ADVANCED INFORMATION SYSTEM SECURITY

Hoping to get a better grade this time around.

Fabio Lorenzato

October 6, 2024

Contents

Chapter 1

Transport Layer Security

Chapter 2

Transport Layer Security

TLS, or Transport Layer Security, was originally proposed by Netscape in 1995 as a way to secure communications between a web browser and a web server. It is the successor to SSL, or Secure Sockets Layer, which was first introduced by Netscape in 1995. The two terms are often used interchangeably, but TLS is the more modern and secure protocol.

The main goal of SSL was to create secure network channel, almost at session level(4.5), between two parties, to provide some security services that neither TCP nor IP provides:

- **peer authentication** based on asymmetric challenge-response authentication(the challenge for the service is implicit, while for the client is explicit). Server authentication is alway compulsory, while client authentication is optional and requested by the server.

- **message confidentiality** base on symmetric encryption

- **message integrity** and authentication based on MAC computed on the trasmitted data

- **replay, filtering and reordering attack protection** using implicit record numbers(the correct order of transmission is provided by TCP, for this reason the number is implicit). This number is used also in the MAC computation.

You can see the TLS packet structure in figure 2.1. The TLS handshake protocol is used to establish a new session or reestablish an existing session. The TLS change cipher spec protocol is used to trigger the change of the algorithms to be used for message protection, or most notably to pass from the previous unprotected session to a protected one. The TLS alert protocol is used to signal errors or signal the end of the connection. The TLS record protocol contains the generic protocols informations and its content depend of the state of the connection and the protocol it is tunneling.
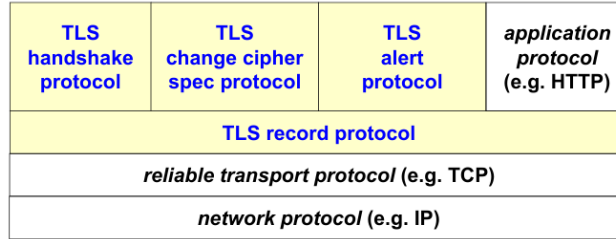
TLS handshake protocol	TLS change cipher spec protocol	TLS alert protocol	*application protocol* (e.g. HTTP)
TLS record protocol			
reliable transport protocol (e.g. TCP)			
network protocol (e.g. IP)			

Figure 2.1: TLS packet structure.

2.1 TLS session and connection

It is important to make a clear distinction between TLS session and connections.

TLS sessions a **logical association** between client and server, created via an handshake protocol and its shared between different TLS connections(1:N).

TLS connections are a **transient TLS channel** between client and server, which means that each connection is associated with only one specific TLS session(1:1).
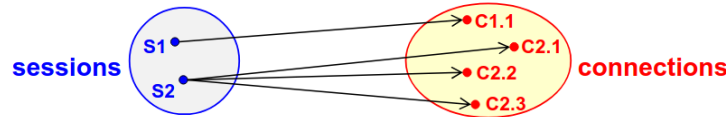
sessions S1 · · S2   **connections** ·C1.1 ·C2.1 · C2.2 · C2.3

Figure 2.2: TLS session and connection.

2.2 TLS handshake protocol

The TLS handshake protocol is used to establish a new session or reestablish an existing session. It's a critical part of the TLS protocol, because the channel pass from an unprotected state to a protected one. During this phase the two parts agree agree on a set of algorithms for confidentiality and integrity, exchange random numbers between the client and the server to be used for the subsequent generation of the keys, establish a symmetric key by means of public key operations (originally RSA and DHKE, but nowadays the elliptic curve versions of algorithms are used) and negotiate the session-id and exchange the necessary public keys certificates for the asymmetric challenge-response authentication.

2.3 Achieving Data protection

Data protection is achieved by using symmetric encryption algorithms to encrypt the data and Message Authentication Codes(MAC) to ensure the integrity of the data and the authentication of the sender.

Figure 2.3 shows how the data protection is achived in TLS using authenticate-then-encrypt approachm, but also encrypt-then-authenticate is possible.

The MAC is computed over the data(compressed or not), the TLS sequence number and the key used for the MAC computation. The padding is also part of the MAC computation to avoid those attacks that change the padding.

