Understanding TLS/SSL encryption technology

TLS is, in few words, a cryptographic protocol that provides end-to-end security of any data sent between applications over the Internet. Most common scenarios are in secure web browsing,. However, it can and should also be used for other applications such as e-mail, file transfers, video/audioconferencing, instant messaging and voice-over-IP, etc, etc.

It is important to understand that TLS does not secure data on end systems. It does ensure the secure delivery of data over the Internet, avoiding possible eavesdropping and/or alteration of the content being sent (in-transit)

TLS is normally implemented on top of TCP in order to encrypt Application Layer protocols such as HTTP, FTP, SMTP and IMAP, although it can also be implemented on UDP, DCCP and SCTP as well.

As you can imagine, to protect the data, TLS relies on cryptography technology in order to encrypt/decrypt the in-transit data being sent over internet. We will be covering terms and concepts such as symmetric and asymmetric cryptography, public key infrastructure (PKI), X.509 digital certificates, etc.

Notice that, to be able to setup, configure and apply NGFW to your network, you need to understand how TLS connections (such as https) works, including digital X.509 certificates configuration and deployment.

Encryption and types

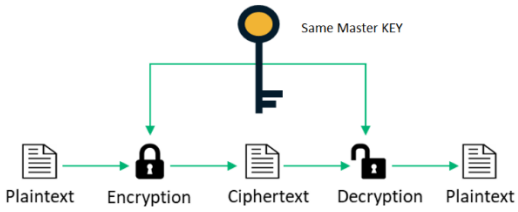
The idea behind of encryption is precisely to conceal or hide (encrypt) the information we want to send from A to B in an “insecure channel”. An insecure channel is a channel or a path where we cannot guarantee that no one will intercept, steal, or modify (eavesdrop) the information transmitted from A to B and the other way around.

Here is where cryptography takes place allowing us to secure the information travelling through an insecure channel, by encrypting (concealing) the original data toward its destination. Once received by the receiver, the receiver will be able to decrypt the received encrypted data to its original value.

Today, there are two types of encryptions: **Symmetric and Asymmetric**

Symmetric Encryption

In order to encrypt/decrypt a chunk of data, symmetric encryption engines use exactly the same key to encrypt and to decrypt the data. This is normally called the "**Master key**" or "**shared secret**"

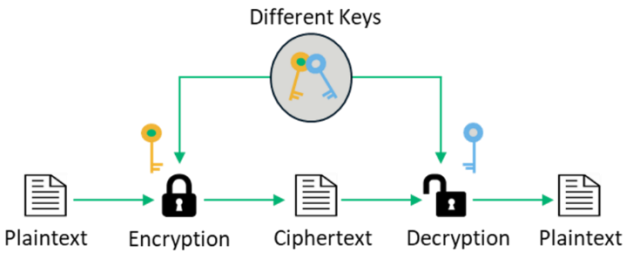


Though it sounds like an easy way to achieve encryption, the main problem here is **distributing** the **master key** between the two parties over an insecure channel. This **master key exchange** process will be address in this blog, using different techniques, including one technique explained in the next chapter: Asymmetric encryption, and other brilliant techniques.

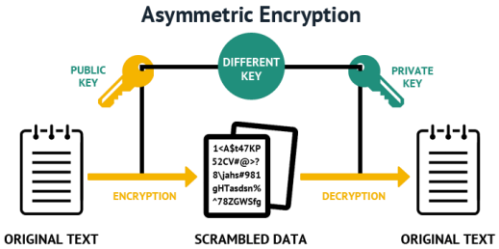
Some of more common symmetric encryption algorithms are: AES128, AES256, Serpent, Camelia, etc.

Asymmetric Encryption

Asymmetric encryption/cryptography, uses a pair of related keys with a special relationship: Any data that you **encrypt** with one of the pairs keys can be **decrypted ONLY BY** the other pair key, and **the other way around**.



We normally refer to this pair keys as **Public key** and **Private key**. The public key can be shared with everyone, whereas the **private key must be kept secret**



Now, it seems we have resolved the problem of distributing the master key needed using symmetric encryption over insecure channels. Well, yes and no.

Asymmetric encryption solves the problem of key distribution by using public keys for encryption and private keys for decryption (or the other way around). The tradeoff, however, is that asymmetric encryption systems are very slow by comparison to symmetric systems and require much more computing power because of their vastly longer key lengths.

