

## **Chapitre II**

# **Le protocole SSL (Secure Socket Layer)**

### **Plan de chapitre :**

- **Introduction**
- **Présentation de protocole SSL**
- **Principe de fonctionnement**
- **Composition de protocole SSL**
- **Le PKI (Public Key Infrastructure)**
- **Conclusion**

## 2.1 Introduction

Toutes les données qui circulent sur internet peuvent être accédées par des personnes qui ne sont pas les récepteurs destinés, et lorsque ces données contiennent des informations privées, comme le mot de passe et numéros de carte de crédit..., on a besoin des mécanismes pour mettre ces données incompréhensibles pour assurer que ne seront pas inspectées durant le transport. Le protocole Secure Socket Layer (SSL) est parmi les protocoles qui sont désignés pour aider à la protection de données durant la transmission.

## 2.2 Origine [3]

SSL (Secure Socket Layer) est un protocole à négociation, développé à l'origine par Netscape en 1994.

Il a pour but de sécuriser les transactions Internet, par authentification du client (un navigateur la plupart du temps) et du serveur, et par chiffrement de la session.

La sécurisation des connexions à l'aide du protocole SSL doit assurer que :

- La connexion assure la confidentialité des données transmises.
- La connexion assure que les données transmises sont intègres.
- L'identité des correspondants peut être authentifiée.
- La connexion est fiable.

La version 2.0 vient de Netscape et la version 3.0, qui est actuellement la plus répandue, a reçu les contributions de la communauté internationale en 1996.

SSL est indépendant des applications qu'il l'utilisées lors des transactions et s'applique sous les protocoles HTTP, FTP, Telnet... etc.

Clients et serveurs commencent par s'authentifier mutuellement (l'un après l'autre), puis négocient une clé symétrique de session qui servira à assurer la confidentialité des transactions. L'intégrité de ces dernières est assurée par l'application de HMAC (Hashed Message Authentication Code).

En 2001, le brevet de SSL a été racheté par l'IETF (Internet Engineering Task Force) qui l'a *rebaptisé* *TLS* (Transport Layer Security).

Le protocole TLS correspond à la version 3.1 de SSL.

TLS semble destiné à supplanter SSL dans les développements futurs. Les détails de TLS sont trouvés dans RFC 2246.

### 2.3. Cas d'utilisation

SSL ou, Secure Socket Layer, permet l'accès sécurisé à un site web ou à des certaines pages d'un site web. Ces connexions se différencient par des connexions "normales" c'est à dire non sécurisées, par le fait que l'adresse n'est plus "http://" mais "https://", où le S indique sécurisé.

Il y a deux manières de vérifier si le site visité est sécurisé ; d'une part en essayant de s'y connecter avec l'adresse https qui dans le cas où le site dispose d'une connexion sécurisée permettrait la connexion avec le site. Et d'autre part en vérifiant si le cadenas figurant à droite de la barre d'état d'un navigateur est fermé.



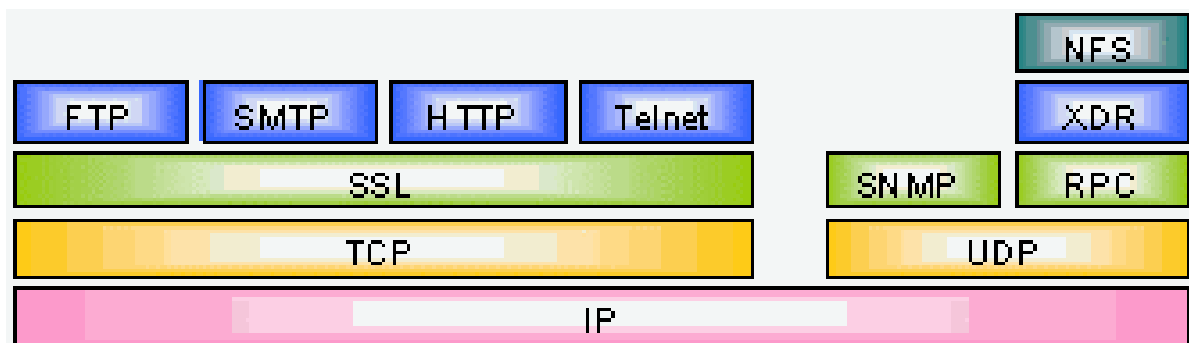
**Figure 2.1.** Site sécurisé dont l'adresse possède le préfixe « https »



**Figure. 2.2.** Différence entre un site sécurisé et un autre qui ne l'ai pas.

SSL est très largement adopté par les messageries existantes comme: Qualcomm Eudora 5.1r, Microsoft Outlook..., entre autre, toutes supportent les connexions SSL sur leurs serveurs.

SSL se situe au sommet de la couche TCP/IP, et au-dessous de la couche d'application. Pour mettre en place une connexion SSL, il faut d'abord établir une connexion TCP/IP, car SSL utilise certaines "primitives" de TCP/IP (tunnel). Ainsi SSL peut être vu comme un canal sûr au sein de TCP/IP, où tout le trafic entre deux applications "Peer to Peer" est échangé de manière cryptée. Tous les appels de la couche d'application à la couche TCP, sont remplacés par des appels de l'application à SSL, et c'est SSL qui se charge des communications avec TCP.



