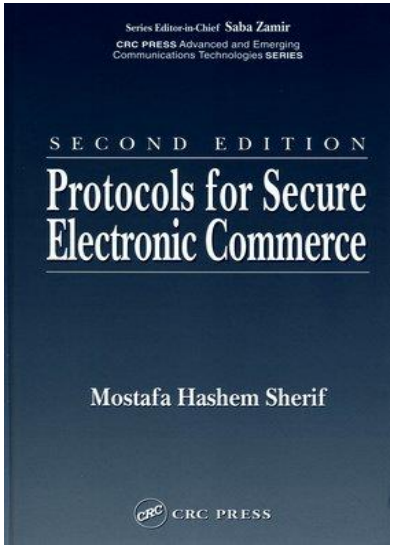
Sécurité du support de communication « SSL/TLS »

Rida Khatoun, maître de conférences, Télécom-ParisTech

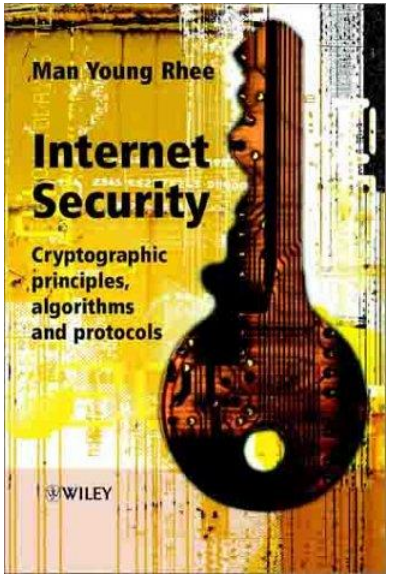
rida.khatoun@telecom-paristech.fr

Références

- **Protocols for Secure Electronic Commerce**
  - Auteur : M. Sherif
  - Editeurs : CRC Press (Taylor and Francis Group)
  - Nb de pages : 640 pages



- **Internet Security : Cryptographic Principles, Algorithms and Protocols**
  - Auteur : Man Young Rhee
  - Nb de pages : 424 pages
  - Editeur : John Wiley & Sons Ltd

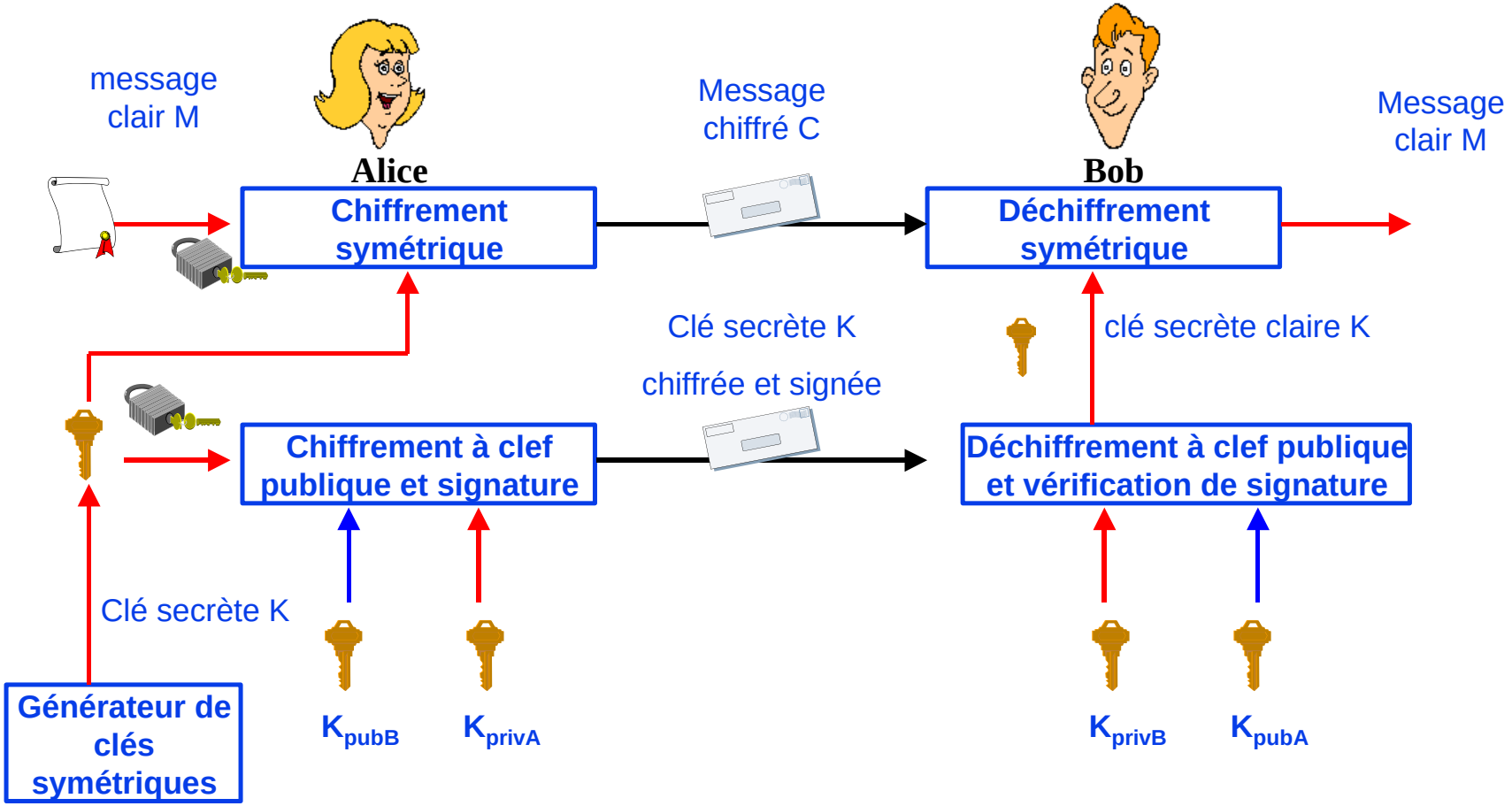


Sommaire

- Le protocole SSL/TLS : *Secure Socket Layer*
  - Rappel
  - Contexte, définition et normes
  - Services
  - Architecture/sous-protocoles
  - Certificat numérique/acteurs
  - Limites et vulnérabilités

Rappel : services sécurité

- Services de sécurité



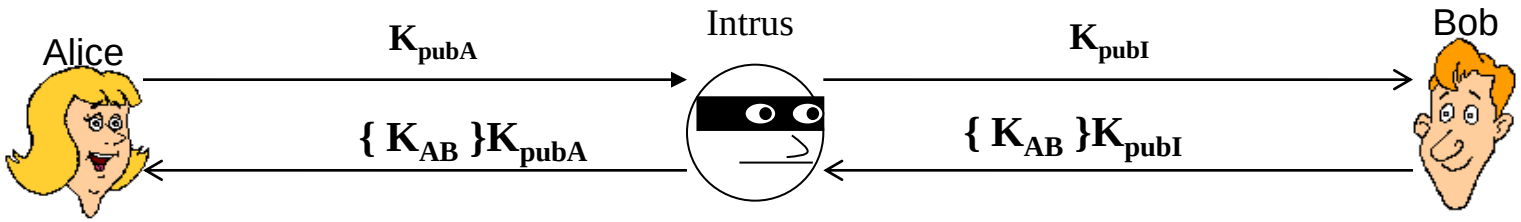
Vulnérabilités : exemple

- Attaques de l'homme au milieu : MITM

Alice dispose de

K_{pubA} clé publique

K_{privA} clé privée



Nécessité d'authentifier les clés publiques

Nécessité de gérer les clés publiques:

Durée de vie, usages, révocation, …

… déployer une infrastructure de confiance

Pourquoi le protocole TLS ?

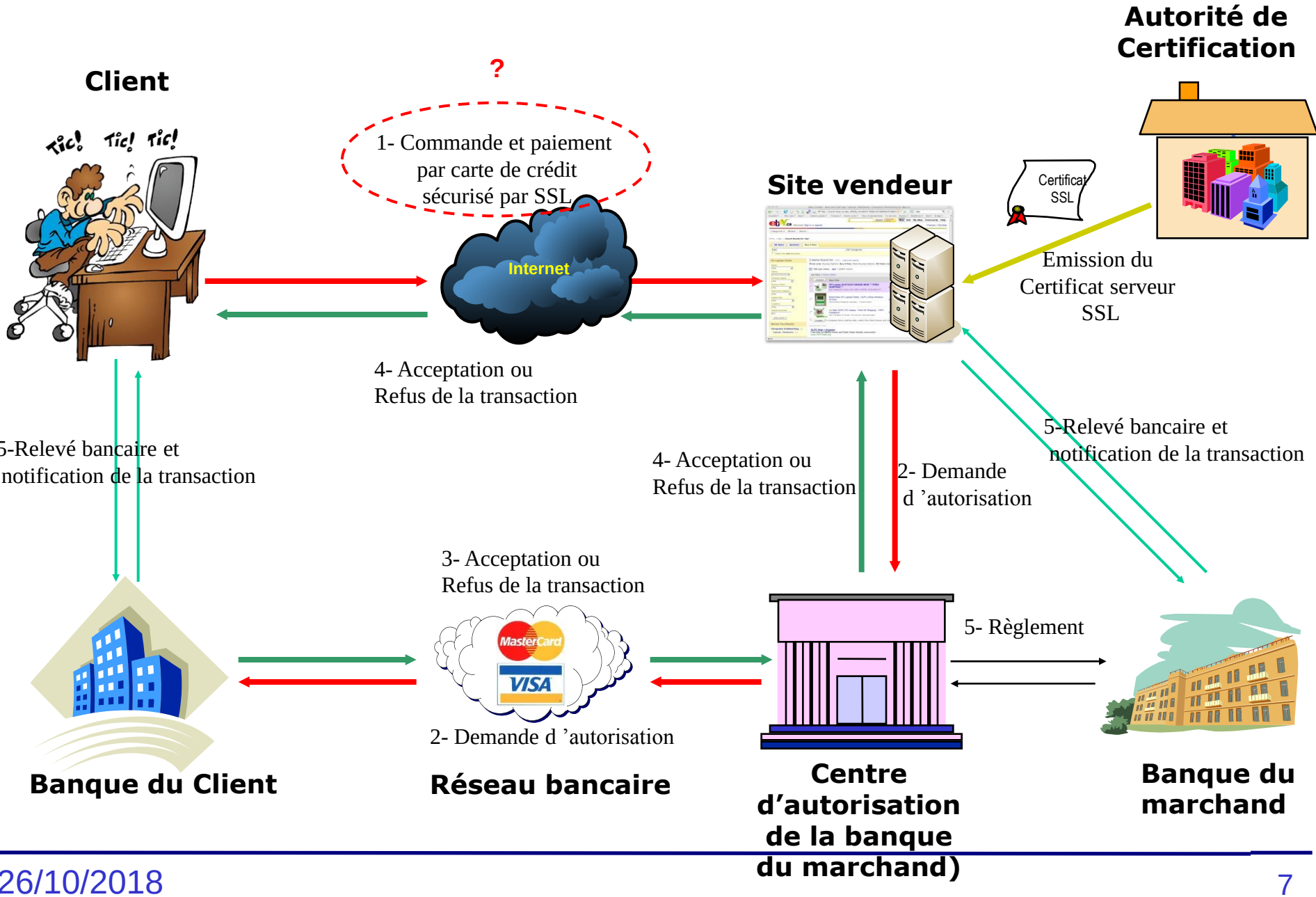
- SSL/TLS Pourquoi ?

