Lecture Introduction into Cyber Security

Transport Layer Security (TLS) (Part 1)

Asya Mitseva, M.Sc.
Prof. Dr.-Ing. Andriy Panchenko

Chair of IT Security
Brandenburg University of Technology Cottbus-Senftenberg

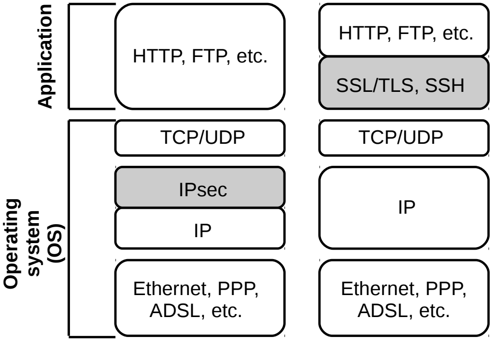
15 January 2019

Brandenburgische Technische Universität Cottbus - Senftenberg

UNIVERSITÉ DU LUXEMBOURG

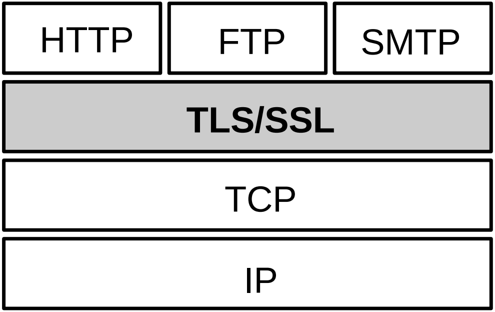
Erasmus+

Introduction



- **Most applications need to distinguish between distinct users**
- **Problem: IPsec only tells which IP address in on the other end**
  - Several users can be behind certain IP address, e.g., use of NAT
  - Single user can access the network from variety of IP addresses

Transport Layer Security (TLS): Overview (1/2)



- **Set of protocols that rely on TCP and provide**
  - Integrity and encryption of application data
  - Authentication of identities of both communicating parties
  - Interoperability, i.e., different applications should be able to use TLS
  - Efficiency, i.e., reduce the number of cryptographic operations needed

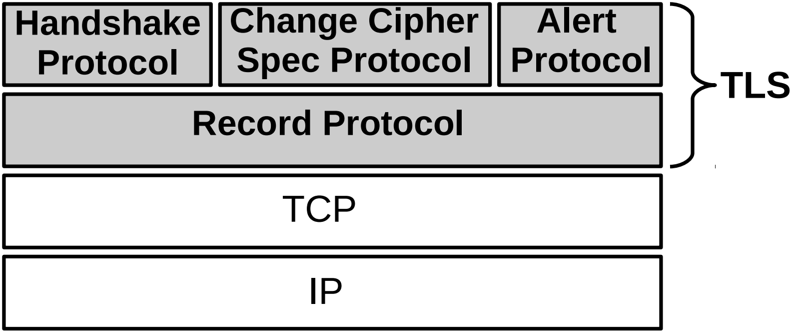
Transport Layer Security (TLS): Overview (2/2)

- Standardized version of *Secure Socket Layer (SSL)*
  - SSL is commercial protocol developed by Netscape
  - Last SSL version: 3.0, published in 1996

- **TLS version history**
  - TLSv1.0 – Upgrade of SSLv3.0, defined in RFC 2246 in 1999
  - TLSv1.1 – Add protection to attacks possible against TLSv1.0 and handle padding errors, defined in RFC 4346 in 2006
  - TLSv1.2 – Update and extension of used cipher suites, defined in RFC 5246 in 2008
  - TLSv1.3 – Defined in RFC 8446 in 2018

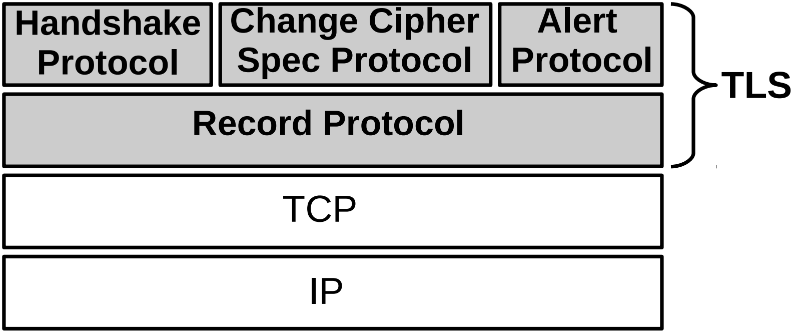
TLS Architecture (1/2)

