A.D. 1308
unipg
DIPARTIMENTO
DI MATEMATICA E INFORMATICA

Security in Computer Networks and TLS/SSL

Corso di Introduzione alla Sicurezza Informatica

Chiara Luchini

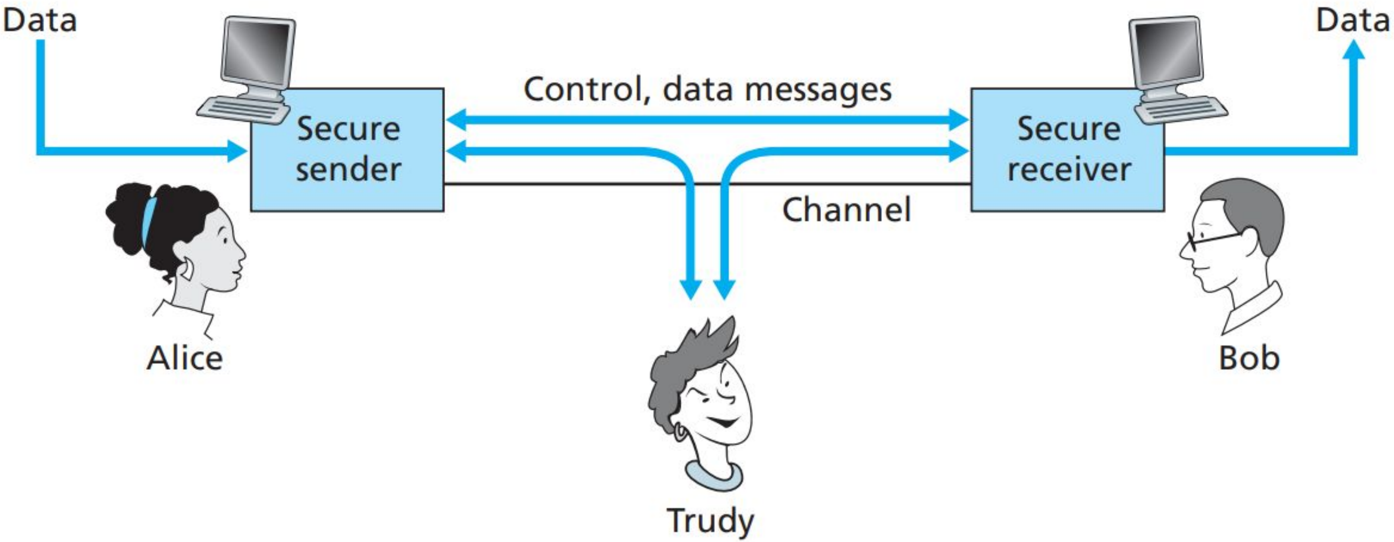
Anno Accademico 2023-2024

Network Security

Properties of Network Security

- **confidentiality**: only sender, intended receiver should "understand" message contents sender encrypts message receiver decrypts message.
- **authentication**: sender, receiver want to confirm identity of each other.
- **message integrity**: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection.
- **access and availability**: services must be accessible and available to users.

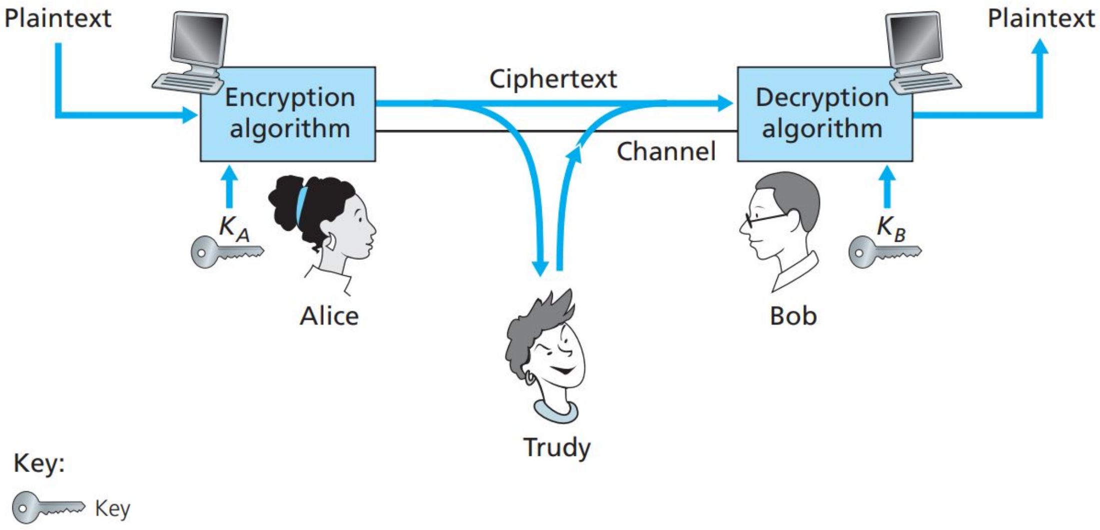
Alice, Bob and Trudy



Principles of Cryptography

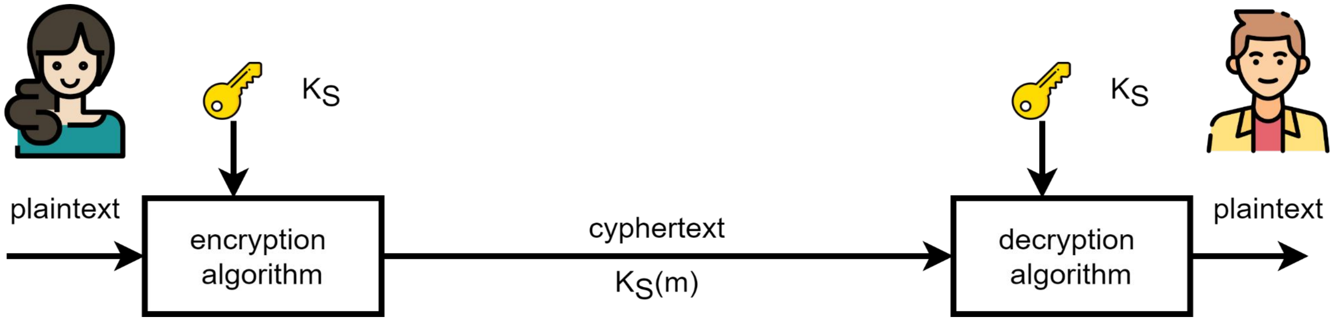
Cryptography components

- **m**= plaintext message
- $\mathbf{K_A(m)}$= ciphertext, encrypted with key K_A
- **m**= K_B $(K_A(m))$



Symmetric Key Cryptography

- **Symmetric key crypto:** Bob and Alice share same (symmetric) key (**DES**, **AES**)



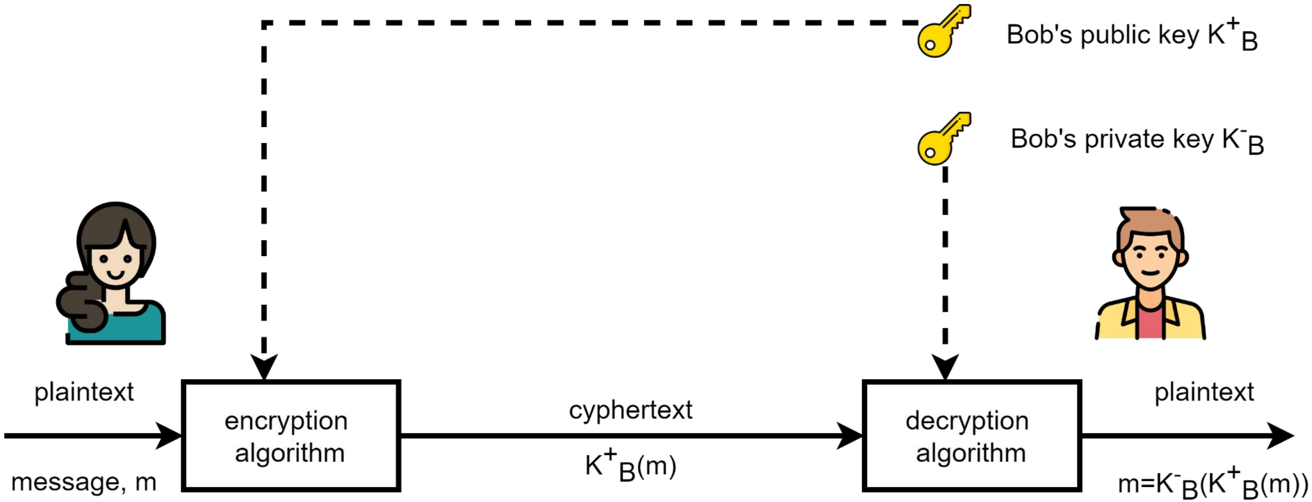

```
plaintext:   abcdefghijklmnopqrstuvwxyz

ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

e.g.: `Plaintext: bob. i love you. alice`

Public Key Cryptography

