

# COMPUTER NETWORKS LAB (CS315)

## Assignment-13

### TLS

Date: 28 March 2024



Indian Institute of Technology Dharwad  
भारतीय प्रौद्योगिकी संस्थान धारवाड़

# Transport-layer security (TLS)

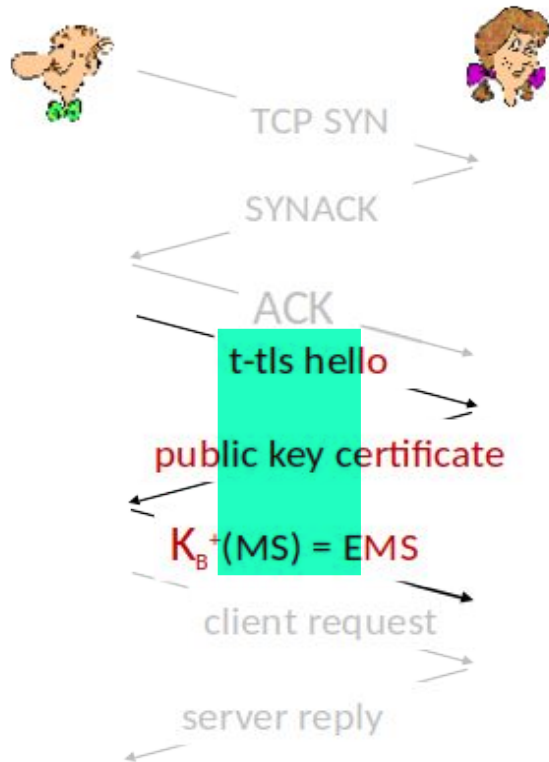
- widely deployed security protocol above the transport layer
  - supported by almost all browsers, web servers: https (port 443)
- provides:
  - confidentiality: via *symmetric encryption*
  - integrity: via *cryptographic hashing*
  - authentication: via *public key cryptography*

} *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

# 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:



# 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

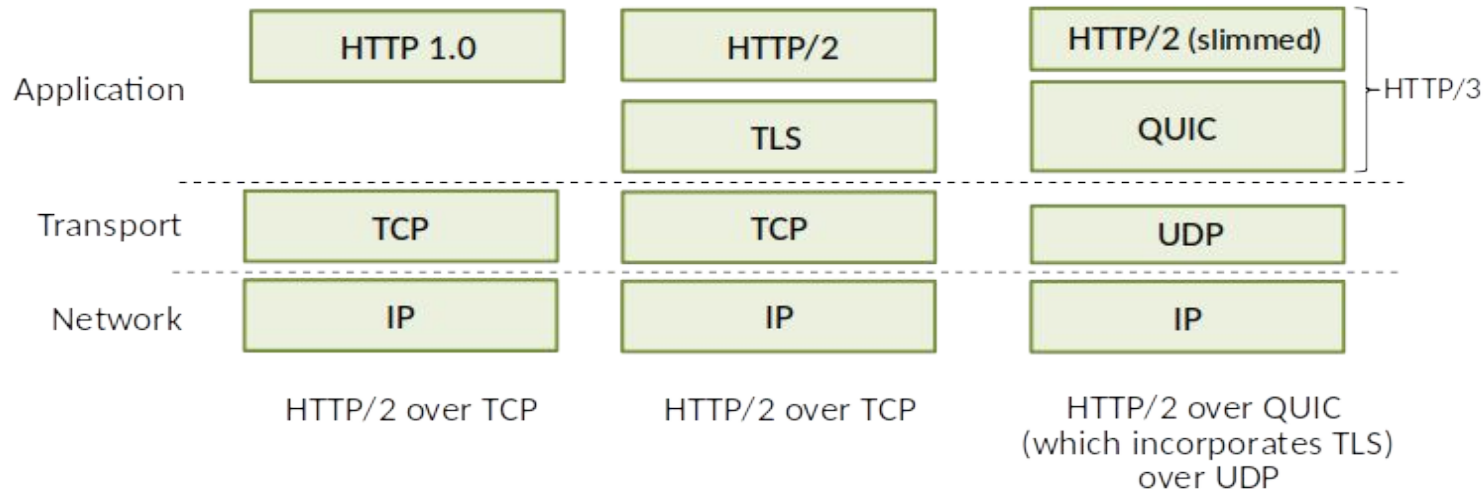
- truncation attack:
  - attacker forges TCP connection close segment
  - one or both sides thinks there is less data than there actually is
- **solution:** record types, with one type for closure
  - type 0 for data; type 1 for close
- MAC now computed using data, type, sequence #

$$K_c \left( \begin{array}{|c|c|c|c|} \hline \text{length} & \text{type} & \text{data} & \text{MAC} \\ \hline \end{array} \right)$$



