



# Network Security

## Chapter 5

### Secure Socket Layer (SSL)/ Transport Layer Security (TLS)



- ❑ Classification in the OSI reference model
- ❑ SSL/TLS history
- ❑ TLS security services and protocol architecture



## Classification in the OSI Reference Model (1)

- ❑ The transport layer provides communication between application processes (instead of communication between end-systems)  
Its main tasks are:
  - Isolation of higher protocol layers from the technology, structure and deficiencies of deployed communications technology
  - Transparent transmission of user data
  - Global addressing of application processes, independently of lower layer addresses (Ethernet addresses, telephone numbers, etc.)
  - Overall goal: provision of a (if required reliable) end-to-end service.



## Classification in the OSI Reference Model (2)

- ❑ *Transport layer security protocols* enhance the transport layer service by assuring additional security properties.
- ❑ As they usually require and are built upon a transport service, they actually represent *session layer protocols* according to the terminology of the *Open Systems Interconnection (OSI) reference model*.
- ❑ However, based on the Internet terminology where the application layer is directly on top of the transport layer, we use the term transport layer security protocols.

## Overview

- Classification in the OSI reference model
- SSL/TLS history
- TLS security services and protocol architecture

## SSL/TLS History (1)

- The Secure Socket Layer (SSL) protocol was originally designed to primarily protect HTTP sessions.
- In the early 1990's there was a similar protocol called S-HTTP.
- However, as S-HTTP capable browsers were not free of charge.
- SSL version 2.0 was included in browsers of Netscape Communications, it quickly became predominant.
- SSL v.2 contained some flaws and so Microsoft Corporation developed a competing protocol called Private Communication Technology (PCT).
- Netscape improved the protocol and SSL v.3 became the de-facto standard protocol for securing HTTP traffic.

## SSL/TLS History (2)

- SSL can be deployed to secure arbitrary applications that run over TCP.
- In 1996 the IETF decided to specify a generic *Transport Layer Security (TLS)* protocol that is based on SSL.
- The IETF started a working group to define the *TLS* protocol.
- Officially, the protocols SSL, SSH and PCT were announced to be taken as input.
- However, only SSL V.3.0 was really considered.

## SSL/TLS History (3)

- TLS V.1.0 was published as an RFC in January 1999 [RFC2246]
- Modifications compared to SSL V.3.0 include:
  - The HMAC construction  $H(K, p_1, H(K, p_2, m))$  for message integrity was adopted instead of hashing in prefix and suffix mode  $H(K, m, K)$
  - The Fortezza based cipher-suites of SSL was removed, as they include an unpublished technology
  - A digital signature standard (DSS) based authentication and key exchange dialogue was included
    - DSS uses SHA-1 to compute a fingerprint of a message and El-Gamal as a signature algorithm.
    - The DSS public key  $g^x$  can be also used for a DH exchange.

## SSL/TLS History (4)

- In order to achieve exportability of TLS compliant products, some cipher-suites specify the use of keys with entropy reduced to 40 bit:
  - These cipher-suites contain the word “export” in their name
  - As the government of the USA changed its policy concerning the export of cryptographic products, this is of less importance today. The use of 40 bits keys is deprecated
- The version presented in this Chapter is TLS V1.1 [RFC4346] ratified in April 2006.

## SSL/TLS History (5)

- TLS V1.2 [RFC5246] was ratified in August 2008.
- TLS V1.2 adds more flexibility in the negotiation of the cipher suite.
  - Provides the ability to negotiate an algorithm for a pseudo-random function PRF. (PRF in V1.1 was pre-defined and makes use of MD5 and SHA-1).
  - Adds more algorithms for cryptographic hash values, e.g. SHA-256, SHA-384, SHA-512.
  - [RFC5246] combines several documents together, e.g. the definition of the AES and ECC cipher suites were defined in separate documents for V1.1.
- For the newest development of the TLS protocol, please refer to the Working Group (WG) web site at <http://www.ietf.org/html.charters/tls-charter.html>

## Overview

- Classification in the OSI reference model
- SSL/TLS history
- TLS security services and protocol architecture

## TLS Security Services

- *Peer entity authentication:*
  - Prior to any communications between a client and a server, an authentication protocol is performed to authenticate the peer entities and establish a shared secret key
    - Either only client performs authentication of the server
    - Or additionally, the server performs authentication of the client
  - Authentication can be performed based on certificates
    - Initially specified public key algorithms for the certificates: RSA and DSS
    - ECC was added in [RFC4492]
  - Authentication based on a long term pre-shared key was added in [RFC4279]
  - Upon successful completion of the authentication dialogue a *TLS session* is established between the peer entities
- *User data confidentiality:*
  - If negotiated upon session establishment, user data is encrypted
  - Encryption algorithms: IDEA / DES / 3DES / RC2 in CBC, RC4, null
  - AES cipher suites was added in [RFC3268]
- *User data integrity:*
  - A MAC based on a cryptographic hash function is appended to user data
  - Hash algorithms: MD5, SHA, null
- Replay protection