- Radically different approach (**Diffie-Hellman**, **RSA**)
- Sender, receiver **do** not share secret key
- **Public** encryption key known to **all**
- **Private** decryption key known only to **receiver**

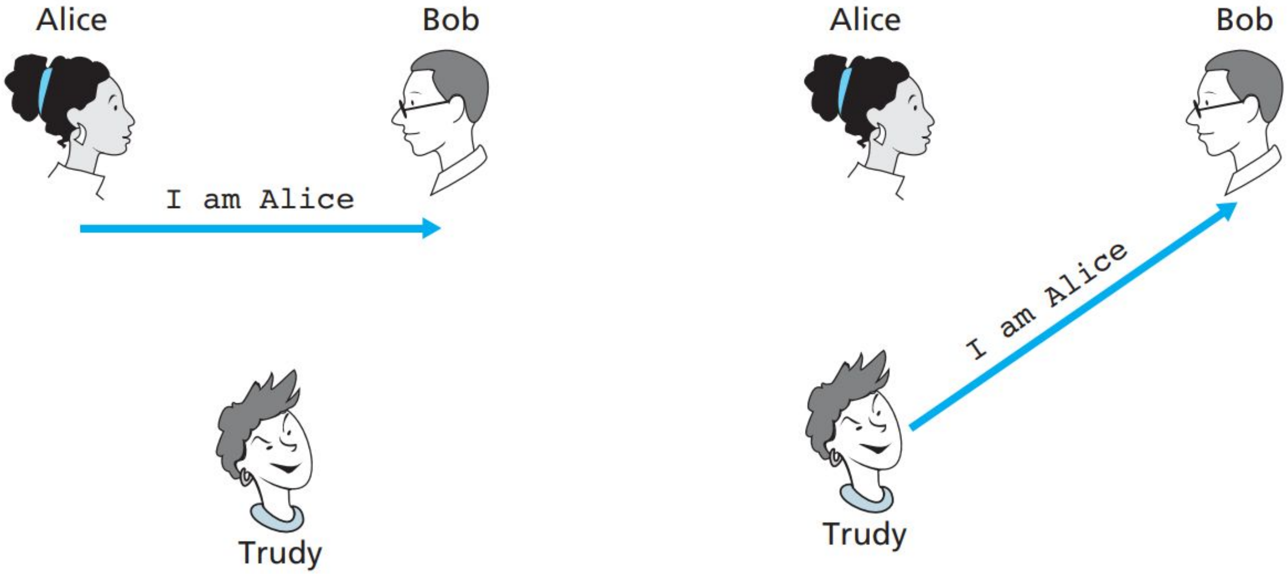


Authentication

Authentication

Goal: Bob wants Alice to “prove” her identity to him

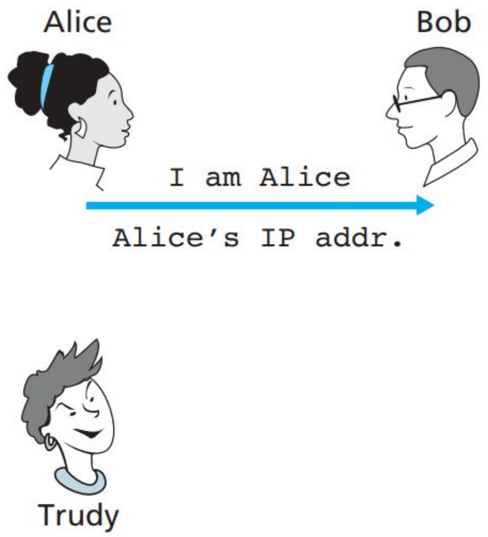
Protocol ap1.0: Alice says “I am Alice”

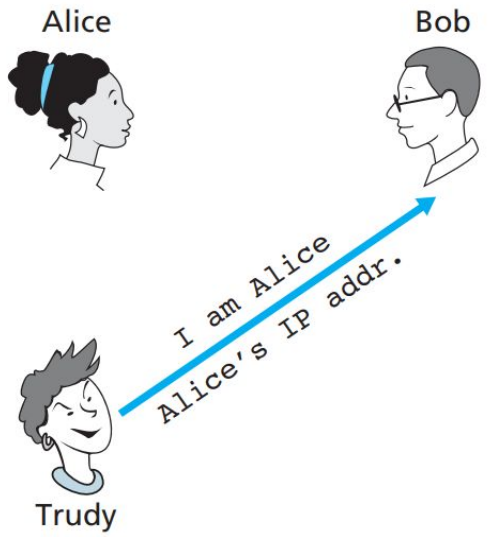


Authentication

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address

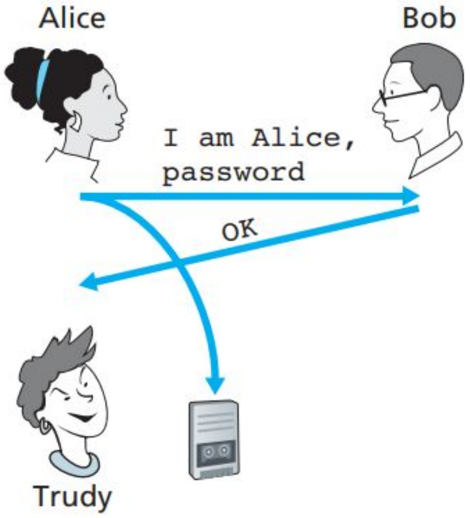


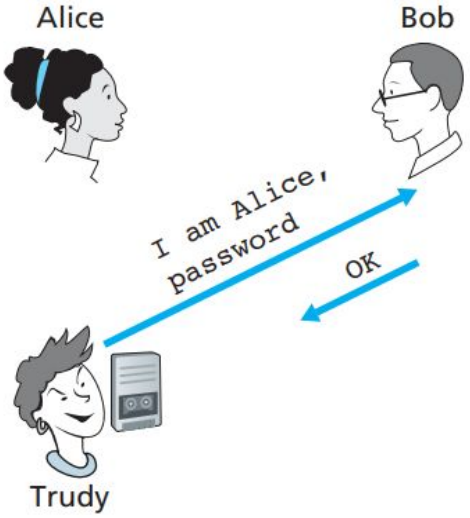


Authentication

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap3.0: Alice says “I am Alice” and sends her encrypted secret password to “prove” it.





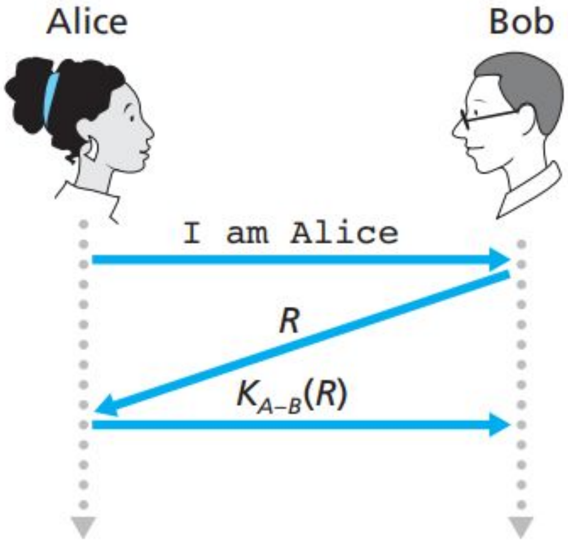
Authentication: symmetric key

Goal: avoid playback attack

nonce: number (R) used only once-in-a-lifetime

Protocol ap4.0: to prove Alice “live”, Bob sends Alice nonce, R

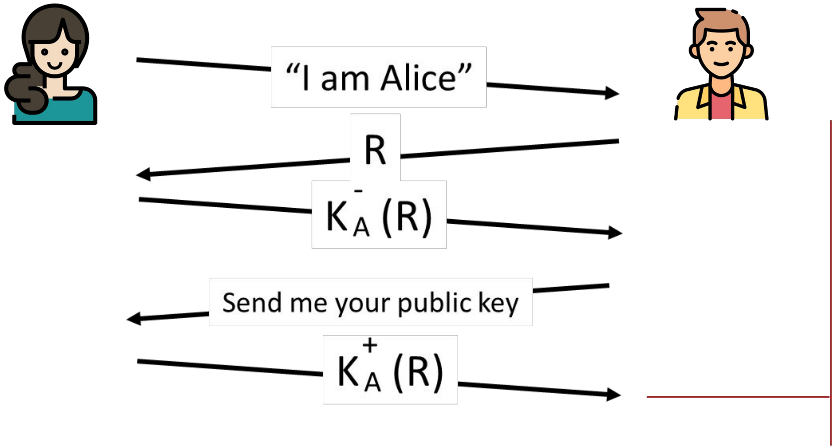
- Alice must return R, encrypted with shared secret key



Authentication: public key

Can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



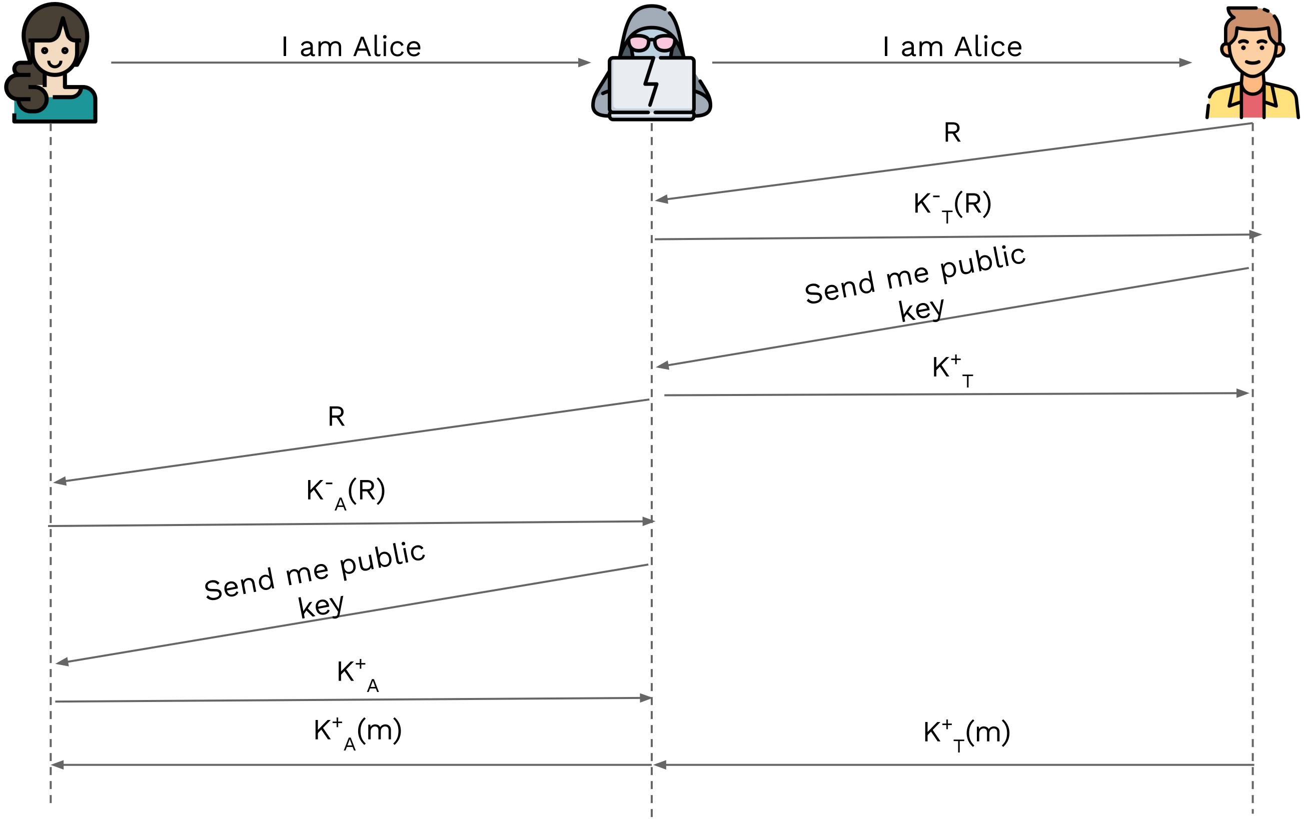
Bob computes

$K_A^+(K_A^-(R)) = R$

