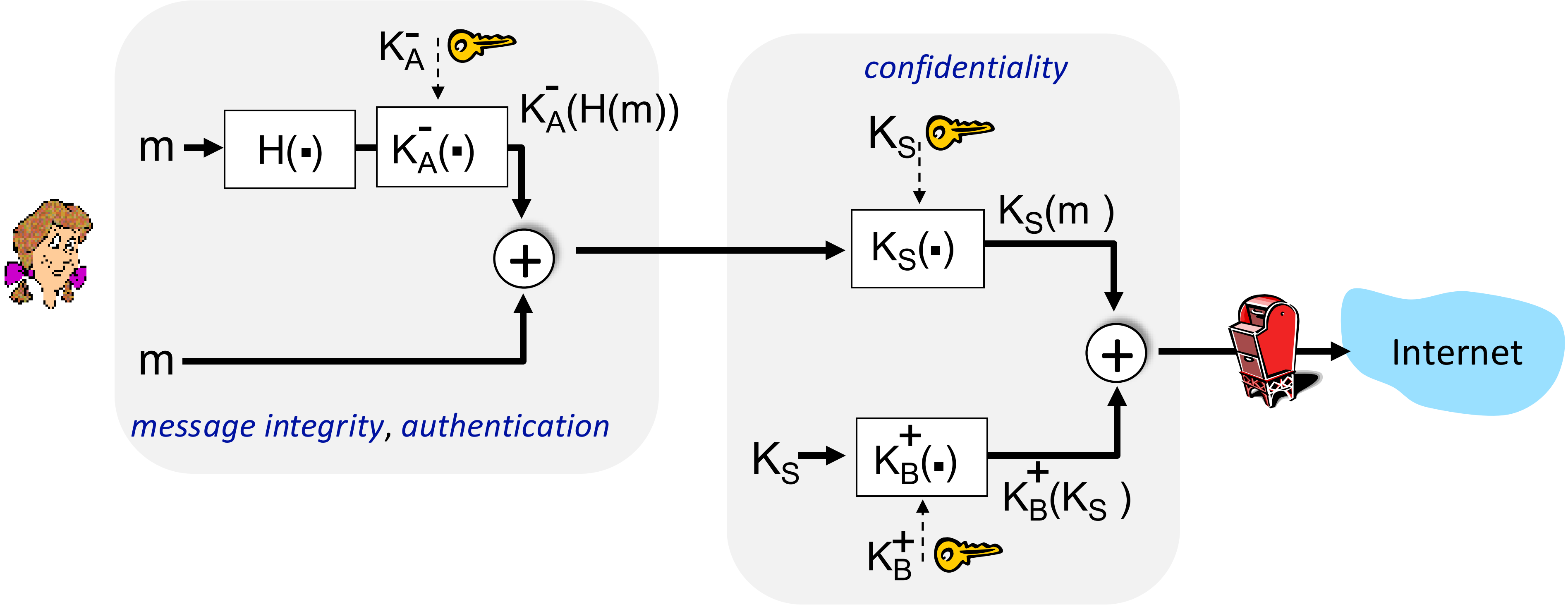
Alice wants to send m to Bob, with *message integrity*, *authentication*



- Alice digitally signs hash of her message with her private key, providing integrity and authentication
- sends both message (in the clear) and digital signature

Secure e-mail: integrity, authentication

Alice sends m to Bob, with *confidentiality, message integrity*, *authentication*



What are Bob's complementary actions?

How do Alice and Bob obtain each other’s public keys?

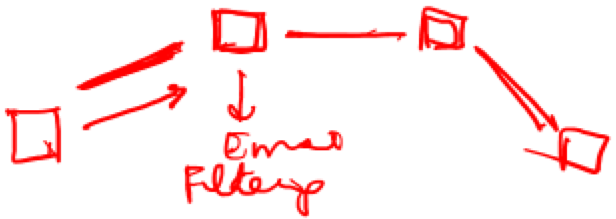
↳ Host your public key on a website

↳ Key signing parties : Decentralized web of Trust

Pretty Good Privacy (PGP)

Network Security

- Securing Protocols
  - Email
  - **TCP** → TLS
  - Network-layer
- Operational security: firewall and IDS



SSL → TLS

HTTPS → TLS

Transport-layer security (TLS)

