

Le protocole SSL (TLS)



Haute Ecole de la Province de Liège



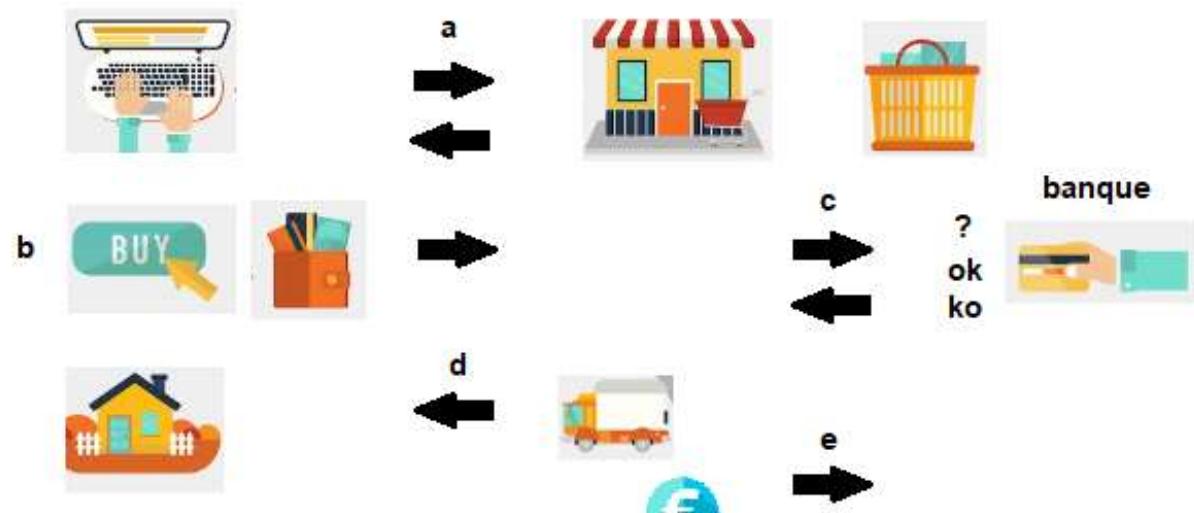
I. INTRODUCTION

La force d'un algorithme n'est pas dans son secret. Tout le monde peut savoir comment il fonctionne.

1. Contexte

■ Le commerce en ligne

- a. le client remplit son caddie virtuel (browser);
- b. Caddy ok => il paie. Il choisit son mode de paiement, l'institution financière utilisée diverses informations demandées par le marchand. Le tout est envoyé vers ce dernier.
- c. le marchand demande l'autorisation de paiement auprès de l'institution financière spécifiée par le client
- d. si ok, le marchand confirme la transaction au client et les articles sont envoyés
- e. le marchand demande le paiement effectif à l'institution financière du client.



1. contexte



Il existe bien entendu beaucoup d'autres exemples:

Pc Banking

Echange avec des sites fédéraux,

...



Tous les échanges passent par le réseau => risques

Le client doit être assuré qu'il échange bien avec le vrai marchand (serveur, site,...)

le marchand (responsable du site) pourrait utiliser les informations de paiement à d'autres esciens

les informations de paiement pourraient être interceptées par des personnes mal intentionnées.

...



et 3D secure, c'est quoi ?

2. Objectifs



AUTHENTIFIER

- Les clients: le serveur doit être sûr que les clients sont bien ceux qu'ils prétendent être
- Les serveurs: le client doit être sûr que les serveurs sont également bien ceux qu'ils prétendent être



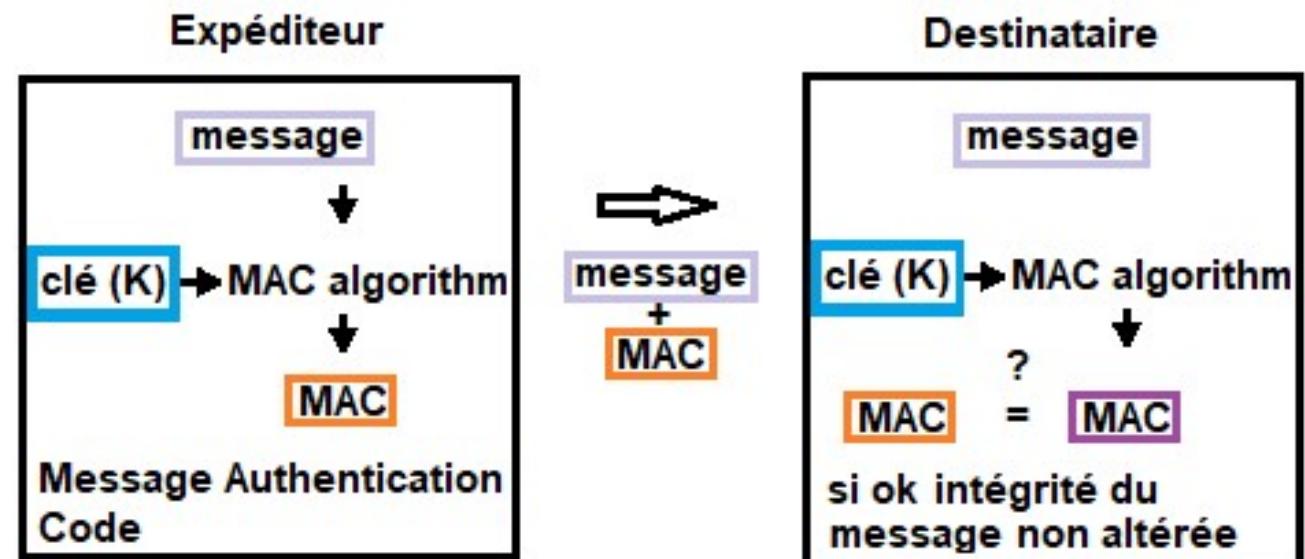
CHIFFRER/DÉCHIFFRER

- DES / Triple DES: chiffrement symétrique (découpage en blocs). Peut être utilisé également pour signer (HMAC)
- RSA: chiffrement asymétrique (paire de clés publique / privée + certificats). Utilisé pour la signature.

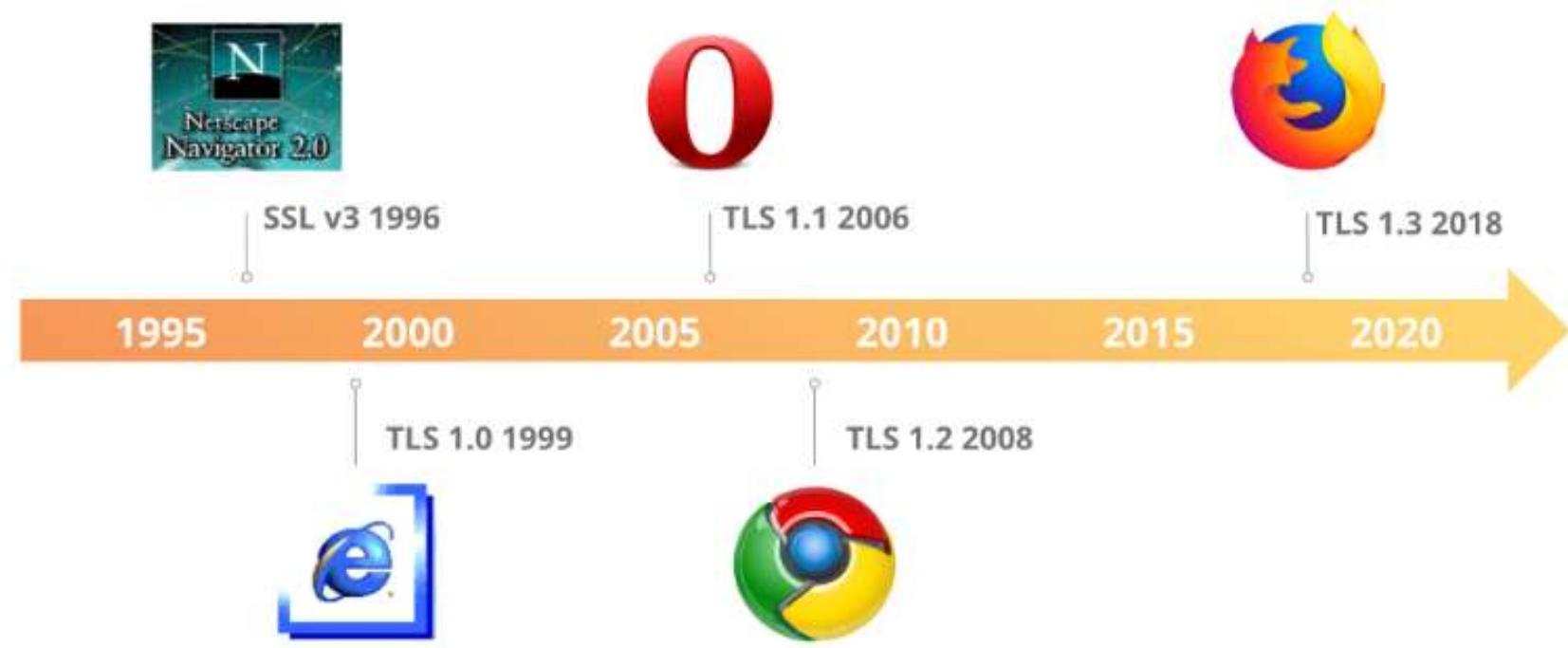
2. Objectifs

- Pour atteindre ces objectifs, SSL utilisera logiquement :