and knows only Alice could have the private key, that encrypted R such that

$K_A^+(K_A^-(R)) = R$

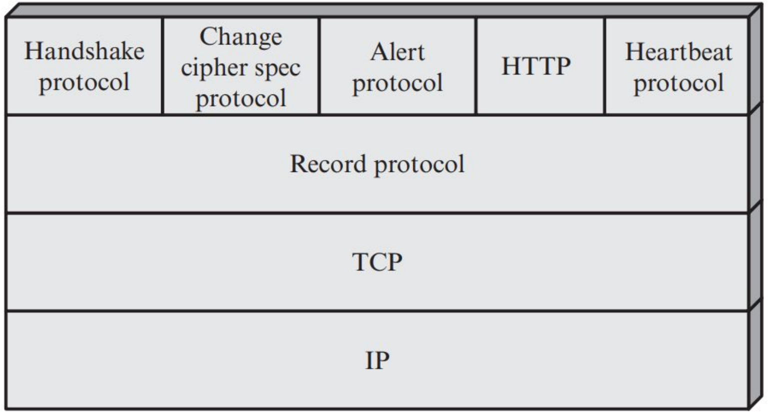
Man in the middle



Transport-layer security (TLS)

Transport-layer security (TLS)

- TLS is an **Internet standard** that evolved from a commercial protocol known as **Secure Sockets Layer (SSL).** Widely deployed security protocol above the transport layer (TCP)
  - supported by almost all browsers, web servers: https (port 443)
- provides:
  - **confidentiality**: via symmetric encryption
  - **integrity**: via cryptographic hashing
  - **authentication**: via public key cryptography



TLS Version

SSL and TLS protocols

Protocol	Published	Status
SSL 1.0	Unpublished	Unpublished
SSL 2.0	1995	Deprecated in 2011 (RFC 6176)
SSL 3.0	1996	Deprecated in 2015 (RFC 7568)
TLS 1.0	1999	Deprecated in 2021 (RFC 8996)[20][21][22]
TLS 1.1	2006	Deprecated in 2021 (RFC 8996)[20][21][22]
TLS 1.2	2008	In use since 2008[23][24]
TLS 1.3	2018	In use since 2018[24][25]

TLS ARCHITECTURE

TLS Connection

- A **transport** that provides a suitable **type of service**
- For TLS such connections are **peer-to-peer relationships**
- Connections are **transient**
- Every connection is associated with **one session**

TLS Session

- An **association** between a **client** and a **server**