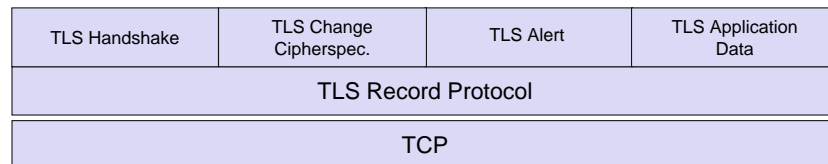
## TLS Sessions and TLS Connections

- The TLS protocol negotiates a so-called „*TLS session*“ with a TLS handshake.
- After the TLS handshake, both communication parties have established a security context.
- It would be useful to re-use this security context for several TCP connections, in order to gain performance.
- This is very important for securing HTTP traffic, as for HTTP 1.0, every item on a web page is transferred in an individual TCP connection.
- TLS can negotiate a TLS sessions with different (parallel or subsequent) TLS connections.
- Furthermore, TLS can guarantee that different keying material for encryption and data integrity is used for different connections.

## TLS Session & Connection State

- TLS Session state:
  - *Session identifier*: a byte sequence chosen by the server
  - *Peer certificate*: X.509 v.3 certificate of the peer (optional)
  - *Compression method*: algorithm to compress data prior to encryption
  - *Cipher spec*: specifies cryptographic algorithms and parameters
  - *Master secret*: a negotiated shared secret of length 48 byte
  - *Is resumable*: a flag indicating if the session supports new connections
- TLS Connection state:
  - *Server and client random*: byte sequences chosen by server and client
  - *Server write MAC secret*: used in MAC computations by the server
  - *Client write MAC secret*: used in MAC computations by the client
  - *Server write key*: used for encryption by server and decryption by client
  - *Client write key*: used for encryption by client and decryption by server

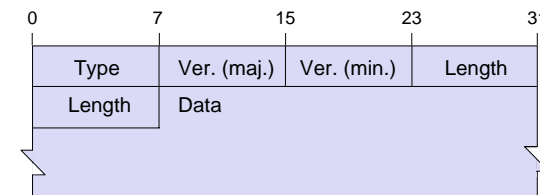
## TLS Protocol Architecture



- TLS is structured as a layered and modular protocol architecture:
  - Handshake: authentication of peers, negotiation of cryptographic parameters and establishment of a shared secret
  - Change Cipherspec.: signaling of transitions in ciphering strategy
  - Alert: signaling of error conditions
  - Application Data: interface for transparent access to the record protocol
  - Record:
    - Fragmentation of user data
    - Compression (optional) of plaintext records
    - Encryption and integrity protection (both optional)

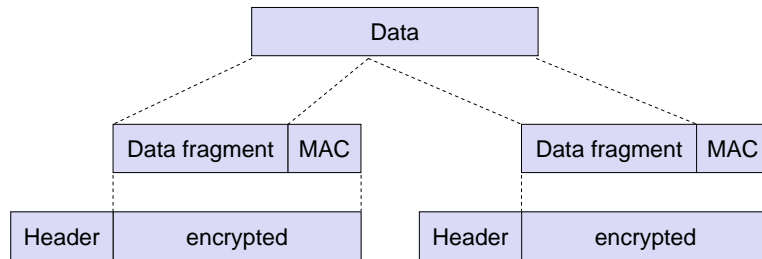
## TLS Record Protocol

- Record protocol format:



- Type:
  - Change Cipherspec. (20)
  - Alert (21)
  - Handshake (22)
  - Application Data (23)
- Version: the protocol version of TLS.
  - For historical reasons and backwards compatibility, it holds:
    - for TLS V1.1 the version value is 3.2
    - for TLS V1.0 the version value is 3.1,
    - for SSL V3.0 the value is 3.0
- Length: the length of the data in bytes

## TLS Record Protocol Processing (1)



- Sending side:
  - The record layer first fragments user data into fragments of a maximum length of  $2^{14}$  octets. More than one message of the same content type can be assembled into one record.
  - After fragmentation the record data is compressed, the default algorithm for this is *null* (~ no compression)
  - A message authentication code is appended to the record data
  - The record data and the MAC are encrypted using the encryption algorithm defined in the negotiated cipherspec (may imply prior padding)