**Figure2.3** SSL selon le modèle OSI

## 2.4. Fonctionnalités [4]

SSL a trois fonctions:

- ***Authentification du serveur :***

Qui permet à un utilisateur d'avoir une confirmation de l'identité du serveur. Cela est fait par les méthodes de chiffrement à clés publiques qu'utilise SSL. Cette opération est importante, car le client doit pouvoir être certain de l'identité de son interlocuteur à qui par exemple, il va communiquer son numéro de carte de crédit.

- **Authentification du client :**

Selon les mêmes modalités que pour le serveur, il s'agit de s'assurer que le client est bien celui qu'il prétend être.

- **Chiffrement des données :**

Toutes les données qui transitent entre l'émetteur et le destinataire, sont chiffrées par l'émetteur, et déchiffrées par le destinataire, ce qui permet de garantir la confidentialité des données, ainsi que leur intégrité grâce souvent à des mécanismes également mis en place dans ce sens.

## 2.5. Composition [3]

SSL se subdivise en quatre sous protocoles; le **SSL record Protocol**, et le **SSL handshake protocol** ; Plus deux autres protocoles, mais qui ont un rôle moins essentiel, c'est le **SSL Change Cipher Spec**, et le **SSL Alert**.

Le SSL record protocole définit le format qui sera utilisé pour l'échange des données. Alors que le SSL **handshake** se charge des différents échanges de messages entre le client et le serveur, au moment où ils établissent la connexion comme l'authentification, la version de protocole, l'algorithme de cryptage, ...

Une session SSL est définie par les paramètres suivants partagés entre un client et un serveur :

- **Session identifier** : un octet fixé par le serveur pour identifier la session
- **Peer certificat** : un certificat pour le serveur, éventuellement un autre pour le client
- **Cipher Spec** : définit l'algorithme de chiffrement et l'algorithme de condensation
- **Master secret** : clé de 48 octets négociée entre le serveur et le client.
- **Compression method** : NULL pour l'instant (en SSLv3 ou TLS)
- **Is resumable**: flag qui indique si de nouvelles connexions peuvent être créées à partir de cette session.

### 2.5.1. Le protocole SSL handshake :

Ce protocole permet s'authentifier un client et un serveur mutuellement, de négocier les algorithmes de chiffrement, négocier les algorithmes de MAC et enfin de négocier les clés symétriques qui vont servir au chiffrement.

Chaque message échangé entre le client et le serveur contient trois champs :

- **Type**, indique l'objet du message (1 octet)
- **Length**, indique la longueur du message (3 octets)
- **Content**, contient les données transmises (plus d'un octet)

#### Phase 1 : Etablissement des paramètres de sécurité

Cette phase a pour but d'établir le lien sécurisé. Les paramètres du premier message, client\_hello, envoyé par le client, sont :

**Version**: la plus haute version de SSL que puisse utiliser le client.

**Random**: Un horodatage de 32 bits et une valeur aléatoire de 28 octets générée par le client.

**Session ID** : Un nombre, qui identifie la connexion. Un zéro signifie la volonté du client d'établir une nouvelle connexion sur une nouvelle session. Un autre nombre signifie la volonté de changer les paramètres ou de créer une nouvelle connexion sur la session existante.

**CipherSuite** : Une liste, par ordre décroissant de préférence, des algorithmes que supporte le client. Il s'agit des algorithmes d'échange de clé et de chiffrement.

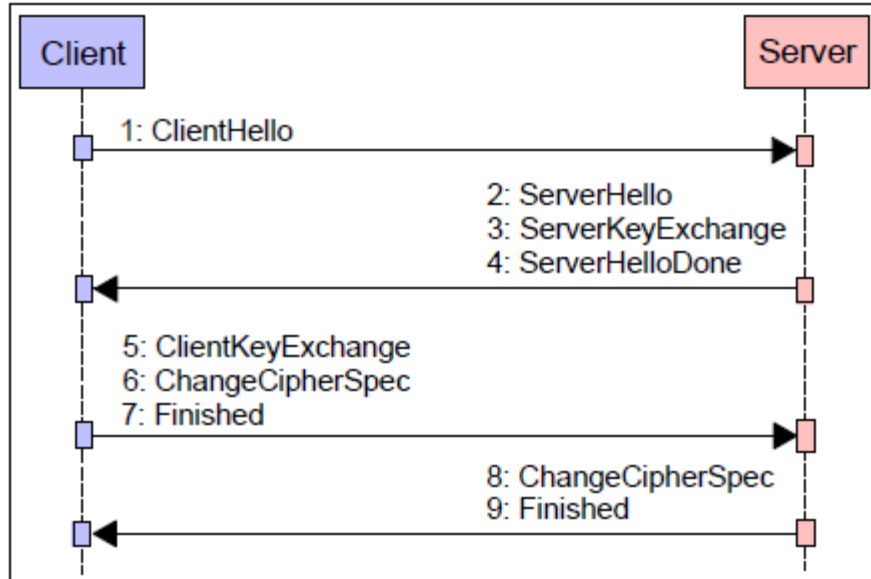
#### KeyExchange :

C'est la clé utilisée pour chiffrer et déchiffrer les données envoyées par les deux côtés.