The MAC is then encrypted with the dedicated symmetric key and a suitable initialization vector(IV)
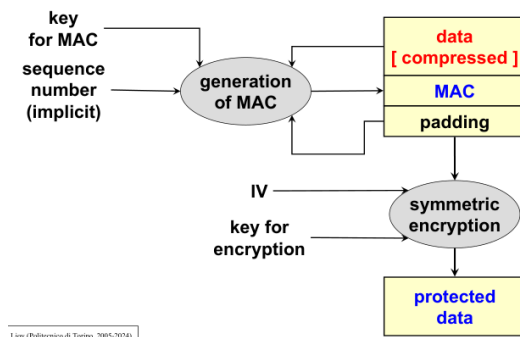

Figure 2.3: TLS data protection.

The keys are **directional**, so there are two keys(one for client to server and one for server to client) to protect against reuse of the sequence number in the opposite direction.

2.4 Relationship among keys and sessions

When a new session is created using the handshake protocol, a new **pre-master secret** is established using public key cryptography. Then from the session a new connection is created, which requires a random number to be generated, and exchanged between the client and the server. Those two values are combined via a KDF, usually HHKDF(HMAC-based key derivation function) to generate the **master secret**. This computation is done only once, and the secret is common to several connections.
The pre-master secret is then discarded, and the keys necessary for the MAC computation, encryption and, if necessary, IVs will be derived from the master secret.
You can notice that the master secret is common to any connections inside a session, but the per-connection keys are different every time. This is another important feature to avoid replay attacks, possible because numbering is per-connection. This solution also allows to reduce the cost of establishing new keys for each connection.
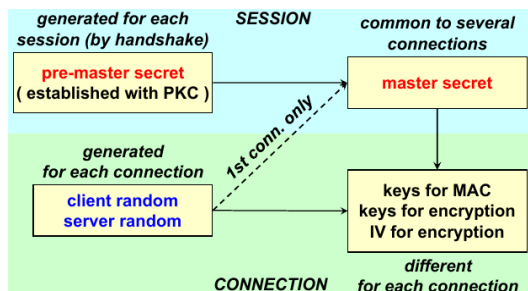

Figure 2.4: Relationship among keys and sessions.

2.5   Perfect Forward Secrecy

Since the keys are generated from symmetric crypto, if the private key used to perform encryption and decryption of the pre-master secret is compromised, all the previous communication can be decrypted because it is possible to derive the master-secret. This is only possible if the server has a certificate valid for both signature and encryption In this context, perfect forward secrecy is desirable.

> **Perfect Forward Secrecy** is a property of key-agreement protocols ensuring that the compromise of the secret key used for will compromise only current (and eventually future) traffic but not the past one

The most common way to achieve this is to use **ephemeral keys**, which are one-time asymmetric keys(used for key exchange). This means that the key pair used for key exchange is not a long term key pair, but a temporary one generated on-the-fly when necessary.
The ephemeral key needs to be authenticated, so only for this purpose the long-term key is used for signing the ephemeral key. This is done using DHKE instead of RSA because the latter one is really slow, while the former one is faster with the compromise of only using the established key for a certain number of session.
Let's now go over some considerations: if the temporary key is compromised, perfect forward secrecy of the communication is still valid because he can only decrypt the traffic exchanged using the temporary key. On the contrary, if the long-term key is compromised, no secret is really disclosed, because no traffic has been exchanged using it for encryption, but is still a problem for server authentication.

2.6   The protocol

The TLS handshake is always initiated by the client.

2.6.1   Client Hello and Server Hello

In version 1.2 the client sends a **Client Hello**, which contains:

- the SSL version preferred by the client, and the highest supported(2=SSL-2, 3.0=SSL-3, 3.1=TLS-1.0, ...)