- **Handshake Protocol**
  - Assure authentication of both communication parties
  - Negotiate encryption and MAC algorithms
  - Negotiate shared keys used to protect application data
- **Change Cipher Spec Protocol**
  - Activate the negotiated cipher suite



TLS Architecture (2/2)

- **Alert Protocol**
  - Used to exchange TLS-related alerts between communicating parties
- **Record Protocol**
  - Compute MAC on application data
  - Encrypt application data
  - Use keys based on master secret negotiated by the *Handshake Protocol*



TLS Concepts (1/2)

- **TLS session**
  - Association between two communicating parties
  - Define set of cryptographic security parameters
  - Created by the *Handshake Protocol*
  - Shared among multiple connections
  - *Goal: Efficiency*
    - Avoid negotiation of security parameters for each connection

- **TLS session is defined by the following parameters**
  - *Session identifier:* Arbitrary byte sequence selected by the server
  - *Peer certificate:* X509.v3 certificate of the corresponding peer
  - *Compression method:* Algorithm used to compress data
  - *Cipher spec:* Specify crypto algorithm used for data encryption and hash algorithm for MAC calculation
  - *Master secret:* Secret shared between both communicating peers
  - *Is resumable:* Indicate if session can be used to initiate new connections

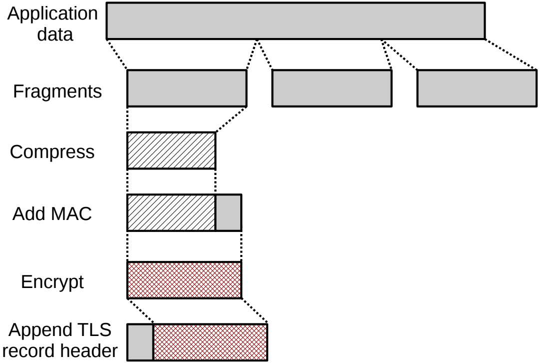
TLS Concepts (2/2)

- **TLS connection**
  - Transport-layer connection that provides suitable type of service
  - Peer-to-peer relationship that is transient
  - Associated with *one* TLS session

- **TLS connection is defined by the following parameters**
  - *Server and client random:* Byte sequences chosen by server and client for each connection
  - *Server/Client write MAC secret:* Secret key used for MAC calculation on data sent by the server/client
  - *Server/Client write key:* Symmetric key for data encrypted by the server/client and decrypted by the client/server
  - *Initialization vectors:* Initialization vector is maintained for each key if necessary
  - *Sequence numbers:* Each peer maintains sequence numbers for sent and received messages for each connection

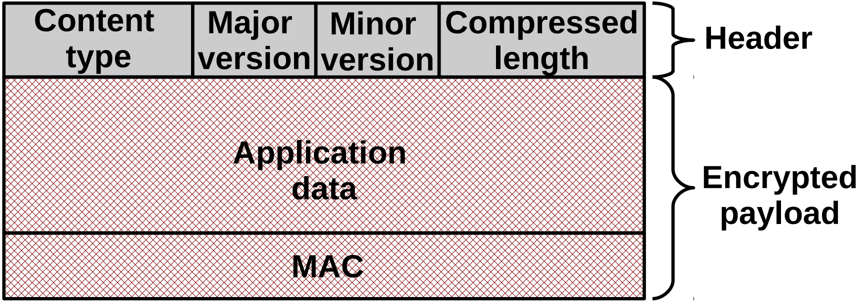
TLS Record Protocol (1/2)

- **Operation**
  1. Application data is fragmented into blocks of 2^{14} bytes
  2. Lossless compression may be optionally applied on each fragment
  3. Compute MAC over the compressed data by using HMAC algorithm
  4. Encrypted compressed fragment & its MAC using symmetric encryption
  5. Prepends TLS record header



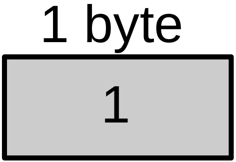
TLS Record Protocol (2/2)