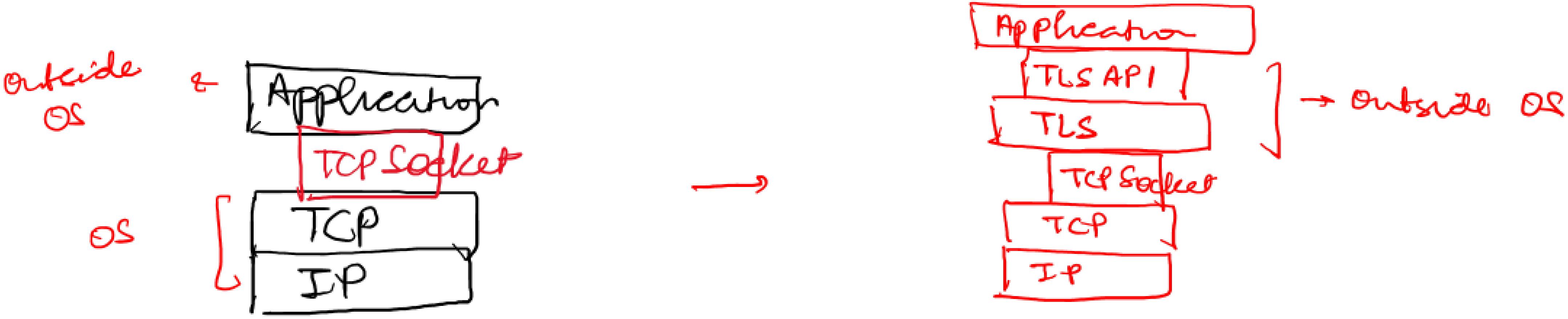
- widely deployed security protocol above the transport layer
  - supported by almost all browsers, web servers: https (port 443)
- provides:
  - confidentiality: via *symmetric encryption*
  - integrity: via *cryptographic hashing*
  - authentication: via *public key cryptography*

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 (TLS)

- TLS provides an API that *any* application can use



TCP API

s. send

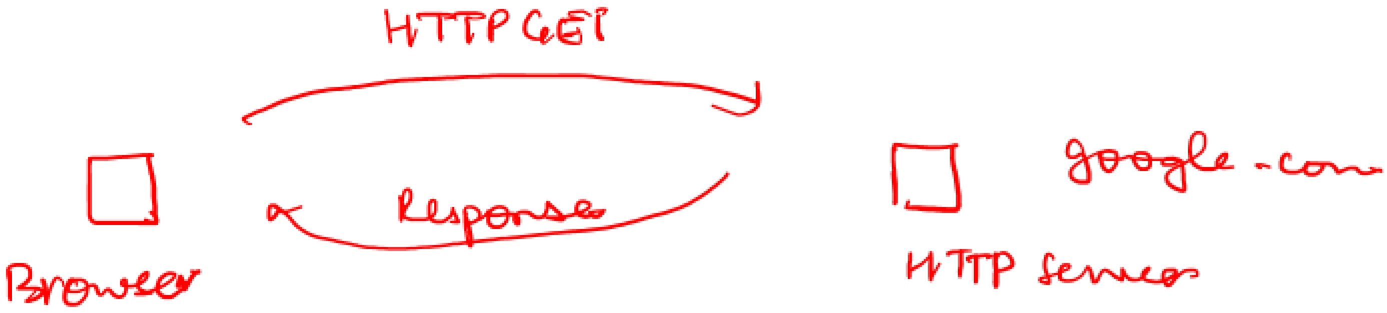
TCP socket

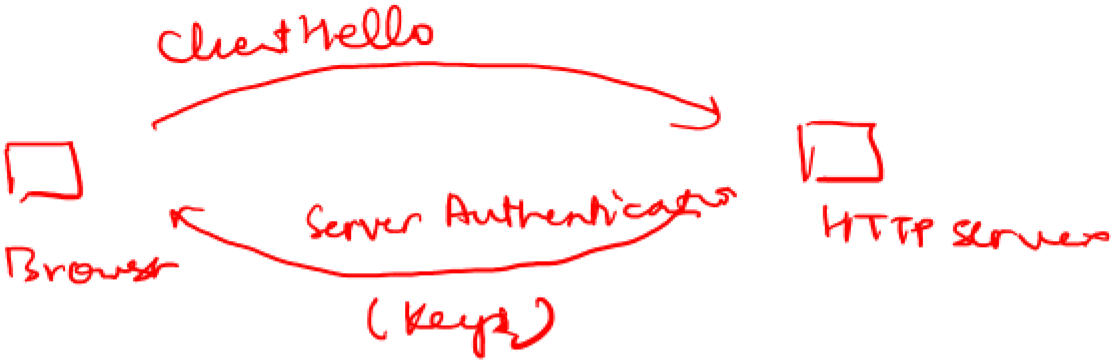
TLS socket

tls.send

Transport-layer Security (TLS): Key Steps

- What needs to be done to implement confidentiality, end-point authentication, and integrity?