- Les certificats et donc les signatures digitales (pour les authentifications)
- les digests et les MACs (pour le contrôle de l'intégrité)
- les chiffrements symétriques (pour la confidentialité - les cryptages et décryptages ne doivent pas réclamer un temps prohibitif).



3. Historique

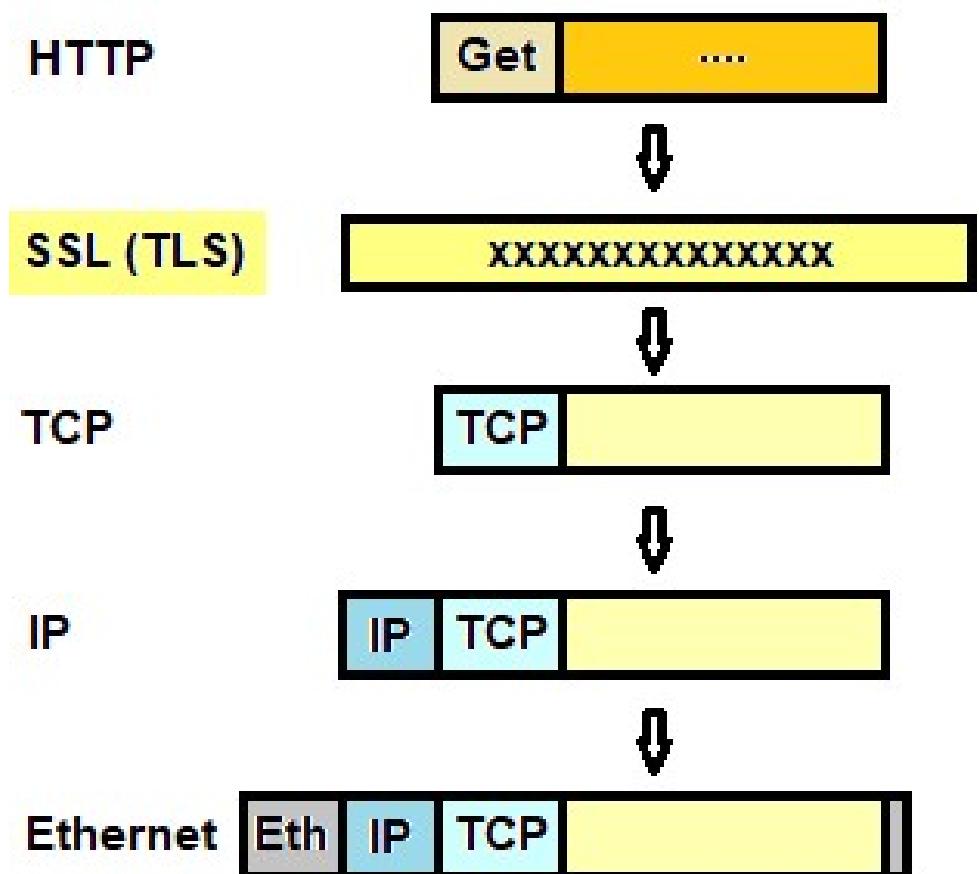


- SSL v1 a été conçu par Netscape en 1994
- SSL v2 fut intégrée au célèbre browser Navigator de Netscape.
- SSL v3 apparut en 1996, permettant notamment l'authentification du client.
- L'IETF racheta le brevet à Netscape en 1999 et s'en inspira pour créer le protocole
- **TLS** (Transport Layer Security Protocol)

4. Protocole entre deux couches

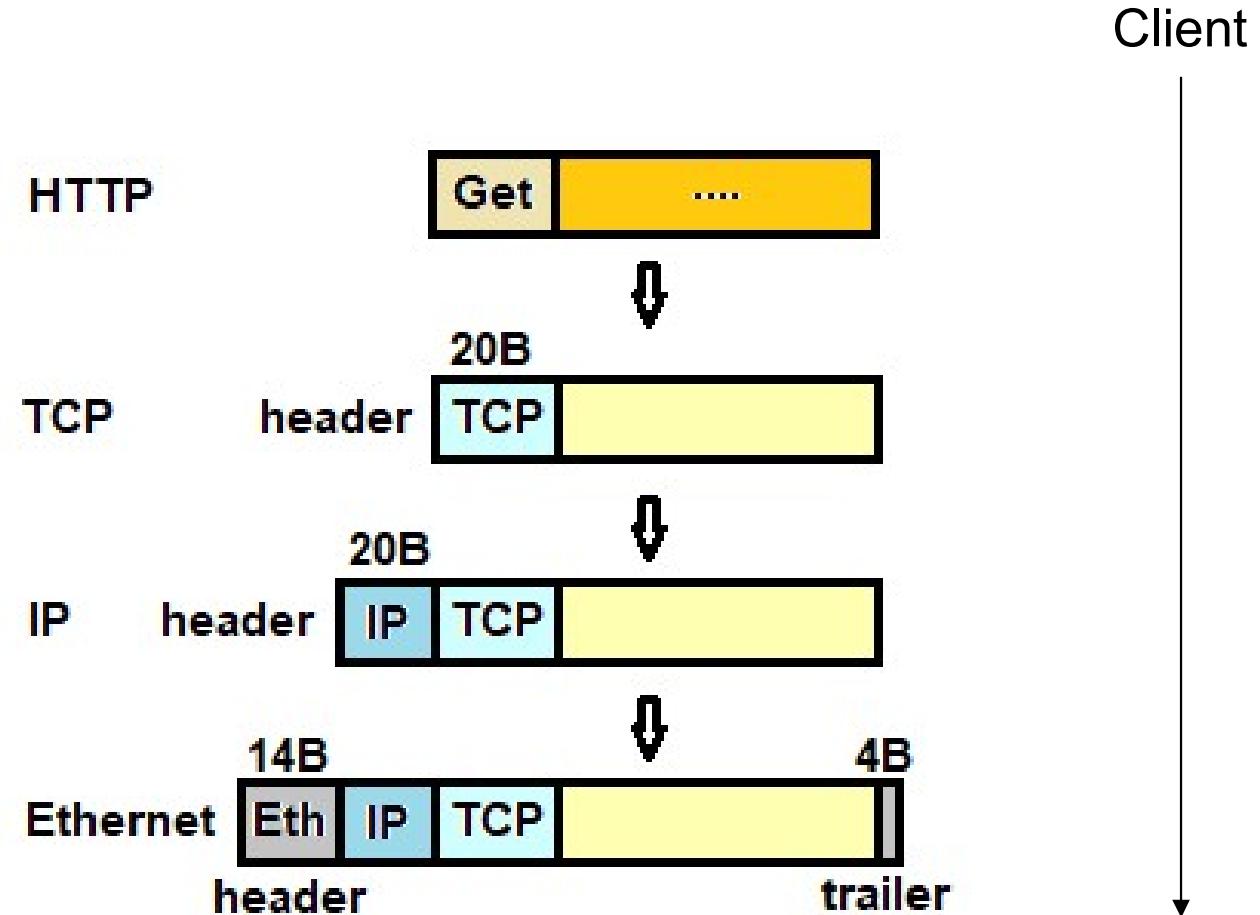
- SSL se situe entre le protocole applicatif et la couche transport (soit TCP)
- il est donc indépendant des applications qui l'utilisent. On ajoute simplement un « s » au nom du protocole: http/https
- Des ports TCP réservés sont attribués à ces protocoles sécurisés :

protocole	port
https	443
smt�	465
ftps	990
ftps-data	989
telnets	992
pop3s	995
imaps	993