#### CipherSpec :

- Cipher Algorithm : DES, 3-DES, RC2, RC4, IDEA...
- MAC Algorithm : MD5 ou SHA-1...
- CipherType : Stream ou Block

Après avoir envoyé ces requêtes, le client attend que le serveur réponde en générant une valeur aléatoire, en indiquant la version, et les meilleurs algorithmes qu'il peut utiliser : ce sera la réponse `server_hello` du serveur.



**Figure 2.4** les étapes d'établissement d'une connexion sécurisée.

1. Premièrement le client envoie un message **ClientHello**.
2. Le serveur répond avec un message **ServerHello** aussi.
3. Le serveur envoie leurs informations de clé publique dans le message **ServerKeyExchange**.
4. le serveur termine la première phase de négociation avec un message **ServerHelloDone**.
5. Le client répond par la clé de session (cryptée avec la clé publique de serveur) dans un message **ClientKeyExchange**.
6. Envoie un autre message **ChangeCipherSpec** pour activer l'option de négociation pour toutes les communications futures.

7. A la fin le client fini la négociation par le message **Finished**.
8. Après la vérification Le serveur envoie un message ChangeCipherSpec pour activer les nouvelles négociations (le serveur peut annuler les cipher de client et proposé des autres).
9. Le serveur fini cette étapes par un message **Finished**.

### Phase 2 : Authentification du serveur et échange des clés

Lors de cette phase, le serveur commence par l'envoi de son certificat. Ce message peut contenir une chaîne de certificats X.509.

Ensuite, le serveur peut demander au client un certificat : c'est le message **certificate\_request** (rarement implémenté dans la réalité), qui comprend les champs **certificate\_type** (algorithme public utilisé) et **certificate\_authorities** (liste des AC valides).

Finalement, le serveur envoie le message **server\_done**, qui signifie la fin de cette phase et que le serveur se met en attente.

