1. PROTOCOLLI SSL E TLS

I protocolli SSL e TLS sono alla base di molte applicazioni che usiamo. Questi due protocolli non sono diversi, infatti TLS è l'evoluzione di SSL.

SSL= Secure **S**ocket **L**ayer. SSL denota un livello di sicurezza delle Socket, che sono delle API inizialmente usate in sistemi UNIX based e poi utilizzate anche in altri sistemi. Queste API consentono di implementare tutte le funzionalità della suite protocollare su cui si basa Internet.

TLS= **T**ransport **L**ayer **S**ecurity è un protocollo che riguarda la sicurezza a livello di trasporto (connessioni connection-oriented).

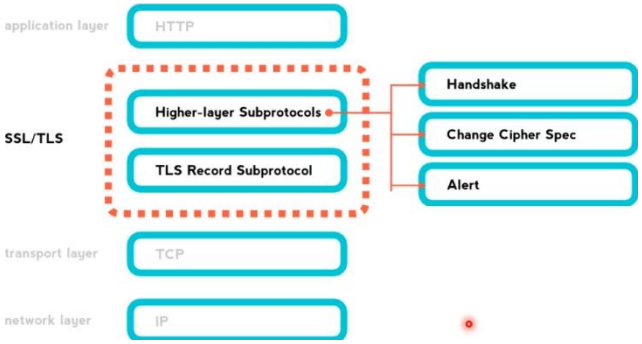
DTLS= **D**atagram **T**ransport **L**ayer **S**ecurity, si basa su connessioni connectionless.

Questi protocolli offrono molte funzionalità per mettere in sicurezza comunicazioni end-to-end che utilizzano protocolli TCP (o UDP), ad esempio HTTP, POP3, IMAP etc….

SSL/TLS è un protocollo per la sicurezza in Internet, e in questa immagine si può vedere che non è l'unico protocollo che è stato definito per la sicurezza in Internet. Ogni livello della suite TCP-IP ha vari protocolli di protezioni sviluppati per proteggere tale livello. Ad esempio, partendo dall'alto, c'è il livello applicativo che si compone di vari protocolli. I principali sono SSH, S/MIME etc..

Communication layers	Security protocols
Application layer	SSH, S/MIME, Kerberos, PGP, WSS, etc
Transport layer	SSL/TLS
Network layer	IPSec
Data Link layer	IEEE 802.1X, IEEE 802.11i (WPA2), etc
Physical layer	Quantum Cryptography

Abbiamo detto che SSL/TLS riguarda la protezione al livello di trasporto. È una cosa esatta, ma esplicitamente, SSL/TLS si frappone tra il livello di traporto e il livello applicativo. Non fa quindi parte del livello di trasporto, ma è una sorta di layer intermedio. SSL/TLS a livello logico è strutturato in 2 livelli: il primo è costituito da vari sottoprotocolli, ciascuno dei quali si occupa di fornire un sottoinsieme delle funzionalità offerte da SSL/TLS. Questi sottoprotocolli sono 3 e sono: Handshake, Change Cipher Spec, Alert. Il secondo livello è formato dal TLS Record Subprotocol. I 2 concetti più importanti di questa struttura sono l'Handshake e il TLS Record Subprotocol.



In che modo SSL/TLS fornisce sicurezza in una comunicazione tra 2 endpoint (Client, Server)? Questa sicurezza è fornita attraverso vari requisiti:

- Fornisce **autenticazione** per applicazioni Server/Client, dove per autenticazione s'intende la possibilità di verificare che ciascuna parte (Client o Server) è effettivamente chi dichiara di essere.
- Cifra i dati prima di inviarli su un canale pubblico. Questo garantisce **confidenzialità**, cioè che i dati possono essere compressi da chi ha il diritto di farlo.
- Garantisce l'**integrità** dell'informazione, realizzato mediante funzioni Hash, MAC e così via. Questo requisito garantisce che un dato inviato su un canale pubblico che è insicuro, non è alterato durante il suo tragitto.
- Le **primitive crittografiche** (ma anche altre informazioni) vengono **negoziate tra le parti** (Client e Server). Per accordarsi s'intende che le 2 parti negoziano quali algoritmi usare per proteggere la comunicazione, anche quali chiavi usare.