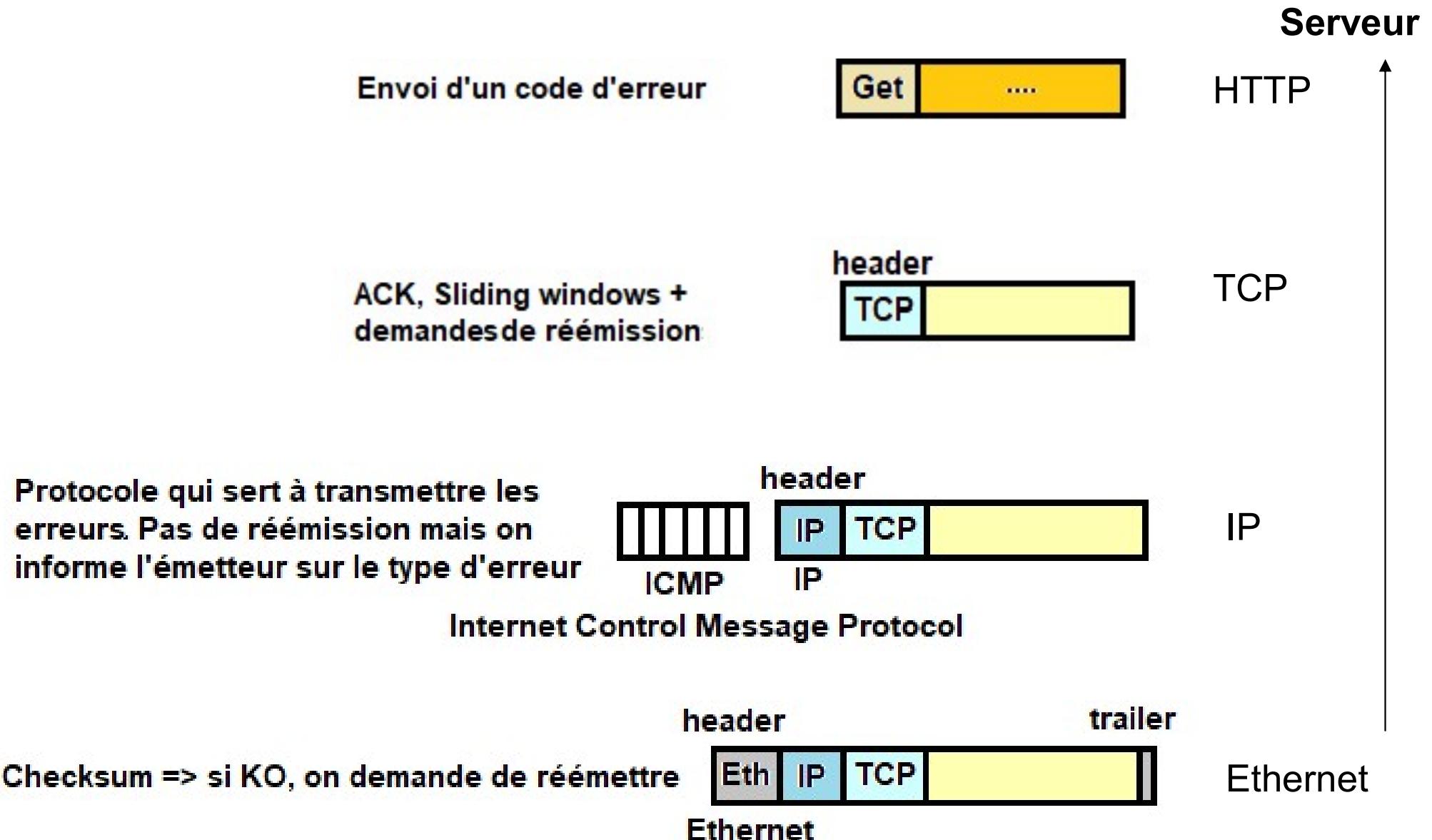


II. LA COMMUNICATION SSL

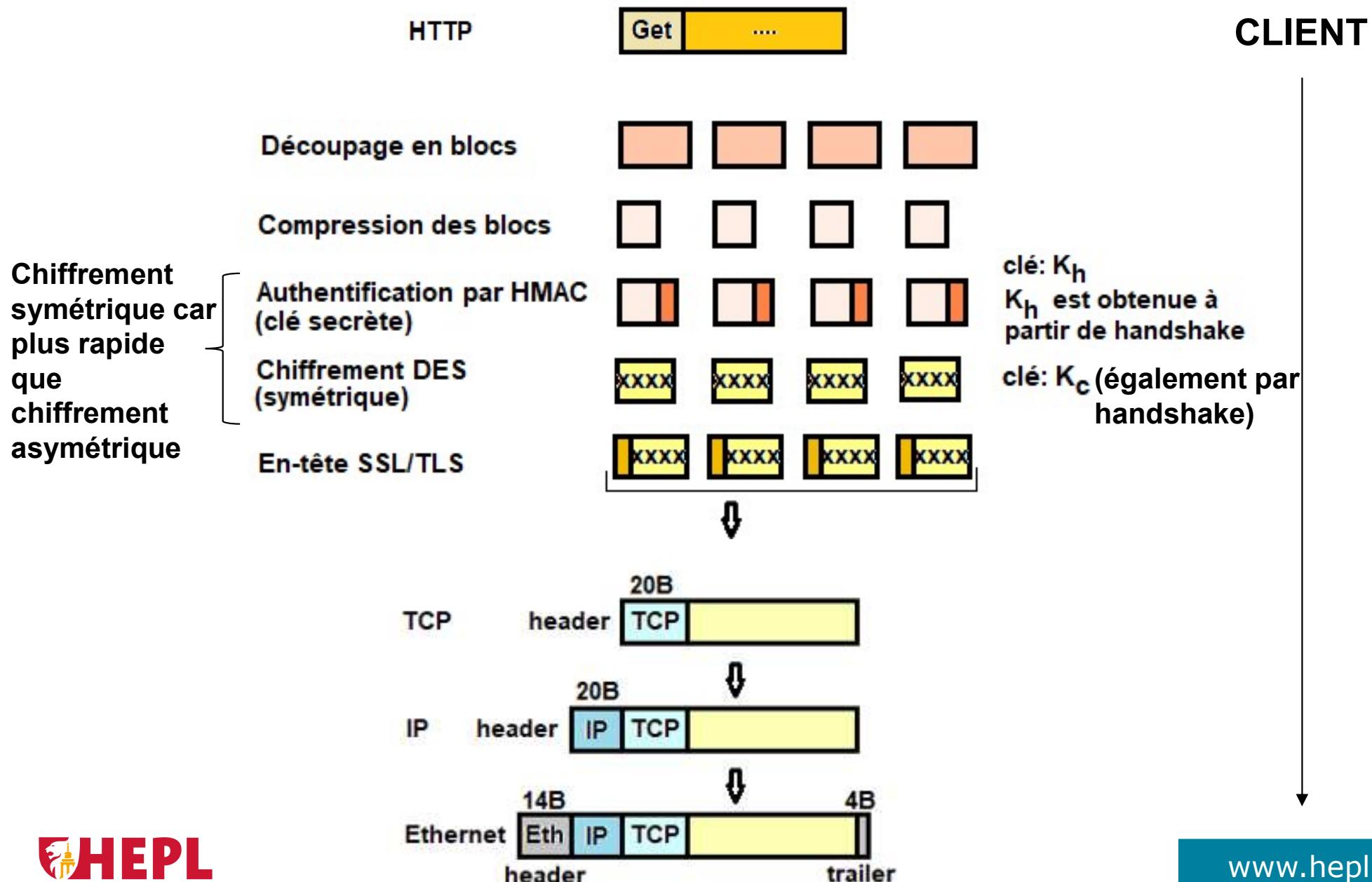
1 Rappel: stack TCP (émission)



