Network Security

Thanks to:
J.F Kurose and K.W. Ross, “Computer Networking: A Top-Down Approach, 8th edition, Pearson, 2020. Teaching material.
Open AI, Content structure.

University of Minho, 2025, pmc

Outline

- Introduction
- Network Security Threats and Attacks
- Securing End-to-End Connections
  - Transport Layer Security (TLS)
- Security at Transport Layer: TCP, UDP, QUIC
- Security Network Layer
  - IP/ICMP, IPSec, VPNs, IPv6 Security

Cybersecurity and Network Security

- cybersecurity – "Prevention of damage to, protection of, and restoration of computers, electronic communications systems, electronic communications services, wire communication, and electronic communication, including information contained therein, to ensure its availability, integrity, authentication, confidentiality, and nonrepudiation."

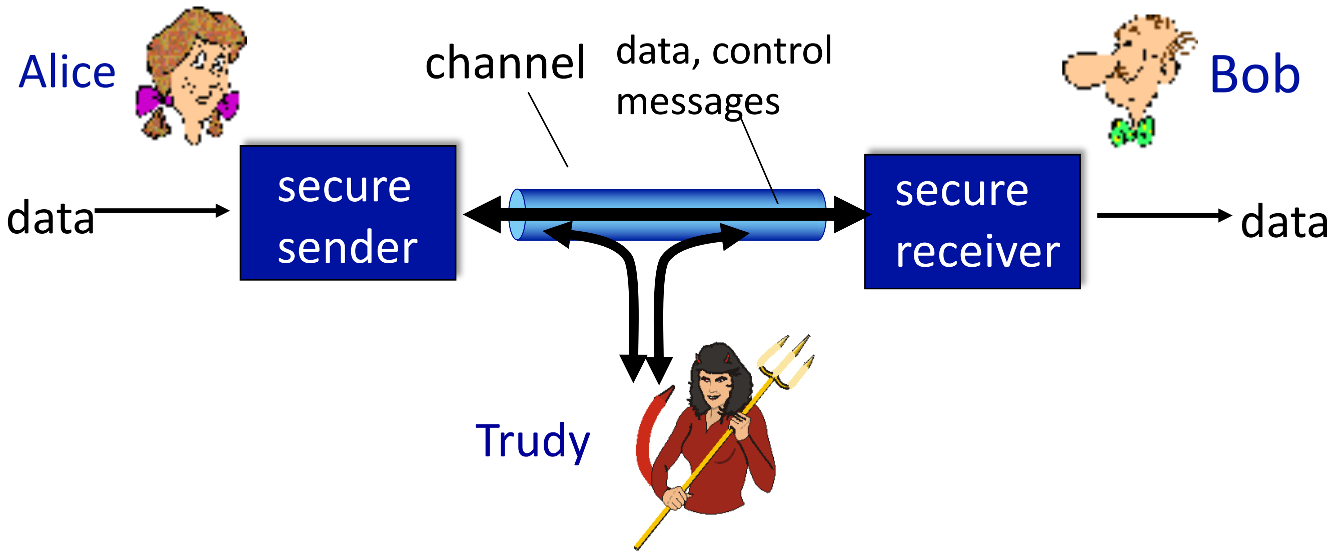
  [NIST, Computer Security Resource Center]

- network security – "Network security is the protection of the underlying networking infrastructure from unauthorized access, misuse, or theft. It involves creating a secure infrastructure for devices, protocols, users, and applications to work in a secure manner."

  [CISCO]

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 service and servers
- BGP service and servers (routers exchanging routing table updates)
- …
- any system or entity in a public or private network

Trudy: there are bad guys (and girls) out there!

Q: What can a "bad guy" do?

A: A lot! (see also section 1.6 of book [Kurose and Ross, 2020])

- eavesdrop: intercept messages
- actively insert messages into connection
- 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
- denial of service: prevent service from being used by others (e.g., by overloading resources)

Network Security

Goal:

- understand network security within TCP/IP protocol stack
  - security at application, transport, network level, link-layer levels

Network Security

- Security needs are transversal to TCP/IP protocol stack

  Recall that:
  - TCP/IP was not originally designed with security concerns in mind
  - application user data may be tampered when in transit
  - transport connections may be reset or hijacked
  - network IP addresses and routing may be spoofed
  - link layer MAC addresses may be spoofed or devices hijacked

  - not enough to focus on a particular functional level
    - protection at higher layers do not eliminate threats at lower ones

Recall (Network) Security Principles

confidentiality: only sender and intended receiver should "understand" the content of messages

- sender encrypts message; receiver decrypts message

integrity of messages: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

authentication: sender, receiver want to confirm identity of each other

access and availability: data and services must be accessible and available to users

no repudiation: ensure that a party (sender or receiver) cannot deny having sent or received a message or transaction.