## TLS Record Protocol Processing (2)

- Receiving side:
  - The record is decrypted, integrity-checked, decompressed, de-fragmented and delivered to the application or TLS higher layer protocol
- The MAC appended to the record is computed as follow
  - $\text{HMAC}_{\langle \text{hash} \rangle}(\text{MAC\_write\_secret}, \text{seq\_num} \parallel \text{record-type} \parallel \text{protocol-version} \parallel \text{fragment-length} \parallel \text{fragment})$ 
    - $\langle \text{hash} \rangle$  denotes the hashing algorithm defined in the negotiated cipherspec
    - $\text{seq\_num}$  is the sequence number for this record.
- Notes on the sequence number:
  - $\text{seq\_num}$  is not explicitly transmitted, as the underlying TCP offers reliable transport
  - $\text{seq\_num}$  is used by TLS for the records and has no correlation with the TCP sequence number
  - $\text{seq\_num}$  is set to zero when a TLS connection is initiated
  - $\text{seq\_num}$  is incremented after each record
  - If  $\text{seq\_num}$  overflows, key re-negotiation is required
  - However, since  $\text{seq\_num}$  has 64 bits length, it should never overflow

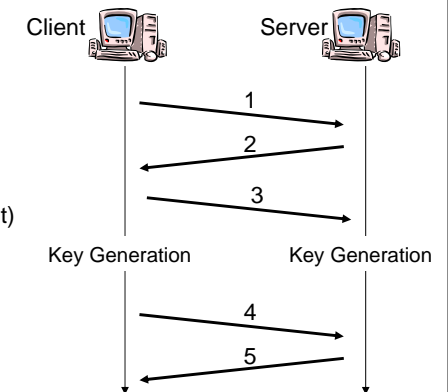
## TLS Handshake Protocol: Introduction

- TLS supports different methods for authentication and key establishment.
- The highest key in the key hierarchy, from which further keying material can be derived, is called the *pre-master-secret*.
  - **RSA:**
    - The *pre-master-secret* is randomly generated by the client and sent to the server encrypted with the servers public key
    - *ClientKeyExchange* is the encrypted pre-master key
    - The server does not send a *ServerKeyExchange* message to the client
  - **Diffie-Hellman:**
    - a standard Diffie-Hellman exchange is performed
    - the established shared secret is taken as *pre-master-secret*
    - *ClientKeyExchange* and *ServerKeyExchange* are respectively the (signed) DH values  $g^x$  and  $g^y$
- Ephemeral (temporary) vs. static DH values
  - The DH secrets may be deleted after the session is over  
In this case, the handshake provides perfect forward secrecy.
  - Alternatively, the DH secrets may be static  
This is the case if the server's (client's) certificate key algorithm is DH

## Overview of the TLS Handshake Protocol with RSA

- Overview:

1. random number, set of cryptographic suites
2. random number, Chosen cryptographic suite, Certificate
3. Key exchange (Pre-master secret)
- Generation of the *master-secret*
4. MAC on all previous messages
5. MAC on all previous messages



- Note: 3 and 4 are actually sent together to reduce latency  
➔ the TLS handshake requires two round trips

## Overview of the TLS Handshake Protocol with DH

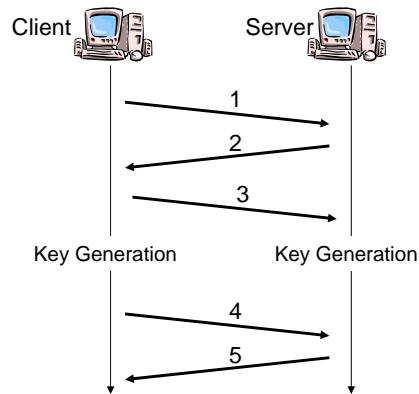
### Overview:

1. random number, set of cryptographic suites
2. random number, Chosen cryptographic suite, Certificate
3. Key exchange (DH value)

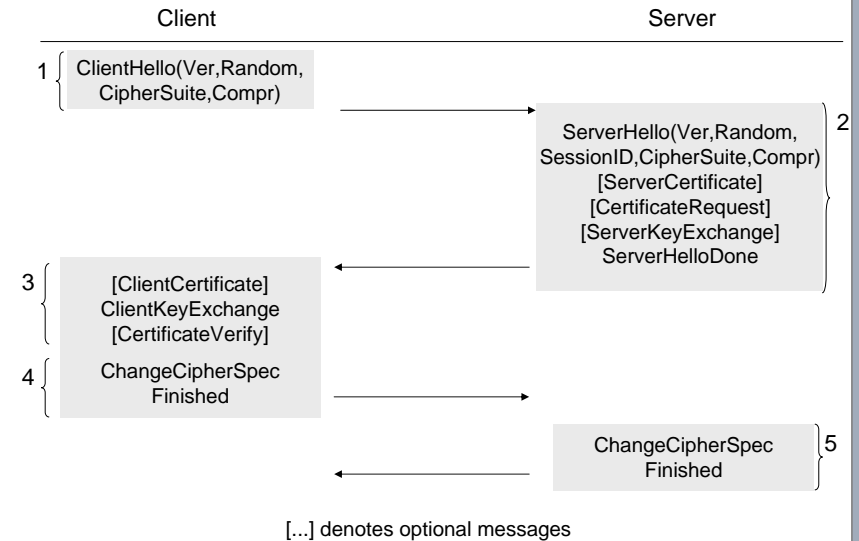
#### Generation of the *master-secret*

4. MAC on all previous messages
5. MAC on all previous messages

- Note: In TLS message 2, "Key exchange" is sent only if the protocol is used with DH.