2. Rappel stack TCP (réception)



3. Stack TCP avec SSL/TLS (émission)



4. Stack TCP avec SSL/TLS (réception)

Gestion des erreurs

SSL-Alert = s

1B

Warning ou fatal error

Warning => on peut continuer
Fatal => STOP (ex)

Error code: ex: client non certifié soit on continue soit on arrête

VALEUR	NOM	ERREUR FATALE	COMMENTAIRE
0	Close_Notify	Non	L'émetteur indique qu'il va terminer la connexion
10	Unexpected_Message	Oui	L'émetteur indique qu'il a reçu un message qu'il ne peut interpréter
20	Bad_Record_Mac	Oui	L'émetteur indique que le message reçu semble être altéré (la somme de contrôle n'est pas la bonne)
30	Decompression_Failure	Oui	L'émetteur indique qu'il a reçu des données qu'il ne peut décompresser
40	Handshake_Failure	Oui	L'émetteur indique qu'il n'a pas été capable de négocier un jeu de sécurité qui lui convient pour cette session
41	No_Certificate	Non	Le client indique qu'il ne dispose d'aucun certificat pouvant répondre aux attentes du serveur
42	Bad_Certificate	Non	L'émetteur indique qu'il a reçu un certificat corrompu
43	Unsupported_Certificate	Non	L'émetteur indique qu'il ne peut gérer le type de certificat reçu
44	Certificate_Revoked	Non	L'émetteur indique que le certificat reçu a été révoqué par l'organisme de certification
45	Certificate_Expired	Non	L'émetteur indique que le certificat a expiré.
46	Certificate_Unknown	Non	L'émetteur indique qu'il y a eu un problème avec le certificat reçu
47	Illegal_Parameter	Non	L'émetteur indique qu'il a reçu un handshake contenant des valeurs de paramètre non conformes



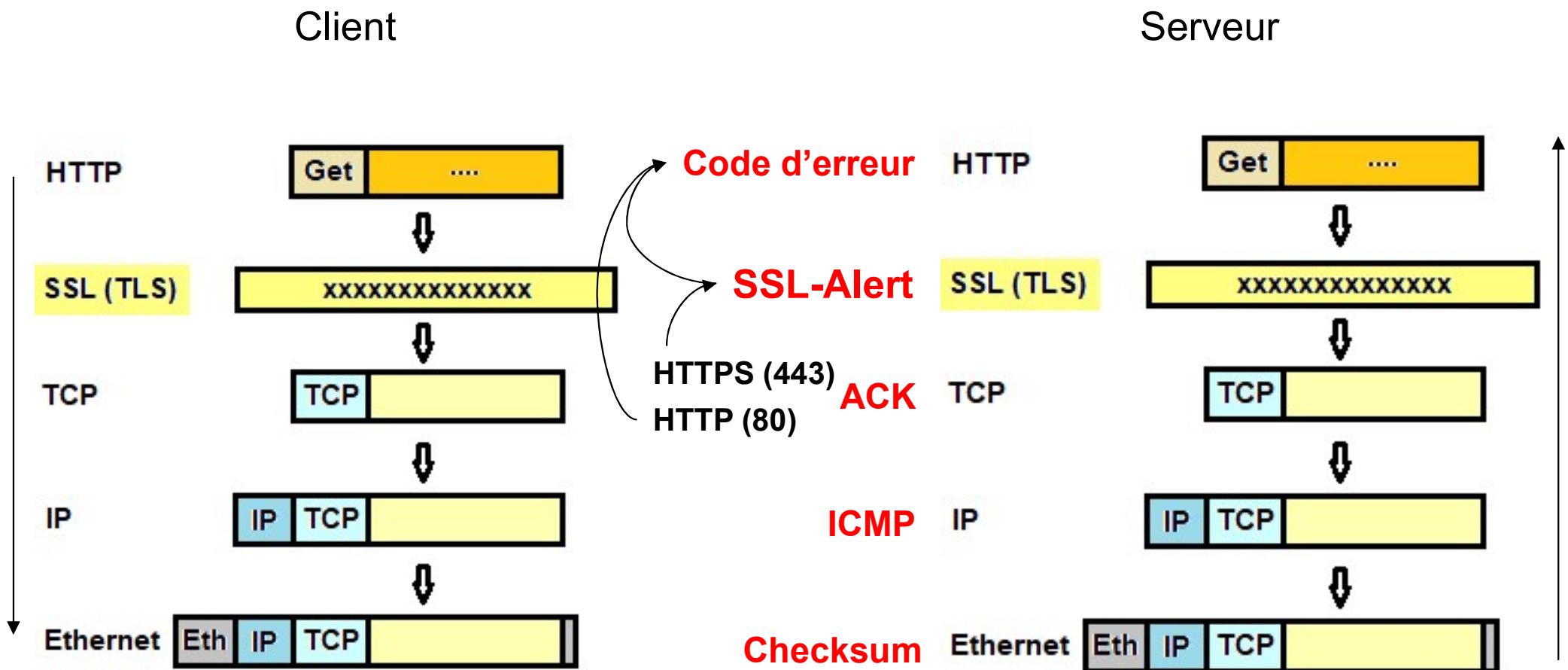
Remarque: K_h est une clé dite « **read** », permet d'authentifier chaque bloc **reçu**

SERVEUR

SSL/TLS



5. Communications



6. Réponse du serveur

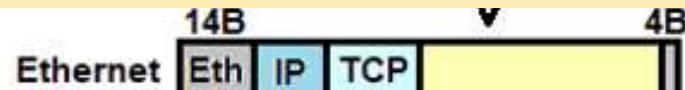
Le SSL handshake doit donc échanger **3 clés**: K_h , $K_{h'}$, K_c .

On **négocie** également les types d'algorithme de :

- **chiffrement (K_c)** (DES, tripleDES, ECB,CBC, ...)
- **compression**
- **hashage (K_h , $K_{h'}$)** (HMAC, MD5, SHA1, SHA256, SHA...)

Et on **échange** toutes ces informations.

Le serveur doit s'authentifier. Le client doit être sûr qu'on se connecte bien sur le serveur légitime. L'inverse peut être vrai également, le client peut être authentifié (mais contrairement au serveur, pas obligatoire). Dans le cas d'une transaction bancaire, le serveur doit être sûr que c'est le bon client => authentification du client impérative.



Le SSL handshake est asymétrique

III. LE SSL HANDSHAKE

4 phases

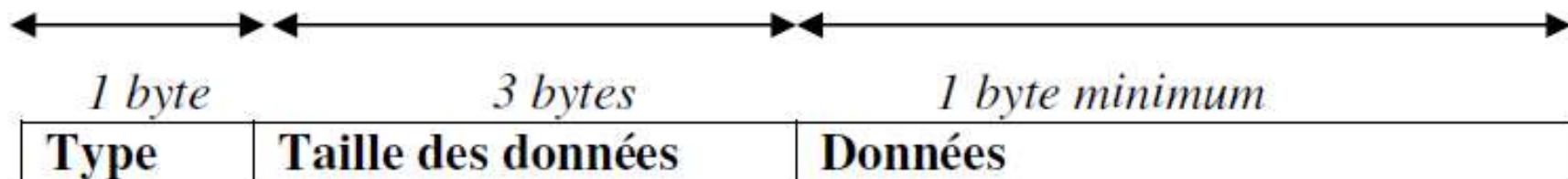
1. Objectifs

Le SSL handshake permet au serveur et au client de :

- s'authentifier (obligatoire pour le serveur, optionnel pour le client);
- négocier les algorithmes de chiffrement et de HMAC;
- négocier les clés symétriques à employer durant une session
(3 clés: K_c , K_h « clé write », $K_{h'}$ « clé read »).

Ces trois clés symétriques sont **les clés de session**.

Ces opérations sont basées, sur des échanges entre le client et le serveur à l'aide de messages au format TLV (Type, Length, Value):



2. Le sous-protocole: « SSL-Handshake »

Le sous-protocole SSL Handshake comporte 4 phases d'échanges de messages :

- 1) Etablissement des possibilités en matière de sécurité. Cette phase détermine entre autres la version, la session, les algorithmes de chiffrement et compression, des nombres aléatoires.
- 2) Authentification du serveur et échange des clés
- 3) Authentification du client et échange des clés
- 4) Clôture.