Exemple : HTTPS est une version sécurisée du protocole HTTP

Confiance ?

Besoin d'un protocole sécurisé pour les sites web qui ont des données sensibles à échanger

Etapes et acteurs dans une transaction



Contexte

- SSL proposé par Netscape* et intégré au browser
  - 1994 : Première version de SSL testé en interne (pas déployée)
  - 1994 : Première version de SSL diffusé : SSL 2.0
  - 1996 : SSL 3.0, version stable du protocole, a été soumise à l'IETF pour une standardisation formelle
  - TLS est la nouvelle version de SSL 3.0, reprise par l'IETF

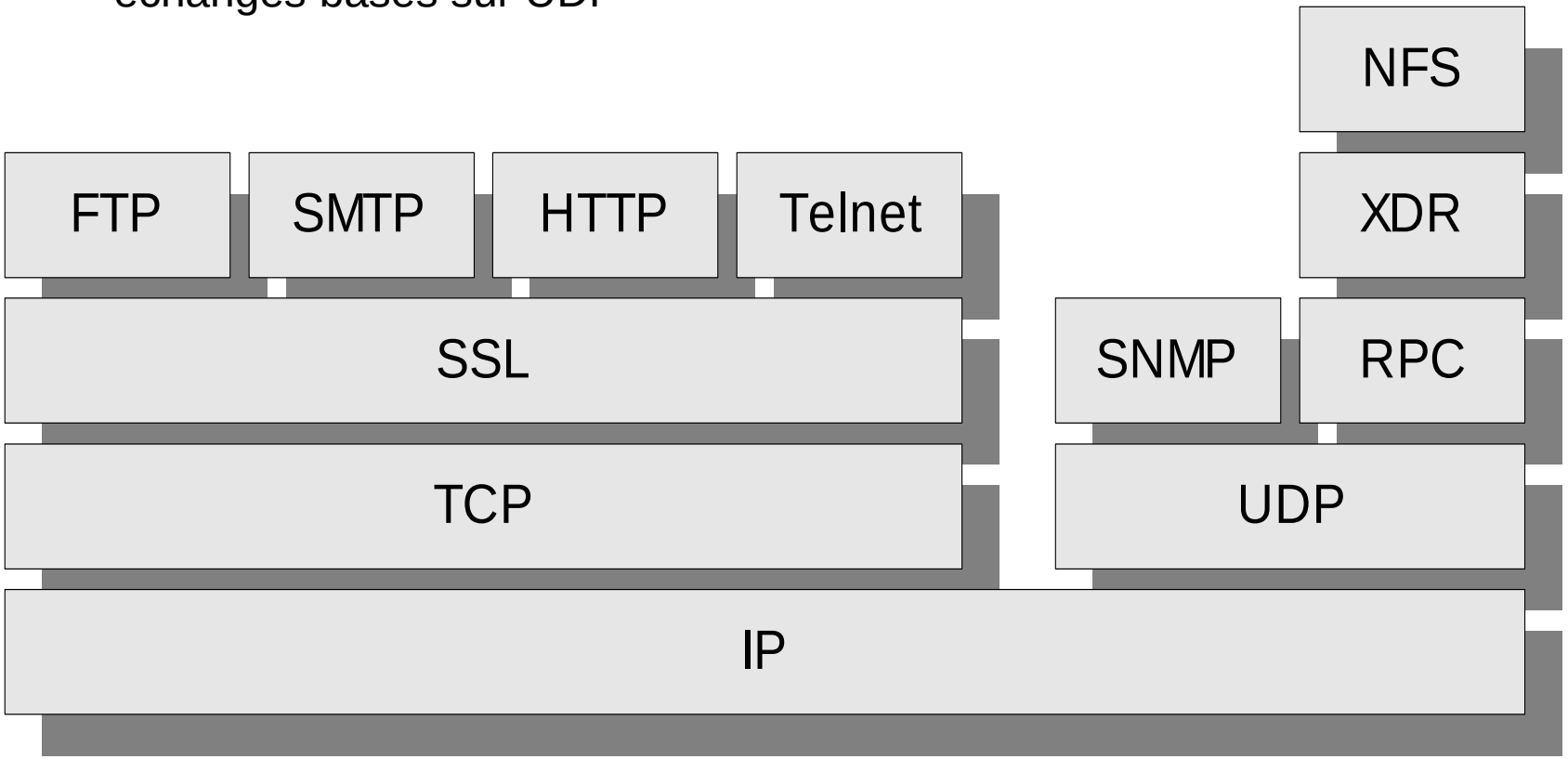
- IETF au sein du groupe Transport Layer Security
  - RFC2246 : première version 1.0 publiée par IETF en 1999
  - RFC2817 : addition de Kerberos** à TLS, 2000
  - RFC2818 : HTTP sur TLS, 2000
  - RFC3268 : utilisation d'AES pour TLS, 2002
  - RFC4346 : TLS version 1.1, 2006
  - RFC5246 : TLS version 1.2, 2008
  - RFC8446 : TLS version 1.3, 2018

** Netscape Navigator : navigateur ayant dominé le marché au milieu des années 1990*

*** Kerberos : protocole d'authentification : tickets au lieu de mots de passe*

Architecture

- SSL/TLS
  - Protocole généraliste chargé de sécuriser les échanges basés sur TCP
  - Transparent pour le protocole de niveau supérieur
- DTLS (RFC 4347)
  - Extension de TLS, appelée Datagram TLS, chargée de sécuriser les échanges basés sur UDP



Ports au dessus de SSL

- Ports accordés par *IANA** aux applications utilisant SSL

Protocole sécurisé	Port	Protocole non sécurisé	Application
HTTPS	443	HTTP	Transactions requêtes-réponses sécurisées
SSMTP	465	SMTP	Messagerie électronique
SSL-LDAP	636	LDAP	Annuaire LDAP
FTPS	990	FTP	Contrôle du transfert de fichiers
TELNETS	992	TELNET	Protocole d'accès à distance

** IANA : Internet Assigned Numbers Authority*

Services de SSL

- SSL/TLS fonctionne suivant un mode client-serveur, et permet d'atteindre les objectifs de sécurité « CIA »
  - Authenticité du serveur : vérifie que le certificat et l'identité publique fournis par le serveur sont valides
  - Confidentialité des données échangées
  - Intégrité des données échangées : garantit que les données reçues n'ont pas été altérées, frauduleusement ou accidentellement

Authentification


Confidentialité

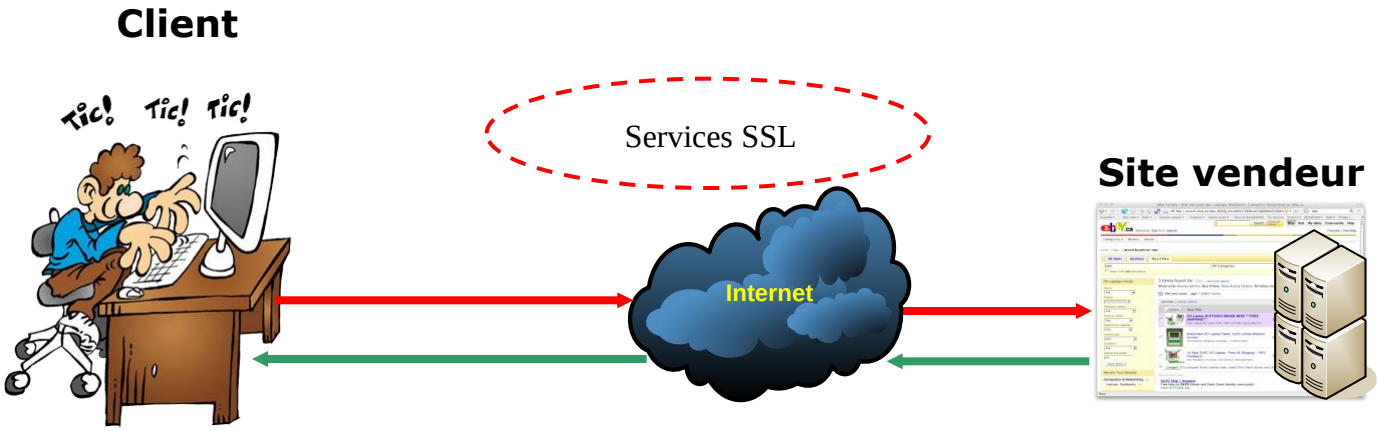
Intégrité

Comment TLS assure-t-il ces services ?

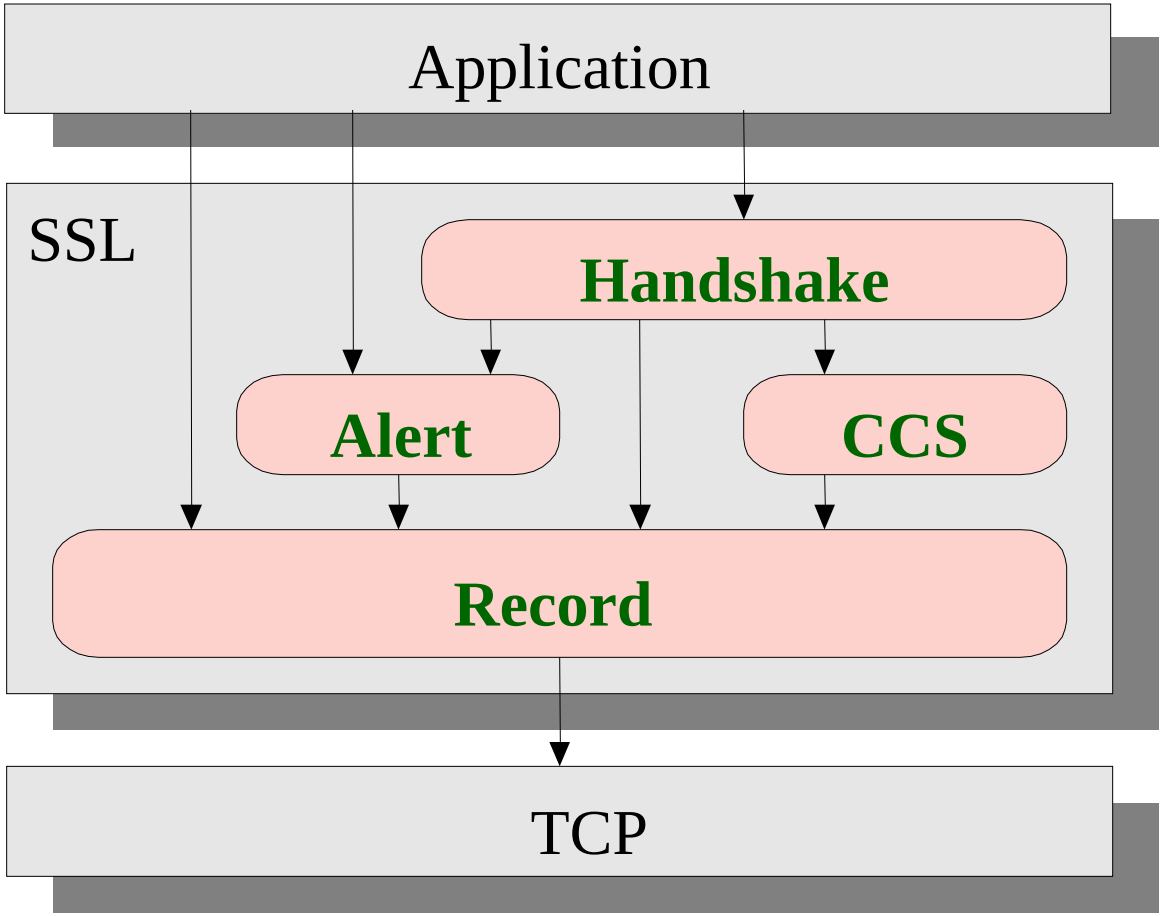
- Pour assurer la confidentialité
  - Chiffrement /déchiffrement symétriques
  - Algorithmes : DES, 3DES, RC2, RC4, IDEA, AES
- Pour assurer l'intégrité
  - Signature numérique : fonction de hachage
  - Algorithmes : MD5, SHA-1, HMAC
- Pour assurer l'authentification du serveur
  - Certificats numériques
  - Infrastructures à clé publique afin de créer une chaîne de confiance
  - Chiffrement/déchiffrement asymétrique RSA