## TLS Handshake Protocol – Details (1)



## TLS Handshake Protocol – Details (2)

- The initial handshake is not protected, since the client and server do not share a secret, nor have they negotiated a cipher spec
- The *ServerCertificate* message is sent by the server in all cases where the server needs to be authenticated
  - which is always the case except for anonymous key establishment
- The *CertificateRequest* message is sent by the server in case the client is also authenticated using a certificate
- The *ServerKeyExchange* message is sent by the server
  - only in case of a Diffie-Hellman key exchange
  - and only if the server's certificate does not contain sufficient information about the Diffie-Hellman public key
- The *ChangeCipherSpec* messages sent respectively by the client and the server in the the initial handshake indicate that from now on the rest of the communication is protected
- The "Finished" message is the first protected message with the just negotiated algorithms and keys
- The "Finished" messages enable both the client and the server to verify that the key exchange and authentication processes were successful

## TLS Handshake Protocol: Key Exchange

- Possible combinations for key exchange algorithm and server's certificate key algorithm

Notation	Key Exchange
RSA	<ul style="list-style-type: none"> <li>•The client generates the pre-master-secret and encrypts it with the server's RSA public key</li> <li>•The RSA key included in the server's certificate is used for encrypting the pre-master secret</li> </ul>
DHE_DSS	Ephemeral DH values signed using a DSS key
DHE_RSA	Ephemeral DH values signed using a RSA key
DH_DSS	a static Diffie-Hellman value signed by a DSS key
DH_RSA	a static Diffie-Hellman value signed by a RSA key
DH_anon	Diffie-Hellman key. No certificate. No authentication is performed. In this case man-in-the-middle attacks can not be defended



## TLS Handshake Protocol: Generation of The Master-Secret

- The pre-master-secret and the random numbers provided by the client and the server in their hello-messages are used to generate the master-secret of length 48 byte
- Computation of the master secret:
  - $master\_secret = PRF(pre\_master\_secret, "master\ secret", ClientHello.random + ServerHello.random)$
- where:
  - PRF is a pseudo-random function that generates an output of arbitrary length,
  - "master secret" is the string "master secret"
  - PRF is defined in [RFC4346] Section 5
  - PRF makes use of HMAC with MD5 as well as HMAC with SHA-1 as hash functions
  - The use of both MD5 and SHA-1 is considered to provide security even in case that one of the cryptographic hash functions is "broken"



## TLS Handshake Protocol: Authentication with RSA-based Exchange

- Server authentication:
  - The client verifies the server's certificate and encrypts the pre-master-secret with the server's public key provided with the certificate
  - The client knows that only the server can decrypt the *pre-master-secret*
  - The *pre-master-secret* is required to compute the *master-secret*
  - The *master-secret* is used to compute a **verify\_data** value included in the "Finished" message
    - $verify\_data = PRF(master\_secret, finished\_label, MD5(handshake\_messages) + SHA-1(handshake\_messages))$
    - **finished\_label** is the string "client finished" for the Client's Finished messages and "server finished" for the Servers' Finished messages
  - Thus, when the server sends the correct Finished message to the client, the client can deduce server-authenticity
- Client authentication
  - The server can not deduce any client authenticity from the received *pre-master-secret*
  - If client authenticity is required, the client additionally sends its certificate and a CertificateVerify message that contains a signature over a hash (MD5 or SHA) of the *master-secret* and all handshake messages exchanged before the CertificateVerify message



## TLS Handshake Protocol: Authentication with DH-based Exchange

- Server authentication
  - The server can
    - 1) either supply a certificate containing fixed Diffie-Hellman parameters
    - 2) or use the "server key exchange" message to send a set of temporary (ephemeral) Diffie-Hellman parameters signed with a DSS or RSA certificate.
 