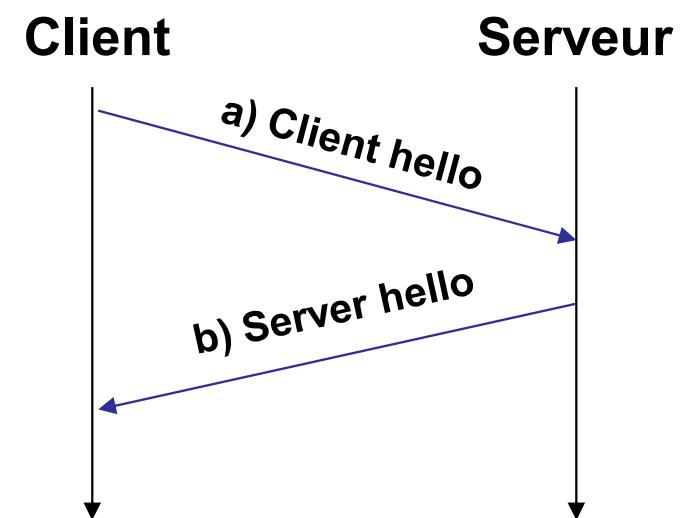
2.1 Phase 1: paramètres de sécurité

Le handshake est **paramétrable**: la succession des échanges peut évoluer (varier) en fonction des évènements (des résultats des échanges) précédents. Redirection vers un plan B ou C au lieu de continuer avec le plan A initial.

a) Le client envoie:

- **Version**: la plus haute que le client est capable d'utiliser
- **rnd_cli** : nombre aléatoire sur 28 bytes (ressemble à Diffie Helmann)
- **date-heure**: 4 bytes
- **session-id**:
 - *0* → créer une nouvelle connexion dans une nouvelle session;
 - *valeur non nulle* → créer une nouvelle connexion dans la session existante. Le handshake complet une fois par ex: pour la journée et si reconnexion durant cette même journée, on ne refait pas tout)

C'est le client qui initie les échanges avec un « Client hello »



2.2 Phase 1: paramètres de sécurité

[vers phase 1: wireshark](#)

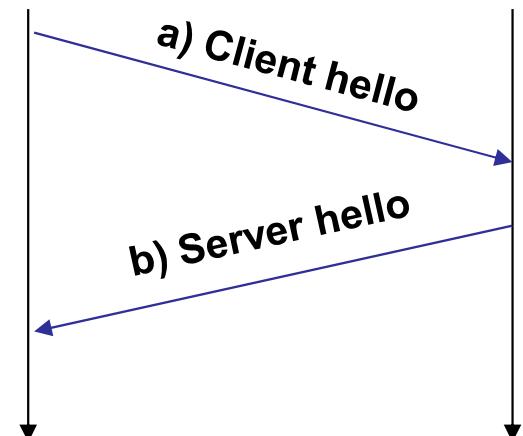
- **cipher-suite** : liste des algorithmes d'échanges (RSA, Diffie-Hellman), de chiffrements (DES, RC4, RC2 + par blocs CBC ou par flux) et digestes-MACs (MD5, SHA-1, SHA256, SHA384, SHA512, ...) et paramètres complémentaires (comme le vecteur d'initialisation IV pour CBC);
- Méthode de compression

b) Le serveur envoie:

- ses choix (combinaison d'algorithmes)
- un nombre aléatoire **rnd_svr** qui servira de numéro de version).

Le serveur choisit la stratégie optimale en tenant compte des renseignements reçus.

Client **Serveur**



Le client, une fois la réponse reçue du serveur, peut alors mémoriser les paramètres de sécurité et ainsi définis dans une structure.

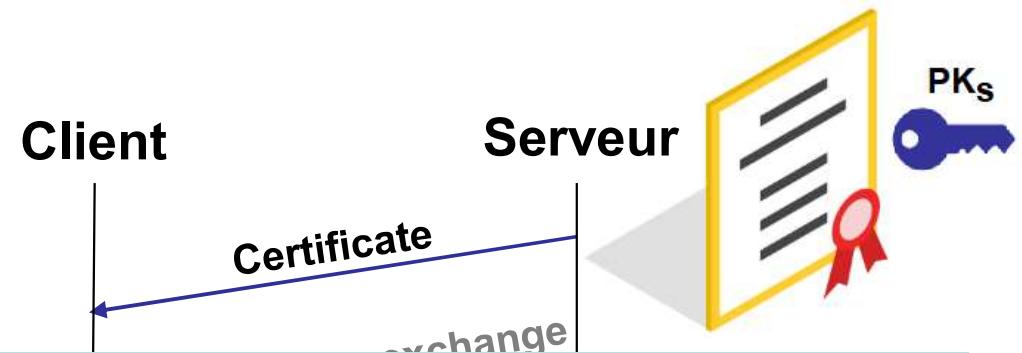
2.3 Phase 2: Authentification du serveur et génération des clés

a) certificate

Le serveur commence par envoyer son certificat dans un **message certificate**. Ce dernier est à l'origine le plus souvent d'un **certificat X509 niveau 3** (càd signé par une autorité de certification + vérifications poussées)

Il sera envoyé au format **ASN.1** (Abstract Syntax Notation version 1), assurant ainsi la plus grande compatibilité possible càd **rendre le certificat inaltérable** par rapport au jeu de caractères utilisé par la machine qui va lire ce dernier.

Dans le cas d'un certificat classique donnant une clé publique, le client est en état d'authentifier le serveur puis d'envoyer un message « **Premaster** » (éléments destinés à la génération de la future clé de session) en le cryptant avec cette clé publique.



Le certificat **doit être vérifié** (ici par le client) avec **la clé publique /le certificat du CA** présente dans les magasins de certificats (autorités de certification racine de confiance) par exemple du browser

Si la date du certificat n'est pas valide, un message **SSL-Alert** (warning => mais pas fatal : attention, se méfier car on peut continuer la procédure) est envoyé

2.4 Phase 2: Si le serveur n'a pas de certificat

Il existe différents types de certificat:

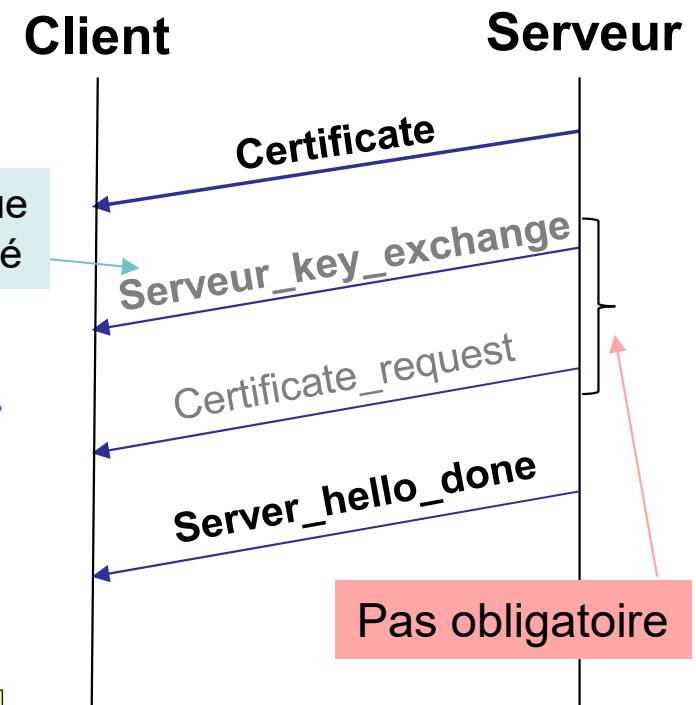
- **D'authentification.** La clé publique vérifie la signature sur un challenge (ex: HMAC)
- **De signature.** Signer un véritable document (pas un challenge).
- **De chiffrement.** On utilise alors la clé publique dans le mécanisme de cryptage

Il faut un plan B

le serveur (s'invente une paire de clés) doit alors fournir d'autres informations pour que le client soit en définitive en mesure de crypter son message **Premaster**.

- **Plan B1.** soit une **clé publique d'une paire de clés RSA générée par le serveur**; cette clé publique est temporaire : elle n'est valide que durant la session en cours. Le message est signé par le serveur selon un algorithme RSA.
- **Plan B2.** soit les **paramètres d'un algorithme de Diffie–Hellman**. Le message peut être alors signé par le serveur selon un algorithme RSA.

