TLS i HTTPS

Què és TLS

TLS (https://datatracker.ietf.org/doc/html/rfc8446) és un protocol de xarxa basat en TCP que aporta privacitat, autenticació i integritat a altres protocols de xarxa (concretament, del nivell d'aplicació) que no en tenen degut a haver sigut dissenyats fa molt de temps, quan la seguretat no era una prioritat. D'aquesta manera, la combinació de, per exemple, HTTP + TLS dona lloc a HTTPS (el qual permet la comunicació segura -és a dir, xifrada, autenticada i amb integritat- entre clients web i servidors web), una combinació de SMTP + TLS dona lloc a SMTPS (el qual permet l'enviament segur de correus), etc.

> NOTA: Existeix una versió del protocol TLS basat en UDP anomenat DTLS, però no en parlarem

TLS se centra en el xifratge del canal per on es transfereix la informació, no en el xifratge de la informació en sí. Això vol dir que si es fa servir TLS en una connexió, es xifra tot el que hi passa "dins" d'aquesta connexió (comandes de petició/resposta, contrasenyes d´accés, capçaleres dels missatges, els propis missatges, etc).

TLS és un protocol híbrid (és a dir, que utilitza tant criptografia asimètrica com simètrica). La primera és usada per verificar la identitat dels extrems (gràcies a l'ús de certificats, és a dir, claus públiques de les quals es pot confiar de la seva autenticitat; a la pràctica, però, normalment es verifica la identitat del "servidor" però la del "client" no és tan habitual fer-ho, llegiu nota següent), i un cop fet això, intercanviar de forma segura, ara sí, una clau simètrica efímera, la qual serà utilitzada, a partir de llavors per realitzar de forma segura tota la transmissió de dades entre els dos extrems (fins que la connexió finalitzi).

> NOTA: El més comú en una comunicació HTTPS és que l'únic extrem que necessiti ser identificat és el servidor; els clients (navegadors, normalment) no són identificats. Això vol dir que els elements previs necessaris per establir una comunicació HTTPS són només dos: la clau privada del servidor (que no sortirà d'ell) i el certificat del servidor (el qual serà descarregat pel client en el moment de l'establiment de la connexió TLS i que, això sí, hauria d'estar signat per una CA reconeguda pel client gràcies a que aquest tingui en el seu poder el seu certificat arrel). En el cas que un client necessités ser identificat per un servidor (com per exemple, tot i que es fa servir una tecnologia diferent, passa en el protocol SSH), en aquest cas, caldria que existís una clau privada del client i un certificat del client (que hauria de ser enviat al servidor en el moment d'establir la connexió TLS).

> NOTA: L'ús de criptografia híbrida en lloc de només criptografia asimètrica per realitzar tota la comunicació és degut a l'alt cost computacional que té aquesta en comparació amb el de la criptografia simètrica (la qual, al seu torn, té el problema de la distribució de claus, solventat per la criptografia asimètrica)

Les "cipher suites"

Una "cipher suite" és un conjunt d'algoritmes que TLS utilitza per realitzar tot el següent:

1. Establir un canal segur asimètric entre els extrems mitjançant l'ús de certificats i claus privades
2. Intercanviar de forma segura la clau simètrica efímera que s'utilitzarà a partir de llavors en tota la comunicació fins el final de la connexió
3. Xifrar tota la comunicació entre els extrems amb la clau simètrica acordada en el pas anterior
4. Garantir, a la vegada, la integritat de la comunicació mitjançant la mateixa clau simètrica (fent ús d'un determinat HMAC)

La llista oficial de "cipher suites" reconegudes pel protocol TLS es troba aquí: https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4 No obstant, cal destacar que de totes les "cipher suites" que allà hi apareixen, només unes poques són les emprades per la darrera versió del protocol TLS (que és la v1.3). Concretament, en podem destacar les següents com les realment utilitzades a la pràctica avui dia:

TLS_AES_128_CCM_SHA256
TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256

Tal com es pot veure, a les "cipher suites" soportades per TLS v1.3 només hi consten dos tipus d'algoritmes possibles: d'una banda estan els algoritmes disponibles per realitzar el xifrat de la informació, (els quals són exclusivament de tipus AEAD -i, per tant, no només asseguren la confidencialitat sinó també la integritat i l'autenticació dels missatges-): AES-CCM, AES-GCM i ChaCha20-Poly1305; i d'una altra estan els algoritmes de tipus HMAC disponibles per realitzar internament diverses operacions, sovint anomenades "HKDF" (de "Hashed Key Derivation Function"), relacionades amb el protocol d'intercanvi de claus (per exemple, per fer el càlcul de forma segura de la clau simètrica efímera a partir dels valors aleatories "pre-master" generats en cada extrem, o per fer altres càlculs secrets accessoris -veure seccions 4.4.1 i 7.1 del RFC 8446 i el RFC 5705-).

