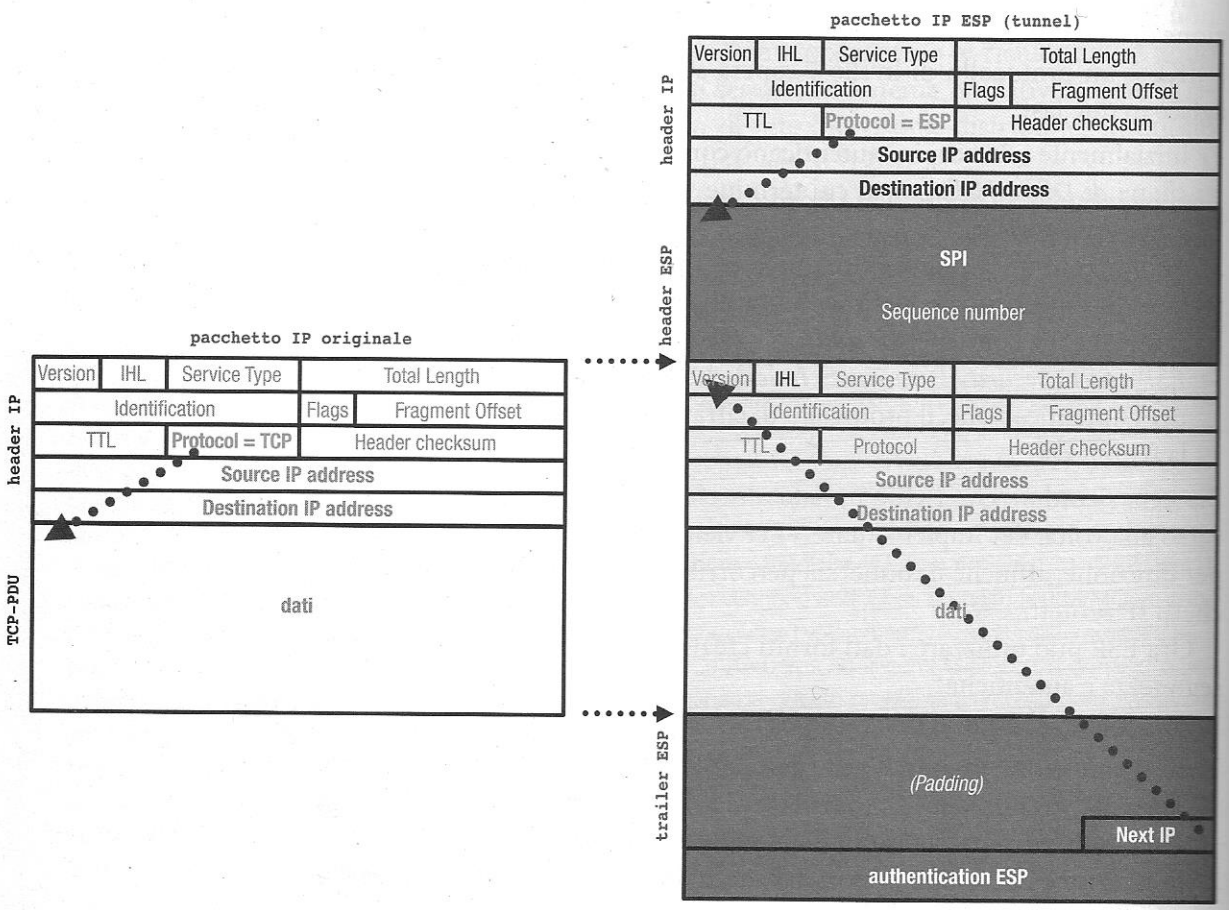
cifrato, quindi ESP autentica sia il proprio header sia il proprio trailer con una funzione di hash come **MD5** o **SHA-1**, sempre a seconda della CA negoziata con IKE.