In this case, temporary parameters are hashed with the hello.random values before signing to ensure that attackers do not replay old parameters.
  - In either case, the client can verify the certificate (1) or signature (2) to ensure that the parameters belong to the server.
- Client authentication
  - If the client has a certificate containing fixed Diffie-Hellman parameters, its certificate contains the information required to complete the key exchange and client authentication.
  - Note that in this case the client and server will generate the same Diffie-Hellman result (i.e., *pre-master-secret*) every time they communicate. However, the *master-key* will be different, since the hello.random value are used to compute the *master-key*.



## TLS Handshake Protocol: Generation of Session Keys

- TLS requires several sessions keys for the encryption and data integrity of the application data (and TLS higher layer protocols) in both directions
- To compute the session keys, a sufficient amount of keying material is generated from the *master-secret* and the client's and server's random numbers in a first step:
  - $key\_block = PRF(master\_secret, "key\ expansion", server\_random + client\_random);$
- Then, the session keying material is truncated consecutively from the *key\_block*:
  - *client\_write\_MAC\_secret*
  - *server\_write\_MAC\_secret*
  - *client\_write\_key*
  - *server\_write\_key*

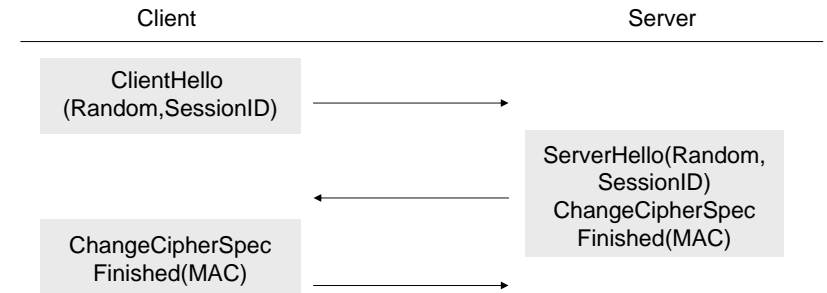


## TLS Handshake Protocol: Abbreviated Handshake (1)

- A TLS session can be negotiated to be “resumable”:
  - Resuming and duplicating TLS sessions allows to re-use established security context for several TCP connections
  - As mentioned above, this is very important for securing HTTP traffic, as for HTTP 1.0, every item on a web page is transferred in an individual TCP connection.
  - When resuming / duplicating an existing session, an abbreviated handshake is performed



## TLS Handshake Protocol: Abbreviated Handshake (2)



- If the server finds the sessionID and decides to resume the session, it answers immediately with the „ChangeCipherSpec” and “Finished” messages
- If the server can not resume / decides not to resume the session it answers with the full handshake
- If the server decides to resume, new keying material is generated using the new random values
  - $key\_block = PRF(master\_secret, "key\ expansion", server\_random + client\_random);$
- The Finished messages are protected with the freshly generated keying material and include a MAC of all the previous abbreviated handshake messages



## SSL/TLS Alert Protocol

- Used to transmit errors and exceptions
- Here is a non-extensive list of exceptions and error cases (please refer to [RFC4346] Section 7.2 for the complete list)
  - closed\_notify: used to initiate the closure of the TLS session
    - This is needed to avoid truncation attacks
  - unexpected\_message
  - bad\_record\_mac
  - decryption\_failed
  - record\_overflow
  - bad\_certificate
  - unknown\_ca
  - ...



## SSL/TLS Change Cipherspec Protocol

- The change cipher spec protocol is used to signal transitions in ciphering strategies.
- The protocol consists of a single message “ChangeCipherSpec”, which is encrypted and compressed under the current (not the pending) connection state



## TLS Cipher-Suites (1)

- Cipher suites: a set of pre-defined cryptographic algorithms
- More than 50 different cipher suites are defined for TLS V1.1
- Notation
  - Prefix: "TLS\_"
  - Consists of 2 selectors
    - Key exchange algorithm
    - Algorithms for protection with the Record protocolseparated by „\_WITH\_“
- Example: TLS\_DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
  - DHE\_RSA:
    - Key exchange with Ephemeral Diffie-Hellman,
    - Authentication with RSA certificate  
(i.e. the RSA public key is used to sign the DH value)
  - 3DES\_EDE\_CBC: Encryption with 3DES in CBC mode  
(EDE means „Encryption → Decryption → Encryption“, which is the usual operation for 3DES that we have seen)
  - Message integrity is provided by SHA-1

