Transport Layer Security: TLS

COMPUTER NETWORKS A.A. 24/25

Leonardo Maccari, DAIS: Ca' Foscari University of Venice,
leonardo.maccari@unive.it

Venice, fall 2024

Sect. 1 Secure Protocols

Securing Internet Communications

- We have studied efficient ways to achieve a bi-directional secure communication
- This involves the use of:
  - Hash functions to guarantee integrity
  - Symmetric key encryption to guarantee fast encryption/decryption
  - Public Key encryption to guarantee authentication and non-repudiation
  - Certificates to obtain a Web of Trust
- At what layer do we insert security?

Pros and Cons

- Depending on the layer you use, there are pros and cons
- The lower the layer, the more information are encrypted, however things may not work at all
- The upper the layer, the less support you need from the Operating System, and the easier it is to support new applications, but more information is leaked
- Plus, we know that to use cryptography between two entities Alice and Bob we need what is called a *security association*, which means that Alice and Bob must exchange some key before using a certain security service.

Questions: at what layer can we encrypt?

Let's focus on secrecy. To answer the questions on next slide, ask yourself for each of them:

- What information is actually encrypted?
- What information is not encrypted?
- Can the entities involved set-up a security association?
- What **stops working** if they can't do it?

Questions: at what layer can we encrypt?

- Can we encrypt the IP header?

Questions: at what layer can we encrypt?

- Can we encrypt the IP header?
- Can we encrypt the TCP header?

Questions: at what layer can we encrypt?

- Can we encrypt the IP header?
- Can we encrypt the TCP header?
- If we provide an encryption service on top of TCP (so TCP is not encrypted), what information is available to an attacker?

Answers to Questions

Can We encrypt the IP Header?

- Not really. The IP header is needed for routing, and routers are unknown to Alice and Bob.
- Alice can not negotiate a security association with the routers.
- So there is no way to send an IP packet with an encrypted header and have it routed.
- There are some exceptions...

Answers to Questions

Ways to encrypt the IP header

- Encrypt at MAC layer. This can be done, but it applies encryption only in the LAN.
- For instance, a wireless AP receives encrypted dataframes, but then it decrypts them and send them on the Internet in the clear
- Alternatively, use a Virtual Private Network, that encapsulates an IP packet (encrypted) inside another IP packet (with headers in the clear).
- For that you need to be the manager of both Alice and Bob computers (or networks).

Answers to Questions

Can We encrypt the TCP Header?

- Technically, yes. TCP is an end-to-end protocol, so it involves only Alice and Bob, and they could set-up a security association.
- However, unfortunately the Internet relies on middleboxes: hosts that need to look at the TCP header.
- If TCP header is encrypted, a NAT does not work anymore (neither firewalls or traffic shapers).
- A VPN encrypts the TCP traffic, but again, it is not of general use.

Answers to Questions

If we provide an encryption service on top of TCP (so TCP is not encrypted), what information is available to an attacker?

- TCP headers contains the port number, so a passive attacker (Eve) is able to know the destination host and the protocol
- Eve can then know that Alice is watching a movie, or texting with someone

Transport Layer Security: TLS

- Since the original Internet protocols were designed without any security principle in mind, we had to add security on top of the existing protocols and running code
- The most manageable solution to provide widespread use of cryptography was to introduce it on top of TCP
- This provided a reasonable trade-off between ease of application (and in fact, nowadays almost all TCP traffic is encrypted) and the leakage of information from the lower layers

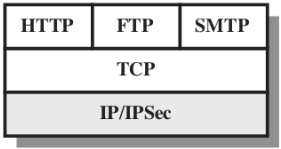
Security in the Network Stack

However, security can be introduced at any layer in the network stack (figure from Tannenbaun's book)