- a 28 bytes pseudo-random number, which is the client random

- a session-id, which is *0* if the client is starting a new session, and *1* if the client is trying to resume a previous session

- a list of cipher suites supported by the client, in order to let the server choose the most secure one (the set of algorithms used for encryption, for key exchange, and for integrity)

- a list of compression methods supported by the client (supported only up to TLS 1.2)

And then a **server hello** is sent back, which contains:

- the SSL version chosen by the server, the highest one supported by both the client and the server

- a 28 bytes pseudo-random number, which is the server random

- a session identifier(session-id), which is a new one if the server is starting a new session, and the same as the client's if the server is resuming a previous session

- the cipher suite chosen by the server, the strongest common one between the client and the server

- the compression method chosen by the server
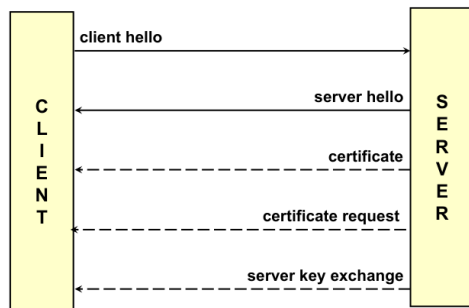
(a) The TLS handshake protocol(TLS 1.2).   (b) The TLS handshake protocol(TLS 1.3).
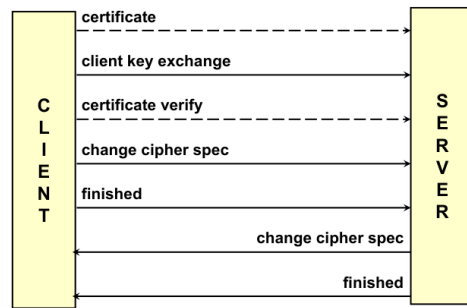
2.6.2 Cipher suite

A cipher suite is a string which contains the set of cryptographic algorithms used in the TLS protocol. A typical cipher suite consists of a key exchange algorithm, the symmetric encryption algorithm, and the hash function used for generating MACs. Some example of those are:

- SSL_NULL_WITH_NULL_NULL (no protection, used for the record protocol to be used in the handshake)

- SSL_RSA_WITH_NULL_SHA

- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5

- SSL_RSA_WITH_3DES_EDE_CBC_SHA

2.6.3 Certificates

After the initial exchange, the server is ready to authenticate itself.
The server sends its long-term public key certificate to the client for server authentication. Actually, the whole certificate chain, up to the root CA, must be sent. Furthermore , the subject of the certificate must match the server name.
Server authentication is implicit, because its private key is used to decrypt the pre-master secret, while client authentication is always explicit.

> The implicit server authentication is based on the fact that the MAC is computed trough the key derived trough knowledge of the private key, so only the server can compute it.

Optionally, the server can request a certificate from the client for client authentication. In this case the server specifies the list of trusted CA's, and the client sends its certificate chain. The browsers show to the users (for a connection) only the certificates issued by trusted CAs. If client certificate verification is required, an explicit request to send the hash computed over all the handshake messages before this one and encrypted with the client private key is sent to the client.

2.6.4 Key exchange

The key exchange is the most important part of the handshake protocol. If the server is using RSA for key exchange, the client generates a pre-master secret, encrypts it with the server's public key(which can be ephemeral or from its x.509 certificate) and sends it to the server. If RSA is no used, DHKE can be used to generate the pre-master secret, and in this case the server computes the value independently and the two parts can derive the master secret.
Another option is to use FORTEZZA, which is a key exchange algorithm based on DH.

2.6.5 Certificate verify

In case the server requested client authentication, the client will be required to send the certificate to prove that he is the owner of it's private key. The message to be signed is the hash of all the messages exchanged up to this point in the handshake protocol. This is to avoid replay attacks too.

2.6.6 Change cipher spec

The change cipher spec message used to trigger the change of the algorithms to be used for message protection. It allows to pass from the previous unprotected messages to the protection of the next messages with algorithms and keys just negotiated, thus is technically a protocol on its own and not part of the handshake. Some analysis even say that it could be removed from it.