Type of Network Security Threats

Phishing – deceptive tactics (masquerade as legitim e-mail messages, websites, etc.) to elude unsuspecting users into share sensitive data such as passwords, financial details, or personal data (also SQL injection, malware, etc.)

- A: promote cybersecurity awareness, two-factor authentication

DNS Spoofing – manipulation of domain name system resolution to redirect users to malicious websites or intercept sensitive information

- A: use of DNSSEC, DNS filtering and monitoring

DDoS attacks – distributed attack aiming at disrupting services by overwhelming a target system with traffic, exhausting resources and limiting legitimate users

- A: use of detection, filtering, analytical and blocking techniques; reduce systems exposure

Man-in-the-Middle (MitM) attacks – interception of communication parties, allowing the attacker to eavesdrop, modify, or inject malicious content into the data stream

- A: use of robust encryption standards and network monitoring

DNS spoofing attacks

Methods:

DNS compromise

- attacker hijacks a DNS server and has access to the name resolution system
- able to change resource records (RR) to return malicious data

MitM

- attacker intercepts DNS queries and manipulates DNS server responses with malicious data

Side effect: DNS cache poisoning

- subsequent requests from other users or applications will get malicious data from DNS caching system
- in addition, the attacker may exploit TTL of entries in the DNS cache to ensure that malicious data persist

DDoS attacks

1. Volumetric attacks

SYN flooding

- attacker launches a massive number of TCP connection requests from clients (possibly using false source IP address to hide identity) to a target server
- at server, TCP replies with SYN/ACK and creates state for the connection
- clients don't send ACK, remaining the connection half open, exhausting memory at server
- A: use authentication cookies, and allocate state only if client is legitimate (e.g. SCTP)

IP, ICMP, UDP flooding

- attacker sends a large volume of traffic (e.g., ping) to the IP broadcast addresses of intermediary networks, with the source IP address spoofed (victim's IP address)

DDoS attacks

1. Volumetric attacks (cont.)

Slow flooding

- same principle but spread over a large time window; mixed with legitimate traffic; more difficult to detect

2. Protocol-based attacks

Reflection / Amplification

- attacker exploits vulnerabilities of application protocols (e.g. DNS, NTP, LDAP, SSDP) to launch IP spoofed queries, causing them to send an amplified response to the victim

Charlie Ciso
How do we recognize a DDoS attack?
Trust me, you'll know.
Charlie Ciso
Sorry, this panel is having trouble loading.
OK

DDoS attacks

3. Application-layer attacks

HTTP / HTTPS flooding

- attacker floods the target web server with HTTP / HTTPS requests, consuming its resources and causing service degradation, or keeping multiple long connections open (Slowloris Attack)

Headless-browser DDoS
Botnet IPs: Day 1
by Incapsula.com
North Atlantic Ocean
South Atlantic Ocean
South Pacific Ocean
Indian Ocean

DDoS attacks

3. Application-layer attacks

- other attacks more focused on server/client side (not properly DDoS)
- HTTP/HTTPS sgnostic

SQL injection (SQLi)

- attacker exploits vulnerabilities in web apps to execute malicious SQL queries, which may lead to database overload or data leakage
  - or vulnerabilities of APIs by sending apparently legitimate requests

Cross Site Scripting (XSS)

- attacker injects malicious code into a webpage, stealing cookies or running code (usually JS) in the victim’s browser

MitM attacks

Application layer

- e.g. DNS spoofing, HTTP flooding, downgrade, malicious JS injection

Transport layer

- e.g. truncation (reset), replay, reordering, downgrade

Network layer

- e.g. IP spoofing, ICMP redirects, forge route advertisements

Link layer

- e.g., ARP spoofing, ARP poisoning, rouge/fake AP, DHCP spoofing

Physical layer

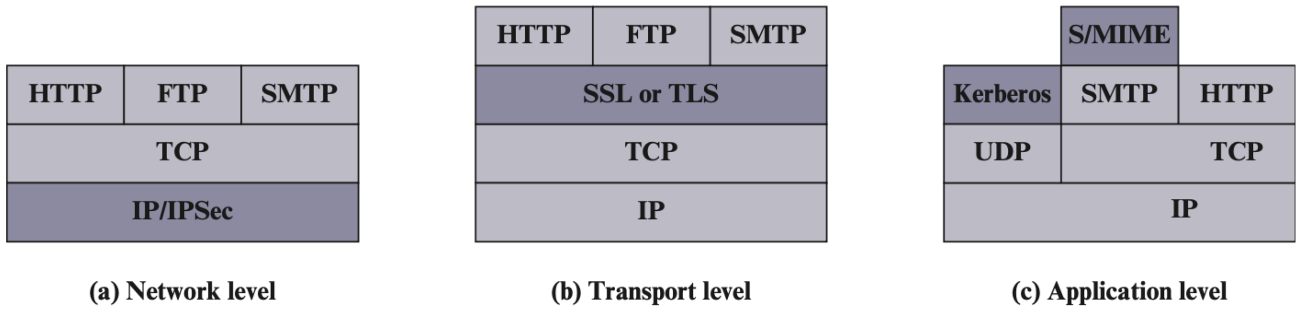
- e.g. jamming, side channel attacks, medium tapping (intercepting cable/radio)