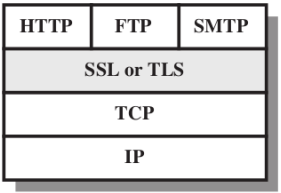
a) At Network Layer: a VPN

b) At the transport layer: TLS

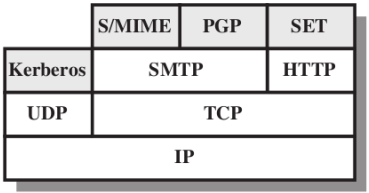
c) At the application layer: PGP



(a) Network Level



(b) Transport Level

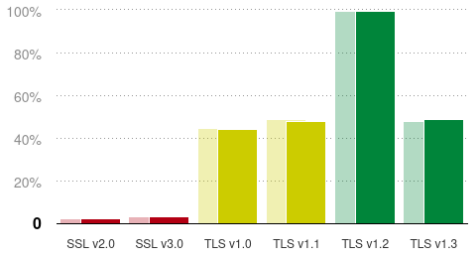


(c) Application Level

SSL/TLS

- SSL (secure socket layer) introduced by Netscape in 1995 adds security services "on top of" TCP
- Over the years it was standardized and renamed TLS (Transport Layer Secuity)
- There are several versions of SSL, all deprecated, and three versions of TLS, with a still not marginal penetration of the old ones (graph from SSL Labs, top 150.000 Alexa websites).
- SSL and TLS are names we still use interchangeably

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]



The '*S*' protocols

- While many application layer Internet protocols were designed before the advent of these tools and were indeed insecure, we have been replacing them with secure versions
- The secure protocols generally use the same name of the insecure protocol, with a final '*S*', that run on a different port:
  - HTTP becomes HTTPS: from port 80 to port 443
  - POP3 becomes POP3S: from port 110 to port 995
  - IMAP becomes IMAPS: from port 143 to port 993
  - ...

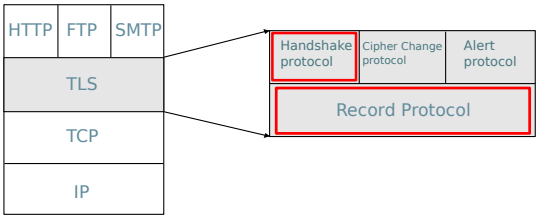
The '*S*' protocols

- The protocols themselves did not have to change, TLS was introduced smoothly and gradually
- Note that this is a big win for the layered Internet design
- There is also a version for UDP (DTLS RFC6347) that is not very much used

SSL/TLS

- TLS provides the following security services to the upper layers:
  - Origin authentication: the server and the client can identify themselves with the use of X.509 certificates (or even with a Pre-Shared Key, extremely unusual)
  - Secrecy: after the authentication a *key agreement* phase follows in which a shared key is generated to encrypt the traffic using symmetric key encryption.
  - Integrity: another symmetrical key is generated that guarantees integrity using an HMAC
- Until TLS 1.2 the standard was progressively enriched, but the principles were more or less maintained. Starting from version 1.3 many things were changed.

TLS 1.2 sub-stack



TLS contains several protocols, we will briefly look at two of them

- TLS handshake protocol: it is used to negotiate keys and security parameters
- TLS record protocol: the security transformation that is applied to the data

TLS 1.2 Encapsulation in the Record Protocol

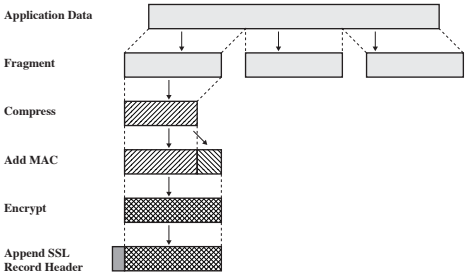


Figure 17.3 SSL Record Protocol Operation

1. The stream is fragmented
2. Compressed (optional, removed in 1.3)
3. a MAC is added
4. It is encrypted
5. a header is added

Note TLS 1.2 used MAC-then-Encrypt, which is deprecated now.

TLS 1.2 Record Protocol (extracted from RFC 5246)

The TLS Record Protocol provides connection security that has two basic properties:

- **The connection is private**. Symmetric cryptography is used for data encryption (e.g., AES). The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol).
- **The connection is reliable**. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA-256, etc.) are used for MAC computations.

TLS Handshake Protocol (RFC5246)

- The TLS Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.
- Some of the things that must be negotiated:
  - What crypto functions will be used: RSA? Elliptic Curve Cryptography?
  - What hash functions will be used: SHA-256? SHA-384?
  - How do we negotiate the symmetric key: DH? RSA exchange?
- The handshake is **critical** because, by definition, it happens before any key was negotiated, so a MiTM can modify it.