Etapes d'une transaction SSL/TLS

- Etape 1 :
  - Etablissement des paramètres de sécurité
  - Authentification du serveur et échange des clés
- Etape 2
  - Validation de l'échange de clés
- Etape 3
  - Chiffrement, fragmentation, compression…
- Etape4
  - Gestion des messages d'erreur



Sous-protocoles TLS



- SSL/TLS est composé de 4 sous-protocoles :
  - SSL Handshake Protocol : établissement de la session
  - SSL Record Protocol : chiffrement, hachage de données
  - SSL Change Cipher Spec Protocol : mise en œuvre des clés re-négociées
  - SSL Alert Protocol : messages d'erreur

Sous-protocoles TLS

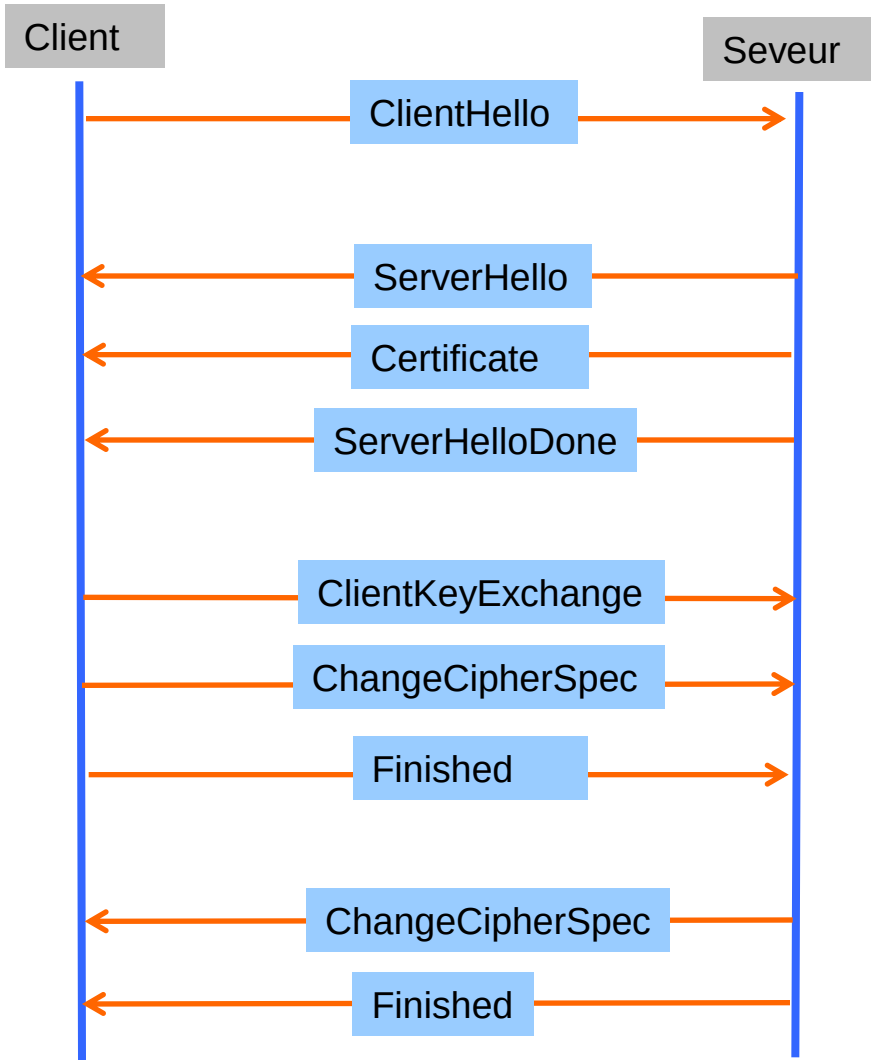
- SSL Handshake Protocol
  - négociation des paramètres et des algorithmes de sécurité
  - échanges de clés
  - authentification du serveur
- SSL Record Protocol
  - fragmentation
  - compression
  - intégrité de messages
  - chiffrement
- SSL Alert Protocol
  - messages d'erreurs (*fatal alerts and warnings*)
- SSL Change Cipher Spec Protocol
  - protocole d'un seul octet
  - indique la fin de la phase *SSL handshake*
  - active, pour la session courante, les algorithmes et les clés négociés dans *SSL handshake*

1- SSL Handshake Protocol

- Permet au client et au serveur de
  - choisir les suites de chiffrement
    - les algorithmes de chiffrement
    - les algorithmes de hachage
    - les clés symétriques qui vont servir au chiffrement
  - s'authentifier mutuellement
- Master secret : clé de 48 octets partagée entre le client et le serveur. Les autres clés sont générées à partir de ce paramètre
- RFC 2246 (*TLS protocol*) :

"When a TLS client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets"

1- SSL Handshake Protocol



- *ClientHello* : ce message contient la version de SSL, un nombre aléatoire permettant de générer les clés secrètes, et l'ensemble d'algorithmes proposés : DH, RSA, 3DES
- *Server Hello* : dans ce message le serveur choisit les algorithmes proposés par le client et choisit la longueur de clés. Ainsi, il génère un nombre aléatoire permettant de générer les clés secrètes
- *Certificate* : le navigateur du client vérifie immédiatement la validité du certificat
- *ServerHelloDone : Ok => à ce moment le client peut vérifier le certificat du serveur et échanger les clés*
- *ClientKeyExchange :* le client génère la clé pre-master et le chiffre par la clé publique du serveur. Enfin, il envoie le message chiffré au serveur
- *ChangeCipherSpec* : le client informe le serveur que tous les messages suivants vont être chiffrés par la clé symétrique échangée dans le message d'avant
- *Finished* : le client envoie un message chiffré par la nouvelle clé pour vérification
- Le serveur fait la même procédure pour vérifier la bonne utilisation de la clé

Exemple : requête *ClientHello*

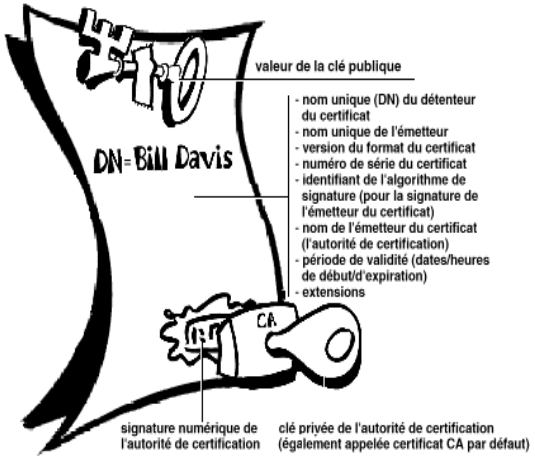
Exemple : réponse *ServerHello*

```
741 17.003003   62.161.94.179          10.10.1.37              TLSv1    Server Hello, Certificate, Server Hello Done
⊞ Frame 741 (1058 bytes on wire, 1058 bytes captured)
⊞ Ethernet II, Src: Cisco_d2:49:3f (00:1f:6c:d2:49:3f), Dst: Dell_2b:76:54 (00:1e:c9:2b:76:54)
⊞ Internet Protocol, Src: 62.161.94.179 (62.161.94.179), Dst: 10.10.1.37 (10.10.1.37)
⊞ Transmission Control Protocol, Src Port: https (443), Dst Port: 55538 (55538), Seq: 1, Ack: 164, Len: 1004
⊟ Secure Socket Layer
  ⊟ TLSv1 Record Layer: Handshake Protocol: Multiple Handshake Messages
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 999
    ⊟ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 70
        Version: TLS 1.0 (0x0301)
      ⊞ Random
        Session ID Length: 32
        Session ID: 08010000EE91A59A714BD60A7F42FCA1FFE867C1207CCFE9...
        Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
        Compression Method: null (0)
    ⊟ Handshake Protocol: Certificate
        Handshake Type: Certificate (11)
        Length: 917
        Certificates Length: 914
      ⊞ Certificates (914 bytes)
    ⊟ Handshake Protocol: Server Hello Done
        Handshake Type: Server Hello Done (14)
        Length: 0
```

Certificat X.509

- Pourquoi utilise-ton un certificat ?
  - Parce que rien ne garantit que la clé publique est bien celle du serveur a qui elle est associée

- Qu'est-ce qu'un certificat ?
  - Normalisé par le standard X.509 de l'UIT* (X.509v3)
  - Petit fichier composé de deux parties :
    - Informations concernant le serveur
    - Signature d'une autorité sur ces informations



- Structure d'un certificat
  - La version du certificat
  - Le numéro de série du certificat
  - L'algorithme de chiffrement utilisé pour signer le certificat
  - Le nom (DN, pour Distinguished Name) de l'autorité de certification émettrice
  - La date de début de validité du certificat
  - La date de fin de validité du certificat
  - L'objet de l'utilisation de la clé publique
  - La clé publique du propriétaire du certificat
  - Signature de données par la clé privée de l'émetteur du certificat (CA)

* UIT : Union internationale des télécommunications

Certificat X.509

Numéro de série du certificat

Spécificités techniques du certificat : format, algorithmes utilisés pour la signature