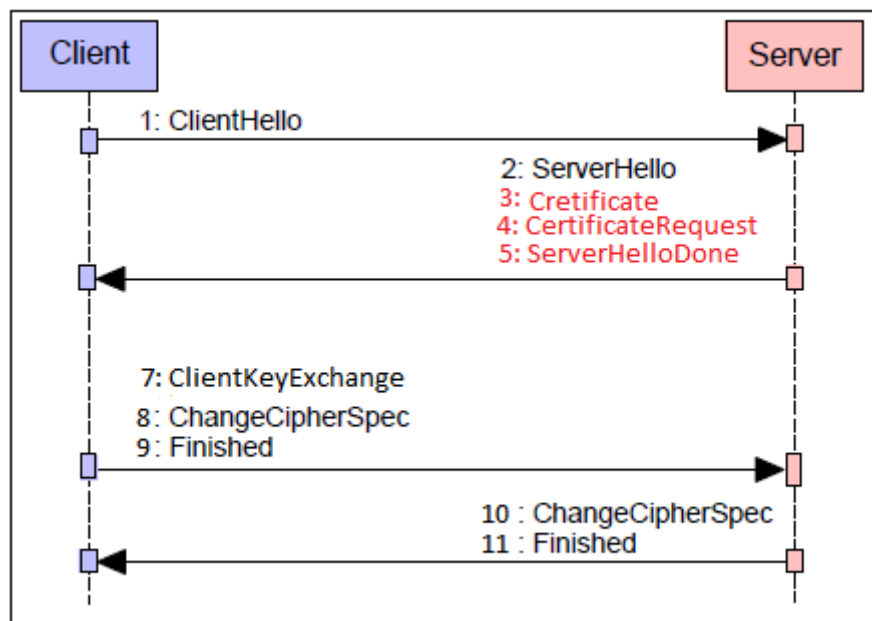


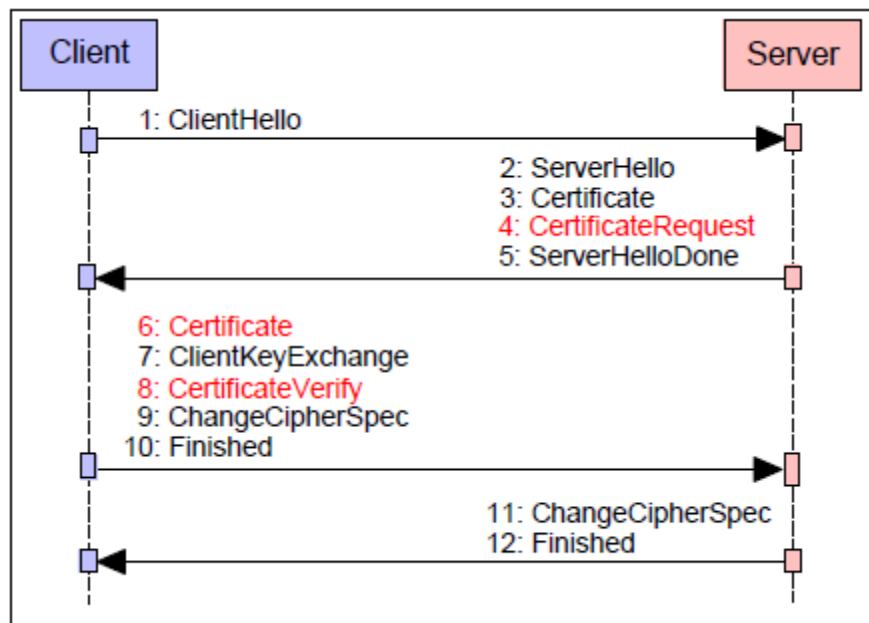
Figure 2.5 authentification de serveur et échange de clés. [5]



### Phase 3 : Authentification du client et échange des clés

Le client doit vérifier que le certificat envoyé par le serveur est valide, et que les autres paramètres sont corrects. Si le serveur a demandé au client d'envoyer un certificat, le client envoie un message **certificate**, contenant le certificat (s'il n'a pas de certificat, il envoie un message **no\_certificate**). Le client envoie ensuite un message **client\_key\_exchange**, qui dépend de l'algorithme choisi.

Pour finir cette phase, le client envoie un message **certificate\_verify**.



**Figure 2.6** Authentification de client et échange de clés.[5]

### Phase 4 : Fin

Le client envoie le message (1 octet) **change\_cipher\_spec**, qui est en fait l'unique message du protocole **Change\_Cipher\_Spec**, et active pour la session courante les algorithmes, clés et sels du handshake. Puis le client envoie le message **finished**, qui authentifie le client et valide l'échange de clé (le message est constitué d'un condensât de la clé symétrique échangée, de l'ensemble des messages échangées durant le handshake et d'un identificateur de l'expéditeur).

Le serveur répond en envoyant son propre **change\_cipher\_spec** et terminer le **handshake**.

### 2.5.2. Le protocole SSL Alert

Ce protocole spécifie les messages d'erreur que peuvent s'envoyer aux clients et serveurs. Les messages sont composés de deux octets, le premier est soit "warning" soit "fatal". Si le niveau est "fatal", la connexion est abandonnée. Les autres connexions sur la même session ne sont pas coupées mais on ne peut pas en établir de nouvelles.

Le deuxième octet donne le code d'erreur.

Les erreurs fatales sont :

- `unexpected_message` : message non reconnu
- `bad_record_mac` : MAC non correct.
- `handshake_failure` : impossible de négocier les bons paramètres.
- `illegal_parameter` : un champ était mal formaté ou ne correspondait à rien.

Les warnings sont :

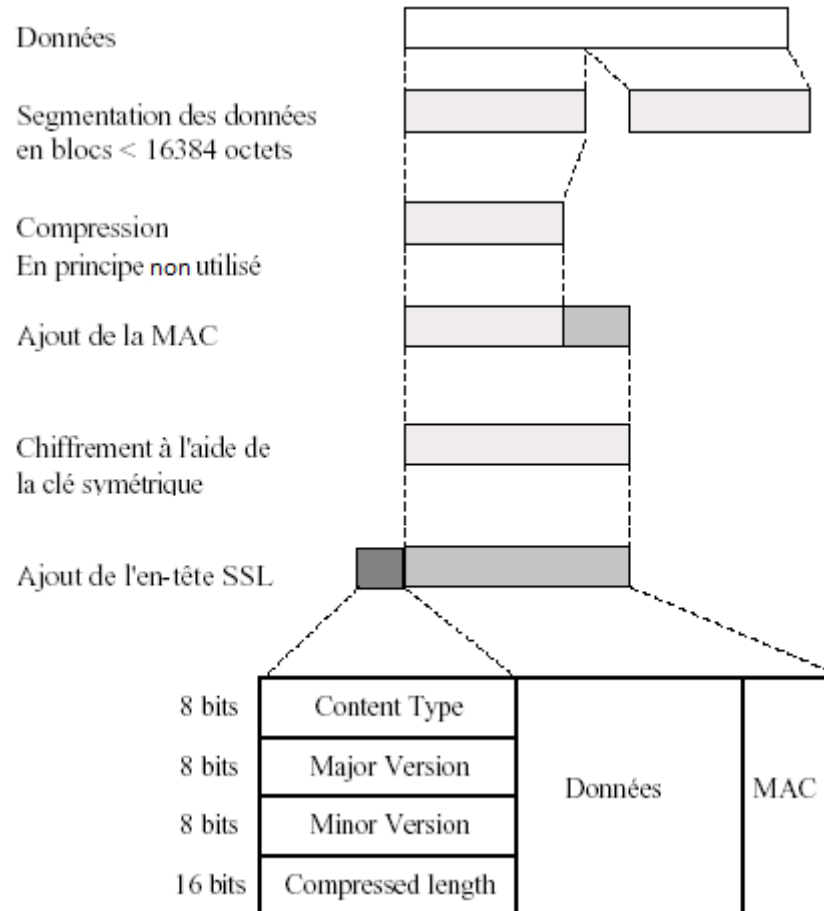
- `close_notify` : annonce la fin d'une connexion
- `no_certificate` : réponse à une demande de certificat s'il n'y en a pas.
- `bad_certificate` : le certificat reçu n'est pas bon (exp. signature non valide)
- `unsupported_certificate` : le certificat reçu n'est pas reconnu...
- `certificate_revoked` : certificat révoqué par l'émetteur.
- `certificate_expired`.
- `certificate_unknown` : tous les problèmes concernant les certificats et non listés au-dessus...