Symmetric encryption algorithms are much faster and require less computational power, but their main weakness is key distribution. Because the same key is used to encrypt and decrypt information, that key must be distributed to anyone who would need to access the data.

So, which one does TLS use for encryption as both have pros and cons? The answer is both, TLS uses asymmetric and symmetric encryption, and not only for encrypting data but for authenticating the parties involved in the connection. Here is where X.509 certificates take action.

X.509 certificates

An X.509 certificate is a digital certificate based on the widely accepted International Telecommunications Union (ITU) X.509 standard, which defines the format of public key infrastructure (**PKI**) certificates. A X.509 certificate is architected using a **key pair** consisting of a related public key and a private key. As we have mentioned above, this key pair is precisely the pair of keys that are used in Asymmetric Encryption, where we choose one to be public and other to be private. Remember, anything that you encrypt with one of the keys can be only decrypted by the other key, and the other way around.

As you can imagine, the **public part** of the key pair, as its name implies, is public and can be **freely distributed**. The **private key** must be **kept safely**, and often encrypted using a master key in symmetric encryption mode, so nobody can use it even if is stolen.

In addition to the public key, **the X.509 certificate** contains other information that represents an **identity** (a hostname, or an organization, or an individual), therefore **the x.509 certificate binds this identity to the contained public key**.

The structure of an X.509 v3 digital certificate is as follows:

Certificate

- Version Number
- Serial Number
- Signature Algorithm ID
- Issuer Name
- Validity period
  - Not Before
  - Not After
- Subject name
- Subject Public Key Info
  - Public Key Algorithm

Subject Public Key ← This is the public KEY

Issuer Unique Identifier (optional)

Subject Unique Identifier (optional)

Extensions (optional)

...

Certificate Signature Algorithm

Certificate Signature

Now, the public key is bound to an identity represented in Subject name which is an string with the following format:

- country (countryName, C),
- organization (organizationName, O),
- organizational unit (organizationalUnitName, OU),
- distinguished name qualifier (dnQualifier),
- state or province name (stateOrProvinceName, ST),
- common name (commonName, CN)
- serial number (serialNumber).

i.e.

Subject: C = US, ST = California, L = Mountain View, O = Google LLC, CN = *.google.com

So, this identity belongs to Google, in US country, State California and a **common name** as ***.google.com.**

So, now we can use this certificate to serve two purposes: **Distribute my public key** AND **authenticate myself (my identity) to others**. Having this in mind, any computer/mobile/device out there, when receiving my certificate, it will own my public key and my Identity, therefore, it will be able to send me encrypted information using the public key, and only me will be able to decrypt it (with the private key). In addition, the certificate has my identity inside, so the device receiving it will be able to trust this identity and be sure it is exchanging information with the right identity.

Everything sounds perfect, but there are still some gaps here. As we have mentioned above, asymmetric encryption via public/private keys is much slower than symmetric encryption (x4 or more) so it is not viable. In addition, we haven't explained how the device receiving the certificate can really trust the identity inside the certificate. At the end, **the X.509 certificate is just a piece of text** received over an **insecure channel**.

Let's resolve first the second issue: To be able to trust/authenticate a received certificate, we must introduce a new player in this cryptography game: **A Certificate Authority**

A **Certificate Authority**, or just **CA**, is an organization that is **trusted** to **sign** digital certificates. CA verifies identity and legitimacy of company or individual that requested a certificate and if

the verification is successful, CA issues signed certificate. Examples of CA organizations are Oracle, VeriSign , D-Trust, DigiCert among many others.

So, from a cryptography perspective, how this signing process works? Again, we will be using the power of asymmetric encryption for this, as follows:

- The company/website that needs to have a signed certificate will create something called CSR (Certificate Signing Request) which is nothing but a X509 certificate as described above, including the public Key and all the identity data.
- This text file with .CSR extension will be sent to a designated CA company, that will evaluate the identity of the certificate, the domain name (Common name or Subject Alternative name, i.e. www.mycompanydomain.com, *.mycompany.com, etc). The identification process will include automatic checking of the private key from the company/website. Notice the CA now has the public key, so it can "challenge" the company/website to be able to decrypt something with is private key (previously encrypted with the public one) to demonstrate ownership of the certificate private key, etc.
- Once the identity of the company has been proved, the **CA will sign the CSR** by **adding a piece of information to the CSR**. The piece of information (the signature) will be the CSR information/content **hashed** (https://en.wikipedia.org/wiki/Cryptographic_hash_function and then encrypted with the **private key of the CA certificate**. The hashing of the CSR information/content will make sure the encrypted data hasn't been manipulated.
- Now, the CSR becomes a real X509 digital certificate ready to be used in any TLS connection: "CSR+the Signature" -> final X509 Certificate