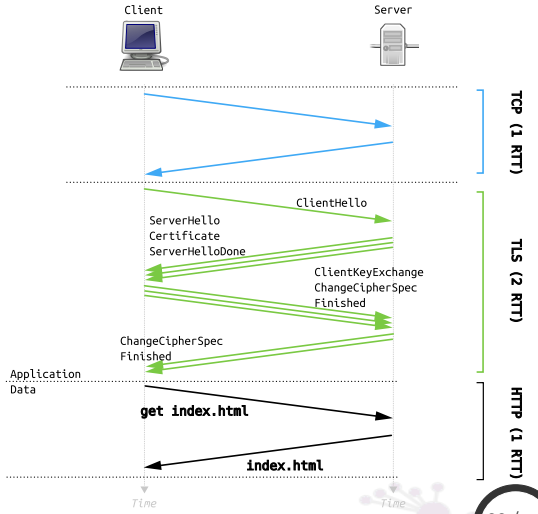
TLS Handshake Protocol (RFC5246)

The TLS Handshake Protocol provides connection security that has three basic properties:

- The peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA, DSA, etc.). This authentication can be made optional, but **is generally required for at least one of the peers**.
- The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, **even by an attacker who can place himself in the middle of the connection**.
- The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the parties to the communication.

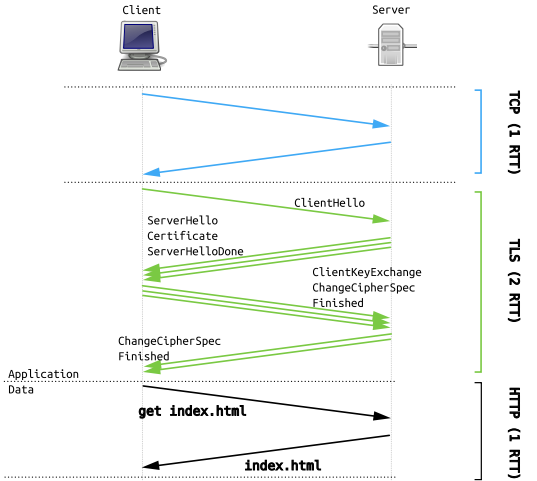
TLS 1.2 Handshake Protocol

1. The first three messages are the 3-way handshake of TCP
2. This includes the `ClientHello` message, that also plays the `ACK` function of TCP handshake
3. Certificates are exchanged in the `Certificate` messages.



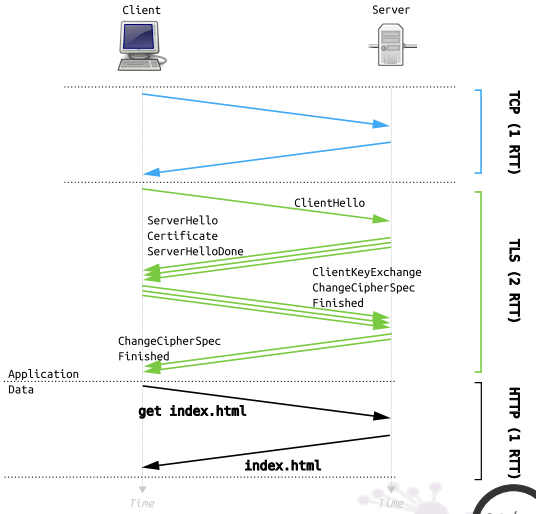
TLS 1.2 Handshake Protocol

4. In `HTTPS` the `Certificate` message contains **the whole tree of certificates, from the root CA down to the website certificate**
5. The `ChangeCipher Spec` message means that key negotiation is over. The next messages will be encrypted and authenticated.
6. The `Finished` message is the first one encrypted and authenticated (so you won't see it in dumps). It contains a hash of all the previous messages to verify (a posteriori) they were not modified by a MiTM.



TLS 1.2 Handshake Protocol

7. For 3 RTT not a single useful byte is received by the client
8. **The first useful byte arrives after 4 RTT** (the packed marked as `index.html`)
9. The connection set-up is way longer than plain TCP.