Outline

- Network Security Threads and Attacks
- Securing end-to-end connections
  - Transport Layer Security (TLS)
- Security at Transport Layer
  - TCP, UDP, QUIC
- Security Network Layer
  - IP/ICMP, IPSec, VPNs, IPv6 Security

Securing end-to-end connections

- attacks target application entities (client, server, peer processes etc.) or traffic between/among entities
  - passive: eavesdropping network traffic
  - active: impersonating user, altering data in transit (or in app/website)
- approaches to security in the protocol stack

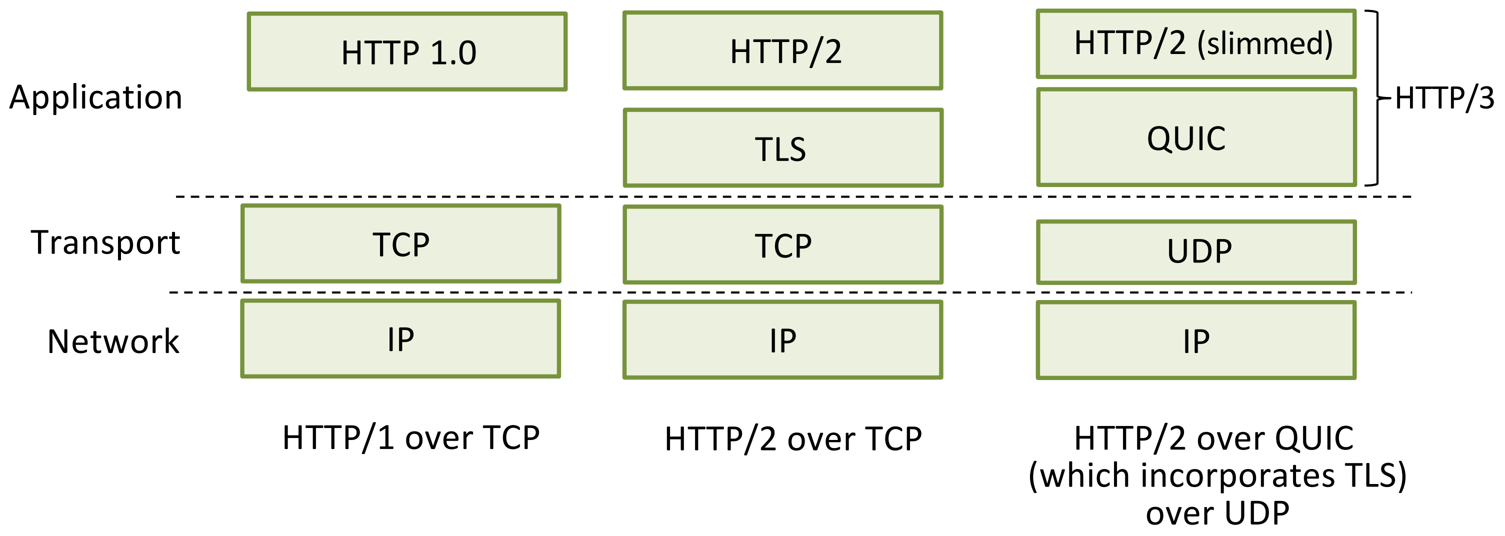


[Stallings, 2023]

(a) general-purpose, transparent L7, IP selective; (b) general purpose, embedded in app or not; (c) app-tailored

Transport-layer security (TLS)

- TLS primary goal is to provide a secure channel between communicating entities or peers
  - TLS primitives* can be accessed through various APIs that *any* application can use
- HTTP view of TLS:



* cryptographic primitives for encryption, authentication, hashing, key exchange, digital signatures

Transport-layer security (TLS)

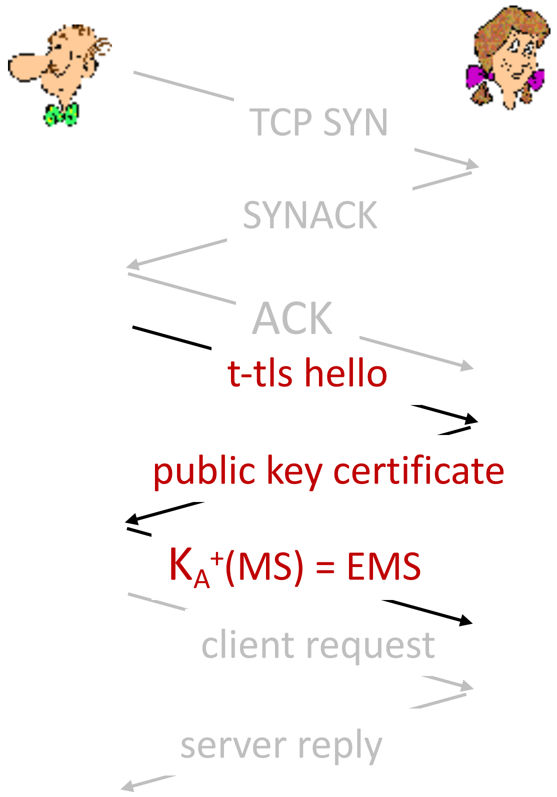
- widely deployed security protocol above the transport layer
  - supported by all recent browsers, web servers: https (port 443)
- provides:
  - confidentiality: via *symmetric encryption*
  - integrity: via *cryptographic hashing*
  - authentication: via *public key cryptography*