- **TLS record header**
  - *Content type:* Higher-layer TLS protocol used to process the fragmet
    - `change_cipher_spec`, `alert`, `handshake`, `application_data`
  - *Major version:* Indicate major version of TLS in use
  - *Minor version:* Indicate minor version of TLS in use
  - *Compressed length:* Length of the plaintext fragment (in bytes)



Change Cipher Spec Protocol

- Specify the `change_cipher_spec` message sent during TLS handshake
- Consist of one message containing single byte of value 1
- Sent by both communicating parties
- Announce that certain cipher suite is used for subsequent messages in session



Alert Protocol

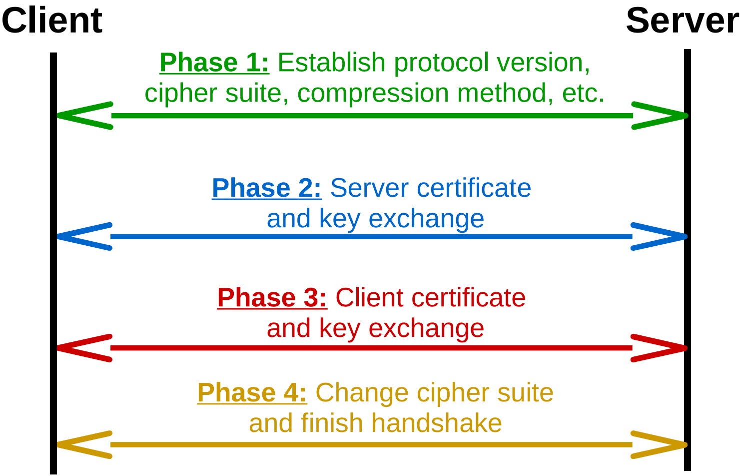
- Deliver TLS-related alerts between communicating parties
- Each alert message consists of two bytes indicating
  - Severity of the alert: *warning* or *fatal*
  - Code of the alert
- Alert messages are compressed and encrypted
- In case of fatal alert, TLS immediately terminates the connection
- Examples for fatal alerts
  - `unexpected_message:` Inappropriate message received
  - `bad_record_mac:` Incorrect MAC received
  - `decompression_failure:` Unable to decompress message
  - `protocol_version:` The protocol version of the client is not supported by the server
  - Etc.

Handshake Protocol: Overview (1/3)

- Negotiate cryptographic security parameters used for TLS session
- Messages exchanged between the client and the server consist of
  - *Type:* Indicate one of ten possible message types
  - *Length:* Length of the message in bytes
  - *Content:* Parameters associated with this message
- 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

- Consist of *four phases*

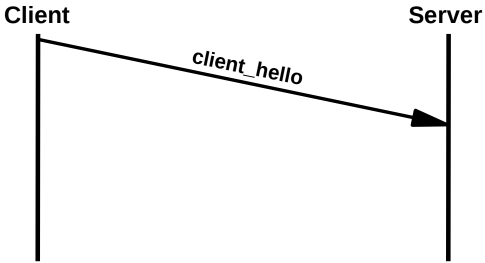


Handshake Protocol: Overview (3/3)

- *Goal:* Negotiate *shared master* secret key between client and server
- Other session keys for encryption and MAC calculation are generated based on the master secret
- Four master secret key exchange methods are supported
  - RSA algorithm
  - Fixed Diffie-Hellman algorithm
  - Ephemeral Diffie-Hellman algorithm
  - Anonymous Diffie-Hellman algorithm
- The set of messages sent in Phase 2 and Phase 3 distinguish depending on the key exchange method selected

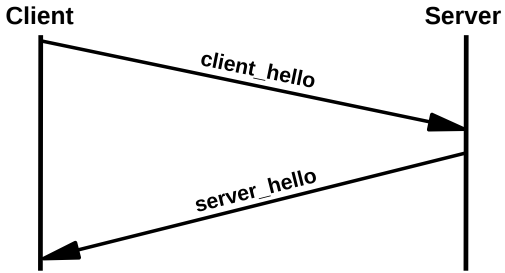
Phase 1: Establish Security Capabilities (1/2)