One way Authentication

Authentication is Normally Unidirectional

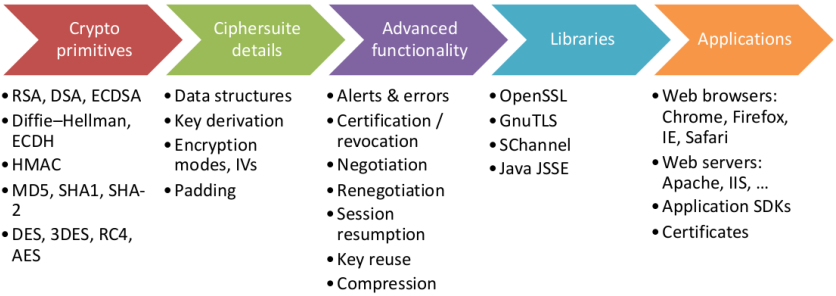
- Note that normally you, as an Internet user, do not need a valid certificate
- With HTTPS, most of the times only the web server delivers a valid certificate to the browser
- The browser is sure of the identity of the server, but not the other way around, which is generally not required.
- However, the client and the server can still negotiate a shared key to make the TCP connection secure in both directions with symmetric key cryptography.

Mutual Authentication: mTLS

- Recently, the paradigm of microservices and cloud-native applications emerged.
- This means that in a cloud system, you have a mesh of services that interact with each other in the same infrastructure
- There, you can better use mutual authentication, because the owner of the infrastructure controls everything.
- In this context the use of mTLS is becoming the best practice to enforce *zero trust* systems: you don't trust a service because it is placed in your infrastructure, but because it uses a valid certificate.

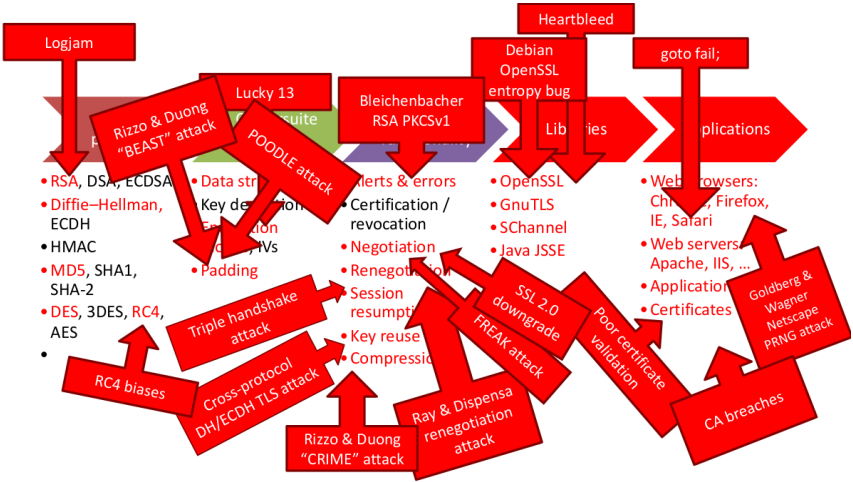
TLS 1.2 Components

The security of TLS depends on many different components. The attack surface of TLS is large.



TLS 1.2 Vulnerabilities

And In fact, there is a long list of vulnerabilities related to TLS.



TLS 1.2 was Severly Bugged

- TLS 1.2 suffered from a large number of critical bugs, hiding in the most complex features of the protocol. These bugs were patched from vendors and with new revisions of the standard.
- TLS 1.3 is a complete rewrite of the protocol, which simplifies TLS, removing a lot of the rarely unused features
- Removing features seems not an improvement, but features increase the attack surface, and if they are rarely used, it makes no sense to maintain them

TLS 1.2 Vulnerabilities (RFC 7457)

Table of Contents

TLS 1.3 Modifications

- Only 5 encrytpion/authentication algoritm, with only one block scheme (AEAD)
- A simplified initial handshake that can work in only one RTT
- The security session can be resumed on a different connection with 0 RTT
- Forward Secrecy is mandatory
- and more...

Forward Secrecy

- Forward Secrecy is a feature of specific key agreement protocols ensures that session keys will not be compromised even if the private key of the server is compromised.
- In practice, let's say that :
  1. Eve intercepts a TLS communication between Alice and Bob and saves its content, even without being able to decrypt it
  2. Later on, Eve breaks into Bob's server and steals the Private Key.
  3. Eve should not be able to decrypt the traffic she saved in advance
- TLS 1.2 does not mandate Forward Secrecy, while TLS 1.3 does.
- Forward Secrecy normally implies using *ephemeral* public keys: keys that are generated during the handshake and deleted after right after they were used.