  all techniques you have studied!
- history:
  - early research, implementation (90's): secure net programming, secure sockets
  - Secure Socket Layer (SSL 3.0) deprecated [IETF, 2015]
  - TLS 1.3: RFC 8846 [2018] (TLS 1.2 RFC 5246 [2008] widely in use)

Transport-layer security: what's needed?

- let's *build* a toy TLS protocol, *t-tls,* to see what's needed!
- required "pieces":
  - 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 handled as a series of data records
    - not just one-time transaction
  - connection closure: special messages to securely close connection

t-tls: initial handshake



t-tls handshake phase:

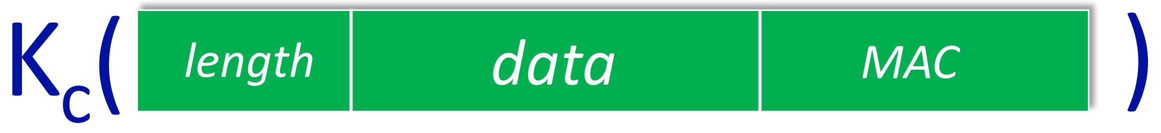
- Bob establishes TCP connection with Alice
- Bob verifies that Alice is really Alice
- Bob sends Alice an encrypted master secret key (EMS)
- Alice uses K_A^- (EMS) to obtain MS and generate all other keys for TLS session
- potential issues:
  - 3 RTT before client can start receiving data (including TCP handshake)

t-tls: cryptographic keys

- it's bad practice to use same key for more than one cryptographic function (important to separate contexts!)
  - 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 (for integrity)
  - K_s : encryption key for data sent from server to client
  - M_s : MAC key for data sent from server to client
- keys are derived from a Key Derivation Function (KDF)
  - takes master secret and 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 data 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, $K_{c,}$ is passed to TCP:



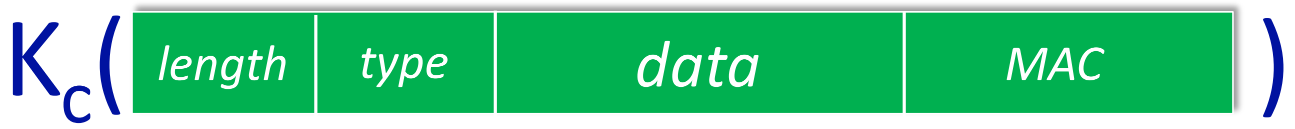
t-tls: encrypting data (more)

- possible attacks on data stream?
  - *re-ordering:* man-in-the-middle intercepts TCP segments and reorders (manipulating sequence #s in unencrypted TCP header)
  - *replay*
- solutions:
  - use local TLS sequence numbers for sent and received records (data, implicit TLS-seq-# incorporated into MAC calculation)
    - e.g. HMAC_hash (M_c, length || seq_num ||data (plaintext))
  - use nonce in symmetric cipher to guarantee unique encryption per record

(note: in TLS, as nonces can be derived from seq_# and IV (Initialization Vector), replay/reuse is prevented as long as seq_# never repeats under same IV/key)