2.6.7 Finished message

The finished message is the last message of the handshake protocol, and the first message protected by the negotiated keys and algorithms. It is necessary to ensure that the handshake has not been tampered with, and it contains contains a MAC computed over all the previous handshake messages (but change cipher spec) using as a key the master secret. Notice that the finished message is different for the client and the server, because the MAC is computed over different messages.
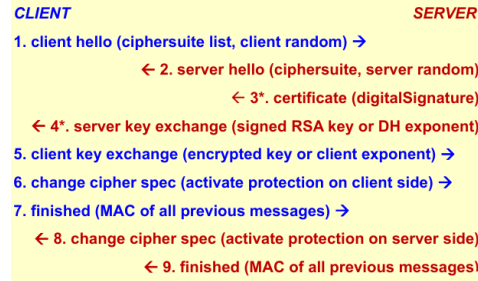This allows to prevent rollback man-in-the-middle attacks (version downgrade or cipher-suite downgrade)
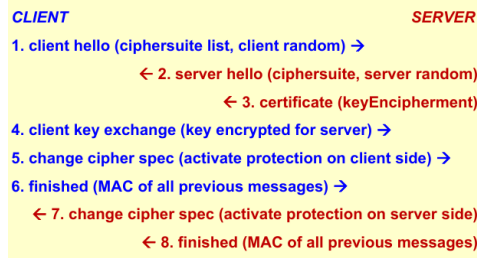
(a) TLS handshake(no ephemeral, no client authN).

(b) TLS handshake(no ephemeral, client authN).

(c) TLS handshake(ephemeral, no client authN).

Figure 2.6: TLS handshake protocol.

2.7 Setup Time

The setup time is the time required to establish a secure connection between the client and the server. TLS depends on TCP, so the TCP handshake must be taken into account. Then the TLS handshake is performed, meaning that typically 3 RTTs (1 for TCP and 2 for TLS) are required to establish a secure connection. Usually after 180ms the two parties are ready to send protected data(assuming 30ms delay one-way).

(a) TLS link teardown.
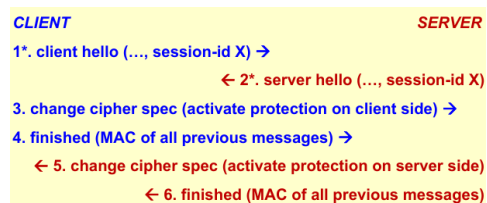
(b) TLS resume session.

Figure 2.7: TLS link teardown and resume session.

2.8 TLS versions

2.8.1 TLS 1.0

TLS 1.0, or SSL 3.1, was released in 1999. It is the first version of the protocol, and it is based on SSL 3.0. Previous version were using proprietary solutions, so the adoption of open standards was strongly encouraged.

2.8.2 TLS 1.1

TLS 1.1 was released in 2006, and it introduced some security fixes especially to protect against CBC attacks. In fact, the implicit IV is replaced with an explicit IV to protect against CBC attacks. Also protection against padding oracle attacks were introduces to reduce the information leaks. For this reason Passing errors now use the bad_record_mac alert message (rather than the decryption_failed one). Furthermore, premature closes no longer cause a session to be non- resumable.

2.8.3 TLS 1.2

TLS 1.2 was released in 2008, and it introduced some new features and improvements. The chipersuite also specifies the pseudo random function instead of leaving the choice to the implementation. The sha-1 algorithm was replaced with SHA-256, and its also added support for authenticated encryption, such as AES in GCN or CCM mode.

All the chipersuites tat use IDEA and DES are deprecated.

2.9 TLS attacks

2.9.1 Heartbleed

Heartbleed is a security bug in the OpenSSL cryptography library, which is a widely used implementation of the TLS protocol. It was able to exploit the fact that the heartbeat extension keeps the connection alive without the need to negotiate the SSL session again. The attacker could send a heartbeat request, but the length of the response is much longer(up to 64KB) than the actual data sent by the client. This attack could then allow to leak memory contents.