Dates de validité du certificat

Objet d'utilisation

Nom, adresse, e-mail…du propriétaire du certificat

Clé publique du propriétaire

Nom du tiers de confiance, identifiant du tiers de confiance, adresse,…

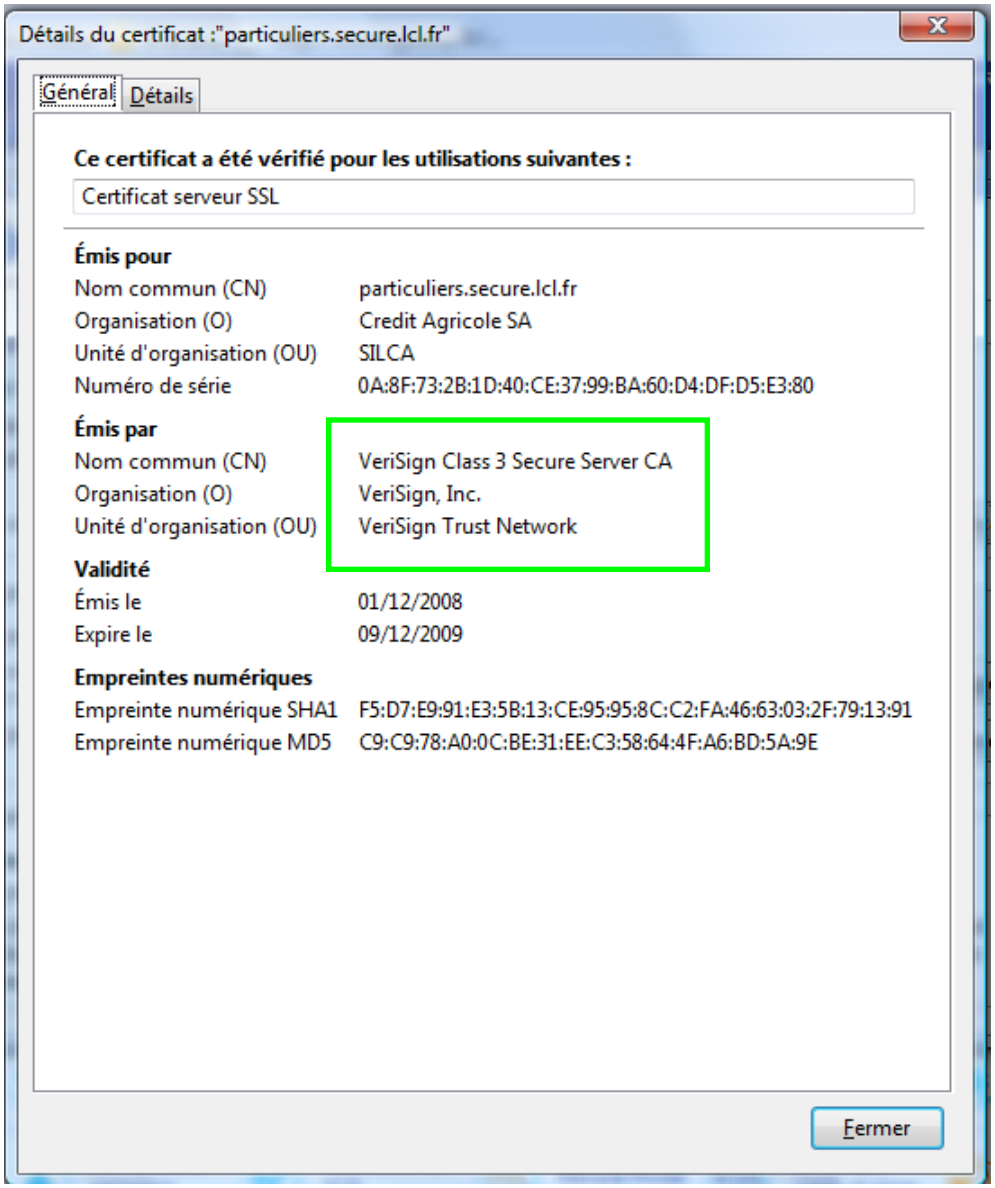
hachage

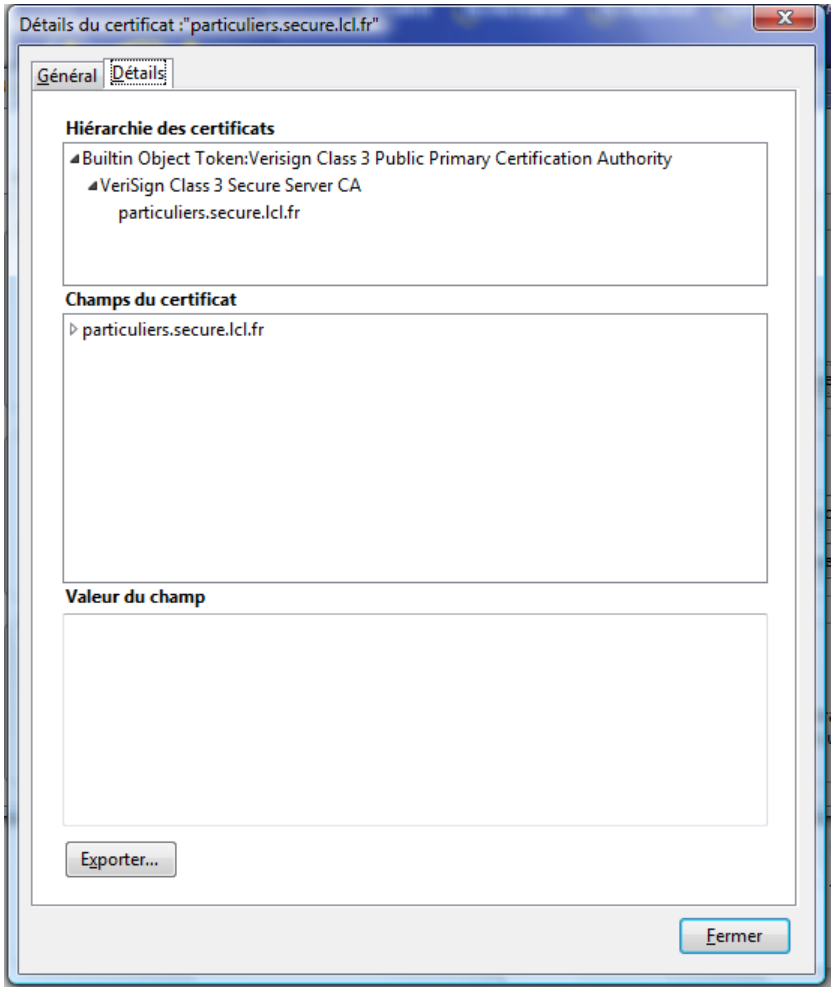
Clé privée du tiers de confiance

Signature du tiers de confiance

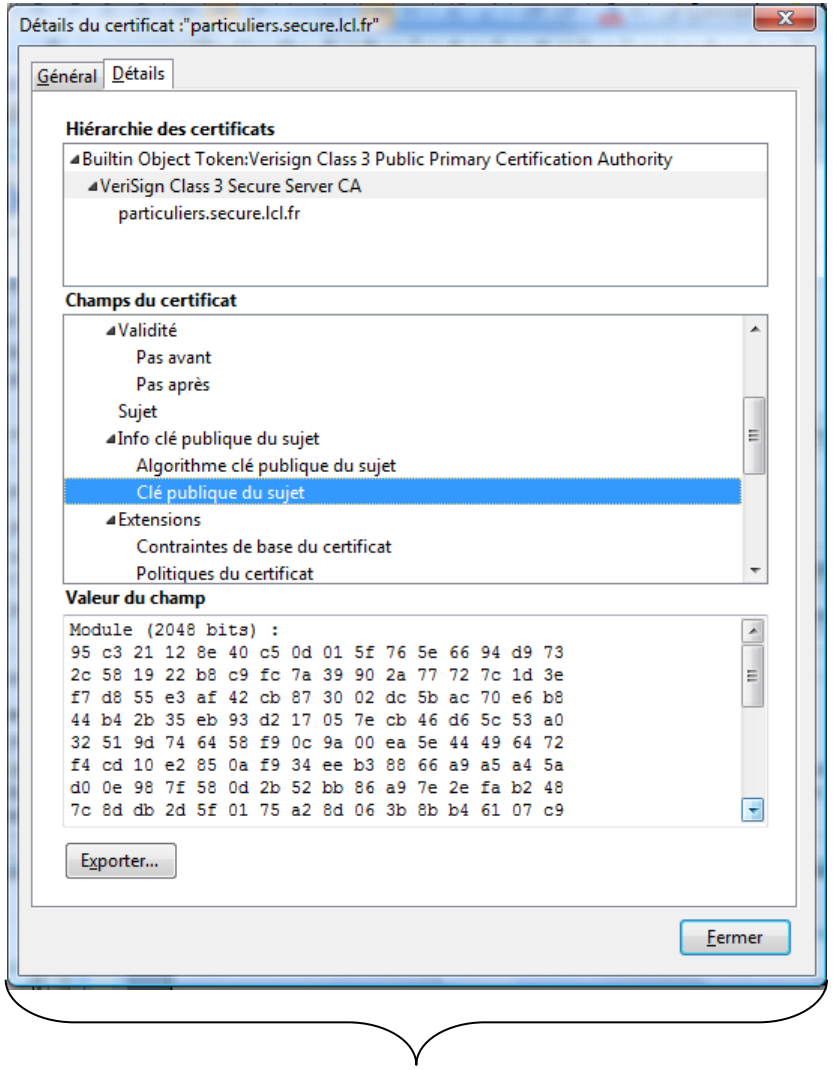
Certificat= $Nom_{Serveur}$ + Nom_{AC} + $K_{P\text{-}SERV}$ + $K_{S\text{-}AC}$ {$Nom_{Serveur}$ + Nom_{AC} + $K_{P\text{-}SERV}$ }