Fixeu-vos que a la "cipher suite" no estan especificats altres algoritmes importants, com són els diferents formats acceptats dels certificats o els protocols reconeguts d'intercanvi de claus. Això és perquè l'estàndard TLS v1.3 ja en defineix de forma estricta quins han de ser aquests algoritmes, així que no cal "negociar-los" explícitament. En concret, els únics formats acceptats de certificats són els de tipus RSA amb claus a partir de 2048 bits o de tipus corba el·líptica amb claus a partir de 256 bits. D'altra banda, els únics protocols reconegut per fer l'intercanvi de claus és el Diffie-Hellman efímer (DHE) o la seva variant de corba el·líptica (ECDHE), podent realitzar en tots dos casos l'autenticació mútua dels extrems o bé mitjançant signatures de tipus RSA, ECDSA o EdDSA o bé mitjançant claus precompartides (PSK, de "Pre-Shared Keys").

> NOTA: The ephemeral key is only good for the duration of the TLS session, which means that once the session is over, the session key is no longer usable. Now, even if the session was recorded via a man-in-the-middle, there is no key that could later be found and used to decrypt the session. That's what is called "perfect forward secrecy". However, while Ephemeral Diffie-Hellman (or DHE) provides strong security, it has a big impact on performance. Because the server is now generating these ephemeral keys for each session, the handshake takes significantly longer. To improve the speed of the Diffie-Hellman process, you can use Elliptic Curve (together, this is called ECDHE).

El TLS "Handhshake"

El primer pas que marca el protocol TCP (sobre el qual es basa TLS) per establir una connexió entre dos extrems és realitzar l'anomenat "TCP 3-way handshake", el qual consisteix en un intercanvi de paquets (concretament, tres: el paquet "SYN" enviat des del client -és a dir, qui vol iniciar la comunicació- al servidor, el paquet "SYN-ACK" enviat del servidor al client com a resposta del paquet anterior i finalment el paquet "ACK" com a confirmació final enviat del client al servidor de nou). Aquest intercanvi de paquet inicial serveix únicament per crear un canal de comunicació (una "connexió") entre els dos extrems, però aquest primer pas és imprescindible per, a partir de llavors, poder començar a transmetre qualsevol tipus de dada justament a través d'aquesta connexió establerta. En tot cas, però, tota transmissió feta en un canal TCP es realitza de forma desxifrada i sense cap tipus de seguretat. Si, a més de poder enviar i rebre dades, es vol que aquest enviament i rebuda es pugui fer de forma segura, caldrà realitzar, un cop s'hagi establert la connexió TCP, un segon "handshake", aquest cop de tipus TLS, per "blindar" el canal, per "assegurar" la connexió.

Per tant, el "TLS handshake" és el primer pas (un cop, això sí, la connexió TCP ja hagi estat realitzada gràcies a haver fet un "TCP 3-way handshake" previ) que els dos extrems han de realitzar per tal de protegir la seva interconnexió tot just acabada d'establir. Aquest "TLS handshake" consisteix, bàsicament, en posar-se d'acord en diversos elements (com ara la "cipher suite" concreta que es farà servir, per exemple) que asseguraran la comunicació entre els extrems. Més en concret, el "TLS handshake" consisteix en realitzar tres accions específiques entre els dos extrems:

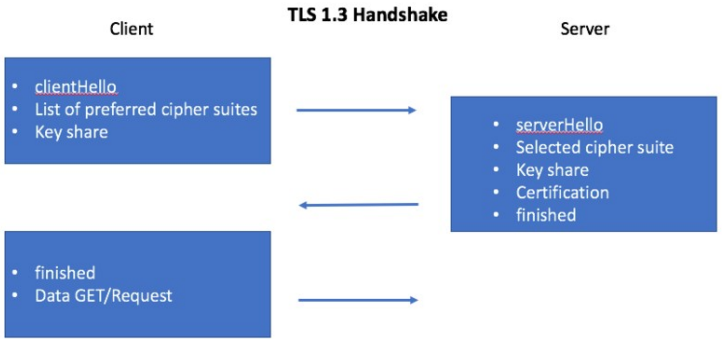
- **.** Posar-se d'acord en la "cipher suite" a utilitzar
- **.** Posar-se d'acord en la clau simètrica efímera que compartiran en aquella sessió
- **.** Establir una confiança entre els extrems mitjançant l'ntercanvi de certificats per tal d'autenticar-se mútuament (o bé només l'extrem "servidor", com sol passar a HTTPS)