### 2.5.3. Le protocole SSL Record

Ce protocole permet de garantir :

- La confidentialité des données transmises.
- L'intégrité des données transmises.

Schématiquement, le protocole SSL Record ressemble à :



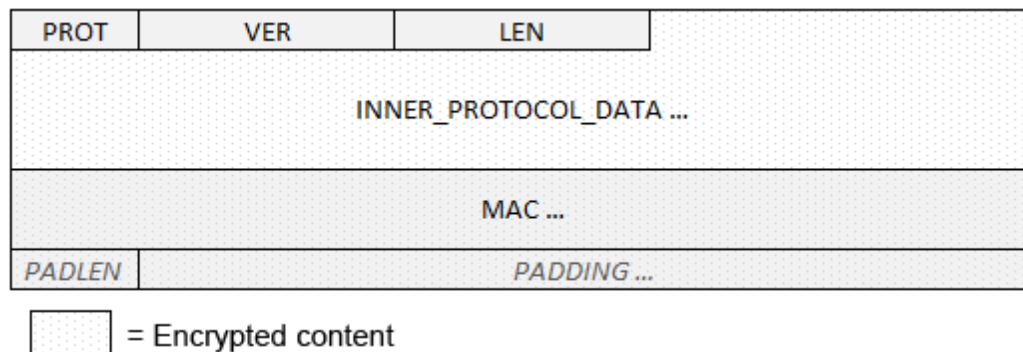
**Figure2.8** Composition du sous-protocole SSL Record

Dans l'entête qui est ajoutée :

- Content Type : le plus haut protocole utilisé pour traiter ce fragment
- Major Version : plus haute version de SSL utilisée (3 pour SSLv3)
- Minor Version : plus basse version utilisée (0 pour SSLv3)
- Compressed Length : la longueur en octets des données (éventuellement compressées) à chiffrer.

Le **ssl/tls Record protocol** est défini comme une couche protocolaire. Dans chaque couche, le message peut inclure des champs pour la longueur, description, et le contenu. Le Record Protocol prend le message à transmettre et le fragmente en sous-blocks, puis les compresse (optionnel), applique le hachage MAC, les encrypte et finalement le résultat est transmis. Dès la réception, les données seront décryptées, vérifiées, décompressées, et réassemblées, puis délivrées au client.

Le Record Protocol généralement n'est pas intéressé par le contenu à traiter. Il distingue seulement trois types de messages *SSL/TLSPlaintext*, *SSL/TLSCompressed*, et *SSL/TLSCiphertext*. Et comme ses noms l'indiquent, *SSL/TLSPlaintext* sont les données en texte clair, le *SSL/TLSCompressed* sont les données compressées et le *SSL/TLSCiphertext* sont les données cryptées (où les données sont compressées et contiennent le MAC).



**Figure 2.9** SSL/TLSCiphertext message

**PROT** – le type de protocole interne, 1 Octet (20 (ChangeCipherSpe), 21 (Alert), 22 (Handshake) or 23 (Application data))