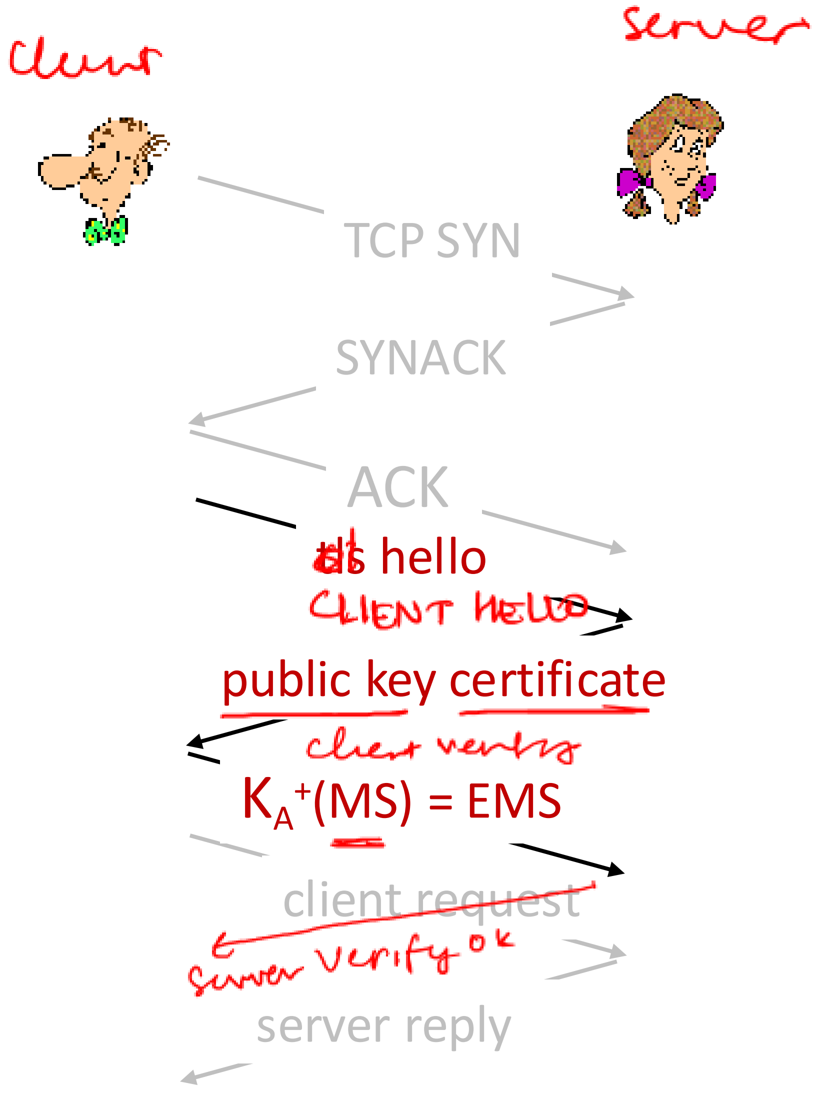




Transport-layer security: what’s needed?

- Step 1 - Handshake and Key Derivation: Alice, Bob use their certificates, private keys to authenticate each other, exchange or create shared secret and use the shared secret to derive set of keys

