**Le Protocole SSL**

**When to use SSL**

A common problem with applications in production is securing data. We have data that is being sent between applications across a untrusted network. It is commonly used in web browsers but can be used with any protocol that uses TCP as the transport layer. For instance, you can use SSL to authenticate servers and clients and then use it to encrypt messages between the authenticated parties.

**SSL Handshake**

Before we continue let’s have a look at SSL Handshake picture below. Starting with the three blue lines, these represent the [three-way handshake](https://en.wikipedia.org/wiki/Handshaking#TCP_three-way_handshake). If everything went well we have a connection established between Client and Server. Beware we have a connection, but we are not secure in terms of data, this means we currently communicate plain-text over the wire.

Now the Client will start with a **CLIENT HELLO** message to the server with the IP and port number obtained during the TCP handshake. When the Server receives this message the following data is known:

* SSL/TLS Protocol Version
* Session ID
* List of Cipher Suites
* List of **CLIENT HELLO** Extensions.

The server will now check if it will support the Protocol Version and Cipher Suites provided.

The server will then respond with a **SERVER HELLO** message:

* SSL/TLS Protocol version.
* A Cipher Suites from the list of Cipher Suites provided by Client.
* Certificate of the server, without the private key :)
* List of **SERVER HELLO** Extensions.



When the client receives the **SERVER HELLO** it will do a **SERVER AUTHENTICATION.**The following checks are done by the client:

* Is today’s date within the validity period?
* Is the issuing Certificate Authority (CA) a trusted CA?
* Does the issuing CA’s public key validate the issuer’s digital signature?
* Does the domain name in the server’s certificate match the domain name of the server itself?

After this the server is **authenticated** the client will continue with the following steps¹:

* At this point, we have everything we need to generate the actual secrets. The client creates the pre-master secret for the session and encrypts it with the server’s public key(received earlier)and sends the encrypted pre-master secret to the server.
* If the server had requested Client authentication(this is optional), the server will try to authenticate the client. If the client cannot be authenticated, the session ends. If the client is successfully authenticated, the server uses its **private key** to decrypt the **pre-master secret** and then performs a series of steps (the client will do the same, starting from the same pre-master secret) to generate the **master secret**.
* The client and the server use the master secret to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its **integrity.**
* The **CLIENT** & **SERVER** now will send each other a message informing that future messages from now on will be encrypted with the session key. It then sends a separate (encrypted) message indicating that its portion of the handshake is finished.
* The SSL Handshake is done. The Client and the Server send each other messages which are **encrypted**using the session keys generated in the previous steps.

**CipherSuite**

A CipherSuite is a suite of cryptographic algorithms used by an SSL connection. A suite contains three different algorithms:

* The key exchange and authentication algorithm used during the handshake
* The encryption algorithm used to encipher the data
* The MAC (Message Authentication Code) algorithm, used to generate the message digest

A *Message digest*or *one-way hash* is a fingerprint of the data*.*If the data changes, the fingerprint changes in ways you cannot predict. So if you alter the data, you have no idea what the corresponding digest will be for the altered data. The content of the hashed data cannot be determined from the hash. This is why it is called *one-way*.

There are several options for each component of the suite, but only certain combinations are valid when specified for an SSL connection. The name of a valid CipherSuite defines the combination of algorithms used. For example, the CipherSuite TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256 specifies:

* The RSA key exchange and authentication algorithm
* The AES encryption algorithm, using a 128–bit key
* The SHA256 MAC algorithm

## **Certificate exchange**

If public keys are sent directly by their owner to another entity, there is a risk that the message could be intercepted and the public key substituted by another. This is also known as a man-in-the-middle attack. To avoid this problem, you should exchange the public keys through a trusted third party, giving you a strong **confidentiality** that the public key really belongs to the entity with which you are communicating. Instead of sending your public key directly, you ask the trusted third party (CA) to incorporate it into a digital certificate. The trusted third party that issues digital certificates is called a Certificate Authority (CA)

# Certificates

The certificates that we previously mentioned are in fact X.509 certificates. X.509 is a standard that defines the format of [public key certificates](https://en.wikipedia.org/wiki/Public_key_certificate). An X.509 certificate is composed of two parts of information:

1. Identifying information, such as your name, address and organisation
2. The public key

With this piece of information anyone who obtains your X509 certificate from a CA, can verify your identity and encrypt data for you. Only you are then able to decrypt and read the message.

# Conclusion

* SSL can be used to secure communication between two entities.
* The Client initiates the SSL Handshake.
* The Server selects the Cipher.
* The Server gives the X.509 certificate and client authenticates the server. Its optional that the server requires the client X.509 certificate.
* After a successful SSL handshake, the server and client can communicate encrypted messages. The messages can only be decrypted by the receiver.

[1] Wm. Arthur Conklin, Principles of Computer Security, (McGraw Hill Professional, 2016), 533