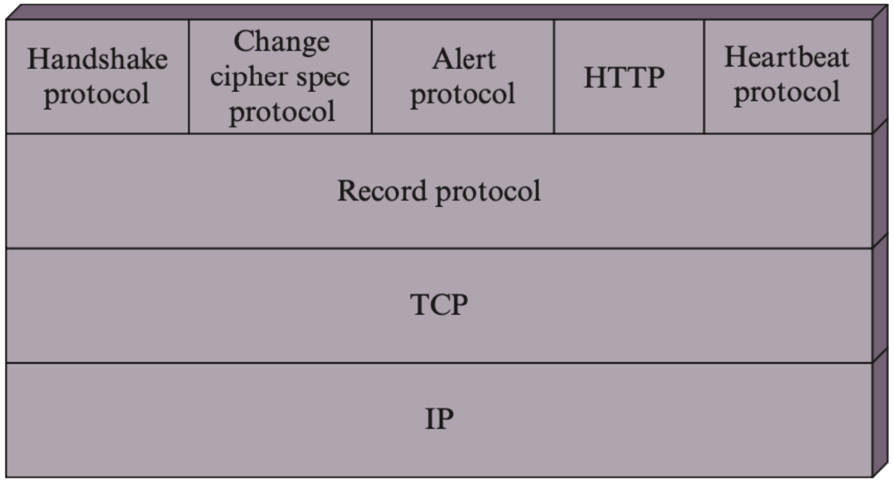
t-tls: connection close

- truncation attack:
  - attacker forges TCP connection close segment (FIN, RST)
  - one or both sides thinks there is less data than there actually is
- solution: use of record types, with one type for closure
  - type 0 for data; type 1 for close
- MAC now computed using length, type, data, sequence #
  - e.g. HMAC_hash (M_c, length || type || seq_num ||data (plaintext))



Transport Layer Security (TLS)

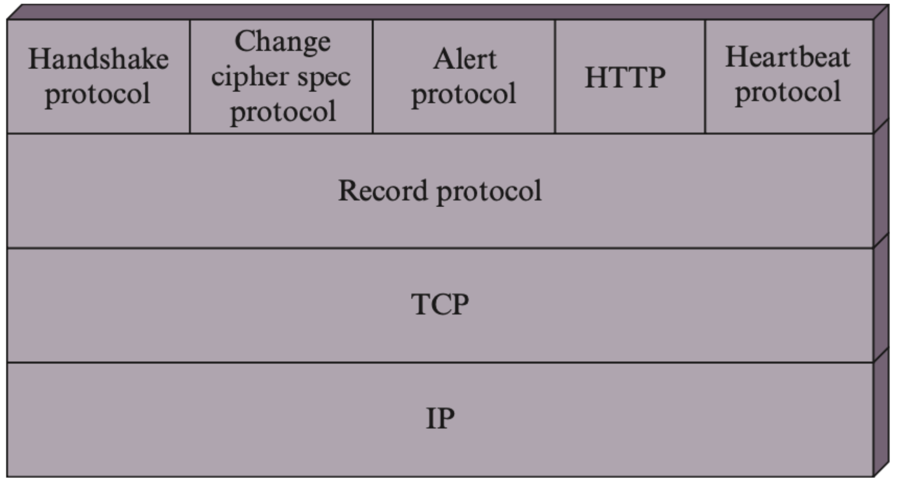
- TLS is a set of layered protocols
  - Handshaking
    - negotiate and establish a secure session
  - Change cipher spec
    - change to new encryption
  - Alert
    - report notification of problems
  - Application
    - send application-layer data, e.g. via HTTP
  - Heartbeat
    - keep alive session sensing
  - Record
    - package and secure application data



[Stallings, 2023]

Transport Layer Security (TLS)

- TLS concepts:
  - Session
    - association between client, server
    - created by TLS Handshake Protocol
    - define a set of cryptographic security parameters to be used by connections
  - Connection
    - end-to-end temporary transport path between entites
    - each connection is associated with a session



[Stallings, 2023]

Transport Layer Security (TLS) – TLS 1.2

- Session state
  - Session ID – ID set by server for the session (active, resumable)
    Session Ticket – encrypted unit with all session state sent to client
  - Peer certificate – ITU-T X.509 v3 certificate (may be null)
  - Compression method – to be used before encryption (mostly null)
  - Cipher spec – defines the bulk data encryption algorithm (e.g. AES), a hash algorithm used for MAC calculation, other attributes (e.g. hash size)
  - Master Secret (MS) – secret shared between parties (48 bytes)
    - derived locally from C-S randoms, Pre MS (PMS), and used in a KDF (Key Derivation Function) to obtain the session keys

Transport Layer Security (TLS) – TLS 1.2

- Connection state
  - C-S Randoms – 32-bit nonces sent in Hello messages to use in KDF
  - MAC Keys – for integrity checks (e.g., HMAC)
  - Session Keys – r/w keys for the connection
  - Cipher Suite – negotiated, for client and server
  - Initialization Vector (IV) – maintained for each key
    - initiated during handshaking, then the last ciphertext block of each record is used as IV, for randomness
  - Sequence Numbers – separate seq #s for sending/receiving (for data ordering integrity, and replay avoidance)
    - initiated when a "change cipher spec" message is sent, received
    - incremented monotonically per record sent, received

Transport Layer Security (TLS) – TLS 1.3

- Session state
  - state management is simplified at client and server, namely:

  - Pre-Shared Key (PSK) – instead of using session IDs or session tickets within the initial handshaking, PSK is introduced for fast session resumption
    - when a client establishes a connection, the server issues a session ticket that contains the PSK
    - PSK is used for session resumption without full handshaking
    - only the PSK and some session-related data (e.g., cipher suite) are stored by the client
    - minimal state is required by the server, typically just the PSK decryption key