- Created by the **Handshake Protocol**
- Define a **set of cryptographic security parameters** which can be shared among multiple connections
- Are used to **avoid** the **expensive negotiation** of new security parameters for each connection

Session state parameters

Session Identifier	An arbitrary byte sequence chosen by the server to identify an active or resumable session state
Peer Certificate	An X509.v3 certificate of the peer; this element of the state may be null
Compression method	The algorithm used to compress data prior to encryption
Cipher spec	Specifies the bulk data encryption algorithm and a hash algorithm used for MAC calculation
Master secret	48-byte secret shared between the client and the server
Is resumable	A flag indicating whether the session can be used to initiate new connections

Connection state parameters

Server and client random	Byte sequences that are chosen by the server and client for each connection
Server with MAC secret	The secret key used in MAC operations on data sent by the server
Client writes MAC secret	The secret key used in MAC operations on data sent by the client
Server writes key	The secret encryption key for data encrypted by the server and decrypted by the client
Client writes key	The symmetric encryption key for data encrypted by the client and decrypted by the server
Initialization vectors	When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol
Sequence numbers	Each party maintains separate sequence numbers for transmitted and received messages for each connection

TLS Record Protocol

The TLS Record Protocol provides **two services** for TLS connections

Confidentiality

The Handshake Protocol defines a **shared secret key** that is used for conventional **encryption** of **TLS payloads**

Message Integrity

The Handshake Protocol also defines a **shared secret key** that is used to form a **message authentication code (MAC)**

TLS Record Protocol Operation

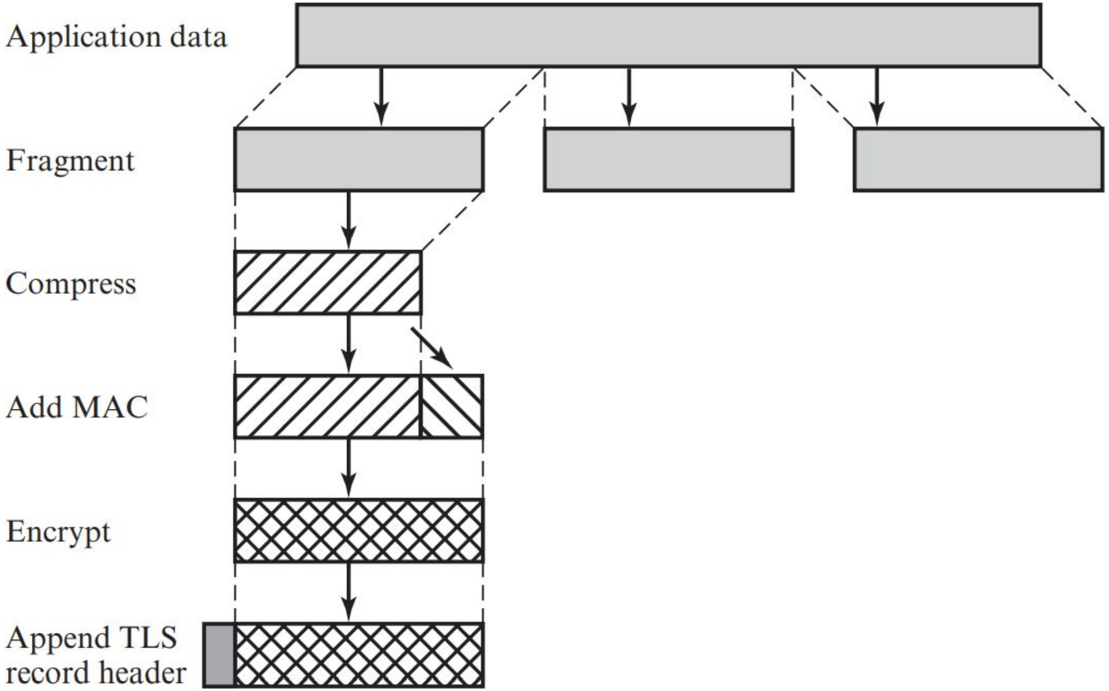
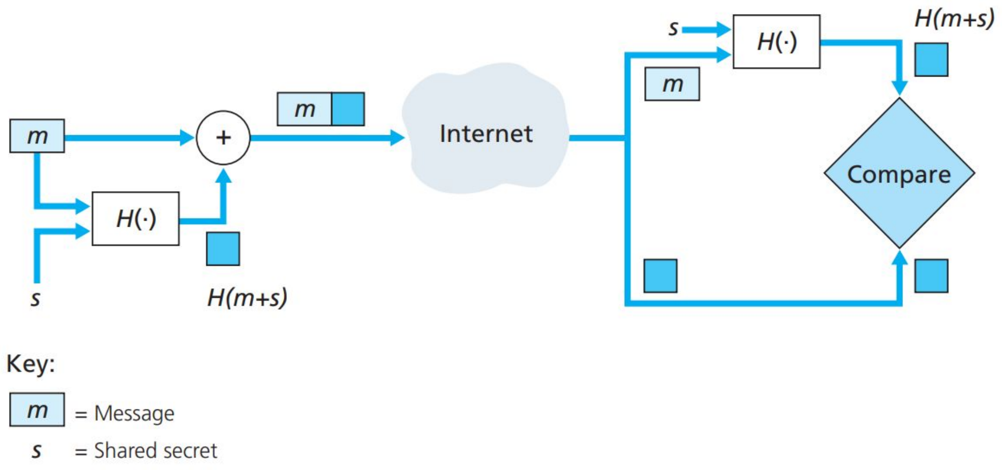


Figure 6.3 TLS Record Protocol Operation

Message Authentication Code (MAC)



- Alice creates message **m**, concatenates **s** with **m** to create **m** + **s**, and calculates the **hash *H*(m + s)**. ***H*(m + s)** is called the **message authentication code (MAC)**
- Alice creates an extended message **(m, *H*(m + s))** and sends it to Bob
- Bob the message **(m, h)** and knowing **s**, calculates the MAC ***H*(m + s)**. If ***H*(m + s)** = **h**, Bob concludes that everything is fine.