Altri due concetti che sono alla base di SSL/TLS sono **Connessione** e **Sessione**. Sebbene sembra che denotino la stessa cosa, così non è. Infatti, una connessione:

- Definisce un canale di comunicazione tra due endpoint (Client e Server).
- Ciascuna connessione è associata ad una **sessione**.

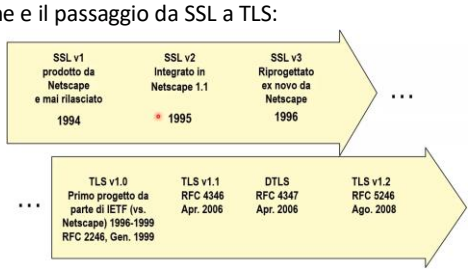
Una sessione:

- Definisce un insieme di parametri crittografici, negoziati tra le parti, che possono "sopravvivere" tra diverse connessioni. Per "sopravvivenza" tra due connessioni s'intende che se tra due persone che vogliono dialogare vengono stabiliti degli algoritmi per rendere sicura una comunicazione, e dopo un po' s'interrompe il dialogo, la creazione di un nuovo dialogo avviene "riesumando" la precedente, cioè ripartire da dove il dialogo era stato interrotto, utilizzando gli stessi algoritmi e chiavi concordati in precedenza. Naturalmente per poter applicare il concetto di "sopravvivenza", è necessario salvare i parametri di sessione. La "sopravvivenza" può ricominciare anche su macchine diverse da quelle che sono state utilizzate per i dialoghi precedenti.
- Usata per evitare la (costosa) negoziazione di nuovi parametri di sicurezza per ciascuna connessione.

SSL/TLS consente di creare una nuova sessione, senza interrompere la connessione in corso. La connessione rimane attiva, però all'interno di questa connessione si possono negoziare nuovi parametri SSL/TLS. Ad esempio, se si sta proteggendo la comunicazione tramite DES e una persona vuole ancora più sicurezza poiché si deve parlare di informazioni molto riservate, si rinegoziano i parametri senza interrompere la connessione, e si utilizza AES.

Gli ambiti di utilizzo di SSL/TLS sono: commercio elettronico, pagamenti, accesso sicuro ed autenticato alle informazioni e così via…

Viene mostrata una timeline che mostra l'evoluzione e il passaggio da SSL a TLS:



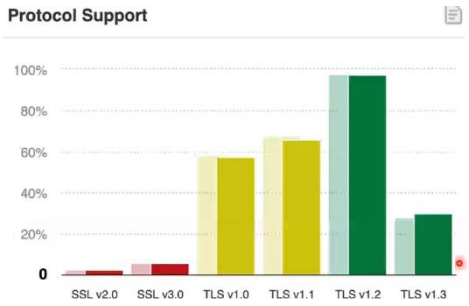
Abbastanza di recente, a fine marzo 2018 è stato approvato TLS v.1.3 caratterizzato da rimozione di:

- Curve ellittiche note come "deboli" o meno utilizzate.
- Funzioni Hash crittografiche MD5 e SHA-224.
- Protocollo SSL e cifrario RC4 (che erano ancora supportati per retrocompatibilità).

E aggiunta di nuovi algoritmi e protocolli:

- Stream Cipher ChaCha20 e algoritmo Poly1305 per il Message Authentication Code (MAC).
- Algoritmi di firma digitale Ed25519 ed Ed448.
- Protocolli di scambio chiavi x25519 ed x448.