- Initiate session and establish security parameters associated with it
- The client sends `client_hello` message indicating
  - *Version:* Highest TLS version supported by the client
  - *Random:* Nonce used to prevent replay attacks during key exchange
  - *Session ID:* Variable-length session identifier
  - *CipherSuite:* List of crypto algorithms supported by the client
  - *Compression Methods:* Compression methods supported by the client



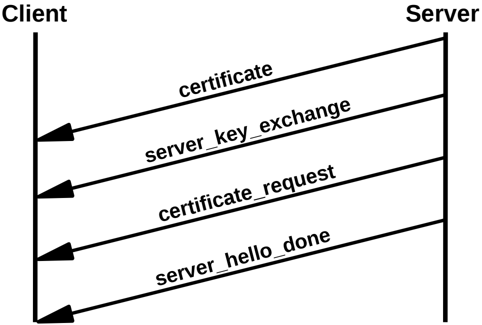
Phase 1: Establish Security Capabilities (2/2)

- The server responds with `server_hello` message indicating
  - *Version:* Highest TLS version supported by the server
  - *Random:* Nonce created by server, independent from client's Random
  - *Session ID:* Variable-length session identifier created by server
  - *CipherSuite:* Cipher suite selected from those supported by client
  - *Compression Methods:* Compression method chosen from those supported by client



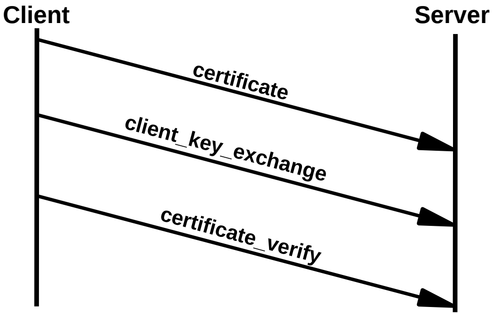
Phase 2: Server Authentication and Key Exchange

- Server sends its `certificate` if its authentication is required
  - E.g., not required for anonymous Diffie-Hellman
- Server sends `server_key_exchange` if it is required
  - E.g., not required for fixed Diffie-Hellman
- Server requests certificate from the client
- Server indicates end of the server hello by `server_done` message



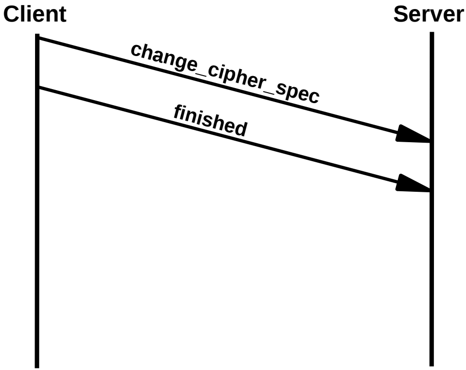
Phase 3: Client Authentication and Key Exchange

- Client checks if server certificate is valid and `server_hello` parameters are acceptable
- Client sends its `certificate` if it is requested
- Client sends `client_key_exchange`
  - If key parameters are sent in the certificate, this message is null
- Client sends explicit certificate verification by `certificate_verify`
  - Signs all sent/received messages starting from `client_hello`



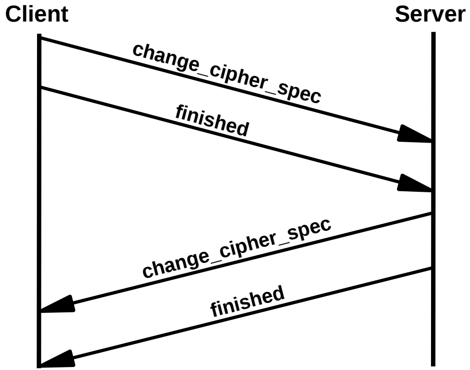
Phase 4: Finish (1/2)

- Client sends `change_cipher_spec` to activate negotiated cipher suite
- Client sends `finished` message to check if the key exchange and authentication succeeded
  - PRF(`master_secret`, `finished_field`, MD5(`handshake_messages`) || SHA-1(`handshake_messages`))
  - `finished_field` = client finished



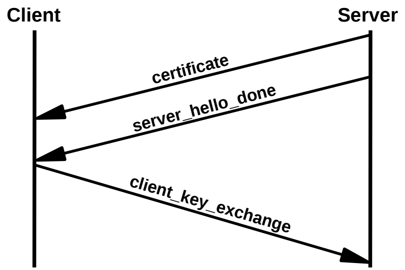
Phase 4: Finish (2/2)