So, now that we have our X.509 certificate ready to take action…. How a computer/mobile/device receiving this certificate during a TLS connection can verify is a valid X.509 certificate? Again, we will use the power of the asymmetric encryption, as follows:

- In order to check/validate that a X.509 is real, we need to validate the signature of the certificate (remember, signature is the encrypted "hashed" part of the certificate). This signature has been "created" by one of the trusted CAs companies in the world by using its private Key, and as we all know already, **anything than has been encrypted by one private key, CAN ONLY BE DECRYPTED by its public key counterpart**.
- The validation process will require having a CA Trusted Store of public keys from all the CA authorities we do trust in the internet (DigiCert, Oracle, VeriSign, etc). Once we receive the certificate (including the signature part), **the validation process of the signature consists in "trying" to decrypt the signature with at least one of the public keys from our CA Trusted Store**. If that is a positive (decryption completed), then that **CA** was the one signing the CSR with its private key. Again, remember, anything encrypted with a private key can ONLY be decrypted with its public key and the other way around.

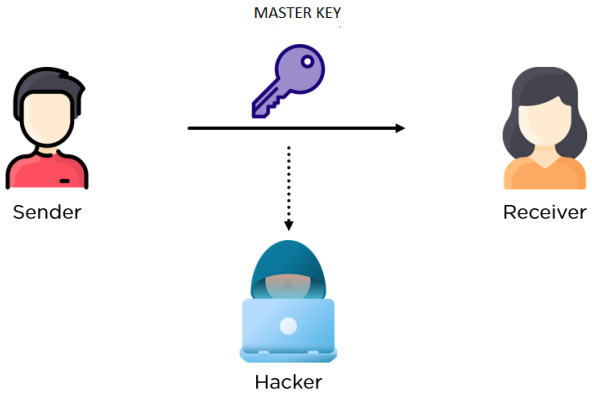
Now that we know we have a valid certificate that includes the domain name where we are trying to connect to, for example, www.mycompanycomain.com (included in the common name or Subject Alternative name fields from Certificate), the computer/mobile/browser will make sure that we are actually connecting to that DNS domain name (www.mycompanycomain.com) and not to other one. This is a mandatory validation, since anyone can steal the X.509 certificate and try to use it from another DNS domain server (www.myfakecompany.com, etc) and it would pass the CA signature validation process.

So, now all the computers/mobiles/devices out there can download the public key of my Certificate and send me data encrypted so I will be the only one being able to decrypt it. Again, yes and no. Regretfully, again, the asymmetric encryption is much slower than symmetric encryption (x4 or more) so it is not viable. The solution is to use both, and this is where SSL/TLS walks on.

TLS Basics

TLS is a cryptographic protocol that provides end-to-end security of data being sent between applications over the Internet. It is mostly familiar to users through its use in secure web browsing, and in particular the padlock icon that appears in web browsers when a secure session is established.

TLS uses a combination of **symmetric and asymmetric cryptography**, as this provides a good compromise between performance and security when transmitting data securely. Remember that asymmetric cryptography requires 4x or more computational power than symmetric cryptography. However, symmetric cryptography requires the same master key in both sides of the communication to be able to encrypt/decrypt the message, so the goal for using symmetric encryption is, to be able to exchange the master key (or shared secret) between the two parties over an insecure channel first, without being capture by a “man-in-the-middle”, and then start using this master key to secure the communication, encrypting/decrypting every message being sent/received using the chosen symmetric engine.