TLS Record Protocol: Record Header

- **Content Type (8 bits)**: the higher-layer protocol used to process the enclosed fragment.
- **Major Version (8 bits)**: indicate the major version of TLS in use. For TLSv2, the value is 3.
- **Minor Version (8 bits):** indicate minor version in use. For TLSv2, the value is 3.
- **Compressed Length (16 bits)**: the length in bytes of the plaintext fragment (or compressed). Maximum value 2^{14} + 2048.

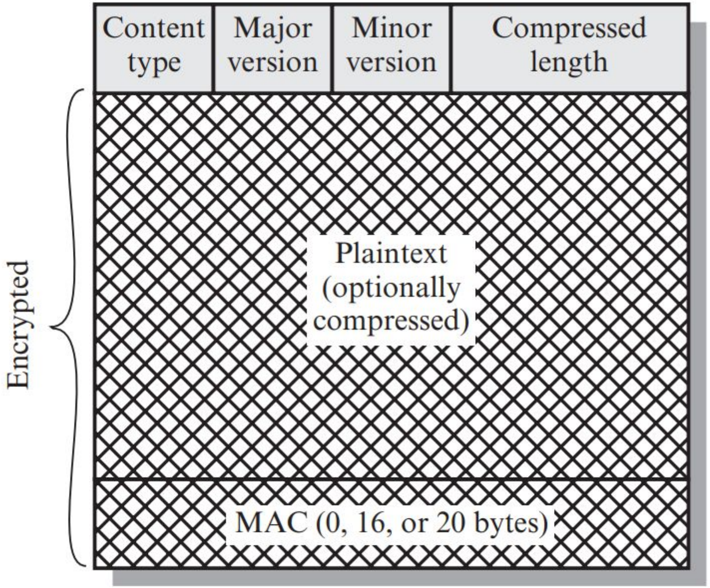


Figure 6.4 TLS Record Format

Content types: Change Cipher and Alert

- **change_cipher_spec**: TLS specific protocol, it consists of a **single byte message** with value 1. The sole purpose is to **update** the **cipher suite** to be used on this connection.
- **alert**: is used to convey **TLS-related alerts** to the peer entity. Each message in this protocol consists of **two bytes.** The first byte takes the value **warning (1)** or **fatal (2)** to convey the severity of the message. The **second byte** contains a code that indicates the **specific alert**.

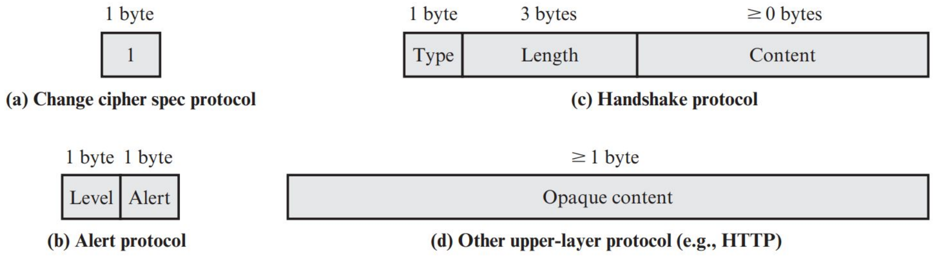


Figure 6.5 TLS Record Protocol Payload

Content types: Handshake Protocol

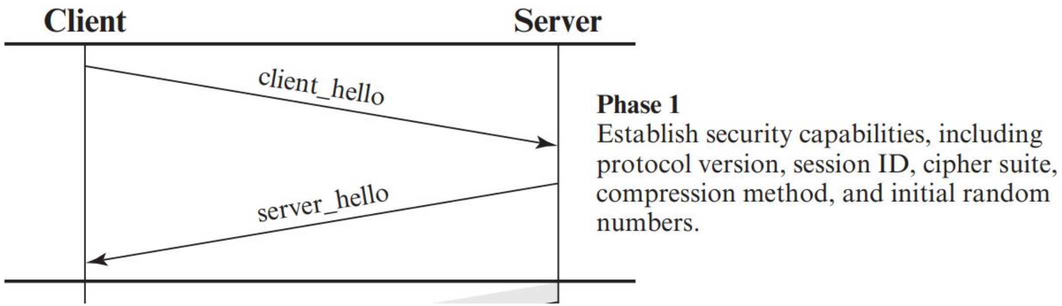
Table 6.2 TLS Handshake Protocol Message Types

Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

The most complex part of TLS is the **Handshake Protocol** . This protocol allows the server and client to **authenticate** each other and to **negotiate an encryption** and **MAC algorithm** and **cryptographic keys** to be used to protect data sent in a TLS record. Each message has 3 field:

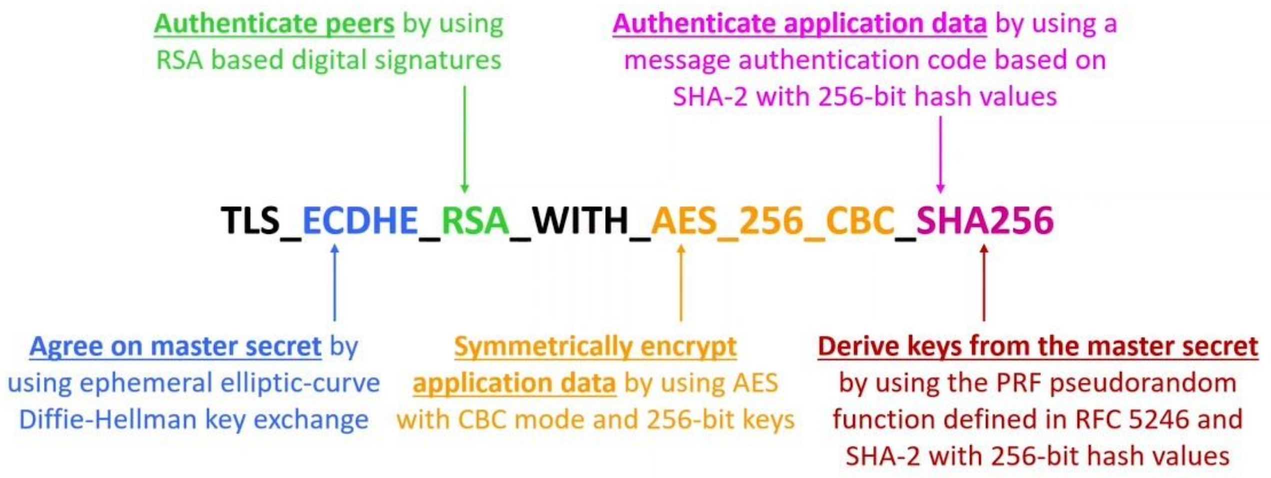
- **Type (1 byte)**: Indicates one of 10 messages.