- Server sends `change_cipher_spec` to activate negotiated cipher suite
- Server sends `finished` message to check if the key exchange and authentication succeeded
  - PRF(`master_secret`, `finished_field`, MD5(`handshake_messages`) || SHA-1(`handshake_messages`))
  - `finished_field` = server finished



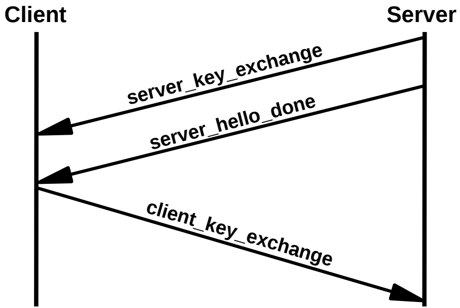
Key Exchange Methods Used in Handshake Protocol (1/4)

- **RSA**
  - Server sends public-key certificate for RSA encryption key
  - Client generates random secret key, called `pre_master_secret`
  - Client encrypts the `pre_master_secret` by using server's public key
  - Client sends the encrypted `pre_master_secret` to the server
  - Server decrypts the `pre_master_secret` with its private key
  - Calculation of `master_secret`: PRF(`pre_master_secret`, "master secret", ClientHello.Random || ServerHello.Random))
  - Longer RSA secret key → higher level of security



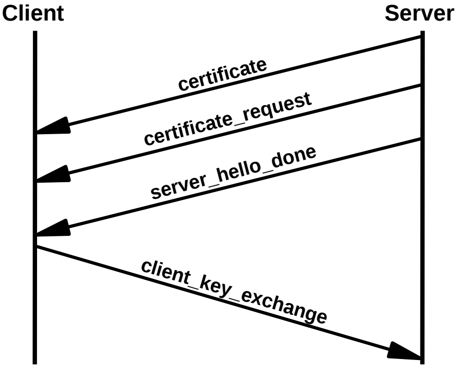
Key Exchange Methods Used in Handshake Protocol (2/4)

- **Anonymous Diffie-Hellman**
  - Use of base Diffie-Hellman algorithm without authentication
  - Peers exchange public Diffie-Hellman parameters to each other
  - Peers perform Diffie-Hellman calculation to obtain `pre_master_secret`
  - Calculation of `master_secret`: PRF(`pre_master_secret`, "master secret", ClientHello.Random || ServerHello.Random))
  - *Problem:* Vulnerable to man-in-the-middle attacks



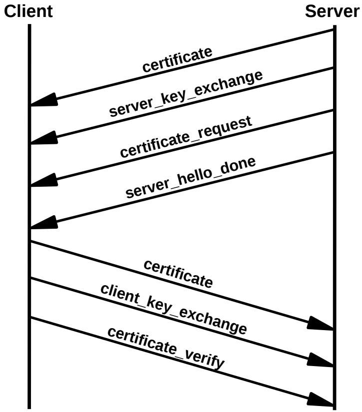
Key Exchange Methods Used in Handshake Protocol (3/4)

- **Fixed Diffie-Hellman**
  - Server certificate contains fixed public Diffie-Hellman parameters
  - The Diffie-Hellman parameters signed by certification authority
  - Client sends its public Diffie-Hellman parameters in certificate if authentication is required
  - Secure as long as the certification authority is not compromised



Key Exchange Methods Used in Handshake Protocol (4/4)

- **Ephemeral Diffie-Hellman**
  - Client and server create *fresh, temporal* Diffie-Hellman parameters
  - Diffie-Hellman parameters are signed with client's or server's private key
  - Client and server exchange its public keys for signature verification
  - Provide the highest level of security compared to the other methods



Pseudo-random Function (PRF)

- Expand secrets into blocks of data
- Make use of small, shared secret to generate longer blocks of data
- Guarantee security from the kinds of attacks made on hash functions
- HMAC_hash(secret, A(1) || seed) || HMAC_hash(secret, A(2) || seed) || HMAC_hash(secret, A(3) || seed) || ...
- $A(0) =$ seed, $A(i) =$ HMAC_hash(secret, A(i - 1))