In order to exchange a master key over an insecure channel (remember internet without encryption/decryption is naturally insecure) **TLS will use two different techniques** that follows the same goal, which is **to exchange the master key/shared secret** securely in an insecure channel:

- Rivest, Shamir, Adleman (**RSA**)
- Diffie-Hellman exchange (**DHE**) and Elliptic Curve Diffie-Hellman exchange (**ECDHE**)

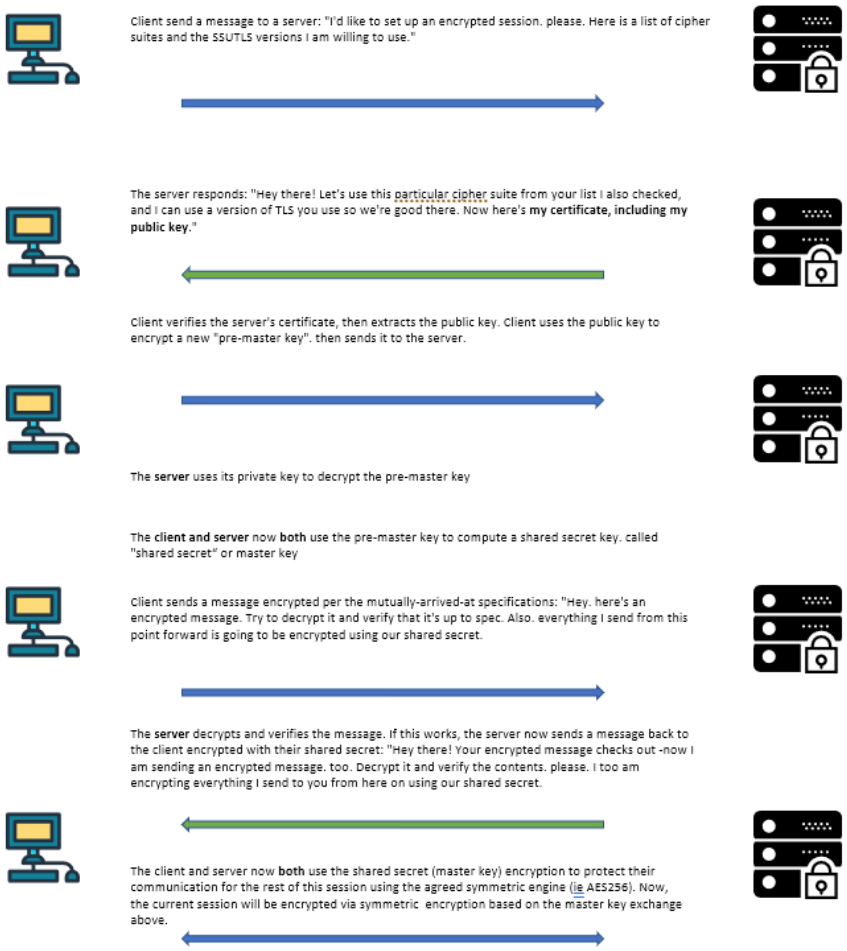
RSA methodology will rely on using the power of asymmetric encryption (private and public key) to **exchange** the master key over an insecure channel.

DHE/ECDHE will rely on using mathematical calculations to **generate** the same master key between two parties over an insecure channel. Notice I'm using the word "**generate**" and **not exchange** because the actual **master key is never exchange between the two parties**. Instead, both parties will exchange certain benign information that will be mixed with their privileged information as it travels over an insecure channel. The benign information can be captured without any risk by any man-in-the middle. At the end of the negotiation, both mathematical calculations of the two parties will end up on the same common secret or master key. More information https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

These two key exchange methodologies take place during a process called **TLS handshake**

TLS handshake unmasked

An **SSL/TLS handshake** is a **negotiation** between two parties on a network – such as a browser and web server – to **establish the details of their connection**. During that negotiation, the two parties will **authenticate** (using X509 certificates) , **agree a cipher suite** (how the master key will be exchanged and what cryptography engines will be used, such as AES256, 128, hashing such as SHA1, 2, etc) and finally, when the authentication occurs (valid certificate received) and agreement on how to establish a **secure channel is done**, then, we will have a secure channel from both parties. These are the actual steps:



Notice in the graph above, we are using RSA (private/public key) to exchange the master key (or shared secret) to be used in the agreed symmetric encryption engine (i.e. AES256) during the cipher suite negotiation. If DHE/ECDHE were used, instead of using public and private key to exchange the master key, both parties would use DHE calculations, exchanging benign material, to end up calculating the same result in both sides. That result will become the master key (shared secret) to be used for symmetric encryption.