## TLS Cipher-Suites (2)

- Note:
  - this list is not extensive. For more cipher specs, please refer to [RFC4346] and other related documents on <http://www.ietf.org/html.charters/tls-charter.html>
  - These cipher-suites, of course, do not need to be memorized and are listed here only to illustrate the flexibility of the TLS protocol
- No protection (initial suite):
  - CipherSuite TLS\_NULL\_WITH\_NULL\_NULL = { 0x00,0x00 }
- Server provides an RSA key suitable for encryption:
  - TLS\_RSA\_WITH\_NULL\_MD5 = { 0x00,0x01 }
  - TLS\_RSA\_WITH\_NULL\_SHA = { 0x00,0x02 }
  - TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5 = { 0x00,0x03 }
    - *deprecated*
  - TLS\_RSA\_WITH\_RC4\_128\_MD5 = { 0x00,0x04 }
  - TLS\_RSA\_WITH\_RC4\_128\_SHA = { 0x00,0x05 }
  - TLS\_RSA\_EXPORT\_WITH\_RC2\_CBC\_40\_MD5 = { 0x00,0x06 }
  - TLS\_RSA\_WITH\_IDEA\_CBC\_SHA = { 0x00,0x07 }
  - TLS\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA = { 0x00,0x08 }
  - TLS\_RSA\_WITH\_DES\_CBC\_SHA = { 0x00,0x09 }
  - TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA = { 0x00,0x0A }

## TLS Cipher-Suites (3)

- Cipher-Suites with an authenticated DH-Key-Exchange
  - TLS\_DH\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA = { 0x00,0x0B }
    - *deprecated*
  - TLS\_DH\_DSS\_WITH\_DES\_CBC\_SHA = { 0x00,0x0C }
  - TLS\_DH\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA = { 0x00,0x0D }
  - TLS\_DH\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA = { 0x00,0x0E }
    - *deprecated*
  - TLS\_DH\_RSA\_WITH\_DES\_CBC\_SHA = { 0x00,0x0F }
  - TLS\_DH\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA = { 0x00,0x10 }
  - TLS\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA = { 0x00,0x11 }
  - TLS\_DHE\_DSS\_WITH\_DES\_CBC\_SHA = { 0x00,0x12 }
  - TLS\_DHE\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA = { 0x00,0x13 }
  - TLS\_DHE\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA = { 0x00,0x14 }
    - *deprecated*
  - TLS\_DHE\_RSA\_WITH\_DES\_CBC\_SHA = { 0x00,0x15 }
  - TLS\_DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA = { 0x00,0x16 }

## TLS Cipher-Suites (4)

- The use of the following cipher-suites without any entity authentication is discouraged, as they are vulnerable to man-in-the-middle attacks:
  - TLS\_DH\_anon\_EXPORT\_WITH\_DES40\_CBC\_SHA = { 0x00,0x19 }
  - TLS\_DH\_anon\_WITH\_DES\_CBC\_SHA = { 0x00,0x1A }
  - TLS\_DH\_anon\_WITH\_3DES\_EDE\_CBC\_SHA = { 0x00,0x1B }
- AES cipher suites with 128 bit key length
  - TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA = { 0x00, 0x2F }
  - TLS\_DH\_DSS\_WITH\_AES\_128\_CBC\_SHA = { 0x00, 0x30 }
  - TLS\_DH\_RSA\_WITH\_AES\_128\_CBC\_SHA = { 0x00, 0x31 }
  - TLS\_DHE\_DSS\_WITH\_AES\_128\_CBC\_SHA = { 0x00, 0x32 }
  - TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA = { 0x00, 0x33 }
  - TLS\_DH\_anon\_WITH\_AES\_128\_CBC\_SHA = { 0x00, 0x34 }
- AES cipher suites with 256 bit key length
  - TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA = { 0x00, 0x35 }
  - TLS\_DH\_DSS\_WITH\_AES\_256\_CBC\_SHA = { 0x00, 0x36 }
  - TLS\_DH\_RSA\_WITH\_AES\_256\_CBC\_SHA = { 0x00, 0x37 }
  - TLS\_DHE\_DSS\_WITH\_AES\_256\_CBC\_SHA = { 0x00, 0x38 }
  - TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA = { 0x00, 0x39 }
  - TLS\_DH\_anon\_WITH\_AES\_256\_CBC\_SHA = { 0x00, 0x3A }

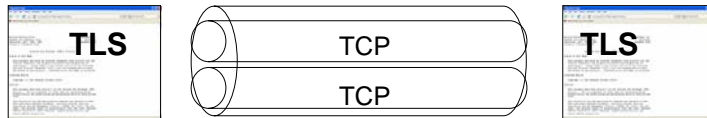
## Datagram TLS (DTLS) (1)

- SSL/TLS was originally designed to run on top of TCP
- Datagram TLS (DTLS) is a *new protocol* defined in [RFC4347] that can be used on top of unreliable transport protocols, such as UDP
- DTLS is already supported by the open source implementation of TLS: OpenSSL
- DTLS is very similar to TLS.
- Therefore, [RFC4347] refers to the TLS specification and specifies only the differences to TLS.