Le statistiche dicono che, sui siti indicizzati da Alexa, l'utilizzo dei vari protocolli è:



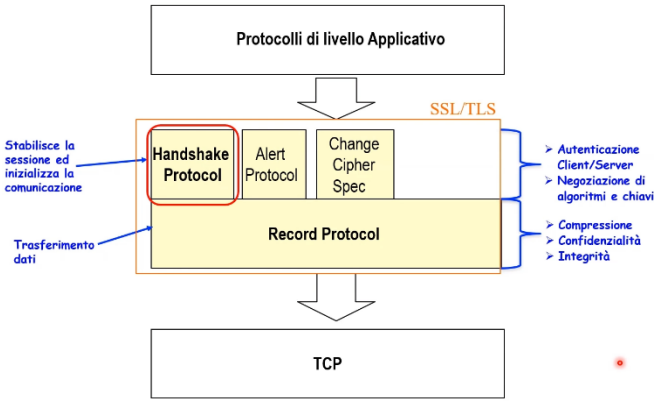
Il 30% fornisce il supporto a TLS v1.3. Quasi il 100% fornisce il supporto a TLS v1.2 e così via. Si ricordi che un singolo sito può supportare più versioni dello stesso protocollo.

Google Chrome ha abilitato di default il supporto a TLS v1.3 a partire dal 2017 (versioni beta, in quanto, come scritto in precedenza, è stato rilasciato ufficialmente a 2018. Non bisogna sorprendersi di questa cosa.). Tale supporto di default è stato poi rimosso a causa di problemi di retrocompatibilità con alcuni web proxy. A partire dalla release 65, Google Chrome (marzo 2018) ha di nuovo abilitato il supporto a TLS v1.3. Una situazione analoga è avvenuta per Firefox.

SSL/TLS è un protocollo che si compone di quattro sottoprotocolli (elencati in ordine di importanza):

- **HANDSHAKE PROTOCOL:**
  Permette alle parti di negoziare i diversi algoritmi (es. primitive crittografiche) e parametri necessari per la sicurezza della comunicazione. Consente l'eventuale autenticazione tra le parti.
- **RECORD PROTOCOL**:
  Utilizza algoritmi e parametri negoziato durante l'Handshake Protocol. Si occupa della compressione, del MAC e della cifratura.
- **CHANGE CIPHER SPEC PROTOCOL**:
  Impone l'esecuzione di un nuovo handshake per rinegoziare i parametri di sicurezza e ripetere l'autenticazione.
- **ALERT PROTOCOL**:
  Notifica situazioni anomale o segnala eventuali problemi.

Tutti questi protocolli menzionati, vanno a formare un vero e proprio stack protocollare, impilati uno sull'altro. Si nota che SSL/TLS, come detto prima, si frappone tra i protocolli di livello applicativo e il livello di trasporto (TCP). Si osserva anche che l'Handshake Protocol opera a monte del Record Protocol.



Andiamo ora a descrivere con più dettaglio ogni singolo sottoprotocollo.

HANDSHAKE PROTOCOL

Viene utilizzato dalle due parti (endpoint). Permette alle parti di:

- Negoziare la versione del protocollo SSL/TLS e l'insieme delle primitive crittografiche da utilizzare (**Ciphersuite**), garantendo interoperabilità tra le diverse implementazioni del protocollo.
- Stabilire le chiavi crittografiche da utilizzare.
- Autenticare il Server (ed opzionalmente il Client).

È importante sottolineare che mediante Handshake Protocol, le due parti coinvolte non solo si mettono d'accordo per quale protocollo, insieme di primitive crittografiche utilizzare, ma di scegliere sempre la migliore soluzione di sicurezza possibile. Questa migliore soluzione è implementata da SSL/TLS tramite la Ciphersuite. Una Ciphersuite è una sorta di regola che stabilisce quali sono le cose da utilizzare per garantire la sicurezza della sessione. Una Ciphersuite definisce:

- **Schema per l'accordo/scambio di chiavi**: stabilisce il modo in cui verranno scambiate le chiavi utilizzate dagli algoritmi a chiave simmetrica. Algoritmi per lo scambio di chiavi sono: RSA, DH, **ECDH**, etc..
- **Schema per l'autenticazione**: stabilisce in che modo verrà eseguita l'autenticazione del Server e (se necessario) quella del Client. Algoritmi per l'autenticazione sono: RSA, DSA, **ECDSA**, etc..
- **Schema per la cifratura simmetrica**: stabilisce quale algoritmo a chiave simmetrica verrà utilizzato per cifrare i dati. Algoritmi per la cifratura sono: **AES**, 3DES, CAMELLIA, etc..
- **Schema per l'autenticazione del messaggio (MAC):** stabilisce il metodo che la sessione utilizzerà per il controllo di integrità dei dati. Algoritmi per il calcolo del MAC sono: **SHA**, MD5, etc..

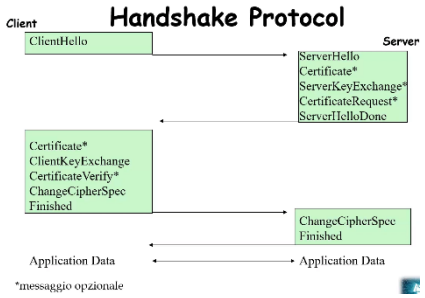
Un esempio tipico di una Ciphersuite è il seguente:

TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 dove:

- **TLS** indica il protocollo.
- **ECDH** indica l'algoritmo per lo scambio di chiavi.
- **ECDSA** indica l'algoritmo per l'autenticazione.
- **AES_256_CBC** indica l'algoritmo per la cifratura
- **SHA384** indica l'algoritmo per il MAC

I dettagli dell'Handshake Protocol vengono mostrati con un esempio pratico (ricordiamo che "protocollo" indica un insieme di regole che definisce lo scambio di messaggi):

Ci sono due entità: Client, Server. Il flusso di esecuzione è stabilito dalle frecce. Ciascun messaggio può essere composto da 1 o più campi. In questa figura ci sono tutti i messaggi che costituiscono il protocollo. Ci sono anche dei messaggi opzionali che possono essere scambiati a seconda della Ciphersuite utilizzata. Questa versione mostrata è semplificata, in quanto sono mostrati i messaggi che sono semanticamente rilevanti.



Il funzionamento è il seguente:

CLIENT

- Si parte dal primo messaggio, che è un protocollo inizializzato dal Client. Questo è un messaggio che viene chiamato di **ClientHello**, come se fosse un saluto che il Client invia al Server. Questo messaggio contiene varie informazioni riguardanti il Client. Ogni informazione è fornita tramite un opportuno campo:
  - La prima informazione, contenuta nel campo è ***client_version***, ed indica la versione di TLS più recente supportata dal Client.
  - Successivamente c'è il campo ***random***, al cui interno c'è un valore di 32 byte (4 byte che specificano data e ora del Client e 28 byte sono un numero casuale generato dal Client).
  - ***session_id***: permette al Client di proporre al Server di riutilizzare una sessione precedentemente stabilita (valore diverso da zero per una nuova connessione sulla sessione corrente. Valore uguale a zero per una nuova connessione su una nuova sessione).
  - ***cipher_suites***: lista di Ciphersuite supportate dal Client.
  - ***compression_methods***: lista degli algoritmi di compressione supportati dal Client.

SERVER

- Il messaggio che il Server invia al Client è un messaggio di **ServerHello**, composto dai seguenti campi:
  - ***server_version***: versione di TLS più recente supportata da ambedue le parti.
  - ***random***: numero di 4 byte che specifica data e ora del Server + numero casuale di 28 byte generato dal Server.
  - ***session_id*** (se presente): new session_id o resumed session_id, lo stesso ID indicato dal **ClientHello**: indica che il Server è d'accordo nel riutilizzare una vecchia sessione.
  - ***cipher_suite***: migliore Ciphersuite supportata dalle due parti: se le parti non hanno una Ciphersuite in comune, la sessione termina con il messaggio di alerà "handshake failure".
  - ***compression_method***: miglior algoritmo di compressione supportato dalle due parti.