En detall, el "TLS handshake" està format pel següent intercanvi (simplificat) de paquets:

1. El client envia un missatge anomenat "**Client hello**" indicant la versió que vol utilitzar del protocol TLS (idealment, la 1.3), la llista de "cipher suites" que pot utilitzar ordenades per preferència (en el cas de TLSv1.3, la llista és molt reduïda, com hem vist a l'apartat anterior) i un valor aleatori. Aquest valor aleatori, que anomenarem "client pre-master key", és el valor que s'utilitzarà per generar la clau compartida efímera (l'anomenada "master key") fent servir un algoritme de la família Diffie-Hellman (l'algoritme concret a utilitzar -DHE, ECDHE,...- i els seus paràmetres concrets són suggerits també com a part del missatge "Client hello")

2. El servidor calcula un valor aleatori propi (el "server pre-master key"), el qual, juntament amb el "client pre-master key" rebut, servirà per generar la clau simètrica efímera. Al client li enviarà llavors un missatge anomenat "**Server hello**" incloent el seu certificat, la "cipher suite" triada d'entre les que li va suggerir el client, un valor anomenat "Session ID" (del qual en parlarem més endavant) i el "server pre-master key" (a més de confirmar-li que l'algoritme concret Diffie-Hellman triat es correspon al suggerit pel client...només en rares ocasions no és així). Si no es requereix cap certificat per part del client (en aquest cas, li enviaria primer un missatge de tipus "Client certificate request"), li enviarà a més el missatge "**Server finished**"

3.- El client verifica el certificat rebut, i si aquest és correcte, genera la clau simètrica efímera amb el seu "client pre-master key" i el "server pre-master key" rebut i li envia el missatge "**Client finished**" A partir d'aquí la comunicació segura ja pot començar



NOTA: Es pot consultar amb més detall i de forma il·lustrada tot els passos d'aquest procés a https://tls13.xargs.org

NOTA: En un "TLS handshake" normal, el servidor envia un identificador de sessió com a part del missatge *ServerHello*. El client associa llavors aquest identificador de sessió amb l'adreça IP i el port TCP del servidor, de manera que quan el client es connecti de nou a aquest servidor (per exemple, si l'usuari visita de nou el lloc web HTTPS en qüestió), podrà utilitzar l'identificador de sessió com a drecera per reestablir el canal segur directament des del seu seu primer missatge *ClientHello* i començar a enviar dades xifrades inmediatament (sense haver de repetir de nou, doncs, el "TLS handshake" complet), ja que el servidor tindrà associat aquest identificador de sessió als paràmetres criptogràfics negociats prèviament i guardats en la seva memòria cau, (tot i que, això sí, la clau efímera emprada serà cada cop diferent). Això és el que se'n diu molts cops "handshake abreujat", "handshake reiniciat" o simplement "**0-RTT**"

Implementació d'un un servidor HTTPS (Apache2)

El mòdul "ssl" és una "pegament" entre l'Apache i la llibreria OpenSSL que permet que el primer pugui fer servir la segona per autentificar a altres màquines (o, el que és el más habitual, a sí mateixa davant d'altres màquines) amb certificats digitals i, per tant, poder encriptar la comunicació amb els clients via TLS, entre altres aspectes relacionats amb la seguretat de les conexions. A l'Ubuntu aquest mòdul s'instal.la per defecte juntament amb el propi servidor Apache (està dins del paquet "apache2-bin") però a Fedora cal instal.lar el paquet "**mod_ssl**" (https://httpd.apache.org/docs/current/mod/mod_ssl.html)

NOTA: També existeix el mòdul "gnutls" (a l'Ubuntu el paquet que l'instal.la es diu "libapache2-mod-gnutls" i a Fedora, "mod_gnutls") similar en objectius al "ssl" però que utilitza GnuTLS.

El primer que haurem de fer per a què l'Apache pugui funcionar com a servidor HTTP**S** és crear (amb les comandes adients de la suite OpenSSL, per exemple) la parella clau privada + certificat que tot seguit configurarem com la parella criptogràfica que l'Apache farà servir.