# Transport-layer security (TLS)

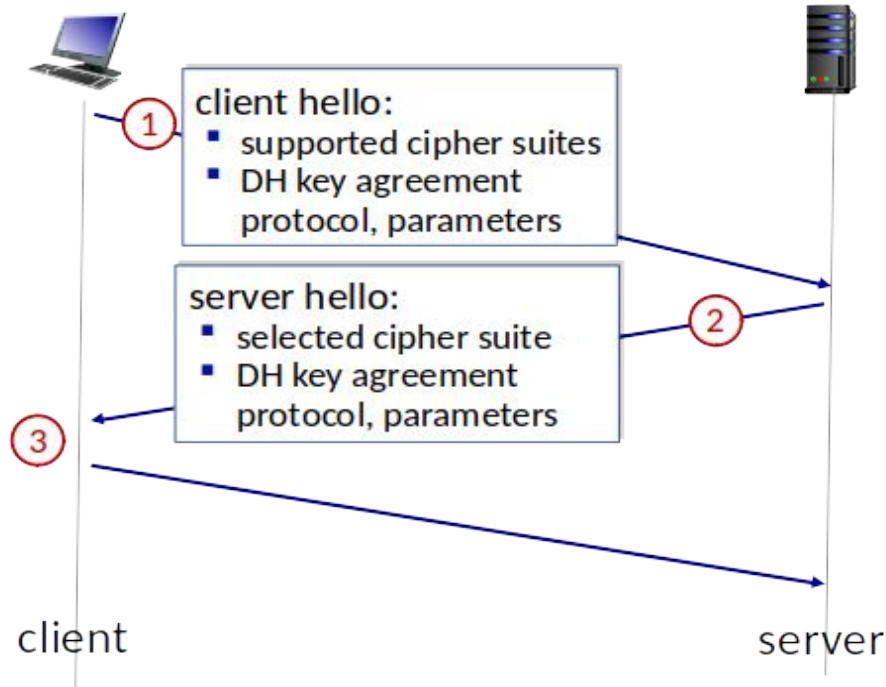
- TLS provides an API that *any* application can use
- an HTTP view of TLS:



# TLS: 1.3 cipher suite

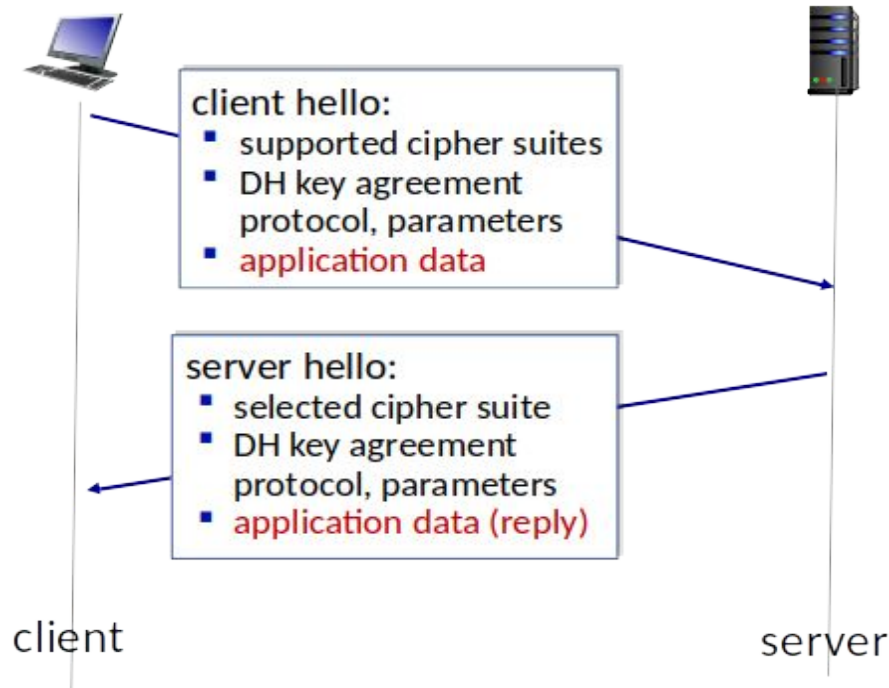
- “cipher suite”: algorithms that can be used for key generation, encryption, MAC, digital signature
- TLS: 1.3 (2018): more limited cipher suite choice than TLS 1.2 (2008)
  - only 5 choices, rather than 37 choices
  - *requires* Diffie-Hellman (DH) for key exchange, rather than DH or RSA
  - combined encryption and authentication algorithm (“authenticated encryption”) for data rather than serial encryption, authentication
    - 4 based on AES
  - HMAC uses SHA (256 or 284) cryptographic hash function

# TLS 1.3 handshake: 1 RTT



- 1** client TLS hello msg:
  - *guesses* key agreement protocol, parameters
  - indicates cipher suites it supports
- 2** server TLS hello msg chooses
  - key agreement protocol, parameters
  - cipher suite
  - server-signed certificate
- 3** client:
  - checks server certificate
  - generates key
  - can now make application request (e.g., HTTPS GET)

# TLS 1.3 handshake: 0 RTT



- initial hello message contains encrypted application data!
  - “resuming” earlier connection between client and server
  - application data encrypted using “resumption master secret” from earlier connection
- vulnerable to replay attacks!
  - maybe OK for get HTTP GET or client requests not modifying server state

# Thank you