## Datagram TLS (DTLS) (2)

- DTLS provides
  - an unreliable transport for application payload.
  - replay protection with an explicit sequence number that is included in the record header, since message re-ordering, message duplication and message loss are possible
  - protection against Denial-of-Service attacks
    - TLS operates on TCP. Therefore, the IP address of the client is known by the time the TLS handshake is initiated.
    - This reduces the potential for DoS attack.
    - Note however: TCP SYN attacks are still possible (see later in this lecture)
    - DTLS supports protection against DoS attacks with cookies similar to IKEv2
- DTLS needs to take care of a re-transmitting its own control messages (e.g. DTLS handshake messages) since the underlying transport protocol does not provide it.
- For more details about the differences between TLS and DTLS, please refer to [RFC4347]

## TLS - Summary



- TLS provides
  - server authentication and eventually also client authentication
  - key establishment and negotiation of cryptographic algorithms
  - data integrity, confidentiality and replay protection
- a TLS session can be used to protect several TCP connections
- TLS consists of a record protocol, handshake protocol, change cipher spec protocol and alert protocol
- DTLS is a similar protocol to TLS with some extensions and modifications that are required due to the unreliable transport

## Additional References

- [FKK96a] A. O. Freier, P. Karlton, P. C. Kocher. *The SSL Protocol Version 3.0*. Netscape Communications Corporation, 1996.
- [RESC01] E. Rescorla, „SSL and TLS – Designing and Building Secure Systems“. Addison – Wesley. 2001
- [RFC3268] P. Chown, “AES Ciphersuites for TLS”, RFC 3268, 2002
- [RFC4279] P. Eronen, H. Tschofenig. “Pre-Shared Key Ciphersuites for Transport Layer Security (TLS) ”. RFC 4279, 2005
- [RFC4346] T. Dierks, E. Rescorla. “*The TLS Protocol Version 1.1*”. RFC 4346, 2006
- [RFC4347] E. Rescorla. N. Modadugu “*Datagram Transport Layer Security*”. RFC 4347, 2006
- [RFC4492] S. Blake-Wilson, et al, “Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)”, RFC 4492, 2006
- [RFC5246] T. Dierks, E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2”, RFC 5246, August 2008.



# Appendix

## IPSec vs. TLS a comparison



### IPSec vs. TLS (1)

- Both protocols provides the following security services:
  - Peer-authentication
  - Confidentiality
  - Message integrity
  - Replay protection
- Protocol components and equivalence/similarity between them
  - IPSec consists of several protocols; AH, ESP und IKE(v1/v2)
  - TLS is basically a single protocol, although different message categories are called protocols: the Record protocol, the Handshake Protocol, Change cipher suite protocol and Alert protocol
  - The TLS *Handshake Protocol*, *Change Cipher Suite Protocol* and *Alert Protocol* provide similar functionality to IKEv2
  - The TLS *Record protocol* provides similar functionality for application data as IPSec “AH and ESP” protocols provide for IP traffic



### IPSec vs. TLS (2)

- Scope of the protection: End points of the „secure channel“
  - TLS operates end-to-end between two *applications*
  - IPSec can operate end-to-end between two hosts, middle-to-middle (i.e. gateway-to-gateway) or end-to-middle (host-to-gateway)
- Scope of the protection: Protocol headers
  - TLS protects only the payload of the application. It does not protect the transport header. It does not protect the IP header.
  - IPSec protects the application payload, the transport header and the IP header (except if ESP is used in transport mode without AH. In this case, the IP header is not protected)
- Scope of the protection: data flows to be protected
  - TLS is used to establish a “secure channel”, which can protect several subsequent TCP connections
  - IPSec can be used to protect several data flows between two hosts A and B.
  - For each data flow to be protected, usually four security associations are required
  - The data flows to be protected are defined by so-called traffic selectors, which consist of the source/destination IP address and port numbers and the transport protocol, e.g. TCP or UDP



### IPSec vs. TLS (3)

- Terminology: Client and Server vs. Initiator and Responder
  - In IKEv2, the terminology used for the communication partners is “Initiator” and “Responder”. In TLS “Client” and “Server”.
  - Note however, that in many environments where IKEv2 is used, there is a “client” behind the “Initiator” and a “server” behind the “Responder”, e.g. a VPN client connecting to a VPN gateway.
- Multiplexing of „secure channels“
  - IKEv2 negotiates an IKE\_SA, which can be used to negotiate several CHILD\_SAs that are used to protect the data traffic
  - TLS negotiates an initial connection. Based on it further connections can be negotiated if the session is resumable.