**VER** : la version de protocole SSL/TLS, 2 Octet (Exp. 0x0301 pour TLS 1.0)

**LEN** : la longueur de message SSL/TLS Record Layer sans PROT,

**INNER\_PROTOCOL\_DATA** : notre message, une taille variable

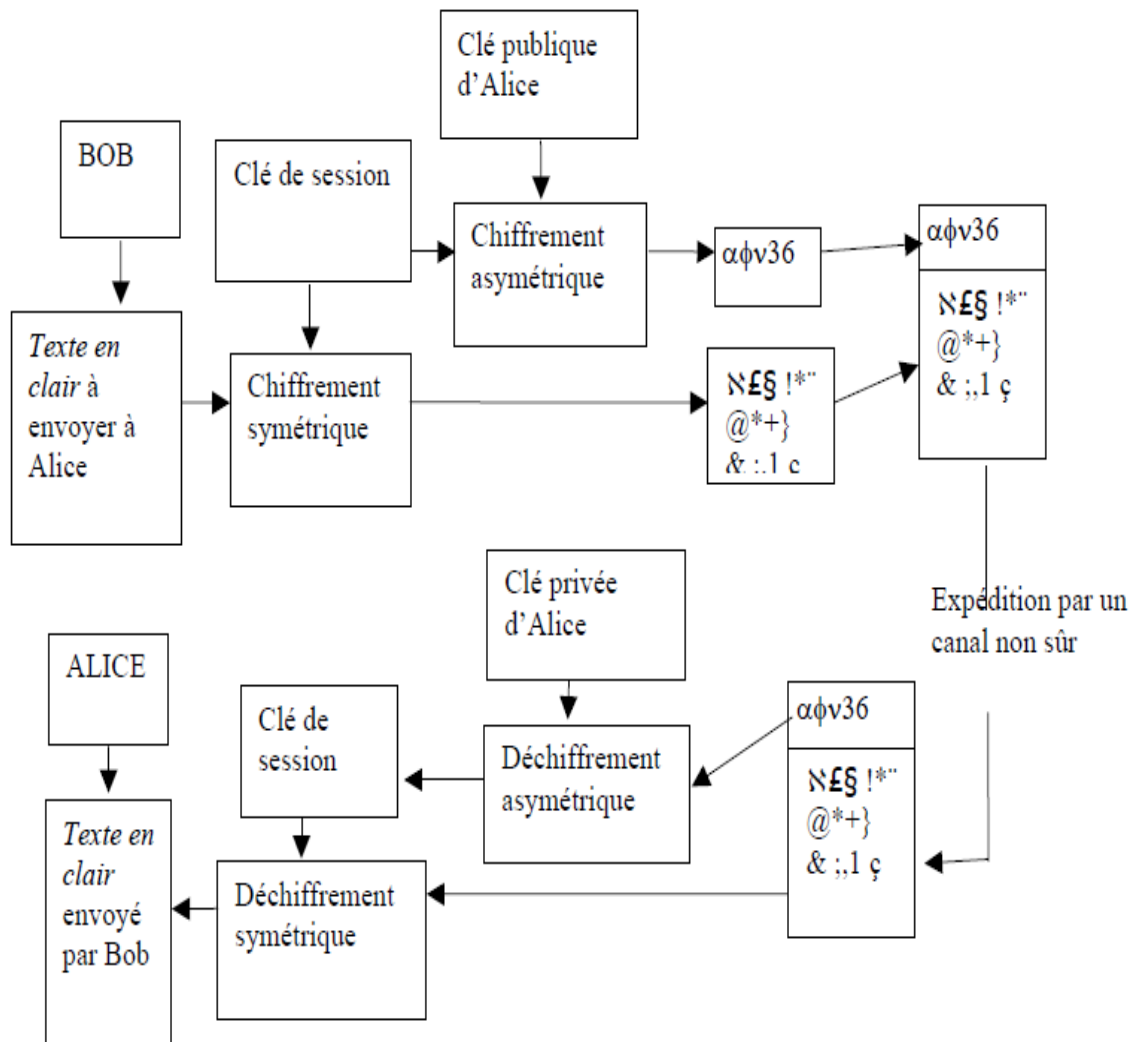
**MAC** : Message Authentication Code

La fonction de condensation (HASH()) est soit MD5 soit SHA-1

**Padding** : La valeur du PADDING spécifie le nombre de bytes de données ajoutés à la structure originale par le destinataire.

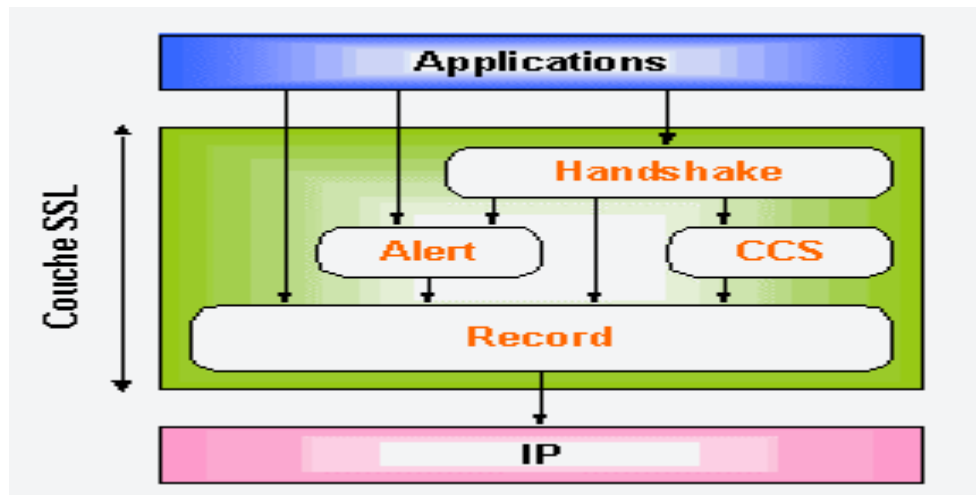
**Encrypted content:** le message SSL/TLS Record Layer encrypté par l'algorithme de chiffrement sans PROT, VER et LEN.

La figure ci-dessous représente la démarche de chiffrement et de déchiffrement.