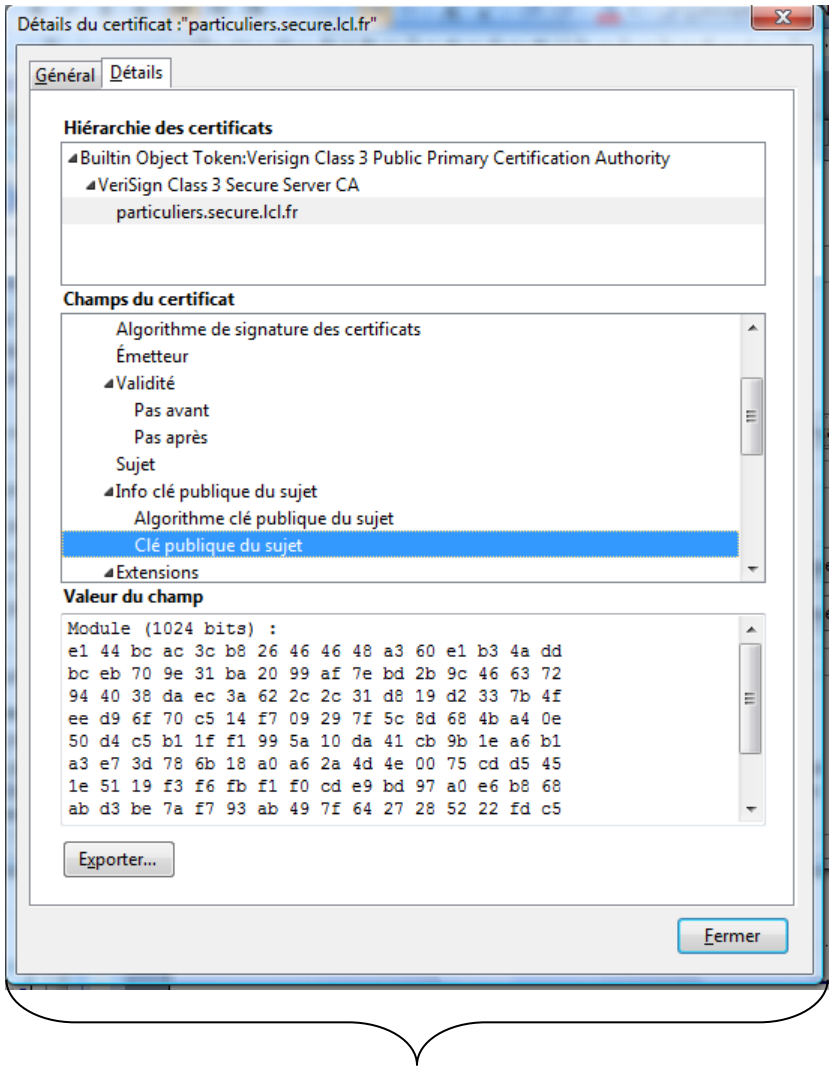
Exemple : certificat X.509





Exemple : certificat X.509

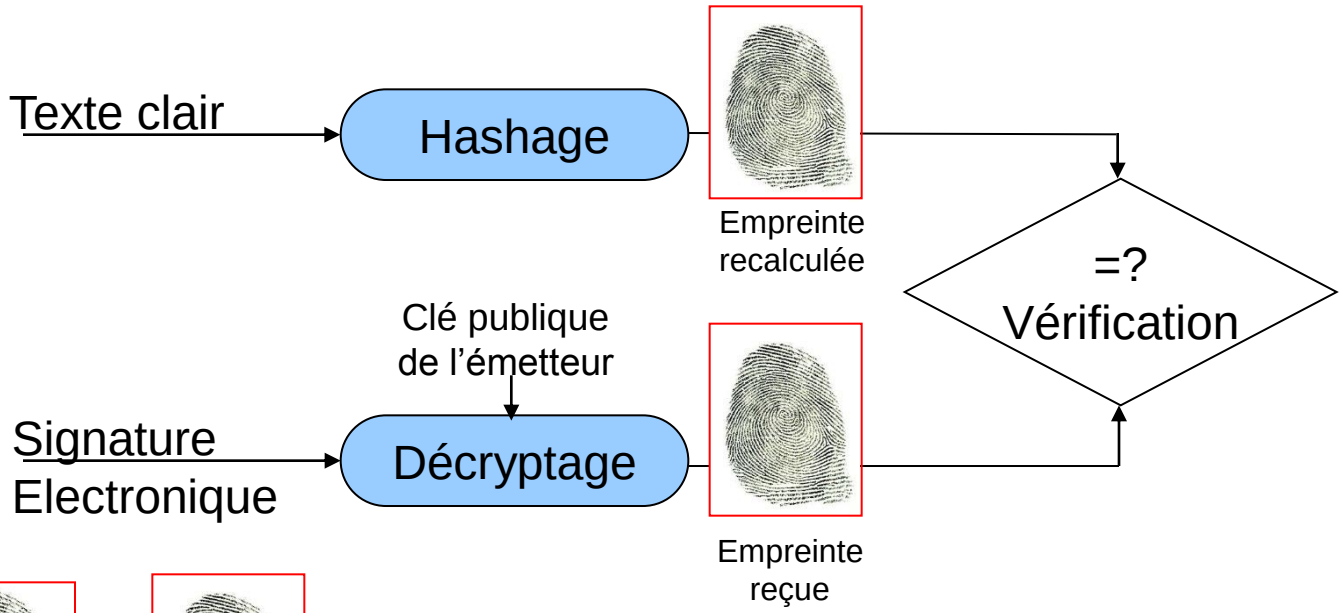




Clé publique de l'autorité du certificat (CA)

Clé publique du serveur

Authentification du serveur



1)  = 

Empreinte reçue — Empreinte recalculée

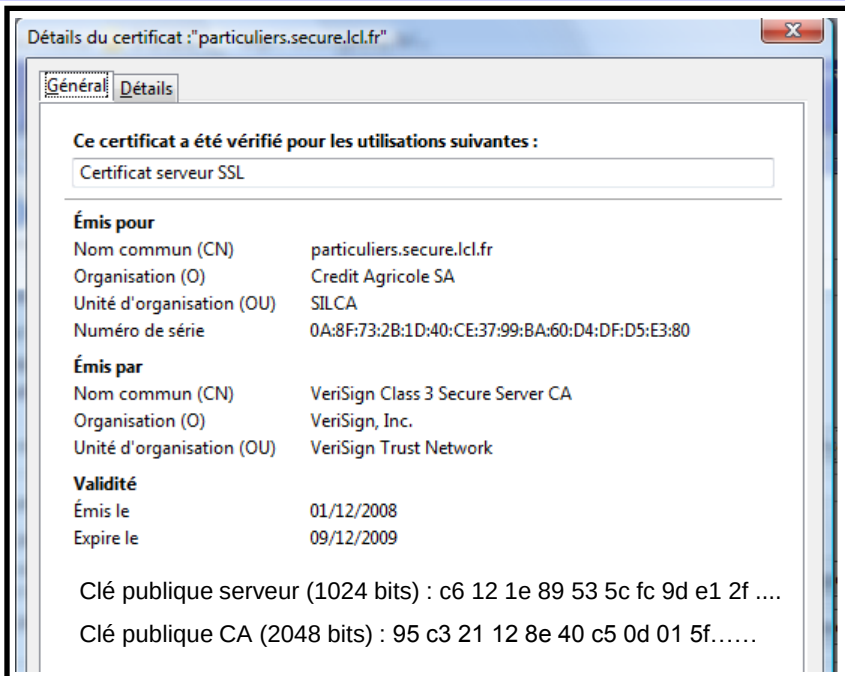
La signature reçue est correcte

2)  ≠ 

Empreinte reçue — Empreinte recalculée

La signature reçue est incorrecte

Authentification du serveur



Hash : SHA-1

C (SHA1(Certificat))

$K_{S\text{-}CA}$

Empreinte numérique chiffrée par la clé privée de CA

Hash : SHA-1

Comparaison

Validation ?

Authentification du serveur

- Signature électronique du certificat
  - Est faite par la clé privée de l'autorité de certification (CA)
  - Attestation forte de l'authenticité du message car personne ne peut avoir la clé privée de la CA
- Vérification du certificat
  - Peut être effectuée par tout service qui possède la clé publique de l'autorité de certification
  - Déchiffrement / vérification par la clé publique

Révocation d'un certificat

- Certificat révoqué
  - Annulé
  - Plus valable
  - Plus digne de confiance

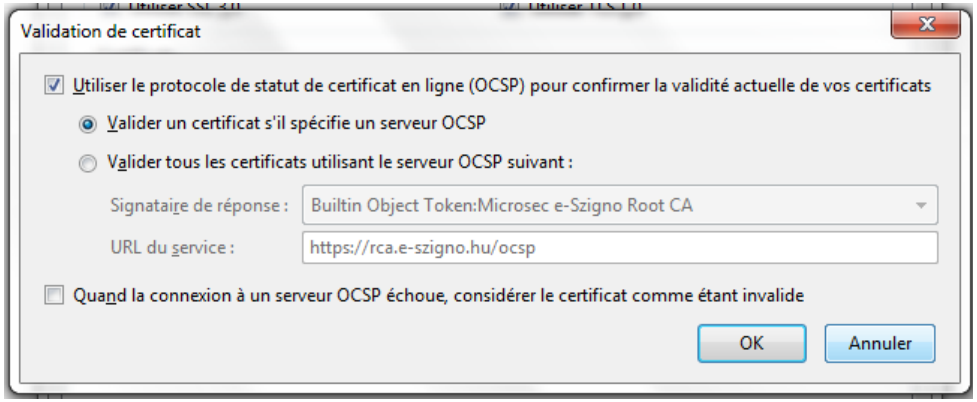
- Un certificat peut devenir invalide pour de nombreuses raisons
  - Clé privée du serveur est compromise/perdue
  - Clé privée de l'AC est compromise/perdue
  - Changement des paramètres (CN, sujet, …)
  - Date d'expiration

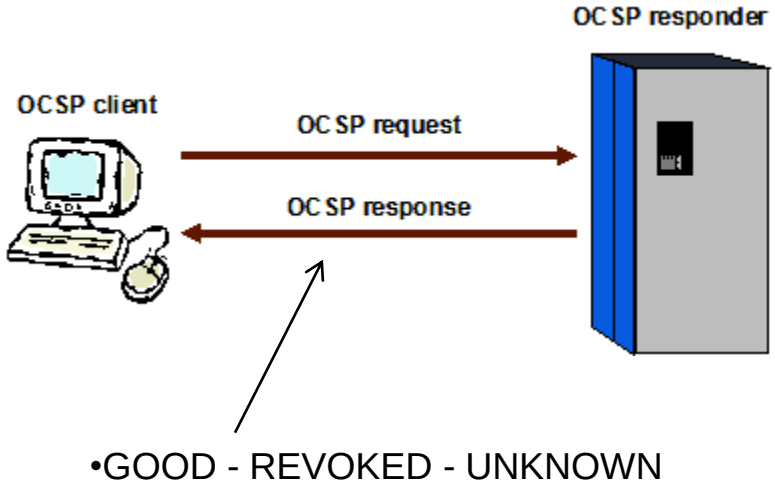
Révocation d’un certificat

- Listes de révocation de certificats (*CRL, certificate revocation lists*)
  - Listes complètes signées numériquement composées de certificats non expirés qui ont été révoqués
  - Liste est récupérée par les clients qui peuvent alors la mettre en cache et l'utiliser pour vérifier les certificats présentés à l'utilisation
- Bonne infrastructure de clé publique (PKI)
- *OCSP : Online Certificate Status Protocol* protocole de vérification en ligne de certificat