È importante ricordare che IPsec è *completamente trasparente* alle applicazioni, dato che agisce **sotto** IP: una volta installato e avviato su un sistema (sia che si tratti del client, sia che si tratti del server), le applicazioni non si accorgono che sta agendo IPsec, e quindi non vanno nemmeno modificate o configurate per utilizzarlo.

6 SSL/TLS

SSL/TLS (*Transport Layer Security*, RFC 5246; SSL, *Secure Sockets Layer* è un protocollo predecessore, che poi è stato standardizzato come TLS) ha come obiettivo quello di fornire l'autenticazione (con identificazione), la riservatezza e l'integrità dei dati inviati da un'applicazione. TLS, infatti, agisce sia con uno schema di cifratura simmetrica sia asimmetrica.

In realtà, ancora una volta, con TLS si intende una suite di protocolli, alcuni collocati a livello applicativo, altri a livello di trasporto. Solitamente TLS viene etichettato come un protocollo di livello 4 Trasporto (a volte come di livello 5 Sessione).

Naturalmente TLS è client/server e opera sul numero di porta TCP usato dall'applicazione che intende utilizzarlo (per esempio, sulla porta TCP 443 quando viene utilizzato da HTTPS). Benché client/server, TLS si può realizzare in modo completamente simmetrico, se implementato da entrambi i versi in modo completo.

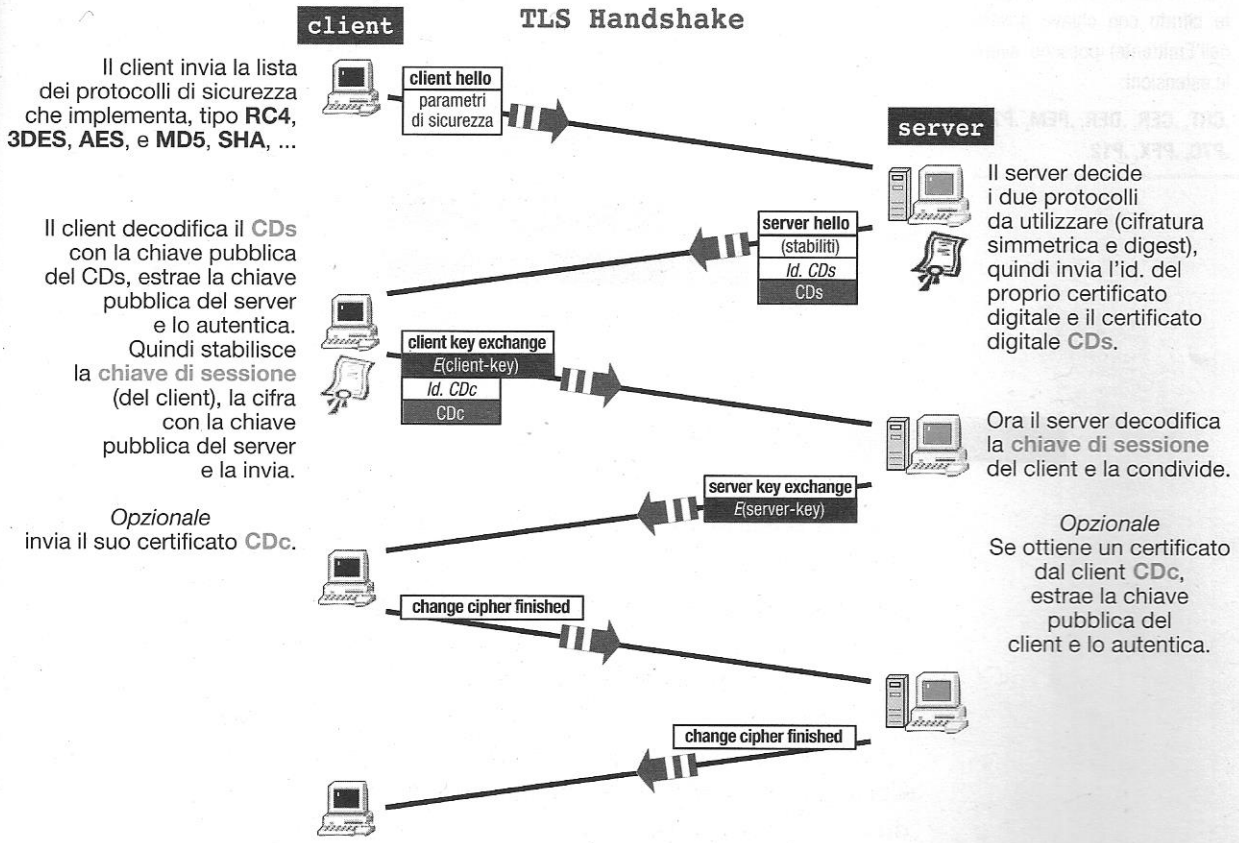
I sottoprotocolli fondamentali di TLS sono:

a) **Handshake** (livello 7), preliminare, negozia i parametri della sicurezza e li stabilisce.
b) **Record** (livello 4), crea i pacchetti dei dati dell'applicazione (che sta usando TLS) con cifratura simmetrica e integrità (digest). I pacchetti *Record* saranno incapsulati in TCP.

Per questa ragione, esattamente come IPsec, TLS è trasparente alle applicazioni. Rispetto a IPsec, TLS fornisce in modo nativo l'identificazione degli interlocutori tramite certificati digitali.

La fase di *Handshake* è la più critica: gli interlocutori devono autenticarsi (singolarmente o mutuamente) tramite chiavi asimmetriche prestabilite e certificato digitale, quindi stabilire una chiave di sessione segreta condivisa per cifrare la comunicazione seguente. In realtà le chiavi simmetriche possono essere due, una per il client e l'altra per il server, dato che è possibile usare TLS anche in un solo verso.

Al termine dell'handshake TLS ha creato una **connessione**. Ora, all'interno di una connessione definita con tutti i parametri necessari TLS può aprire più **sessioni** senza rinegoziare da capo i parametri di sicurezza.



X.509

Gli standard «X» sono emanati da **ITU-T**, uno degli enti di standardizzazione più noti per le reti informatiche (cfr. *Vol.1, Enti di standardizzazione*).

X.509 è lo standard per PKI che definisce, tra l'altro, i formati per i certificati digitali a chiave pubblica utilizzati dalle CA, anche se nella sua forma modificata da IETF denominata X.509 v3 (RFC 3280).

X.509 v3 prevede che il certificato contenga (l'**emittente** è la CA; il **soggetto** è chi lo ha richiesto):

Versione, Numero seriale, ID dell'algoritmo, **Ente emittente**, Validità (non prima, non dopo), **Soggetto**, Info chiave pubblica del soggetto, **Algoritmo per la chiave pubblica**, **Chiave pubblica del soggetto**, Codice identificativo dell'emittente, Codice identificativo del soggetto, Algoritmo di firma del certificato, Firma del certificato.

I file contenenti un certificato digitale X.509 v3 (normalmente cifrato con chiave privata dell'Emittente) possono avere le estensioni:

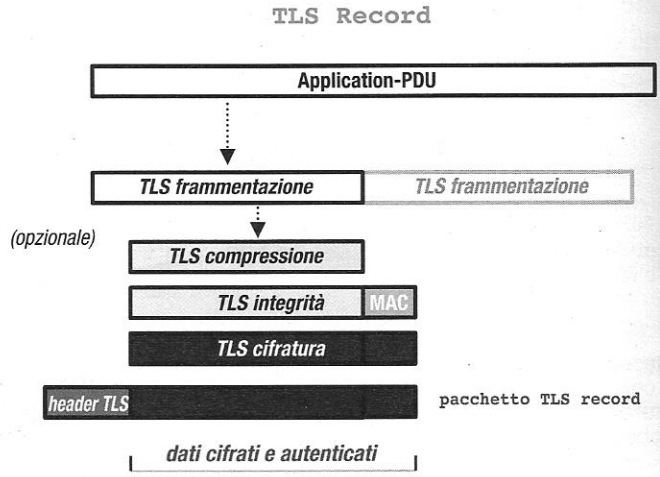
.CRT, **.CER**, **.DER**, **.PEM**, **.P7B**, **.P7C**, **.PFX**, **.P12**.