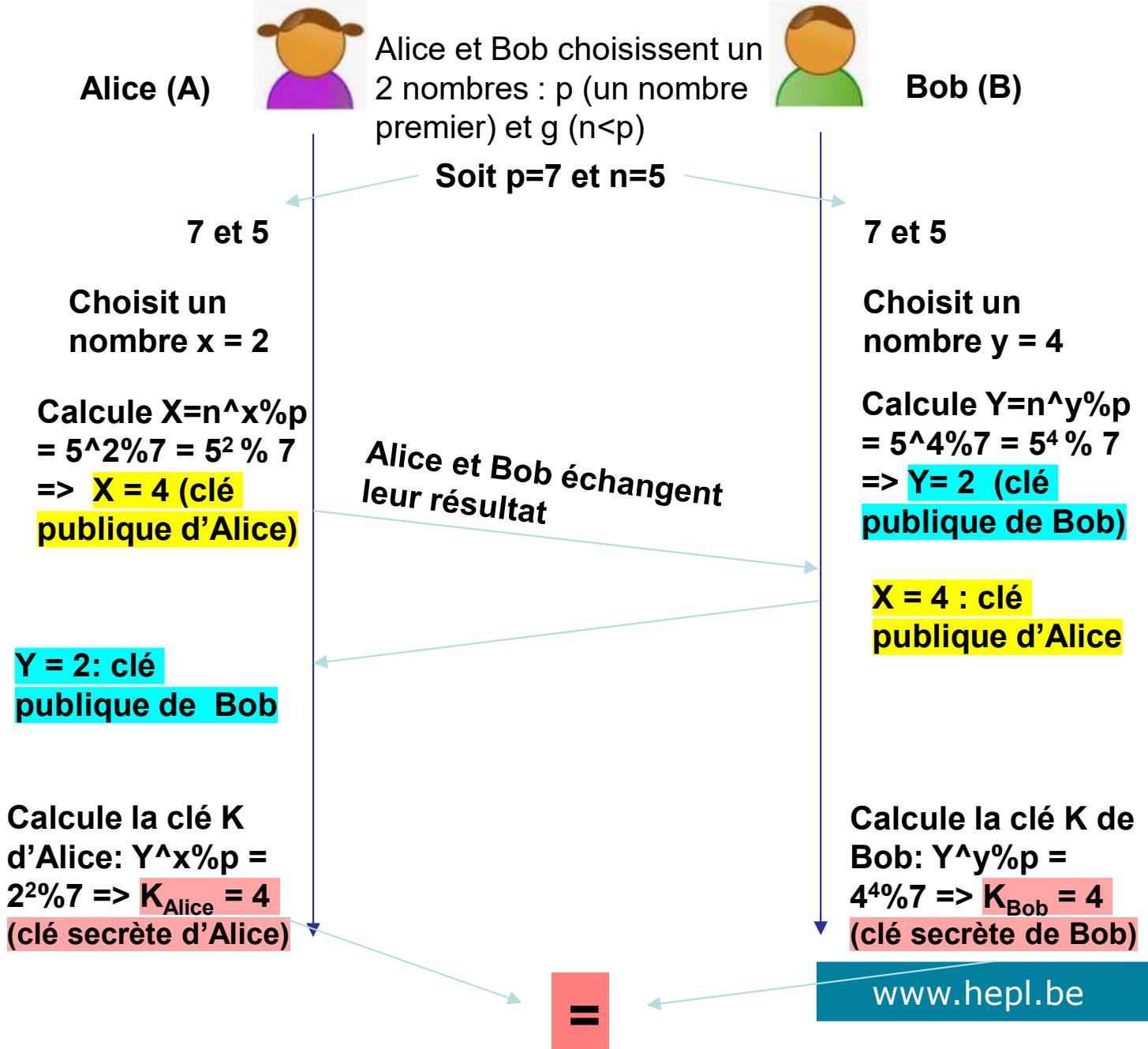
PK_{SAS} : clé publique
Serveur Auto-Signé



Algorithme de Diffie-Hellman

L'algorithme de Diffie-Hellman permet à deux parties A et B de générer la même clé symétrique (clé de session dans notre cas) en échangeant que deux grands entiers premiers entre eux (disons p et n); A (B) choisit un grand entier aléatoire x (y), calcule $X = n^x \% p$ ($Y = n^y \% p$) et l'envoie à B (A); - A (B) calcule sa clé selon $K_{Alice} = Y^x \% p$ ($K_{Bob} = X^y \% p$).

Les deux clés sont bien les mêmes.



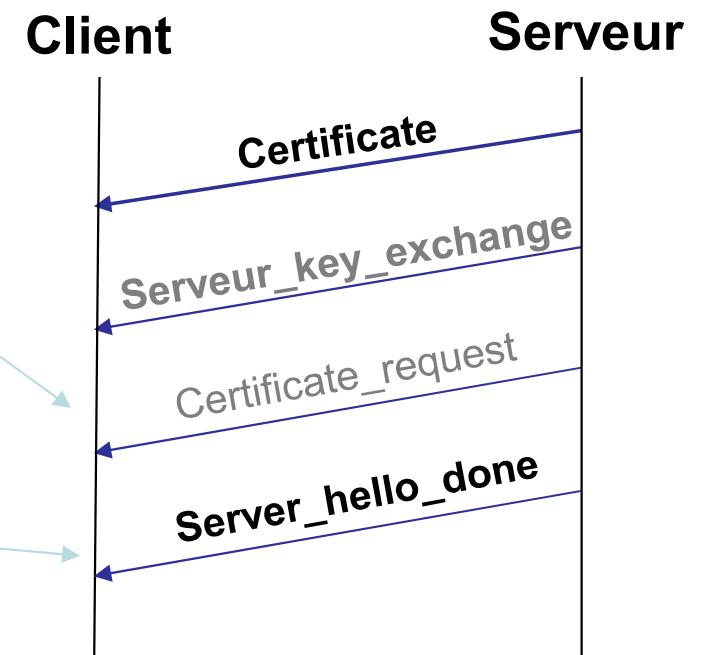
2.5 Phase 2 : certificat client (optionnel) et synchronisation

Certificate_request

Le serveur peut demander au client un certificat (afin notamment d'obtenir la clé publique du client). Il envoie alors un message **certificate_request**, qui comporte le type d'algorithme utilisé et la liste des CA valides. La réponse est dans la phase 3.

Server_hello_done

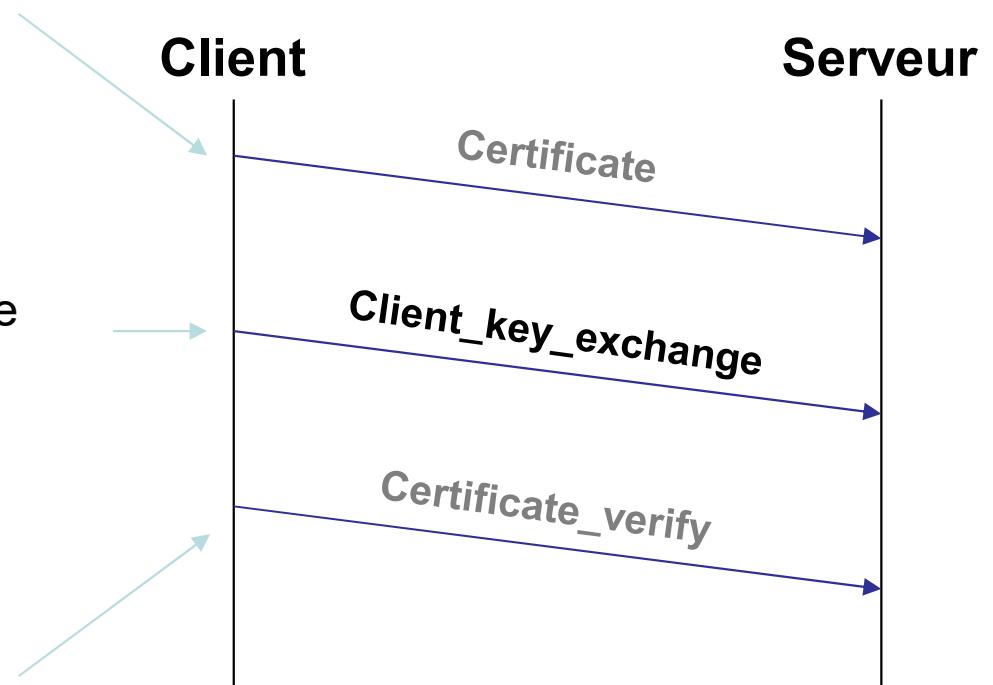
Le serveur envoie enfin un message **server_hello_done** pour signifier la fin de la phase