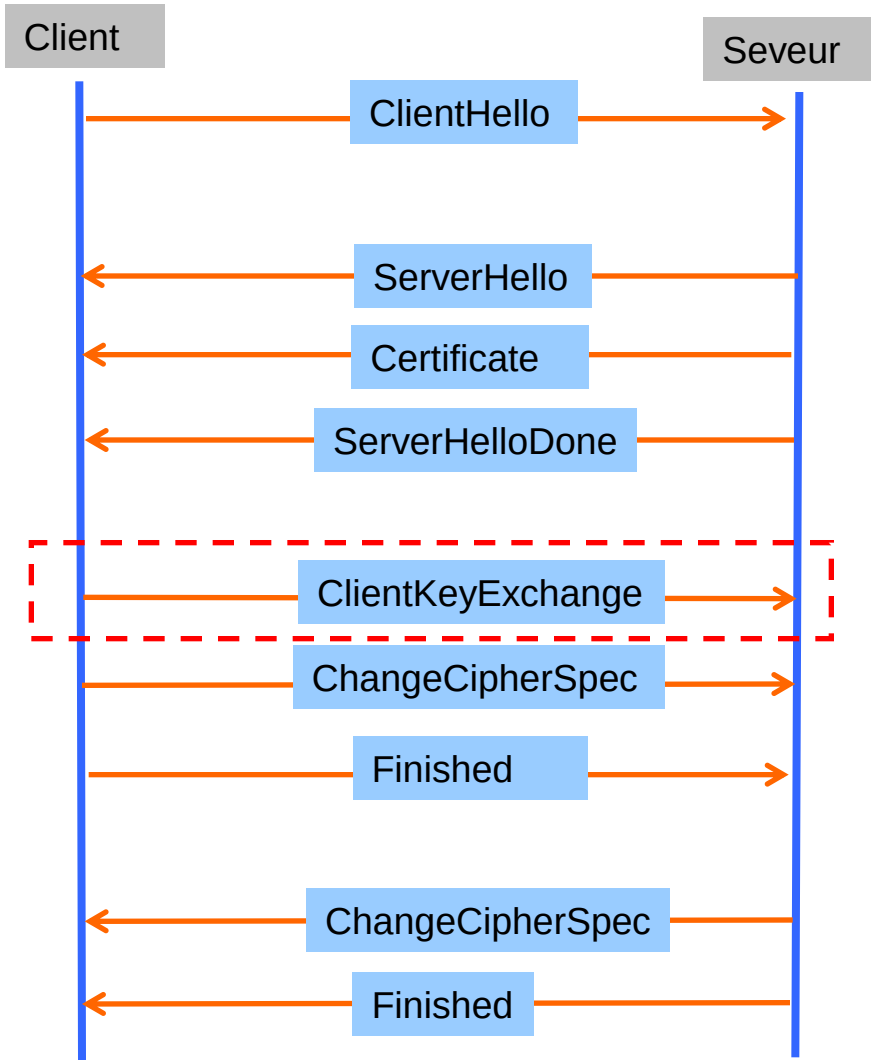
Révocation d'un certificat

- *OCSP : Online Certificate Status Protocol*
  - *RFC 2560*
  - *Utilisé pour valider un certificat X.509*





Calcul des paramètres de sécurité



- *ClientHello* : ce message contient la version de SSL, un nombre aléatoire permettant de générer les clés secrètes, et l'ensemble d'algorithmes proposés : DH, RSA, 3DES
- *Server Hello* : dans ce message le serveur choisit les algorithmes proposés par le client et choisit la longueur de clés. Ainisi, il génère un nombre aléatoire permettant de générer les clés secrètes
- *Certificate* : le navigateur du client vérifie immédiatement la validité du certificat
- *ServerHelloDone : Ok => à ce moment le client peut vérifier le certificat du serveur et échanger les clés*
- *ClientKeyExchange :* le client génère la clé pre-master et le chiffre par la clé publique du serveur. Enfin, il envoie le message chiffré au serveur
- *ChangeCipherSpec* : le client informe le serveur que tous les messages suivants vont être chiffrés par la clé symétrique (échangée dans le message d'avant)
- *Finished* : le client envoie un message chiffré par la nouvelle clé pour vérification
- Le serveur fait la même procédure pour vérifier la bonne utilisation de la clé

Calcul des paramètres de sécurité

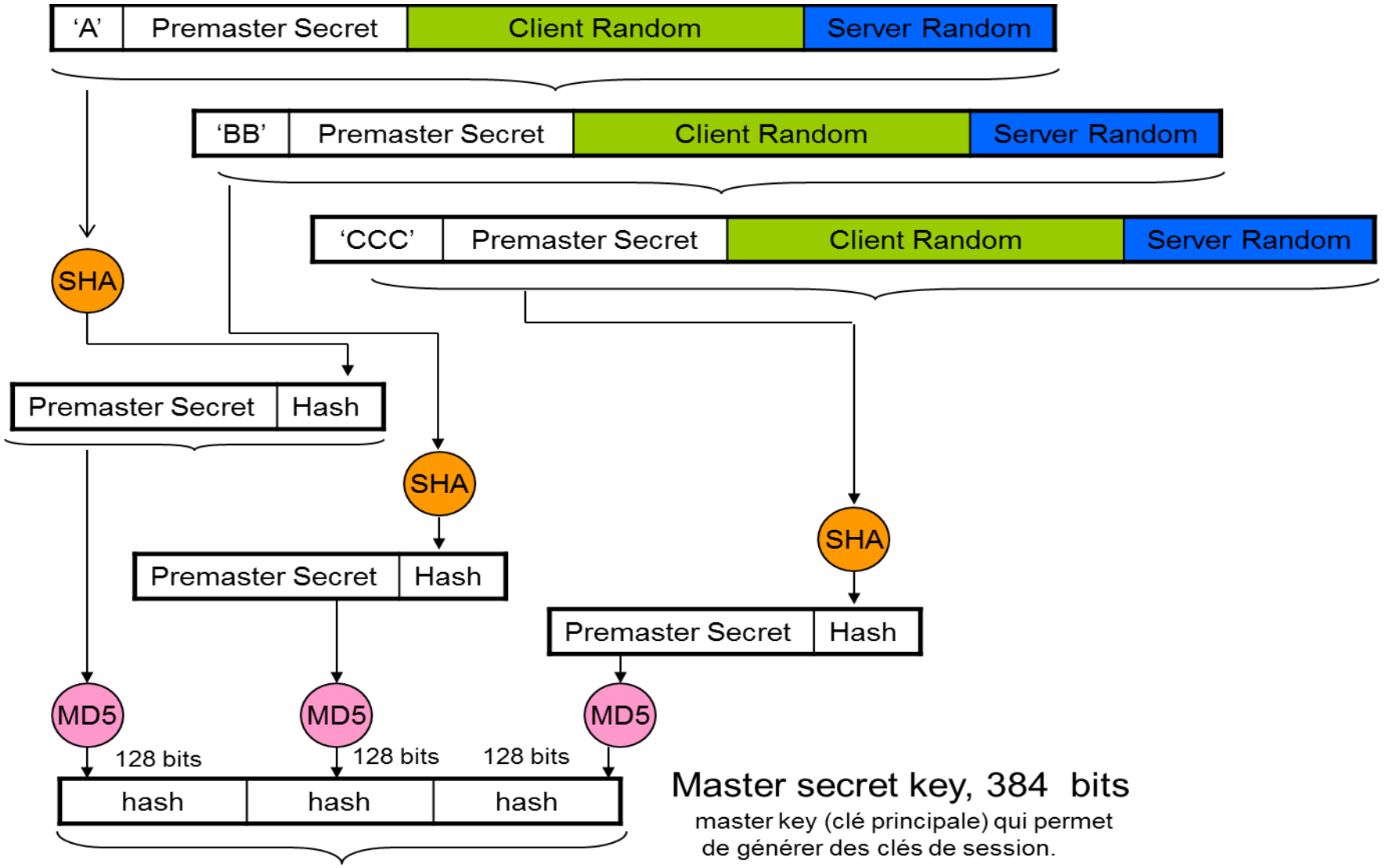
- Construction du *Master secret* à l'ouverture d'une session
  - Calculé par le client et le serveur
  - master_secret =

    MD5(pre_master_secret || SHA('A' || pre_master_secret || ClientHello.random || ServerHello.random))||
    MD5(pre_master_secret || SHA('BB' || pre_master_secret || ClientHello.random ||ServerHello.random)) ||
    MD5(pre_master_secret || SHA('CCC' || pre_master_secret || ClientHello.random || ServerHello.random))

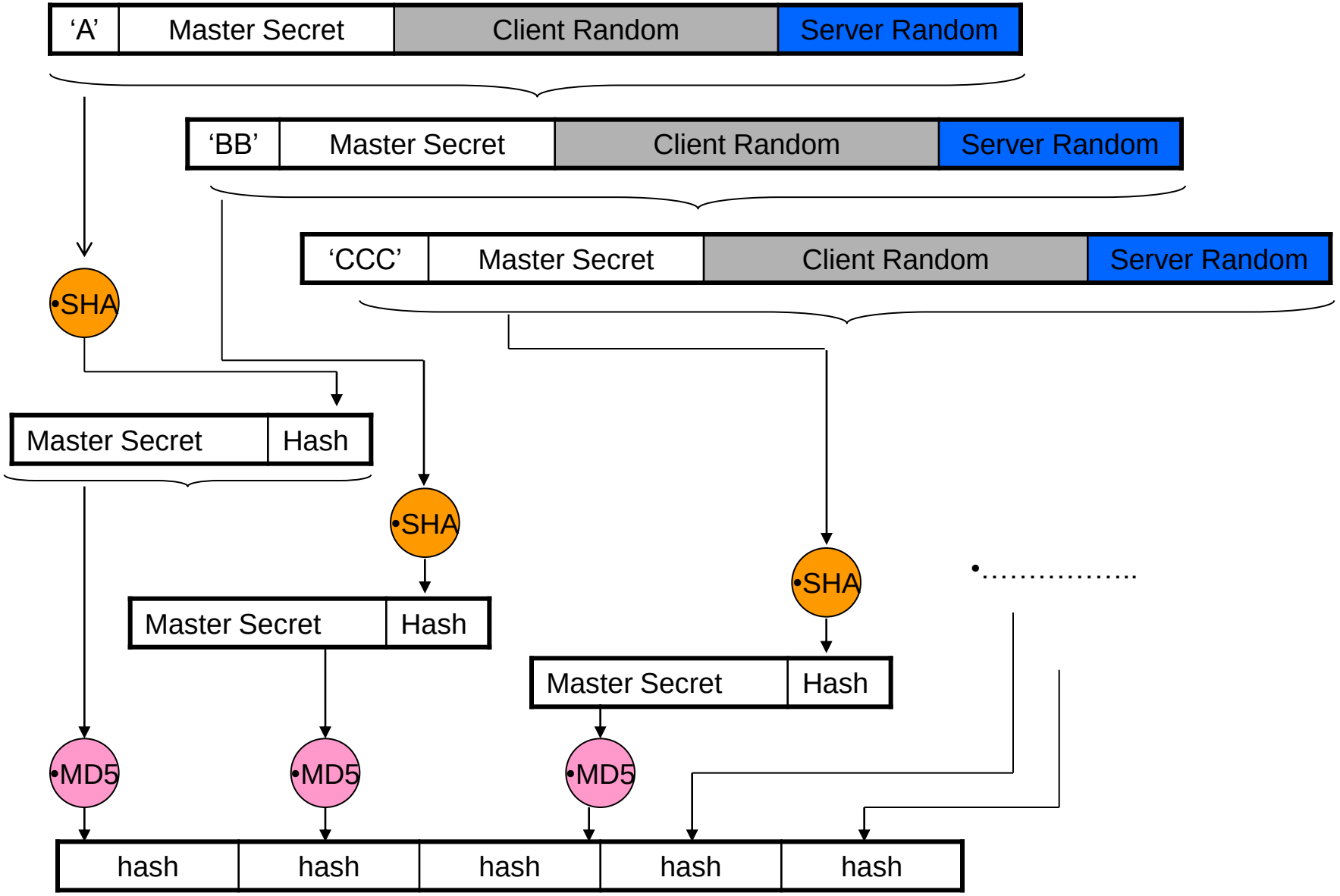
- Génération de secrets à l'ouverture d'une session ou connexion
  - key_block =