t-tls: initial handshake



client

server

TCP SYN

SYNACK

ACK

tls hello

CLIENT HELLO

public key certificate

client verify

$K_A^+(MS) = EMS$

client request

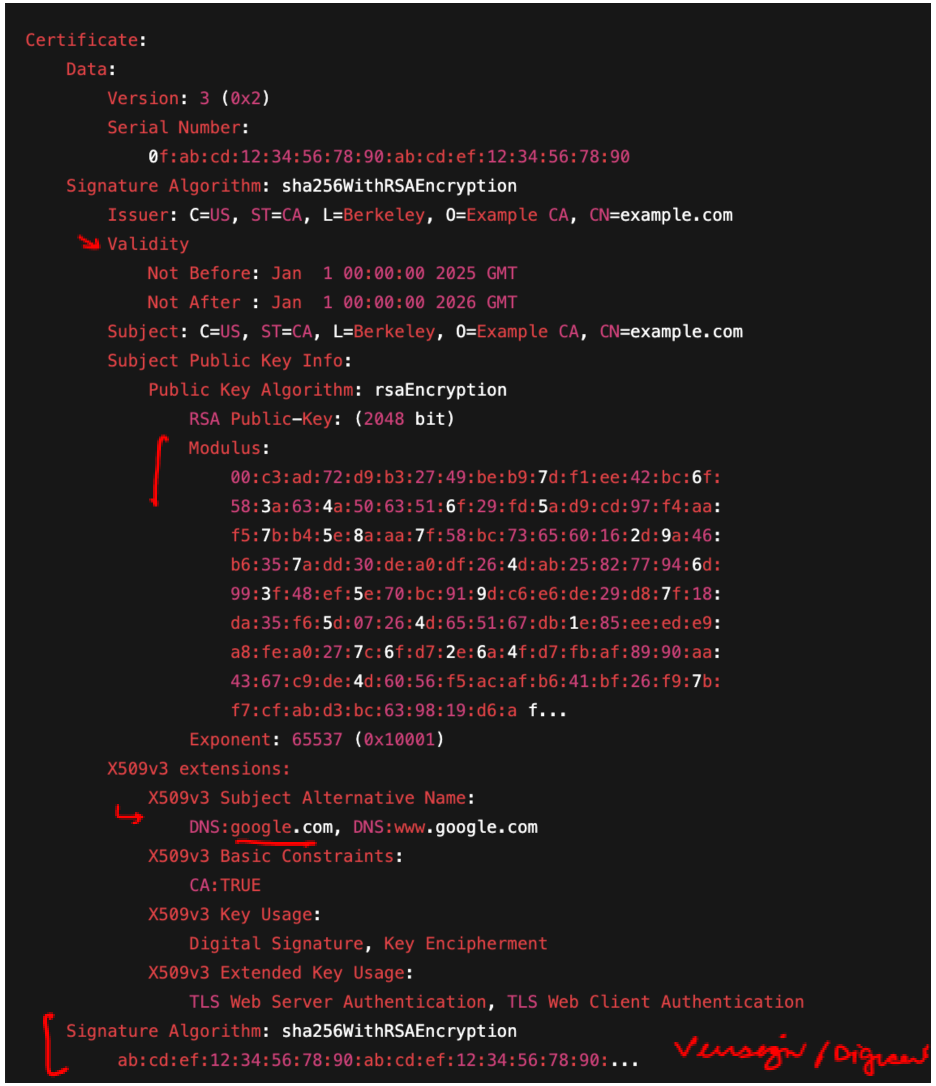
server verify ok

server reply

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)

Example public key certificate



```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0f:ab:cd:12:34:56:78:90:ab:cd:ef:12:34:56:78:90
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=CA, L=Berkeley, O=Example CA, CN=example.com
        Validity
            Not Before: Jan  1 00:00:00 2025 GMT
            Not After : Jan  1 00:00:00 2026 GMT
        Subject: C=US, ST=CA, L=Berkeley, O=Example CA, CN=example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c3:ad:72:d9:b3:27:49:be:b9:7d:f1:ee:42:bc:6f:
                    58:3a:63:4a:50:63:51:6f:29:fd:5a:d9:cd:97:f4:aa:
                    f5:7b:b4:5e:8a:aa:7f:58:bc:73:65:60:16:2d:9a:46:
                    b6:35:7a:dd:30:de:a0:df:26:4d:ab:25:82:77:94:6d:
                    99:3f:48:ef:5e:70:bc:91:9d:c6:e6:de:29:d8:7f:18:
                    da:35:f6:5d:07:26:4d:65:51:67:db:1e:85:ee:ed:e9:
                    a8:fe:a0:27:7c:6f:d7:2e:6a:4f:d7:fb:af:89:90:aa:
                    43:67:c9:de:4d:60:56:f5:ac:af:b6:41:bf:26:f9:7b:
                    f7:cf:ab:d3:bc:63:98:19:d6:a f...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:google.com, DNS:www.google.com
            X509v3 Basic Constraints:
                CA:TRUE
            X509v3 Key Usage:
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
    Signature Algorithm: sha256WithRSAEncryption
         ab:cd:ef:12:34:56:78:90:ab:cd:ef:12:34:56:78:90:...
```