Transport Layer Security (TLS) – TLS 1.3

- Connection state
  - Session Keys – uses more secure key derivation and ephemeral key exchanges so that new set of keys is generated for every connection
  - Cipher Suite – only supports strong, modern cipher suites (e.g., `TLS_AES_128_GCM_SHA256`, five in total)
  - Finished Message – to ensure that both parties agree on the negotiated session parameters (handshake validation)
  - Initialization Vector (IV) – not negotiated, is generated internally as part of the Authenticated Encryption with Associated Data (AEAD) encryption process
  - Sequence Numbers – each record uses seq #, always implicit, never transmitted

Transport Layer Security (TLS) – TLS 1.2 vs 1.3

Aspect	**TLS 1.2**	**TLS 1.3**
IV Management	IVs are explicitly negotiated or transmitted alongside encrypted data	IVs are derived from the handshake key and sequence number, handled internally
Sequence Numbers	Separate read/write sequence numbers are maintained and explicitly managed	Sequence numbers are implicit, combined with IV for AEAD encryption
Encryption Algorithm	Both block ciphers (with CBC) and AEAD ciphers can be used	Only AEAD ciphers (e.g., AES-GCM, ChaCha20) are allowed
Complexity	More complex due to separate management of IVs, sequence numbers, and cipher modes	Simplified with AEAD handling encryption, authentication, IVs, and sequence numbers

In summary:

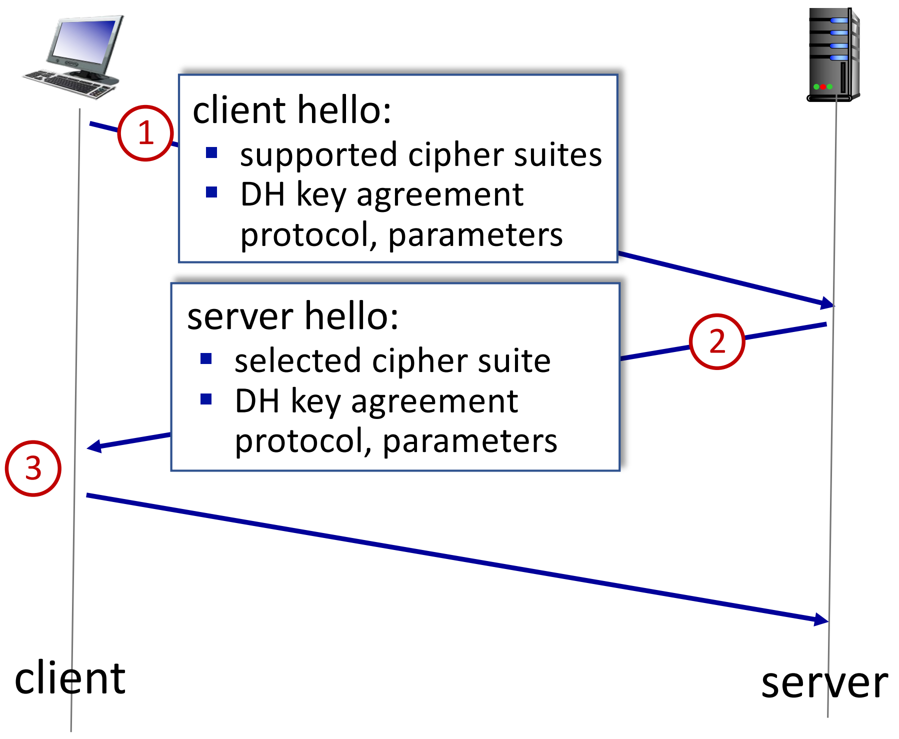
- TLS 1.3 simplifies the use of IVs and sequence numbers by incorporating them into the AEAD cipher process. The sequence #s are used to derive IVs internally, eliminating the need to send or explicitly manage them
- TLS 1.3 reduces the complexity of session and connection state and improves both security and performance