**Figure 2.10** l'architecture de chiffrement mixte.

Pour résumer, les protocoles s'assemblent de la façon suivante :



**Fig. 2.11** Pile des sous-protocoles de SSL

En effet, lors de connexion avec un site sécurisé toutes les transactions sont faites dans un tunnel sécurisé nommé **tunnel HTTPS**.

## 2.6. Tunnel HTTPS

Le protocole HTTPS (HyperText Transfert Protocol Secure) est une combinaison de protocole HTTP avec une couche de chiffrement de type **SSL/TLS**. Il garantit (théoriquement) la confidentialité et l'intégrité des données envoyées et reçues par l'utilisateur. Il permet aussi de garantir l'identité du serveur et/ou du client à l'aide de certificats d'authentification.

## 2.7. Limitations

Malgré tous ces évolutions de protocole SSL ; il reste souffrir de plusieurs problèmes tel que l'héritage des faiblesses des protocoles utilisés (algorithmes de cryptages,...),Le deuxième est les identités (l'authentification)des partenaires et le management de clés puisque ces traitement sont faites à distance. Le management de clés se fait par des organisations spécifiées ; les plus répandus est l'organisation **Public Key Infrastructure (PKI)**.

## 2.8. Infrastructure à clés publiques : [06]

Une **infrastructure à clés publiques** (ICP) ou **infrastructure de gestion de clés** (IGC) ou encore **Public Key Infrastructure** (PKI), est un ensemble de composants physiques (des ordinateurs, des équipements cryptographiques logiciels ou matériel type HSM ou encore des cartes à puces), de procédures humaines (vérifications, validation) et de logiciels (système et application) en vue de gérer le cycle de vie des certificats numérique.

Une infrastructure à clés publiques délivre un ensemble de services pour le compte de ses utilisateurs.

En résumé, ces services sont les suivants :

- enregistrement des utilisateurs (ou équipement informatique) ;
- génération de certificats ;
- renouvellement de certificats ;
- révocation de certificats ;
- publication de certificats ;
- publication des listes de révocation (comprenant la liste des certificats révoqués) ;
- identification et authentification des utilisateurs (administrateurs ou utilisateurs qui accèdent à l'ICP) ;

- archivage, séquestre et recouvrement des certificats (option).

### 2.8.1. Composition

Le PKI à cause de sa complexité de fonctionnement il est divisé en plusieurs entités :

- **L'autorité de certification (AC ou CA)** qui a pour mission de signer les **demandes de certificat (CSR : *Certificate Signing Request*)** et de signer les **listes de révocation (CRL : *Certificate Revocation List*)**. Cette autorité est la plus critique.
- **L'autorité d'enregistrement (AE ou RA)** qui a pour mission de générer les certificats, et d'effectuer les vérifications d'usage sur l'identité de l'utilisateur final (les certificats numériques sont nominatifs et uniques pour l'ensemble de l'ICP).
- **L'autorité de dépôt (*Repository*)** qui a pour mission de stocker les certificats numériques ainsi que les listes de révocation (CRL).
- **L'entité finale (EE : *End Entity*)**. L'utilisateur ou le système qui est le sujet d'un certificat (En général, le terme « entité d'extrémité » [10])

### 2.8.2. Les certificats numériques

Usuellement, on distingue deux familles de certificats numériques :

- **les *certificats de signature***, utilisés pour signer des documents ou s'authentifier sur un site web,
- **les *certificats de chiffrement*** (les gens qui vous envoient des courriels utilisent la partie publique de votre certificat pour chiffrer le contenu que vous serez seul à pouvoir déchiffrer),

#### 2.8.2.1. Nature et composition d'un certificat

Un certificat électronique est une donnée publique. Suivant la technique des clés asymétriques, à chaque certificat électronique correspond une clé privée, qui doit être soigneusement protégée.



L'ensemble de ces informations (comprenant la clé publique) est signé par l'autorité de certification de l'organisation émettrice. Cette autorité a la charge de:

- s'assurer que les informations portées par le certificat numérique sont correctes.
- s'assurer qu'il n'existe, pour une personne et pour une même fonction, qu'un et un seul certificat valide à un moment donné.

Le certificat numérique est donc, à l'échelle d'une organisation, un outil pour témoigner, de façon électroniquement sûre, d'une identité.

## **Modes de création**

Il existe deux façons distinctes de créer des certificats électroniques : le mode centralisé et le mode décentralisé.

- le mode décentralisé est le mode le plus courant : il consiste à faire créer, par l'utilisateur (ou, plus exactement par son logiciel) la bclé cryptographique et de joindre la partie publique de la clef dans la CSR. L'Infrastructure n'a donc jamais connaissance de la clé privée de l'utilisateur, qui reste confinée sur son poste de travail ou dans sa carte à puce.
- le mode centralisé consiste en la création de la bclé par l'AC : au début du cycle de la demande, la CSR ne contient pas la clé publique, c'est l'AC qui la produit. Elle peut ainsi avoir de bonnes garanties sur la qualité de la clé (aléa) et peut... en détenir une copie protégée. En revanche, il faut transmettre à l'utilisateur certes son certificat (qui ne contient que des données publiques) mais aussi sa clé privée ! L'ensemble de ces deux données est un fichier créé sous le format PKCS#12. Son acheminement vers l'utilisateur doit être entrepris avec beaucoup de précaution et de sécurité, car toute personne mettant la main sur un fichier PKCS#12 peut détenir la clé de l'utilisateur.