Lo schema TLS Handshake andrebbe consultato assieme allo schema presente al termine della sezione B3, *Verifica dell'identità sul Web tramite CA*.

Si ricorda soltanto che le chiavi pubbliche degli enti certificatori sono sempre disponibili sui sistemi operativi, quindi il client non ha alcuna difficoltà a decifrare il CDs spedito dal server che, a sua volta, contiene la chiave pubblica del server (con cui il client cifrerà la chiave di sessione) oltre ai valori di certificazione.

Nello schema viene rappresentato un handshake completo, in cui sia il client sia il server si autenticano a vicenda e si scambiano le due chiavi simmetriche per la cifratura. In realtà è abbastanza raro che un client possegga un certificato emesso da un ente certificatore ufficiale, quindi spesso TLS si accontenta della sola autenticazione del server (presso il client). In questi casi anche la sicurezza vale in un solo verso (una sola chiave simmetrica scambiata): le sessioni successive sono cifrate sui messaggi che vanno dal server al client e solo decifrate dal client.

La fase di *Record* si limita ad applicare le operazioni negoziate nella fase di *Handshake* con *quegli* algoritmi e *quelle* chiavi. TLS *Record*, inserito prima di TCP, frammenta i pacchetti dell'applicazione, li comprime (opzionalmente), ne calcola il MAC (con la funzione di hash negoziata), cifra in modo simmetrico sia i dati sia il MAC (con algoritmo e chiavi negoziate), aggiunge un header e passa il tutto a TCP.



7 HTTPS

HTTPS (*HyperText Transfer Protocol over Secure Socket Layer*, RFC 2818), come recita l'RFC:

```
[...]2. HTTP Over TLS
     Conceptually, HTTP/TLS is very simple. Simply use
     HTTP over TLS precisely as you would use HTTP over
     TCP. [...]
```

Come si nota, l'RFC di HTTPS fa riferimento a TLS e non a SSL come indicato nell'acronimo.

Un client HTTPS come un browser, quindi, seleziona l'uso di TLS quando lo schema dell'URL di un sito vale `https` (e non `http`, cfr. *HTTP*).

A questo punto scatta il TLS *Handshake* con le relative implicazioni che sono state illustrate in precedenza.

Al termine, il colloquio tra un client HTTPS (tipicamente un client Web) e un server HTTPS (tipicamente un server Web) risulta autenticato, integro e segreto.

Purtroppo gli host che sono equipaggiati con i client HTTPS più diffusi, ovvero browser come *MS Explorer*, *Google Chrome*, *Mozilla Firefox*, *Opera* e *Safari*, quasi sempre non hanno alcuna necessità di acquistare un certificato digitale da un ente autorizzato ben noto (CA).

Un utente privato, per esempio, non ha nessuna esigenza di questo, come la stragrande maggioranza degli utenti che operano in modo «client» sul Web.

Al contrario, i siti di maggiore prestigio o quelli che hanno l'esigenza di gestire traffico sicuro sono indotti a ottenere certificati a pagamento da una CA. Pertanto il traffico HTTPS, quasi sempre, è sicuro solo in una direzione: l'utente ha la certezza di comunicare in modo sicuro con un server Web HTTPS autentico e di identità certificata, mentre il server Web non ha garanzie di sicurezza circa il client Web che sta richiedendo le sue pagine.