Au minimum lors de la phase 2, il y a 1 seul message « **Certificate** ». Il pourrait être suivi d'un « **Server_key_exchange** » et/ou éventuellement d'un « **Certificate_request** ». Le client ne sait donc pas précisément combien il va recevoir de message(s): 1 à 3 => le dernier envoi « **Server_hello_done** » permet de clôturer les échanges = **synchronisation**

2.6 Phase 3: La génération du "master secret" + certificat client (optionnel)

- Si le serveur a effectivement demandé un certificat au client, celui-ci commence par envoyer son certificat dans un message `certificate` si il en possède un, un message `no_certificate` s'il n'en possède pas.
- Le client envoie ensuite un message `client_key_exchange` dont la nature dépend de l'algorithme choisi en phase 2
- Si le client a envoyé un certificat, on vérifie qu'il est bien en possession de la clé privée associée à la clé publique faisant l'objet du certificat. Pour ce faire, le client va envoyer un message **`certificate_verify`** qui comporte une espèce de **digest calculé sur les messages du handshake et sur le "master secret"** (donc un "**challenge**")

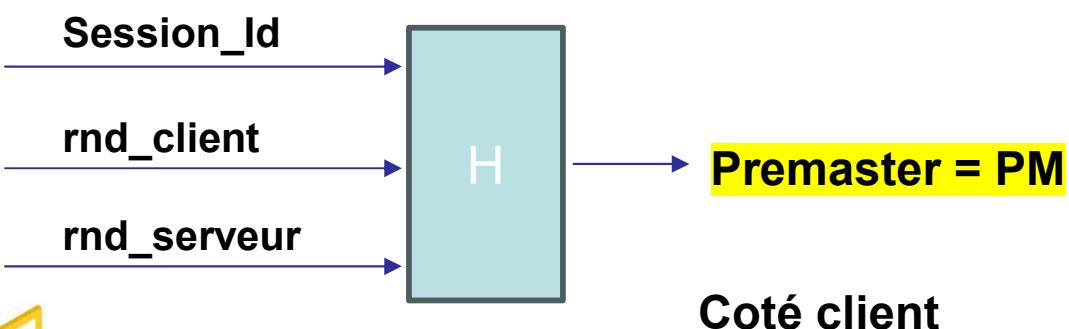


2.7 Phase 3: la génération du « master secret »

Client_key_exchange (mandatory):

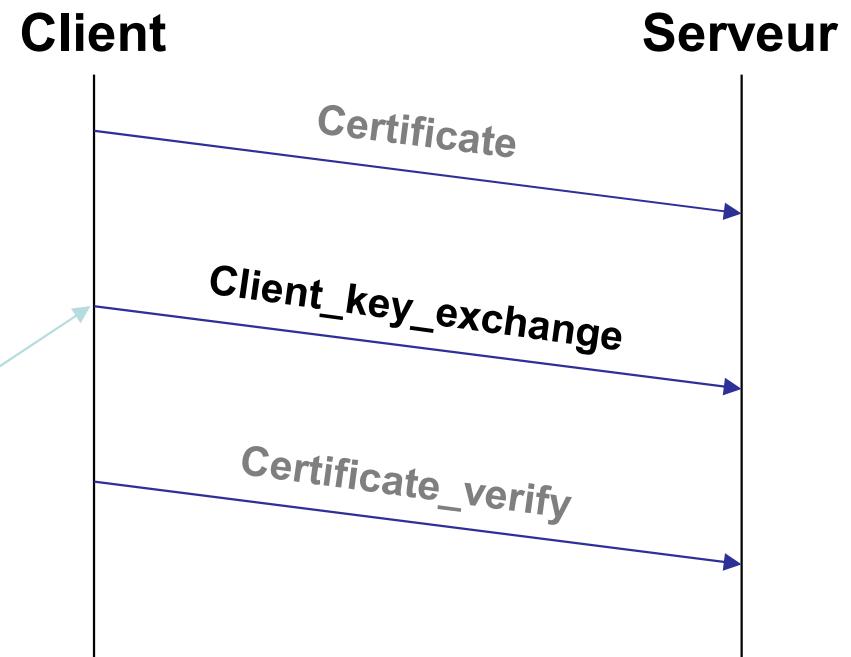
- RSA (basé sur le certificat du serveur ou sur un RSA temporaire)

Le client génère une séquence de 48 octets, le "**premaster secret**", **la crypte avec la clé publique du serveur** et la lui envoie



Coté client

Certificat du serveur	RSA temporaire
$\{ \text{PM} \} \text{PK}_S = P$	$\{ \text{PM} \} \text{PK}_{SAS} = P$ (plan B1)



2.8 Phase 3: la génération du « master secret »



Côté serveur

PrK_S = clé privée du serveur

Certificat du serveur	RSA temporaire
-----------------------	----------------

$$\{ P \} \text{PrK}_S = PM$$

$$\{ P \} \text{PrK}_{SAS} = PM \text{ (plan B1)}$$

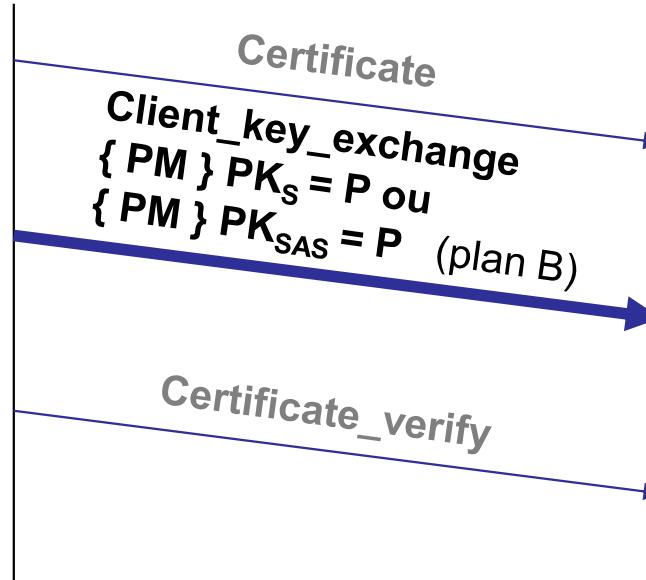


PK_S

PK_{SAS} (plan B1)

Client

Serveur



PrK_S

PrK_{SAS} (plan B1)

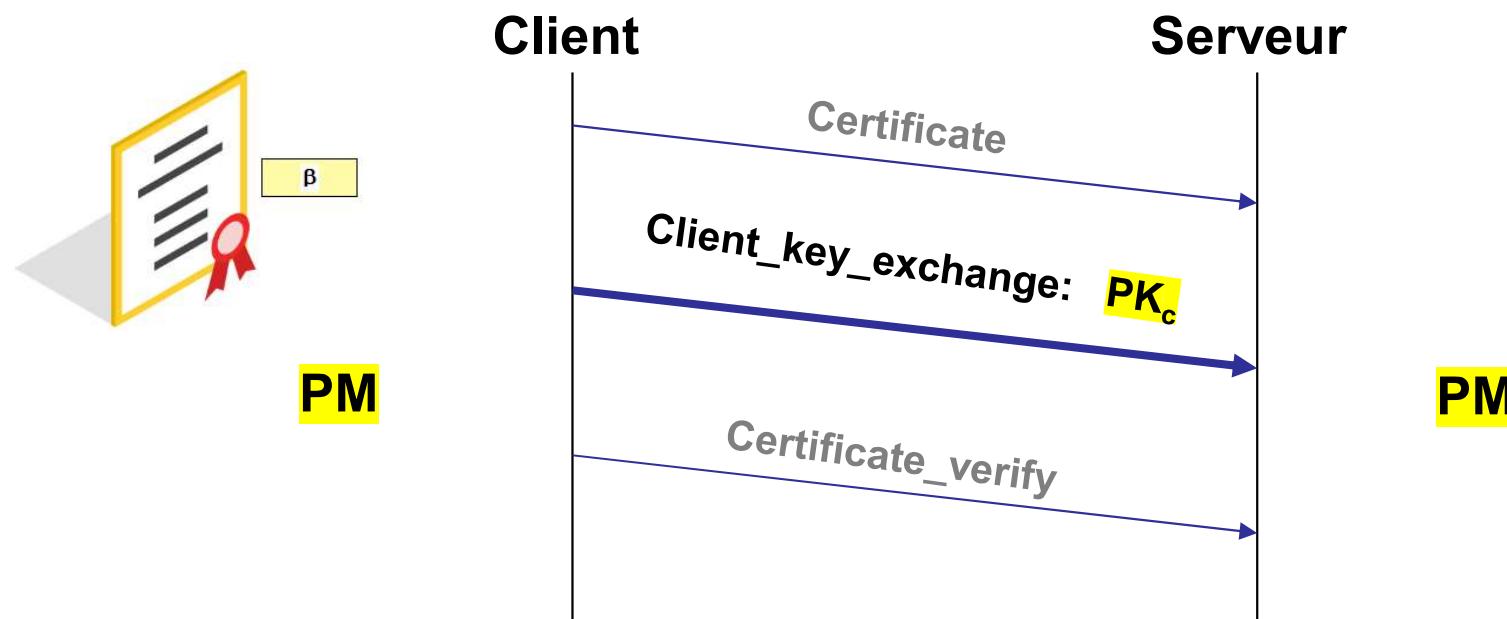
2.9 Phase 3: la génération du « master secret » plan B2