    MD5(master_secret || SHA('A' || master_secret || ServerHello.random || ClientHello.random))||
    MD5(master_secret || SHA('BB' || master_secret || ServerHello.random || ClientHello.random))||
    MD5(master_secret || SHA('CCC' || master_secret || ServerHello.random || ClientHello.random))||
  - Key_block= 2 clés MAC + 2 clés chiffrement

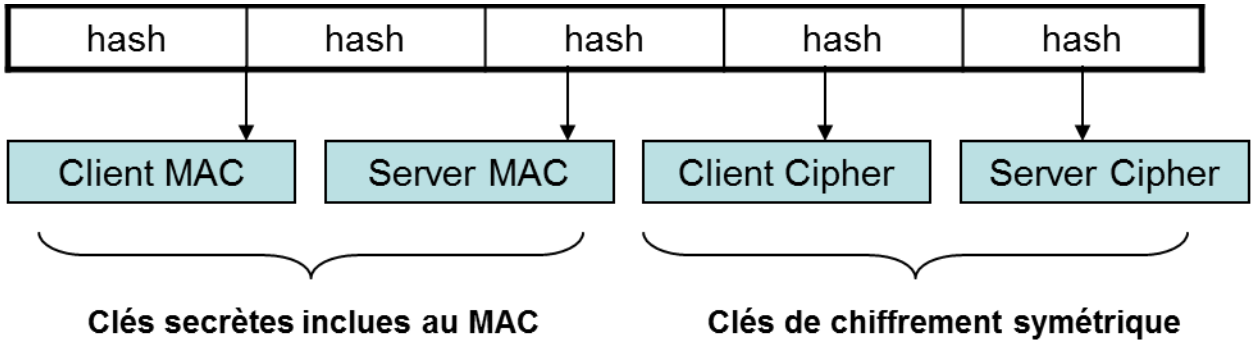
Construction du *master_secret*



Génération des secrets



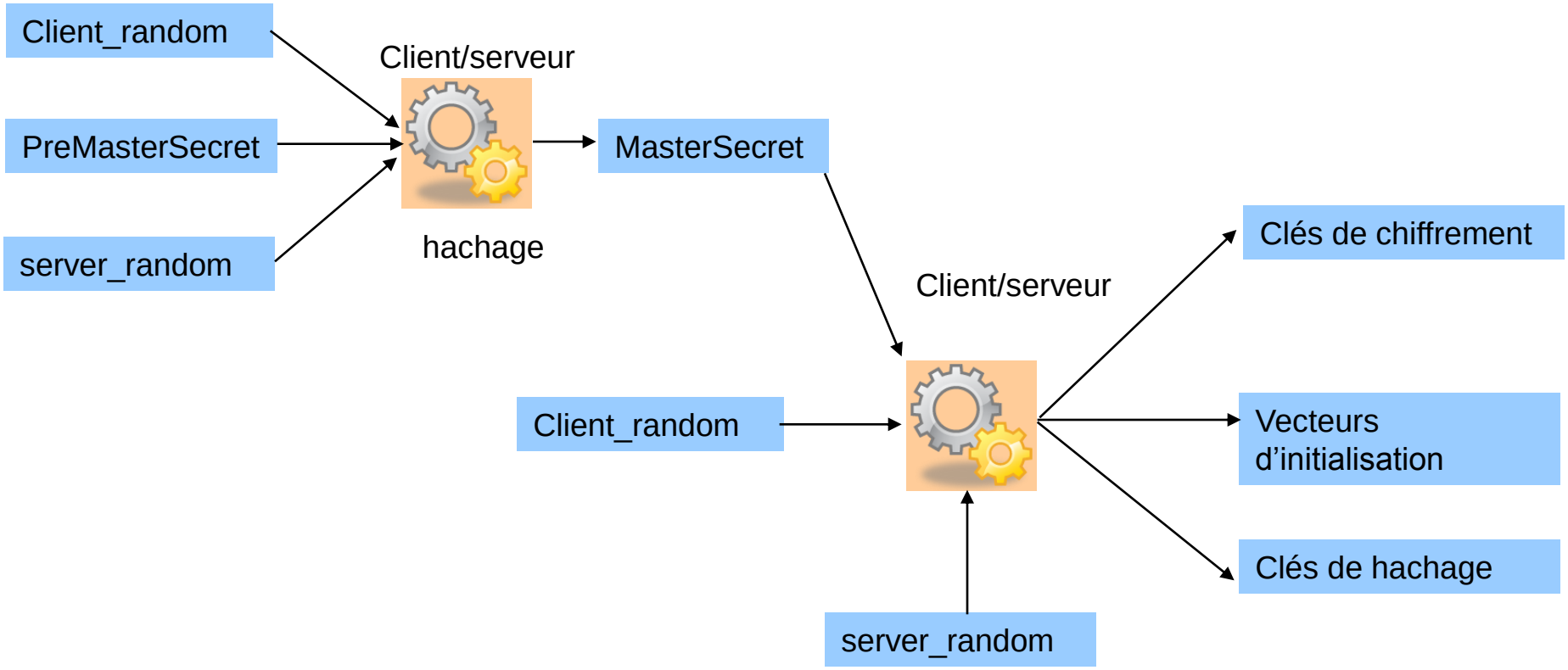
Utilisation des secrets



hash(MAC_write_secret || pad_2 ||
hash(MAC_write_secret || pad_1 || seq_num || SSLCompressed.type ||
SSLCompressed.length || SSLCompressed.fragment))

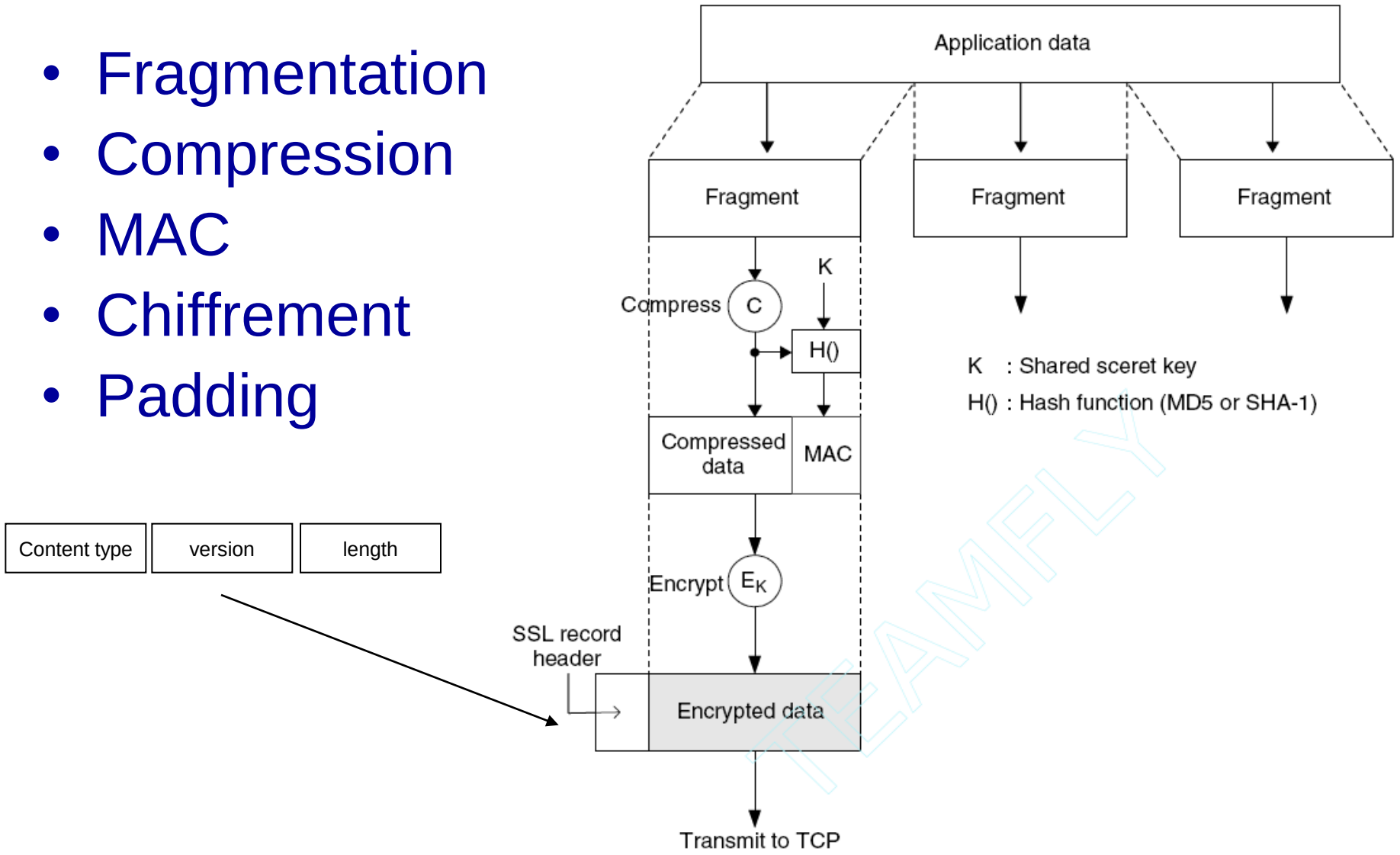
MAC_write_secret = shared secret key
hash = cryptographic hash algorithm; either MD5 or SHA-1
pad_1 = the byte 0x36(0011 0110) repeated 48*(384 bits) for MD5 and 40*(320 bits) for SHA-1
pad_2 = the byte 0x5C (0101 1100) repeated 48*for MD5 and 40*for SHA-1
seq_num = the sequence number for this message
SSLCompressed.type = the higher-level protocol used to process this fragment
SSLCompressed.length = the length of the compressed fragment
SSLCompressed.fragment = the compressed fragment (if compression is not used, the plaintext fragment)