7.1 Gestione dei certificati

Spesso i siti Web non richiedono certificati digitali agli enti CA autorizzati, ma hanno comunque l'intenzione di sfruttare HTTPS con i propri visitatori. In questi casi un'organizzazione può *autocostruirsi* un certificato digitale del tutto analogo a quelli ufficiali, in modo che i client che si connetteranno al suo sito possano usufruire di HTTPS.

Con particolari tool e programmi, la **creazione di un certificato digitale** in formato X.509 v3 è pressoché immediata. Sotto un sistema Windows server, per esempio, si può usare l'utility denominata *SelfSSL* nell'*IIS 6.0 Resource Kit* che permette di automatizzare con un wizard il processo di generazione e di installazione di un certificato auto-firmato.

Anche sotto Linux si possono creare certificati digitali «autoprodotti» e trasformare un server in un'autorità di certificazione.

L'unico inconveniente rimarrà il controllo che i browser effettueranno sul certificato digitale ricevuto, e rilasciato da un'autorità di certificazione «non trust»: non vedendolo rilasciato da una CA ufficiale, i browser segnaleranno con un messaggio a video la proposta della sua accettazione.

Tutti i principali browser, in ogni caso, possono accettare l'importazione di nuove autorità di certificazione non ufficiali da aggiungere a quelli ufficiali presenti nel sistema operativo (i cosiddetti **CA root**).

In questo caso non mostreranno più i messaggi di allerta sulla ricezione di certificati digitali emessi da un ente non ufficiale.

ESEMPIO

Importazione di un certificato digitale (Windows)

L'importazione dei certificati di nuove CA (certificati **CA root**) nei sistemi operativi Microsoft non è così semplice come in ambiente Linux, dove è sufficiente copiare il certificato in una determinata directory di configurazione.

Per importare un certificato digitale (per esempio non rilasciato da una CA ufficiale) si può usare il tool *Microsoft Management Console* (MMC).

Le operazioni da eseguire sono le seguenti:

1. *START/Esegui/mmc*
2. *File/Aggiungi/Rimuovi Snap-in/Aggiungi*
3. *Certificati/Aggiungi/Account del Computer/Computer Locale/Fine*
4. *Gestione Criteri di Protezione IP/Aggiungi/Computer Locale/Fine*
5. *Chiudi/OK*

Ora si è costruita una MMC personalizzata in grado di gestire i certificati digitali.

Per importare un certificato ora si può usare questa MMC:

1. Certificati/Personale/Tasto destro mouse/Tutte le Attività/ Importa
2. Avanti/Seleziona File (selezionare il file)
3. *Avanti/Inserisci Passphrase/Avanti/Attivare 'Selezionare automaticamente...' /Avanti/Fine*

Definire un'autorità di certificazione CA «privata» per gestire dei certificati digitali su una macchina server «privata» significa impostare vari elementi affinché il sistema funzioni correttamente.

Ricordare che nello schema agiscono i seguenti attori:

1) la Certification Authority autocostruita (per esempio, **CA locale**);
2) l'**organizzazione richiedente** un certificato digitale dalla CA locale, un server Web;
3) i **client Web** ignari della CA locale, ma che visiteranno il sito dell'organizzazione richiedente.

I passi dello schema sono:

a) creare il certificato **CA root** per la Certification Authority (CA locale), quindi assegnargli un nome, un periodo di validità, l'ampiezza in bit delle chiavi asimmetriche. Il file che conterrà la chiave privata della CA locale dovrà essere protetto in modo speciale.