Example Protocol 1

1. Server $\rightarrow$ Client: Server Certificate
2. The client verifies the Certificate
3. Client $\rightarrow$ Server: The client generates a random number R, transmits it encrypted with the public key of the server
4. A symmetric key is generated by both with a hash function: $K = hash(R)$
5. From now on the server and the client use AES and HMAC with K
6. Now the Client uses a password method to authenticate

Is this Forward secure?

Example Protocol 2

1. Server $\rightarrow$ Client: Server Certificate
2. The client verifies the Certificate
3. Client $\rightarrow$ Server: The client initiates a Diffie-Hellman session with some random ephemeral key
4. Server $\rightarrow$ Client: The server completes the Diffie-Hellman session with some random ephemeral key
5. Both now own a key K
6. From now on the server and the client use AES and HMAC with K
7. Both server and client delete the ephemeral keys
8. Now the Client uses a password method to authenticate

Is this Forward secure?

Answers

1. No. If the attacker sniffs and saves the whole session, and only later on enters in possession of the private key of the server, `K` is revealed, and thus all the saved traffic can be decrypted. In fact this was dropped from TLS 1.3
2. Yes. If the attacker enters in possession of the private key of the server, this was not used to generate `K`. Some ephemeral key was used that is not available anymore. In fact this is mandatory in TLS 1.3
3. Note that without mTLS it always takes a password method to authenticate the client.

Sect. 2 Socket Programming

Socket Programming Introduction

- The POSIX standard includes the base interfaces for any language that wants to implement a transport layer socket.
- Typically for performance reasons these are implemented in C, but similar interfaces are available in any language
- We will briefly review normal sockets and TLS sockets

Socket Programming

↳2.1 Berkeley Sockets

Network Sockets

- A socket is a standard interface, defined by POSIX between applications and layer 4. The terms *network socket*, *Berkeley socket* or *BSD socket* are used interchangeably
- That is, when an application needs to send or receive data from the network, it uses an API that creates, manages and uses sockets.

UDP and TCP sockets in C Language

- Professor Campbell from Dartmouth has an excellent page describing how to program network sockets in C:
  `https://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html`
- We will look at that instead than copying it here...

improving the code

- Andy Campbell is an excellent professor and a very nice guy...
- ...however :-)
- Do the following:
  1. Run the server and then connect with the client
  2. Kill the server (CTRL+C) and re-launch it immediately
  3. Kill the client and re-launch it
- can the client re-connect? if not, try to find the problem, and fix the code.
- And then, find an overall motivation, solution, next week

Hints

- `netstat -tpna` tells the state of all ongoing connections in your POSIX machine
- essentially, it dumps the list of the TCB
- Run it in another terminal right after killing the server. Is the server connection dead?

Socket Programming

↳2.2 TLS Socket

Code for a TLS/SSL socket

- In a similar way we can set-up a TCP socket that supports TLS
- We are going to look at the code from the OpenSSL examples: `https://wiki.openssl.org/index.php/Simple_TLS_Server`
- Some example functions that deserve a little explaining

  `TLS_server_method` This returns a method that can be used in a TLS connection. It is used when all TLS versions need to be supported, while instead one can restrict a specific version using `TLSv1_2_server_method`. If none is specified, all are allowed and the higher layer protocols will decide on the best one

  `SSL_CTX_new` creates a new `SSL_CTX` object, which holds various configuration and data relevant to SSL/TLS for session establishment

Code for a TLS/SSL socket

- In a similar way we can set-up a TCP socket that supports TLS
- We are going to look at the code from the OpenSSL examples: `https://wiki.openssl.org/index.php/Simple_TLS_Server`
- Some example functions that deserve a little explaining

  `SSL_CTX_use_certificate_file` Load a file containing a certificate
  `SSL_CTX_use_PrivateKey_file` Load a file containing a key
  `configure_context(ctx)` create the context data struct that will be used by all connections

Code for a TLS/SSL socket

- In a similar way we can set-up a TCP socket that supports TLS
- We are going to look at the code from the OpenSSL examples: `https://wiki.openssl.org/index.php/Simple_TLS_Server`
- Some example functions that deserve a little explaining

  `ssl = SSL_new(ctx)` The context needed for this connection
  `SSL_set_fd(ssl, client)` Bind the context to this connection
  `if (SSL_accept(ssl) <= 0)` Do the TLS handshake
  `SSL_write(ssl, reply, strlen(reply))` Send test data