### 2.8.3. Le certificat X.509

Est un standard international publié la première fois en 1988 comme un format pour les certificats de clés publiques, la liste des certificats révoqués, les attributs de certificat et le chemin de l'algorithme de validation de certificats.

Le certificat X.509 est le standard le plus utilisé dans l'internet. En 1988 la version 01(V1) a été réalisée et en 1993 la création de version 02 et en fin la version 03 réalisée en juin 1996.

Aujourd'hui le certificat X.509 V3 est le plus recommandé à utiliser et son format est la suivantes :

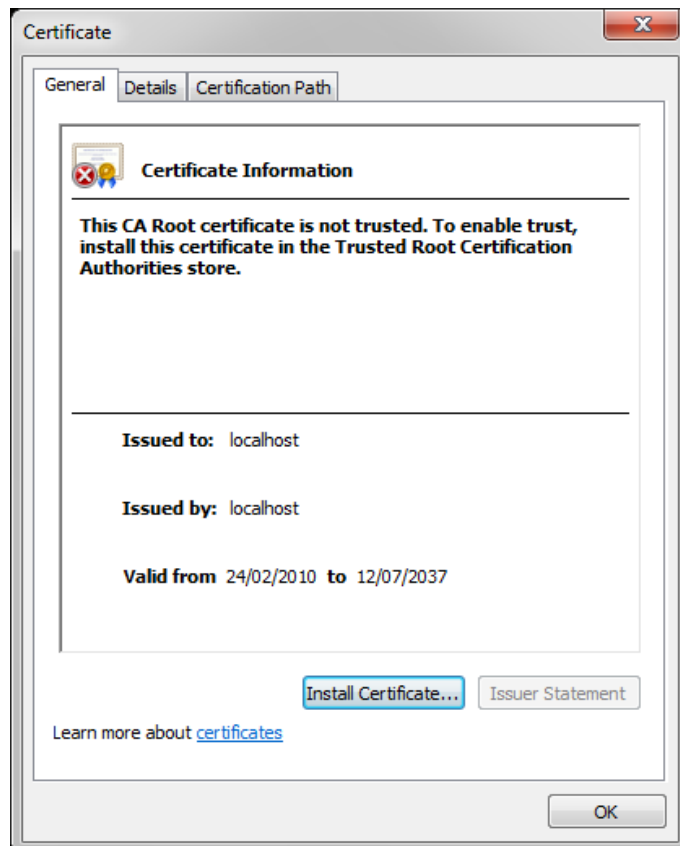
- **Version** : indique à quelle version de X509 correspond ce certificat.
- **Numéro de série** : Numéro de série du certificat.
- **Algorithme de signature**: identifiant du type de signature utilisée.
- **Emetteur** : Distinguished Name (DN) de l'autorité de certification qui a émis ce certificat.
- **Valide à partir de**: la date de début de validité de certificat.
- **Valide jusqu'à** : la date de fin de validité de certificat.
- **Objet**: Distinguished Name (DN) de détenteur de la clef publique.
- **Clé publique** : infos sur la clef publique de ce certificat.
- **Utilisation de la clé** : l'objet d'utilisation de la clé.
- **Thumbprint** : signature numérique de l'autorité de certification sur l'ensemble des champs précédents.

Et pour plus informations autour le certificat X.509 v3 voir RFC 5280 [14].

### 2.8.3.1. Scénario de fin de vie

Il existe deux scénarios possibles de fin de vie d'un certificat numérique :

- le certificat numérique expire (chaque certificat numérique contient une date de « naissance » et une date de « péremption »).
- le certificat est révoqué, pour quelque raison que ce soit (perte de la clé privée associée, etc.) et dans ce cas, l'identifiant du certificat numérique est ajouté à une liste de certificats révoqués (CRL pour Certificate Revocation List) pour informer les applications qu'elles ne doivent plus faire confiance à ce certificat.



**Figure 2.12** exemple de certificat X.509 v3 crée par l'outil de java **Keytool**.

## 2.9. Conclusion

A l'aide de tous ces traitements le protocole SSL devenu un protocole de communication le plus sécurisé dans l'internet. Mais il reste souffrir de plusieurs problèmes comme la centralisation de traitements de PKI c.-à-d. ils y faite en contact avec le CA, et si un tiers se situer dans le chemin de circulation des données il peut les entamer et les modifier avant que sont reçus au destinataire origine c'est ce qu'on appelle l'attaque **man in the middle(MITM)** qui est un grand problème qui intervienne le **SSL**.

Le chapitre suivant comprend une étude détaillé sur cette attaque.