- A questo scambio appena finito, possono seguire una serie messaggi opzionali mandati dal server (**Certificate, ServerKeyExchange, CertificateRequest**). Questa serie di messaggi dipende dalla Ciphersuite su cui si sono accordati. In particolare:
  - **Certificate**: contiene il certificato del Server, cioè la chiave pubblica inclusa nel certificato ed è utilizzata dal Client per autenticare il Server e per cifrare il Pre-master secret (numero casuale generato dal Client che verrà utilizzato dalle due parti per la generazione della chiave di sessione).
  - **ServerKeyExchange**: messaggio usato dal Server per spedire una chiave temporanea al Client, inviato solo se il certificato del Server non contiene informazioni tali da permettere al Client di spedire al Pre-master secret in modo sicuro.
  - **CertificateRequest**: messaggio tramite cui il Server richiede di autenticare il Client. Utilizzato, ad esempio dai siti web delle banche, in cui il Server deve verificare l'identità del Client prima di rilasciare informazioni sensibili.
- Il Server spedisce il suo ultimo messaggio che è un messaggio di **ServerHelloDone** che indica la fine della fase di "presentazione" del Server. A questo punto, la fase di "saluto" tra le due parti termina. Il Server è riuscito ad acquisire tutte le informazioni relative al Client che gli servono per instaurare una connessione sicura.

CLIENT

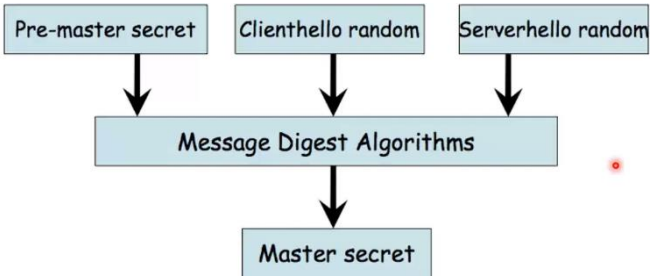
- A seconda della Ciphersuite scelta e dei messaggi opzionali che il Server ha inviato precedentemente, i messaggi che il Client spedisce al Server sono **Certificate** (opzionale) **, ClientKeyExchange** e **CertificateVerify** (opzionale), **ChangeCipherSpec** e **Finished**:
  - **Certificate**: viene spedito solo se il Server lo ha richiesto.
  - **ClientKeyExchange**: Il Pre-master secret viene cifrato con la chiave pubblica del Server (o con la chiave temporanea spedita tramite il messaggio **ServerKeyExchange**) e viene trasmesso a quest'ultimo.
  - **CertificateVerify**: Spedito solo nel caso in cui il Server abbia richiesto l'autenticazione del Client. Il Client invia un messaggio contenente la firma di tutti i messaggi precedentemente scambiati. In questo modo il Server, verificando la firma con la chiave pubblica presente nel certificato del Client, si accerta dell'identità del Client.
  - **ChangeCipherSpec**: viene spedito per confermare al Server che tutti i messaggi che seguono saranno protetti usando le chiavi e gli algoritmi (Ciphersuite) appena negoziati.
  - **Finished**: primo messaggio "sicuro" (cifrato ed autenticato). Questo messaggio contiene l'Hash di quanto scambiato fino a quel momento ed è cifrato usando la chiave di sessione.

SERVER:

- Il Server conclude il protocollo mediante messaggi **ChangeCipherSpec** e **Finished**:
  - **ChangeCipherSpec**: questo messaggio conferma al Client che tutti i messaggi che seguono verranno protetti usando le chiavi e gli algoritmi (Ciphersuite) appena negoziati.
  - **Finished**: questo messaggio contiene l'Hash di tutti i messaggi scambiati fino a quel momento ed è cifrato usando la chiave di sessione.
    OSSERVAZIONE: se il Client è in grado di decifrare tale messaggio e di validare gli Hash in esso contenuti, allora la fase di Handshake SSL/TLS è terminata con successo e la chiave di sessione da lui calcolata coincide con quella calcolata dal Server.