- **Length (3 bytes)**: The length of the message in bytes
- **Content (≥ 0 bytes)**: The parameters associated with this message

Handshake Protocol: Phase 1



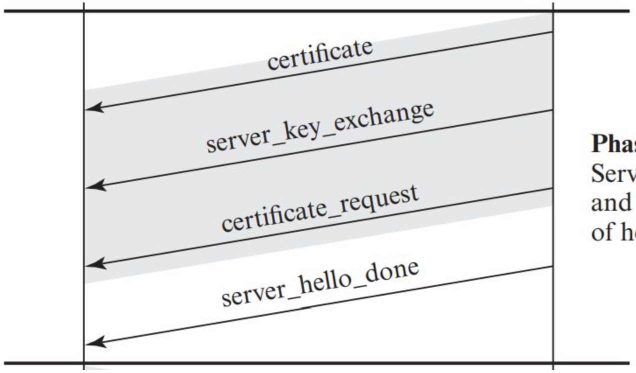
- The **SSL versions** supported by the client.
- 32 bytes of random data **(nonce)** generated by the client.
- A **session ID** created by the client.
- A list of **supported ciphers**.
- A list of **supported compression methods**.

Handshake Protocol: cipher suites



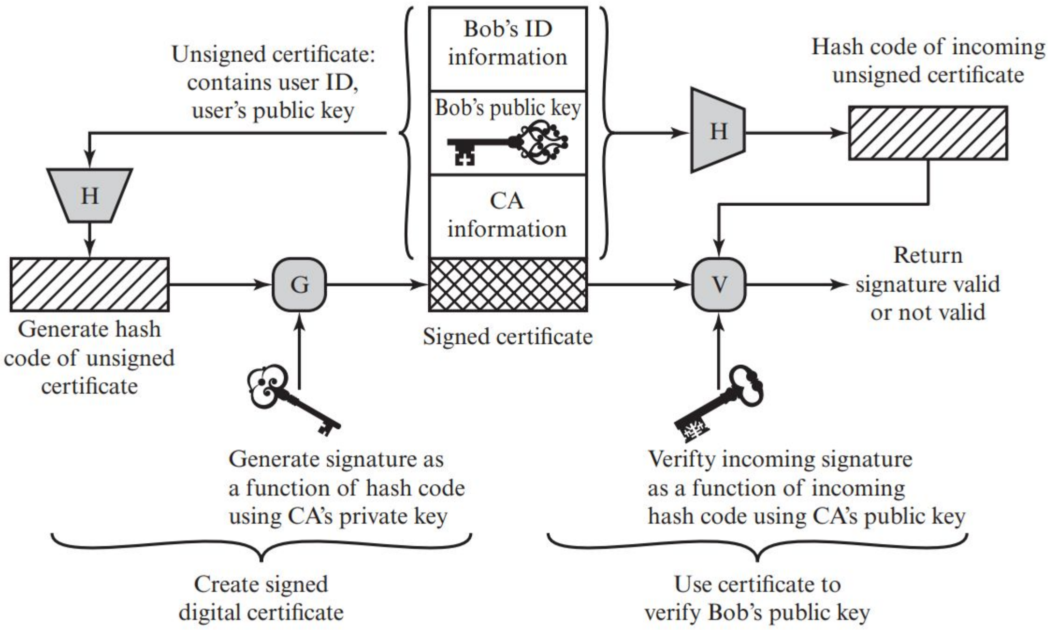
Handshake Protocol: Phase 2

- The **SSL version selected** from the client list.
- 32 bytes of **random data** generated by the server.
- The **session ID**.
- The **ciphers** selected from the client list (RSA and RC4)
- A **selected compression method** (generally no compression is used)



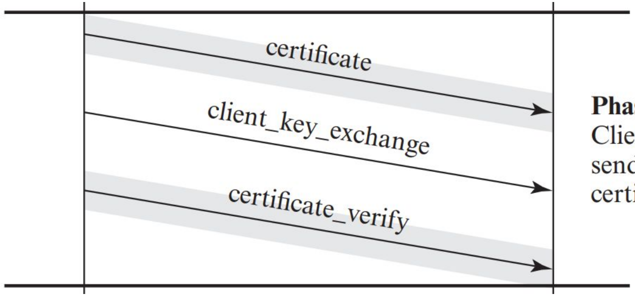
Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Public key Certification Authorities (CA)



- **Certification Authority (CA)**: binds public key to particular entity, E
- entity registers its public key with CE provides **"proof of identity"** to CA
  - CA creates certificate (standard X.509) 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"**

Handshake Protocol: Phase 3



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

- **certificate:** if server requested a certificate, clients sends its certificate.
- **client_key_exchange:** client generates 48 bytes of random data (**pre-master secret**) and encrypts them with server public key.
- **certificate_verify message:** verification of client certificate if sent before. Client hashes all messages sent from client_hello and signs them with its private key – the server validates its signature.