Clés secrètes d'une session SSL



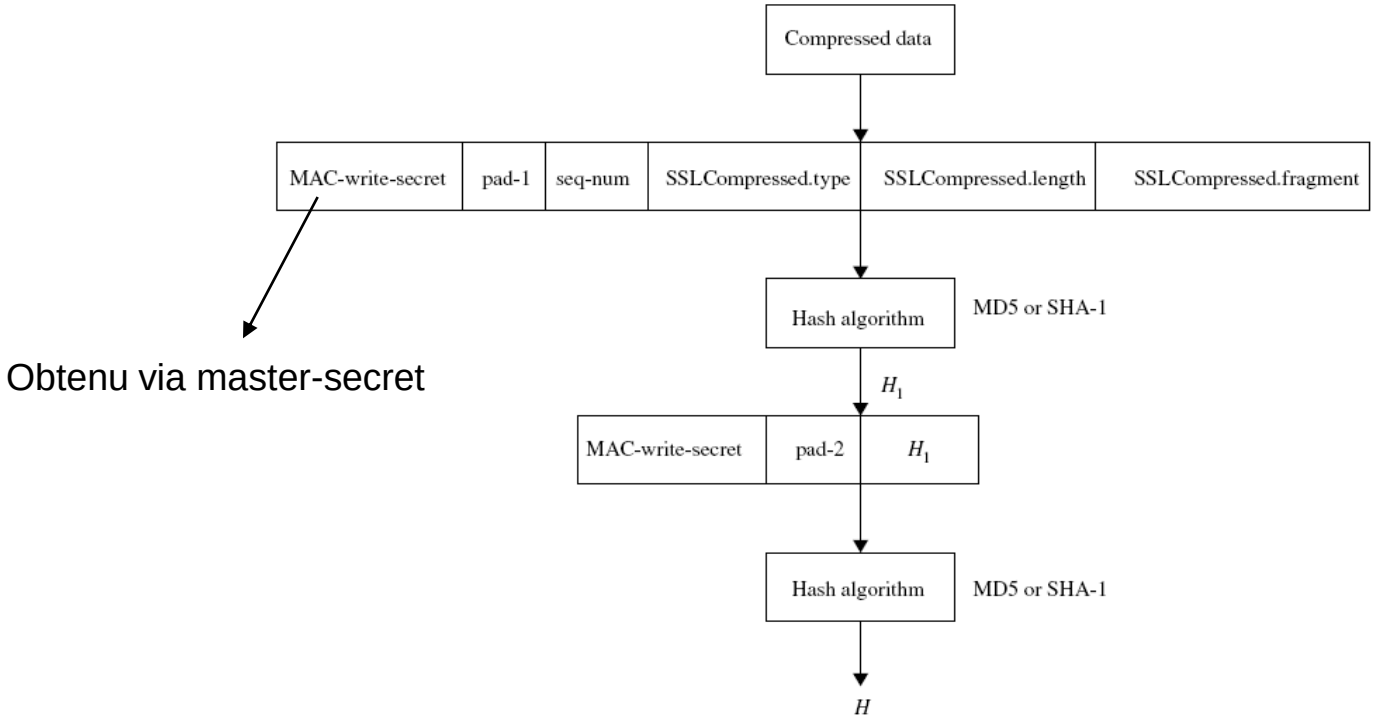
2- SSL record Protocol

- Fragmentation
- Compression
- MAC
- Chiffrement
- Padding



2- SSL record Protocol

- Pad-1 : 0x36
  - répété 48 fois si MD5 est utilisé => 384 bits
  - répété 40 fois si SHA1 est utilisé => 320 bits
- H : MD5 ou SHA1
- SSLCompressed.type : le protocole utilisant ces données
- SSLCompressed.legnth : taille du fragment compressé



3- SSL alert protocol

- spécifie les messages d'erreur envoyés par le client et le serveur
- alertes composées de deux octets,
  - Octet 1 : "warning"
  - Octet 2 : "fatal"
- Erreurs fatales
  - unexpected_message : message non reconnu
  - bad_record_mac : MAC non correct
  - decompression_failure : mauvaise entrée pour la fonction décompression
  - handshake_failure : impossible de négocier les bons paramètres
  - illegal_parameter
- Erreurs « warning »
  - close_notify : fin de la connexion
  - no_certificate : réponse à une demande de certificat s'il n'y en a pas
  - bad_certificate : signature non valide
  - unsupported_certificate : le certificat reçu n'est pas reconnu
  - certificate_revoked
  - certificate_expired
  - certificate_unknown

4- SSL Change Cipher Spec Protocol

- Protocole en un seul octet
- Indique la fin de la phase SSL handshake
- Active pour la session courante
  - les algorithmes
  - les clés négociées dans SSL handshake
- Change le *pending state* en *current state*

Faiblesses de SSL/TLS

- Incapable d'identifier formellement l'acheteur
- Taille de clefs
- Faiblesses dans certaines versions : SSLv3
  - Accepte *Finished* avant *ChangeCipherSpec*
  - Envoi de données chiffrées avant réponse serveur au *Finished*
- Attaques classiques
  - Attaques par force brute
    - Moins sécurisé : clés de 40 bits
    - Plus sécurisé : clés de 128 bits ?
  - Attaques par le milieu : l'attaquant intercepte la requête du client et se fait passer pour le serveur auprès de lui, tout en se faisant passer pour un client auprès du serveur légitime
- Attaques basées sur des propriétés cryptographiques
  - Attaques de type *rollback*
    - l'attaquant cherche à modifier le choix des algorithmes d'échanges de clés de façon à ce que les 2 entités n'utilisent pas les mêmes (RSA et DH par exemple).

Faiblesses de SSL/TLS

- TLS : une succession d'attaques récentes
- Nouvelles vulnérabilités
  1) Attaque de redirection
  2) Mauvais traitement des certificats par les navigateurs
  3) Attaque de renégociation

Faiblesses de SSL/TLS : 3

- Tester les serveurs vulnérables
  - Méthode manuelle
    - Client openssl
    - openssl s_client -connect <serveur>:443

Serveur vulnérable :

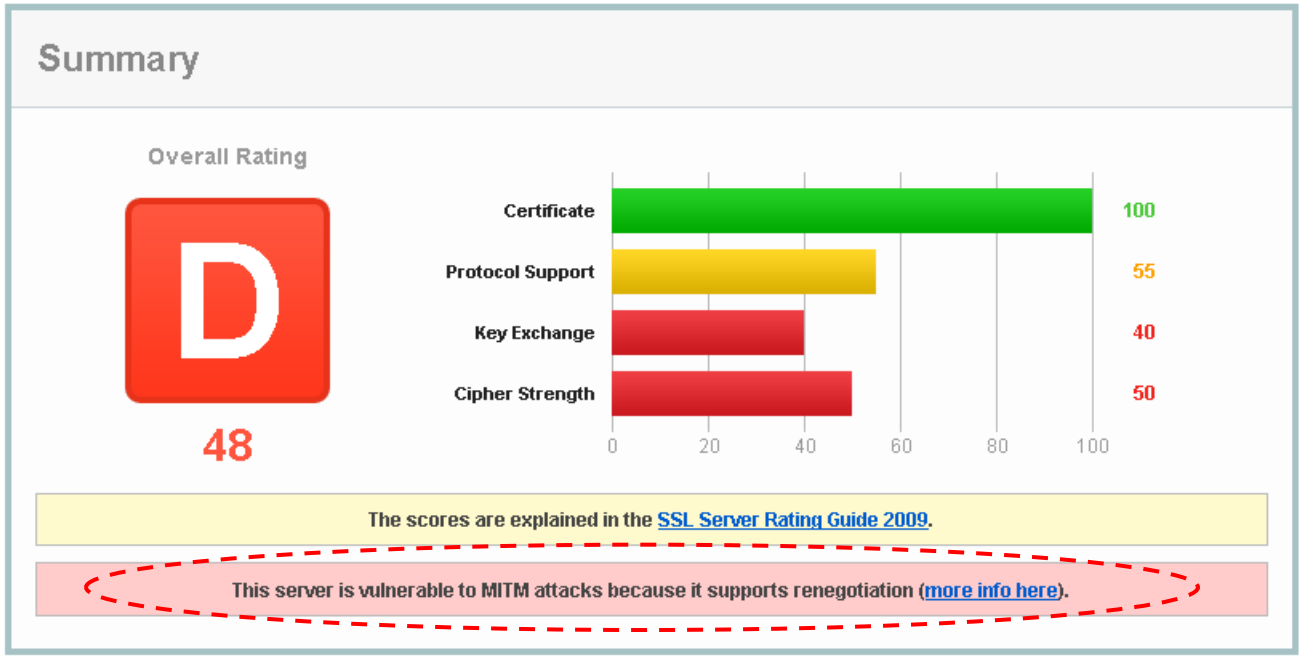
```
RENEGOTIATING
depth=1    /C=US/O=VeriSign,    Inc./
OU=VeriSign Trust Network/OU=Terms of
use  at  https://www.verisign.com/rpa
(c)05/CN=VeriSign  Class  3  Secure
Server CA
verify  error:num=20:unable  to  get
local issuer certificate
verify return:0

HTTP/1.1 302 Found
Server: Apache
Location: https://www.xxx.fr
...
```

Serveur non vulnérable :

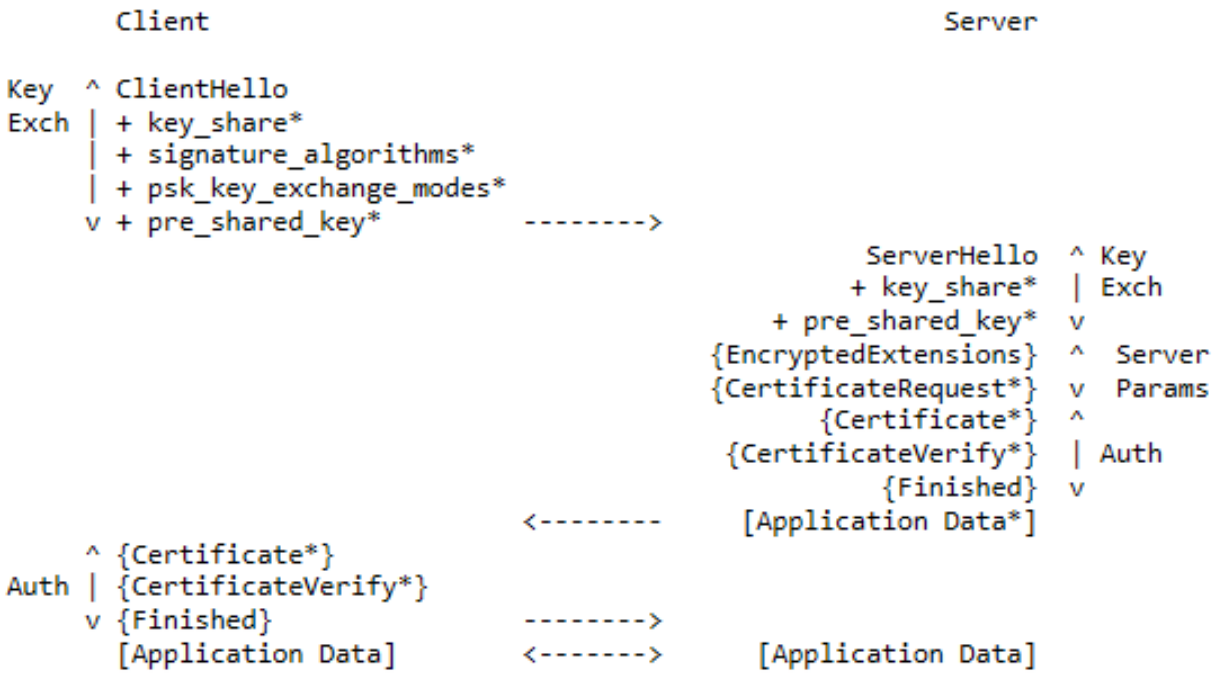
```
RENEGOTIATING
2201:error:1409E0E5:SSL
routines:SSL3_WRITE_BYTES:ssl
handshake failure:s3_pkt.c:530:
```

Faiblesses de SSL/TLS : 3

- Tester les serveurs vulnérables
  - Méthode automatique
    - depuis le site www.ssllabs.com
    - test très intéressant



Passage vers TLS 1.3/ RFC8446



\+ Indicates noteworthy extensions sent in the previously noted message.

* Indicates optional or situation-dependent messages/extensions that are not always sent.

{} Indicates messages protected using keys derived from a [sender]_handshake_traffic_secret.

[] Indicates messages protected using keys derived from [sender]_application_traffic_secret_N

Figure 1: Message flow for full TLS Handshake