   Il certificato CA root potrà essere importato dai client che vorranno connettersi ai siti Web che saranno certificati dalla CA locale. Il CA root contiene la chiave pubblica della CA locale, e servirà ai browser per decifrare i certificati digitali rilasciati dalla CA locale. I browser client dovranno aggiungere il CA root ai certificati ufficiali delle CA autorizzate. Il certificato CA root dovrà essere disponibile sul sito della CA locale affinché possa essere scaricato da chi ne abbia la necessità;

b) creare un modulo di richiesta (CSR, *Certificate Signing Request*), affinché un'organizzazione che vuole un certificato digitale garantito dalla CA locale possa comunicare i suoi dati (tra cui, per esempio, il nome dell'organizzazione richiedente, il nome del sito e la chiave pubblica da utilizzare);

c) creazione del certificato digitale per l'organizzazione richiedente e cifrato dalla CA locale con la sua chiave privata. Il certificato digitale viene creato a partire dal CSR compilato dall'organizzazione richiedente, e viene consegnato all'organizzazione richiedente che lo imposterà presso il suo server Web.

ESEMPIO

Gestione di una CA (Linux)

Tutte le fasi per gestire una CA che possa emettere certificati digitali possono essere fatte, sotto Linux, con l'applicazione *openssl*:

```
linux:~# openssl genrsa -des3 -out itisCA.key 1048
Generating RSA private key, 1048 bit long modulus
.....................+++
.....................+++
e is 65537 (0x10001)
Enter pass phrase for itisCA.key:(password)
Verifying - Enter pass phrase for itisCA.key:(password)
linux:~#
linux:~# openssl req -new -x509 -days 365 -key itisCA.key -out itisCA.crt
 Enter pass phrase: (password)
 Country Name (2 letter code): IT
 State or Province Name (full name): Italia
 Locality Name (eg, city): Parma
 Organization Name (eg, company): ITIS L. da Vinci
 Organizational Unit Name (eg, section) : prova CA
 Common Name (eg, your name): Paolo Ollari
 Email Address: paolo.ollari@itis.pr.it
linux:~#
linux:~# openssl genrsa -des3 -out itisSERVER.key 1024
Generating RSA private key, 1048 bit long modulus
.....................+++
.....................+++
e is 65537 (0x10001)
Enter pass phrase for itisSERVER.key:(password)
Verifying - Enter pass phrase for itisSERVER.key:(password)
linux:~#
linux:~# openssl req -new -key itisSERVER.key -out itisSERVER.csr
 Enter pass phrase: (password)
 Country Name (2 letter code): IT
 State or Province Name (full name): Italia
 Locality Name (eg, city): Parma
 Organization Name (eg, company): Itis L. da Vinci
 Organizational Unit Name (eg, section): prova Server Web
 Common Name (eg, hostname): www.itis.pr.it
 Email Address: paolo.ollari@itis.pr.it
linux:~#
linux:~# openssl -x509 -req -in itisSERVER.csr -out itisSERVER.crt -
sha1 -CA itisCA.crt -CAkey itisCA.key -CAcreateserial -days 365
Signature ok
 subject=/C=IT/ST=Italia/L=Parma/O= Itis L. da Vinci /OU=prova
Server Web/CN= www.itis.pr.it /Email= paolo.ollari@itis.pr.it
 Getting CA Private Key
 Enter pass phrase: (password)
linux:~#
```

Prima di tutto si crea la chiave privata della CA (itisCA.key).

Quindi il certificato CA root tramite la chiave privata, in modo che il CA root contenga la relativa chiave pubblica (itisCA.crt).

Ora una chiave pubblica per un'organizzazione richiedente (itisSERVER.key).

Quindi il modulo di richiesta dell'organizzazione richiedente in cui sarà inserita la chiave pubblica appena creata (itisSERVER.csr).

Infine il certificato digitale da consegnare all'organizzazione richiedente (itisSERVER.crt) a partire dal modulo di richiesta, dal CA root cifrato con la chiave privata della CA root.

Notare che la password, *in questo caso, deve essere quella digitata nel primo passaggio, quello in cui si è creata la chiave privata del CA root.*

Tutti i passaggi possono essere fatti anche con il pacchetto *openSSL* per Windows.