## IPsec vs. TLS (4)

- Mutual authentication
  - IKEv2 and TLS provide both mutual authentication.
  - However, mutual authentication is a “MUST” in IKEv2 whereas in TLS it is allowed (and even very common) that the server is authenticated while the client is not authenticated
- Authentication with a long-term pre-shared secret
  - IKEv2 allows for peer authentication using a long-term shared secret.
  - This option has been added to TLS recently [RFC4279]
- Authentication messages and fields
  - Authentication in TLS is performed either based on the signed KeyExchange messages (in case of a DH key exchange) or the “Finished” message (in case of RSA key exchange)
  - Authentication in IKEv2 is performed based on the AUTH-Payload
- Perfect forward secrecy
  - IKEv2 provides perfect forward secrecy, if the KE value in the IKE\_SA\_INIT exchange are used only once
  - TLS provides perfect forward secrecy only if ephemeral DH exchange is used

## IPsec vs. TLS (5)

- Cryptographic properties: Certificate key algorithms
  - IKEv2 does not specify the public key algorithms to be used in case of authentication is performed via certificates, as the certificate specifies the public key algorithm
  - TLS specifies explicitly which algorithms to be used. Currently: RSA, DSS and ECC
- Cipher suite: negotiation
  - In both cases, the client/Initiator sends a set of proposed cipher suites, and the server/Responder chooses one of them, i.e. the negotiation of the cipher suite can occur usually in a single round trip
- Cipher suite: encoding
  - IPsec uses a hierarchy “SA payload → proposals → protocols → transforms → attributes” to encode a cipher suite
  - TLS uses a single value to encode a cipher suite
- Supported transport protocols
  - IPsec protects the IP packet and does not care about the transport protocol
  - TLS runs on top of TCP
  - DTLS runs on top of UDP and other unreliable transport protocols.
  - TLS and DTLS are *different* protocols (although similar)
  - However, implementation of TLS might support DTLS easily, e.g. OpenSSL supports both

## IPsec vs. TLS (6)

- Cryptographic properties: Generation of keying material: equivalence and similarity in the key hierarchy:
  - The exchanged DH key ( $g^{xy} = KE_i * KE_r$ ) in IKEv2 plays a similar role as the pre-master key in TLS in the key hierarchy
  - SKEYSEED in IKEv2 plays a similar role as the master-secret TLS.
    - Both are respectively the second keys in the key hierarchy
    - In both cases fresh random numbers are involved in the computation
    - Keying material for a TLS session *key\_block* is derived from the master-secret
    - Keying material  $\{SK_d, \dots\}$  in IKEv2 is derived from SKEYSEED
- Cryptographic properties: Pseudo-random function
  - The pseudo-random functions prf is negotiated dynamically in IKEv2 during the IKE\_SA\_INIT exchange
  - The pseudo-random function PRF in TLS is defined in the standard and includes both HMAC\_MD5 and HMAC\_SHA-1
  - Dynamic negotiation of the PRF function will be included in TLS V1.2

## IPsec vs. TLS (7)

- Sequence numbers for replay protection
  - Both protocols provide replay protection
  - In TLS the sequence number is not carried explicitly, since a reliable transport is assumed with TCP.
  - In AH/ESP the sequence number is carried explicitly in the AH/ESP header
  - In DTLS, the sequence number is carried explicitly as well, since no reliable transport is assumed
- Protection against DoS attacks
  - IKEv2 provides protection against DoS attacks using cookies
  - TLS does not provide protection against DoS attacks, since it runs on top of TCP
  - DTLS provides DoS protection using cookies, similar to IKEv2



## IPSec vs. TLS (8)

- Administrative issues: root access
  - The configuration of IPSec policies on a host or a router requires administrative access (e.g. root)
  - TLS on the contrary can be used by any application and does not require administrative access
- Administrative issues: Issues with firewalls and NATs
  - IPSec has incompatibility issues with middleboxes such as firewalls und NATs, since such devices examine and possibly manipulate the IP header and the transport header
  - Note:
    - Incompatibility issues with NATs can be resolved by UDP encapsulation of the protected IP traffic
    - Incompatibility issues with firewalls are resolved if the firewall is co-located with the IPSec end point
  - TLS can operate well in the presence of firewalls and NATs (as long as the port numbers used by the application are not blocked by the firewall), since the IP header and the transport header are not modified by TLS



## IPSec vs. TLS (9)

- Applications
  - IPSec can be used to
    - build VPNs
    - protect a wireless connection in case no secure protocol at the link layer can be provided
    - protect any kind of data traffic between at the IP layer
  - TLS can be used for securing HTTP traffic and any other application's traffic
  - TLS is getting more popular to be used for VPNs due to its flexibility. Especially the administrative issues of IPSec mentioned above make TLS more attractive to use for building VPNs
    - e.g. the open source VPN solution OpenVPN is based on TLS
  - One can say that IPSec and TLS are concurrent solutions for building VPNs
  - Both IPSec and TLS in turn compete with other VPN solutions, e.g. OpenSSH or L2TP