2.9.2 Bleichenbacher attack

2.9.3 Other attacks against SSL/TLS

Some other attacks against SSL/TLS are CRIME, BREACH, BEAST and POODLE.
Crimes is an attack against the compression algorithm used in SSL/TLS, which by injection chosen plaintext in the user requests and then measure the size of the encrypted traffic, an attacker could recover specific plaintext parts exploiting information leaked from the compression, and this is part of the reason why the compression is deprecated in TLS 1.3.
BREACH is an attack against the HTTP compression to deduce a secret within the HTTP response provided by the server.
POODLE, or Padding Oracle On Downgraded Legacy Encryption, is an attack that exploits the fact that SSL 3.0 uses a padding scheme that is vulnerable to a padding oracle attack, by acting as a MITM. This is done by exploiting SSL-3 fallbacks to decrypt data.
For all those reasons SSL-3 has been disabled on most browsers, but its still needed for some browsers, for example IE6 by Microsoft, which is outlasting its expected life.

2.9.4 FREAK attack

FREAK, or Factoring RSA Export Keys, is an attack that exploits the downgrades on TLS to explort-level RSA keys to a factorizable bit length(512 bits). It is also possible to carry this out by downgrading the symmetric key too and then perform a brute force attack(40-bit).
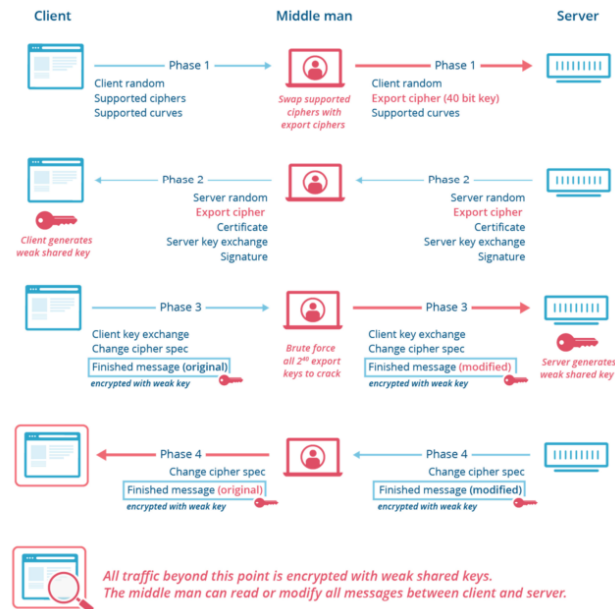

Figure 2.8: FREAK attack.

2.10 ALPN extension

The ALPN extension, or Application-Layer Protocol Negotiation, is an extension that allow to negotiate the application protocol to speed up the connection creation, avoiding additional round- trips for application negotiation. It is used to negotiate the protocol to be used on top of the TLS connection, such as HTTP/2, SPDY, or QUIC. The extension is inserted in the client hello message, by setting the ALPN flag to true and providing a list of supported protocols. This is also useful for those servers that use different certificates for the different application protocols.

2.11 TLS False Start

TLS False Start is another extension that allows the client can send application data together with the ChangeCipherSpec and Finished messages, in a single segment, without waiting for the corresponding server messages. The biggest advantage of using this is the reduction of the latency to 1 RTT. In theory this should work without changes, but to use this in Chrome and Firefox they require the ALPN and the Forward Secrecy enabled, while Safari requires forward secrecy.

2.12   The TLS downgrade problem

In theory, when negotiating the TLS version to be used, the client sends (in ClientHello) the highest supported version, while the server notifies (in ServerHello) the version to be used (highest in common with client)

2.12.1   TLS Fallback Signalling Cipher Suite Value (SCSV)

The TLS Fallback SCSV is a special value that is used to prevent the protocol downgrade attacks. It do so by sending a new (dummy) ciphersuite value(TLS_FALLBACK_SCSV) which is sent by the client when opening a downgraded connection)as last in the chipersuite list.

2.13   TLS session tickets

We know that session resumption is possible with TLS, but the server needs to keep a cache of session IDs, which may become very large for high traffic servers. For this reason, the TLS session tickets were introduced, which are an extension allowing the server to send the session data to the client encrypted with a server secret key. This data is stored by the client, which will send it again when it wants to resume a session. This allows to move the cache to the client side.