From TLS 1.2 to TLS 1.3

1. Handshake Process and Performance:

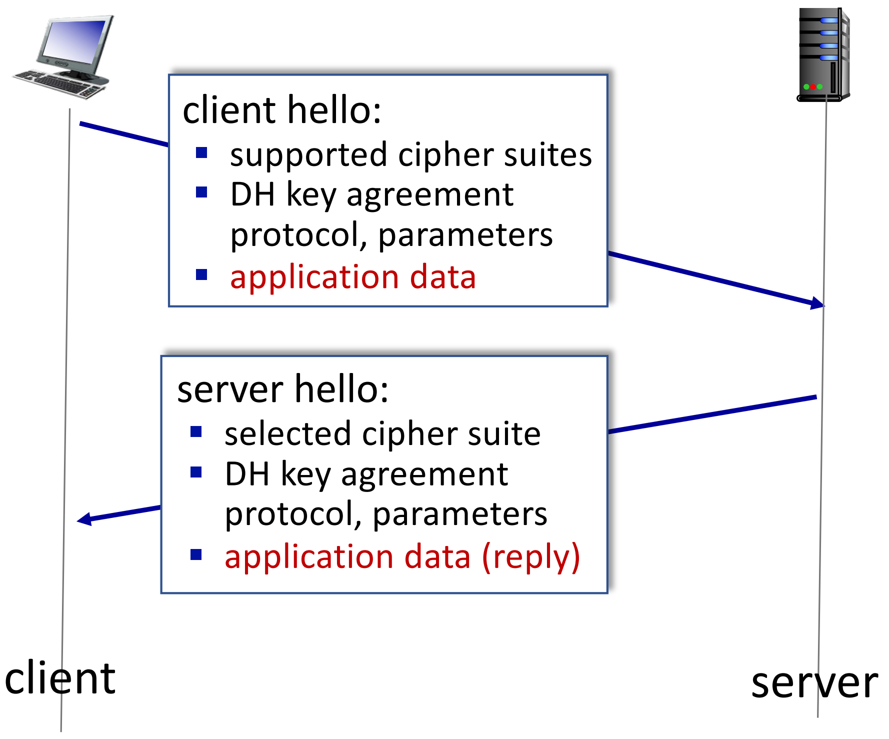
- TLS 1.2:
  - involves a complex handshake with multiple round trips between C and S
  - the full handshake requires 2-RTTs (client hello, server hello, key exchange)
  - supports resuming a session through session IDs or session tickets
  - clients and servers negotiate both the key exchange algorithm (e.g., Diffie-Hellman) and the encryption algorithm (e.g., AES, ChaCha20)
- TLS 1.3:
  - simplifies the handshake process, reducing it to just 1-RTT for full handshakes
  - when resuming a session, it can perform a "0-RTT" handshake, allowing encrypted data to be sent immediately after the client hello, drastically improving performance for repeated connections
  - simplifies key negotiation, reducing the overhead and improving connection speeds

TLS 1.3 handshake: 1 RTT



1. client TLS hello msg:
   - proposes 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 (from master key (PSK) held in server´s cache)
- vulnerable to replay attacks!
  - maybe OK for HTTP GET or client requests not modifying server state
  - application logic must solve

From TLS 1.2 to TLS 1.3

2. Supported Cipher Suites:

- TLS 1.2:
  - supports a wide range of cryptographic algorithms, but not all are secure… e.g., weak algorithms such as RC4 and MD5, currently deprecated
  - allows negotiating both the cipher suite and key exchange method separately, which can lead to security issues if insecure combinations are chosen
  - uses block ciphers (e.g., AES in CBC mode) and stream ciphers (e.g., RC4)
- TLS 1.3:
  - removed many older, insecure algorithms and cipher suites, including:
    - RSA key exchange; RC4, DES, 3DES, and other weak ciphers; MD5 and SHA-1 as hash functions.
  - uses only secure, modern cryptographic algorithms, e.g. AES-GCM, ChaCha20-Poly1305
  - combines key exchange and cipher suite negotiation into a single step, using AEAD (Authenticated Encryption with Associated Data) ciphers, which improve both security and performance

From TLS 1.2 to TLS 1.3

3. Key Exchange Algorithms:

- TLS 1.2:
  - supports both RSA and Diffie-Hellman for key exchange
  - RSA-based key exchange is less secure and vulnerable to certain types of attacks (e.g., forward secrecy is not guaranteed with RSA)
  - Forward Secrecy is optional; therefore, past sessions data may be at risk if server private key is later compromised
- TLS 1.3:
  - TLS 1.3 only supports Diffie-Hellman (DH) and Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange, ensuring Perfect Forward Secrecy (PFS) in every session
  - Forward Secrecy is always enforced, meaning even if long-term keys are compromised, past communication remains secure
  - RSA key exchange has been completely removed

From TLS 1.2 to TLS 1.3

4. Forward Secrecy:

- TLS 1.2:
  - Forward Secrecy (FS) is optional and depends on whether ephemeral key exchange protocol such as ECDHE is used
  - some cipher suites don't offer FS, making it possible for attackers to decrypt past communications if the private key was compromised
- TLS 1.3:
  - Forward Secrecy is mandatory. All sessions use ephemeral key exchanges (such as ECDHE), ensuring that past communications cannot be decrypted even if the private key is compromised

From TLS 1.2 to TLS 1.3

5. Security Enhancements:

- TLS 1.2:
  - vulnerable to several types of attacks, including downgrade attacks (e.g. Logjam, BEAST, FREAK), which exploited weaknesses in older cipher suites and protocols
  - cipher suite negotiation could allow clients and servers to fall back to weak or deprecated cryptographic algorithms
- TLS 1.3:
  - many legacy features and insecure cryptographic elements were removed
  - resists to downgrade attacks, as it no longer allows clients and servers to negotiate weaker cryptographic options
  - simplifies the protocol, reducing the attack surface and improving security
  - removed static RSA and Diffie-Hellman key exchanges, which were vulnerable to passive attacks