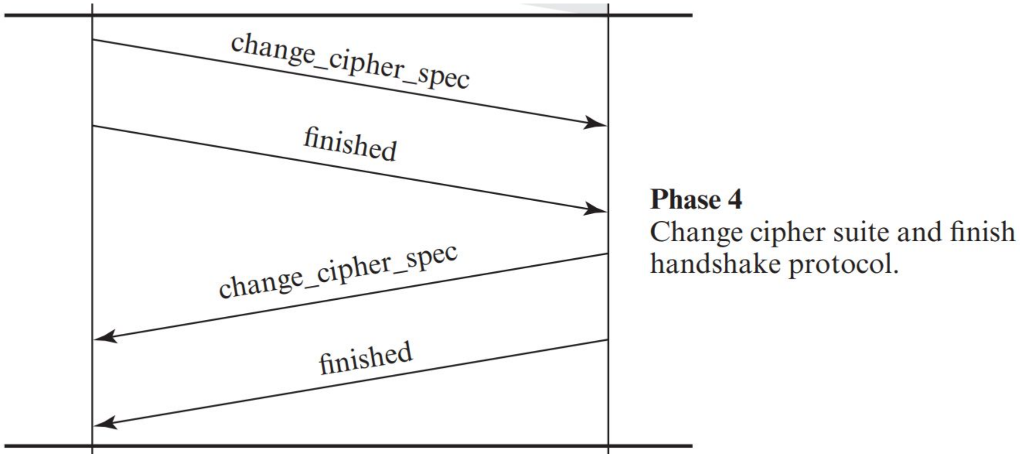
Key Derivation

- After receiving the server certificate, clients creates a **Master Secret (MS)**, encrypts it with the server public key **Encrypted Master Secret (EMS)** and sends it to the server.
- Server decrypts the EMS with its private key. **Now client and server shares a secret!**
- From the MS client and server generate 4 keys for encryption and message integrity:
  - $\mathbf{E_c}$ = session encryption key for data sent from client to server;
  - $\mathbf{M_c}$ = session MAC key for data sent from client to server;
  - $\mathbf{E_s}$ = session encryption for data sent from server to client;
  - $\mathbf{M_s}$ = session MAC key for data sent from server to client.
- In order to prevent a replay attack and a man-in-the-middle attack, server and client use a **sequence number** which is a counter for every TLS record sent by the server/client.

$$\mathbf{MAC = H(m, MAC\ key\ (M_c\ or\ M_s), sequence\ number)}$$

Handshake Protocol: Phase 4

- **change_cipher_spec (client):** client copies the pending CipherSpec into the current CipherSpec indicating that future communications will be handled with these ciphers and parameters. Then it sends a finished message under the new algorithms, keys and secrets.
- **change_cipher_spec (server):** server does the same thing.



Heartbeat protocol

- A heartbeat is a **periodic signal** generated by hardware or software to indicate normal operation or to synchronize other parts of a system
- A heartbeat protocol is typically used to **monitor the availability** of a protocol entity
- The heartbeat protocol runs on top of the TLS Record Protocol
  - Consists of two message types: **heartbeat_request** and **heartbeat_response**
- The heartbeat serves two purposes:
  - It assures the sender that the **recipient is still alive**, even though there may not have been any activity over the underlying TCP connection
  - It generates activity across the connection during **idle periods**, which **avoids closure by a firewall** that does not tolerate idle connections

Check your website

HTTPS

HTTPS (HTTP over SSL)

- Refers to the **combination of HTTP and SSL** to implement secure communication between a Web browser and a Web server
- The **HTTPS** capability is **built** into all modern **Web browsers**
- A user of a Web browser will see URL addresses that begin with https:// rather than http://
- If HTTPS is specified, port **443** is used, which invokes SSL
- Documented in RFC 2818, HTTP Over TLS
- There is no fundamental change in using HTTP over either SSL or TLS and both implementations are referred to as HTTPS
- When HTTPS is used, the following elements of the communication are encrypted:
    - URL of the requested document
    - Contents of the document
    - Contents of browser forms
    - Cookies sent from browser to server and from server to browser
    - Contents of HTTP header

Connection Initiation

For HTTPS, the agent acting as the **HTTP client** also acts as the **TLS client**

- The client initiates a connection to the server on the appropriate port and then sends the **TLS ClientHello** to begin the TLS handshake
- When the **TLS handshake** has **finished**, the client may then initiate the first **HTTP request**
- All **HTTP data** is to be sent as **TLS application data**

There are three levels of awareness of a connection in HTTPS:

- At the HTTP level, an HTTP client requests a connection to an HTTP server by sending a connection request to the next lowest layer
  - Typically the next lowest layer is TCP, but it may also be TLS/SSL
- At the level of TLS, a session is established between a TLS client and a TLS server
  - This session can support one or more connections at any time

Connection closure

- An HTTP client or server can indicate the closing of a connection by including the line **Connection: close in an HTTP record**
- The closure of an HTTPS connection requires that TLS close the connection with the peer TLS entity on the remote side, which will involve closing the underlying TCP connection
- **TLS implementations** must initiate an exchange of **closure alerts** before closing a connection
- A TLS implementation may, after sending a closure alert, close the connection without waiting for the peer to send its closure alert, generating an “incomplete close”
- An **unannounced TCP closure** could be **evidence** of some **sort of attack** so the HTTPS client should issue some sort of security warning when this occurs

References

- Kurose, J. F., & Ross, K. W. (2021). *Computer networking : a top-down approach* (8th ed.). Pearson.
- Stallings, W. (2017). *Network security essentials : applications and standards*. Pearson Education, Inc.
- Gössi, C. (2023, March 14). *TLS Essentials 10: TLS cipher suites explained*. Youtube Video. https://www.youtube.com/watch?app=desktop&v=mFdDap9A9-Q&ab_channel=CyrillG%C3%B6ssi
- Rescorla, E., & Dierks, T. (2008, August). *The Transport Layer Security (TLS) Protocol Version 1.2*. IETF. https://datatracker.ietf.org/doc/html/rfc5246
- Rescorla, E. (2018, August). *The Transport Layer Security (TLS) Protocol Version 1.3*. IETF. https://datatracker.ietf.org/doc/html/rfc8446