2.14   The Virtual Server Problem

Nowadays, virtual servers are very common in web hosting, because they allows to have different logical names associated with the same IP address(ie: home.myweb.it=10.1.2.3, food.myweb.it=10.1.2.3). This is easy to manage in HTTP/1.1 but quite troublesome with HTTPS, because TLS is activated before the HTTP request is sent, which makes it difficult to know which certificate should be provided in advance. The solutions are quire simple:

- use a wildcard certificate, which is a certificate that is valid for all the subdomains of a domain (ie: *.myweb.it)

- use the SNI (Server Name Indication) extension, which is an extension that allows the client to specify the hostname of the server it is trying to connect to, allowing the server to provide the correct certificate. This is sent in the ClientHello message.

- provide a certificate with a list of servers in subjectAltName

2.15   TLS 1.3

TLS 1.3 was released in 2018, and it introduced some new features while solving some of the most common problems of the previous versions:

- reduce the handshake latency in general

- encrypting more of the handshake (for security and privacy)

- improving resiliency to cross-protocol attacks

- removing legacy features

We will now go over the main changes in TLS 1.3.

2.15.1 Key exchange

In this version, the support for static RSA and DH key exchange was removed for many reasons:

- it does not implement forward secrecy

- its difficult to implement correctly

Now Diffie-Hellman ephemeral (DHE) and Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) are the only key exchange methods supported, with some required parameters.

2.15.2 Message protection

2.15.3 Digital signature

2.15.4 Ciphersuites

- TLS_AES_128_GCM_SHA256

- TLS_AES_256_GCM_SHA384

- TLS_CHACHA20_POLY1305_SHA256

- TLS_AES_128_CCM_SHA256

- TLS_AES_128_CCM_8_SHA256 (deprecated)

2.15.5 EdDSA

EdDSA, or Edwards-curve Digital Signature Algorithm, is a digital signature scheme using the EdDSA signature scheme. The EdDSA scheme, unlike standard DSA doesnt require a PRNG, which could in some cases leak the private key if the underlying generation algorithm is broken or predictable.
EdDSA picks a nonce based on a hash of the private key and the message, which means after the private key is generated there's no more need for random number generators. Another advantage is that the EdDSA is faster in signature generation and verification than the standard DSA.

2.15.6 Other improvements

2.15.7 HKDF in TLS 1.3
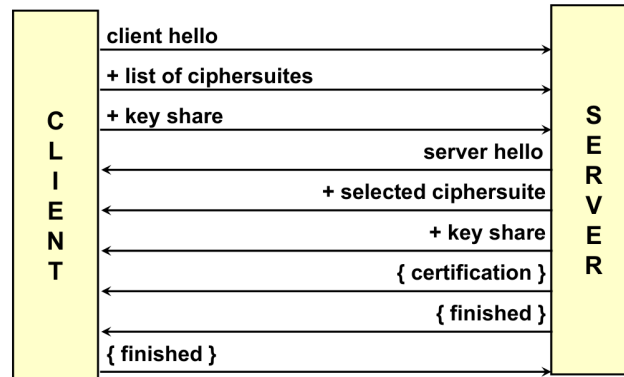
2.15.8 TLS-1.3 handshake

Figure 2.9: TLS 1.3 handshake.

2.15.9 Pre-shared keys

In TLS, the pre-shared key (PSK) replaces the session ID and session ticket. PSKs are agreed upon during a full handshake and can be reused for multiple connections. They can be combined with (EC)DHE to achieve forward secrecy, where the PSK is used for authentication and (EC)DHE for key agreement. While PSKs can be generated out-of-band (OOB) from a passphrase, this is risky due to the potential lack of randomness, making brute-force attacks feasible. Therefore, using OOB PSKs is generally discouraged.

2.15.10 0-RTT connections

In TLS 1.3, when using a pre-shared key (PSK), a client can send "early data" with its initial message, which is protected by a specific key. However, this approach lacks forward secrecy because it relies solely on the PSK and could be vulnerable to replay attacks. While some complex mitigations exist, they are particularly challenging for multi-instance servers.

2.15.11 Incorrect share

In TLS 1.3, if a client sends a list of (EC)DHE groups that the server does not support, the server responds with a HelloRetryRequest, prompting the client to restart the handshake with different groups. If the new groups are also unacceptable, the handshake will be aborted, and the server will send an appropriate alert.