From TLS 1.2 to TLS 1.3

6. Session Resumption:

- TLS 1.2:
  - can be done using session IDs or session tickets, but both methods have limitations, such as requiring multiple round trips and lacking forward secrecy
- TLS 1.3:
  - is handled more efficiently with PSK (Pre-Shared Key) resumption, allowing faster reconnections and optional 0-RTT resumption (though 0-RTT has its own security trade-offs, such as being “replayable”)
  - even session resumption supports forward secrecy

From TLS 1.2 to TLS 1.3

7. Simplification of Protocol:

- TLS 1.2:
  - more complex, with support for a wide range of cryptographic algorithms, cipher suites, and key exchange methods, which add overhead and complexity in deployment and configuration
  - protocol complexity contributes to configuration errors and potential security issues
- TLS 1.3:
  - protocol is simplified by removing older, less secure features and reducing the set of supported cryptographic primitives
  - easier to implement, reducing the chance of misconfiguration or security weaknesses due to legacy settings

From TLS 1.2 to TLS 1.3

8. Privacy Improvements:

- TLS 1.2:
  - limited privacy enhancements, as parts of the handshake (such as the Server Name Indication or SNI) are sent in plaintext
  - attackers may use the initial handshake information to gather data about the server's configuration (e.g., TLS version, cipher suites, server certificate and random)
- TLS 1.3:
  - introduces Encrypted SNI (ESNI), which encrypts the Server Name Indication, to protect the destination server's identity during the handshake
  - improves privacy by encrypting more of the handshake, including key exchange messages

From TLS 1.2 to TLS 1.3

Feature	TLS 1.2	TLS 1.3
Handshake	2-round trip full handshake	1-round trip full handshake, 0-RTT resumption
Key Exchange	RSA, DH, ECDHE (FS optional)	Only ECDHE (FS mandatory)
Encryption Algorithms	AES, RC4, DES, 3DES, etc.	Only secure options like AES-GCM, ChaCha20
Forward Secrecy	Optional, depends on key exchange	Always enforced
Cipher Suites	Wide variety, including weak ones	Simplified, only secure cipher suites
Session Resumption	Session IDs, tickets	PSK, 0-RTT resumption
Protocol Complexity	Complex, supports many legacy features	Simplified, removes insecure/legacy elements
Privacy Features	Minimal (e.g., SNI sent in plaintext)	Encrypted SNI, more handshake privacy
Vulnerabilities	Susceptible to downgrade and other attacks	Resists downgrade and protocol weaknesses

- Improvements in terms of security, performance, simplicity

Transport Layer Security: beyond TLS

- TLS is a unicast protocol

What if end-to-end is multiparty?

Current approaches (ongoing works with similar goals…)

- Group TLS (gTLS) – an extension of TLS for group communication (exp.)
- Multicast Security Protocols – GDOI; mTLS provide secure multicast comm.
  - Group Domain of Interpretation (GDOI) part of Multicast Security WG (MSEC)
  - Multicast TLS (mTLS) – extension of TLS
- Secure Group Messaging (SGM)
  - Messaging Layer Security (MLS) – IETF proposal for secure group messaging with support for large, dynamic groups
- Key Distribution and Management for Multiparty Communication
  - Diffie-Hellman Group Key Exchange (DH-GKE); Tree-based Group Key Exchange (e.g., TreeKEM)

Transport Layer Security: beyond TLS

Current alternatives to TLS:

- DTLS (Datagram Transport Layer Security)
  - protocol based on TLS 1.3 (no order protection), designed for UDP
  - provides encryption and authentication for applications that use unreliable transport like VoIP or VPNs
  - RFC 9147, April 2022
- QUIC
  - recent transport protocol, developed by Google
  - runs over UDP (to ease middlebox traversal)
  - incorporates many security features from TLS 1.3, offering built-in encryption and faster connection establishment
  - RFC 9000, May 2021

Transport Layer Security: beyond TLS

- Signal Protocol
  - End-to-End Encryption:
    - while TLS focuses on securing transport channels (point-to-point), the Signal Protocol emphasizes end-to-end encryption (no intermediaries such as a third part proxy)
  - Perfect Forward Secrecy and Deniability:
    - also present in TLS 1.3
    - adds deniability to allow participants to plausibly deny that they sent specific messages, which adds a layer of privacy (leave no traceable cryptographic evidence)
  - Double Ratchet Mechanism
    - to continuously refresh encryption keys after every message, making it resilient to compromised keys over time
    - TLS uses a more static key agreement system (with occasional key renegotiation)
  - Asynchronous Messaging
    - useful for modern messaging apps where participants may not be online simultaneously

Transport Layer Security

- Assuming that TLS is in place,

  do we still need to concern about transport layer security?

- Clearly, *yes!*