Al termine del protocollo di Handshake, le due parti si sono accordate sulla Ciphersuite da utilizzare. L'Handshake Protocol permette alle due parti di calcolare in maniera indipendente la stessa chiave di sessione. Tale calcolo avviene in due fasi: nella prima fase viene calcolato un **Master Secret**, nella seconda fase viene calcolata una **Chiave di Sessione**.

La prima fase del calcolo della Chiave di Sessione è mostrato nella figura a lato:

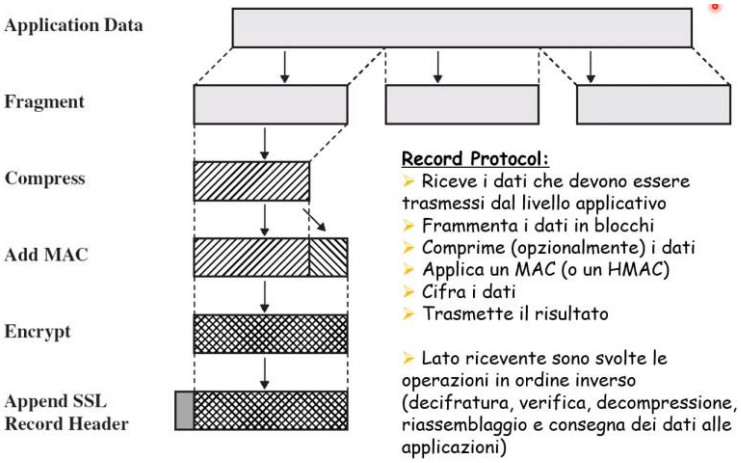


Il calcolo si basa su un algoritmo di **Message Digest Algorithms**, che prende in input la Pre-master secret (inviato dal Client al Server), **Clienthello random** e **Serverhello random** che sono due valori che si sono scambiati durante l'Handshake protocol. Dunque sia Client che Server posseggono queste 3 informazioni e mediante l'algoritmo calcolano la Master secret. Il master secret è sempre di 48 bit, mentre la lunghezza del Pre-master secret dipende dagli algoritmi negoziati.

Nella seconda fase, la costruzione dell'algoritmo Message Digest Algorithms è la stessa di quella descritta in precedenza tranne per il fatto che non viene presa in input la Pre-master secret, ma viene presa in input la **Master secret** calcolata nella fase 1. L'algoritmo genera la **Chiave di Sessione**, tipicamente utilizzata per la cifratura tramite cifrari a blocchi, per garantire confidenzialità. La sicurezza è garantita dal fatto che la Chiave di Sessione è calcolata in maniera indipendente, ed è garantita dalla sicurezza della Pre-master secret.

RECORD PROTOCOL

Si occupa di garantire compressione, confidenzialità ed integrità delle informazioni che vengono trasmette dall'Handshake Protocol. Un esempio di funzionamento logico è il seguente:



Partendo dall'alto, il protocollo riceve i dati provenienti dal livello applicativo, li frammenta, eventualmente li comprime, applica un MAC aggiungendo 1 frammento e poi cifra il tutto. Dopo aver fatto ciò, al messaggio cifrato, si antepone un Header SSL/TLS. Fatto ciò, si inviano i dati sulla rete. Tutti questi algoritmi (compressione, applicare MAC etc..) e chiavi stabilite, sono concordati tramite l'Handshake Protocol. Questo protocollo usa praticamente tutto quello che è stato concordato nella fase di Handshake Protocol.

ALERT PROTOCOL

È usato da una delle due parti per la trasmissione di messaggi di Alert. Viene quindi usato per segnalare anomalie. Ciascun messaggio è caratterizzato da un livello di severità (warning o fatal) e dal tipo di alert (spiega la causa dell'invio di un determinato messaggio).

CHANGE CIPHER SPEC PROTOCOL

Utilizzato per aggiornare la Ciphersuite in uso tra Client e Server. Il protocollo consiste in un unico messaggio inviato da una delle due parti all'altra.