Verisign / Digicert

t-tls: cryptographic keys

$f(K_{ms}, NONCE_c)$

$NONCE_s$

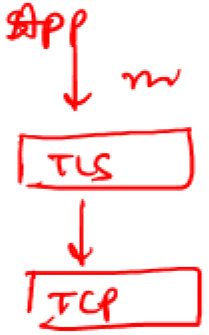
client , server

- 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

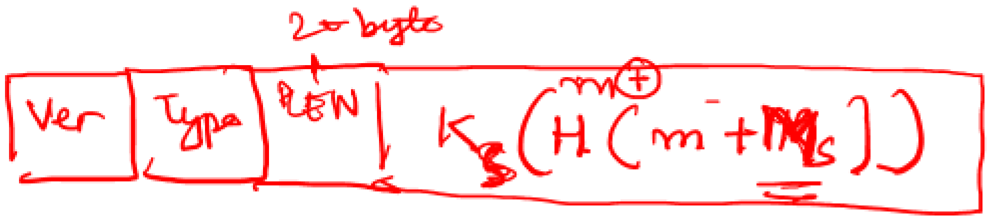
TLS Encrypting Data

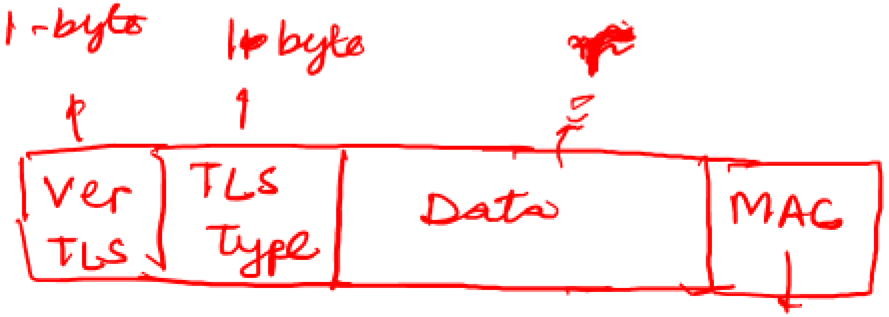
Message Authentication + integrity
↓
End-point Authen

- recall: TCP provides data *byte stream* abstraction
- Q: can we encrypt data in-stream as written into TCP socket?



split ~~TCP~~ data into TLS records
byte stream

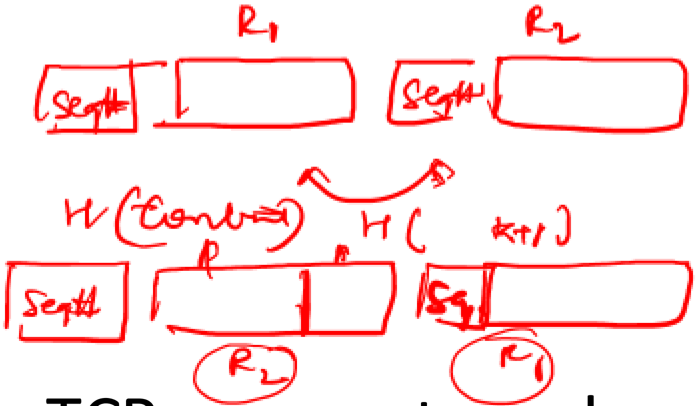




Message Authentication Code

TLS data

TLS: Encrypting Data



- possible attacks on data stream?
  - *re-ordering:* man-in middle intercepts TCP segments and reorders (manipulating sequence #s in unencrypted TCP header)
  - *replay*
- solutions: 0, 1, ... k
  - use TLS sequence numbers (data, TLS-seq-# incorporated into MAC)
  - use nonce

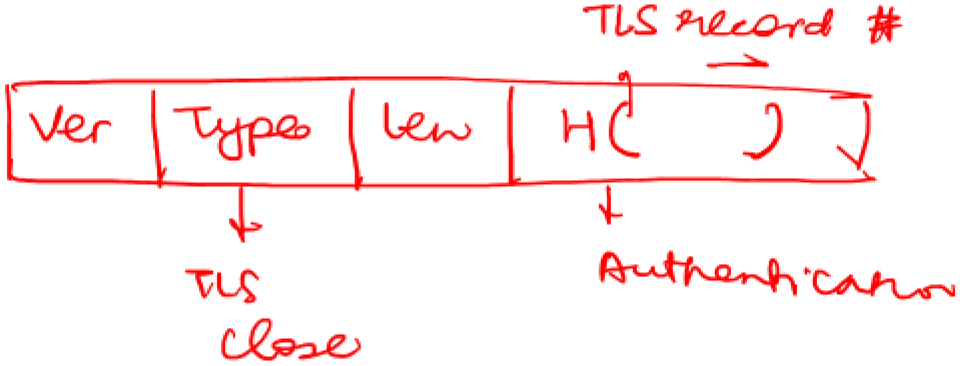
$m \oplus H(m \oplus M_c \oplus \overset{TLS}{\underline{seq\#}} \oplus length)$

t-tls: connection close

FIN

- truncation attack:
  - attacker forges TCP connection close segment
  - one or both sides thinks there is less data than there actually is

TLS close


TLS record #
Ver
Type
len
H( )
TLS close
Authentication