Seguidament, caldrà editar l'arxiu de configuració del VirtualHost "a assegurar" per tal que apareguin les següents línies (sempre dins de la secció *<VirtualHost *:443>...</VirtualHost>*), a més de com a mínim les directives bàsiques *ServerName nom.Dns.Servidor* i *DocumentRoot /ruta/carpeta/on/son/les/pagweb* , entre altres (*ErrorLog ..., CustomLog ..., ServerAlias ...*, etc):

```
SSLEngine on
SSLCertificateFile /ruta/al/certificat/serverHttps.crt       #Per oferir-lo al client per a què pugui xifrar informació
SSLCertificateKeyFile /ruta/a/la/clau/privada/serverHttps.key       #Per poder desxifrar allò rebut del client
```

NOTA:Si estiguéssim en el cas de tenir un fitxer que inclogui tant el certificat com la clau privada tot en un (això també és possible), les línies a escriure serien llavors només les dues primeres anteriors (on a *SSLCertificateFile* caldria indicar en aquest cas la ruta al fitxer que inclou el certificat més la clau privada tot-en-un)

En qualsevol cas, per fer que l'Apache utilitzi la parella clau+certificat indicada, cal habilitar el mòdul SSL; a l'Ubuntu això simplement es fa així: *sudo a2enmod ssl* ; a Fedora ja està habilitat per defecte. Finalment, caldrà activar aquest VirtualHost segur amb la comanda *sudo a2ensite nomfitxerconfigVH* (això només és necessari, de nou, a l'Ubuntu) i reiniciar el servidor (*sudo systemctl reload apache2* o *httpd*)

Per accedir al nou lloc segur simplement caldrà que escriguis a la barra del navegador una direcció tal com https://nom.Dns.Servidor (el port 443 no cal afegir-lo perquè indicant "https" ja se sobrentén, igual que se sobrentenia el 80 quan accedíem a HTTP)

NOTA: En qualsevol moment es pot comprovar si els fitxers de configuració de l'Apache estan correctament escrits amb la comanda *apache2ctl configtest*
NOTA: Si volguéssim redireccionar automàticament al lloc HTTPS un usuari que hagués escrit explícitament "http" en comptes de "https" a la barra de direccions, el més fàcil és utilitzar la directiva *Redirect* dins del VirtualHost "estàndar" per reencaminar-lo a l'altre; és a dir, fer així (una altra manera més complicada també seria fer-ho amb el mòdul "Rewrite", tal com s'explica aquí: https://wiki.apache.org/httpd/RewriteHTTPToHTTPS):

```
<VirtualHost *:80>
    ServerName nom.dns.servidor
    Redirect permanent / https://nom.dns.servidor    #Aquí es pot posar la IP també
</VirtualHost>
```

Si volem que el nostre servidor HTTPS accepti, a més, certificats de clients, hem d'afegir a la configuració del "VirtualHost" (o també es pot sota una directiva *<Directory>* o *<Location>*) les següents directives més:

SSLVerifyClient {optional|require} : Tal com diu el seu nom, si indiquem el valor "optional" voldrà dir que el client pot presentar un certificat (però no és obligator); en canvi si indiquem el valor "require" sí que ho serà

SSLVerifyDepth nº : El nombre indicat serveix per establir el nombre de certificats intermitjos dins de la cadena que l'Apache haurà de recórrer com a màxim fins arribar al certificat arrel que verifiqui el certificat del client presentat (i, per tant, donar-lo per vàlid o no). Un valor de 0 voldrà dir que només s'accepten certificats autosignats; un valor d'1 -per defecte- voldrà dir que només s'accepten aquests darrers o els que han sigut signats per una CA directament coneguda pel servidor Apache (com les configurades a nivell general de sistema o, opcionalment, també el certificat arrel de les quals estigui ubicat en una determinada carpeta -la ruta de la qual haurà de ser indicada explícitament amb la directiva *SSLCACertificatePath /ruta/carpeta* - o, alternativament, mitjançant la directiva *SSLCACertificateFile /ruta/fitxer.crt* si "fitxer.crt" és un fitxer que conté tots els certificats arrel necessaris concatenats)

NOTA: Per més informació (on també es mencionen les múltiples avantatges de fer servir certificats de client en lloc de contrasenyes) es pot consultar https://www.scriptjunkie.us/2013/11/adding-easy-ssl-client-authentication-to-any-webapp
També es pot consultar un tutorial detallat a https://community.hetzner.com/tutorials/apache-ssl-client-auth