Utilisation de l' algorithme de Diffie–Hellman.

Le client envoie sa clé publique: PK_c

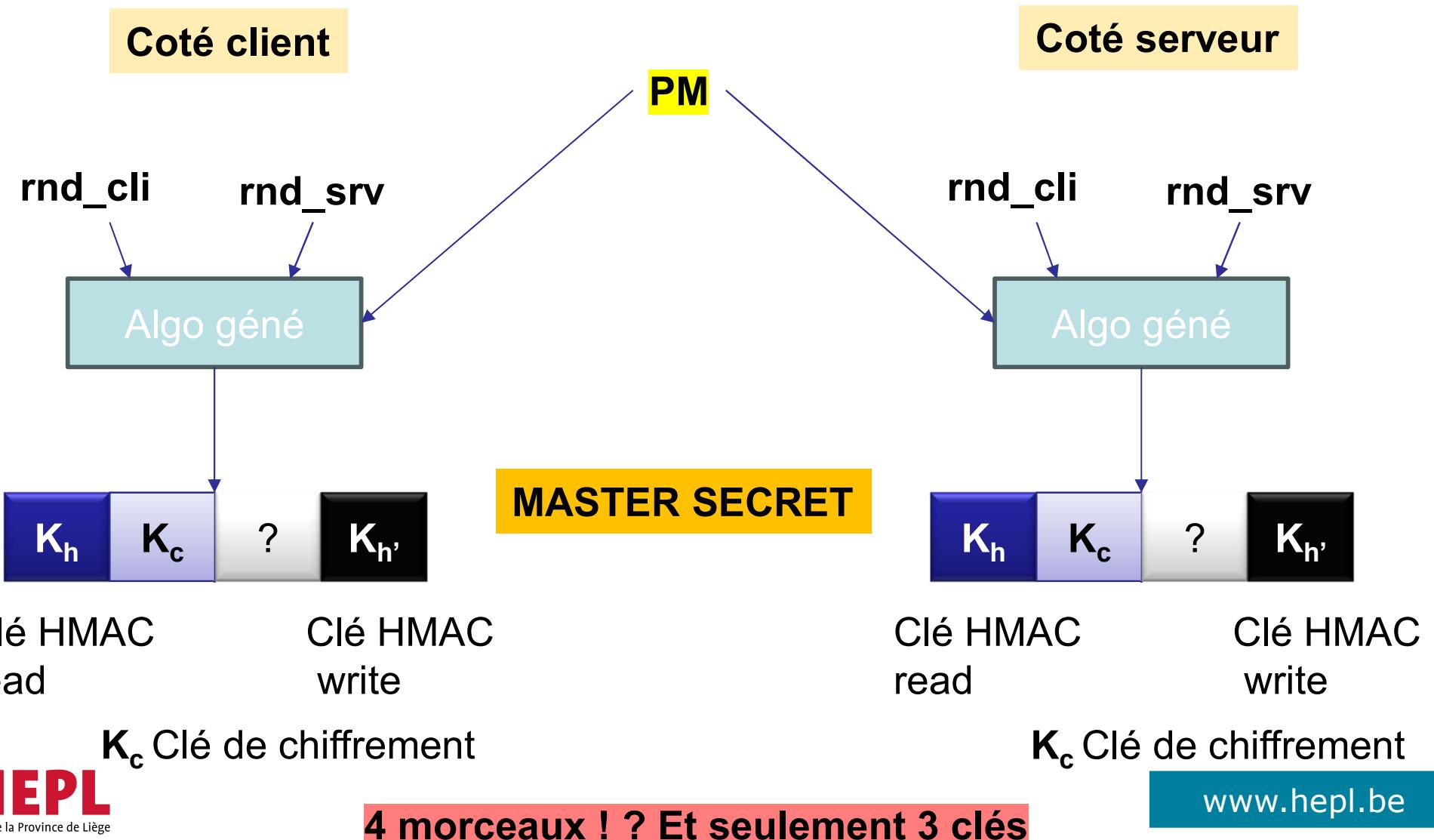


la clé de session ainsi construite des deux côtés fait office de « Premaster secret ».



2.10 Phase 3: Génération du « Master secret » (M)

La **PM** (Premaster secret) est maintenant connue dans les 3 cas d'échanges vus précédemment.

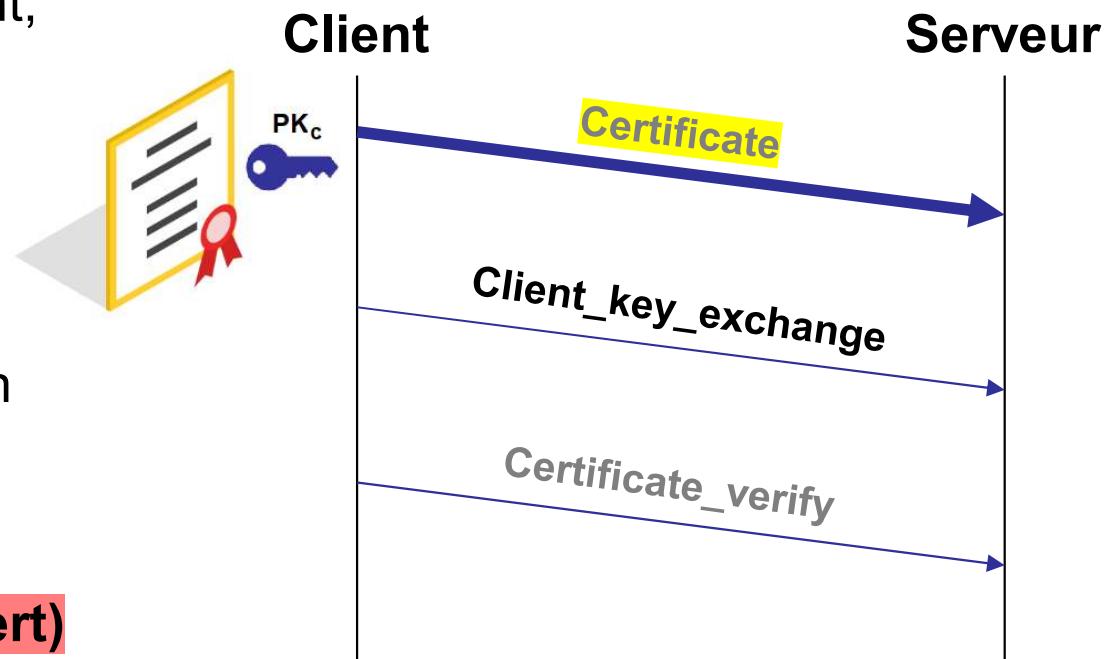


2.11 Phase 3: Les certificats éventuels (optionnel): « Certificate »

Si le serveur a effectivement demandé un certificat (pas mandatory) au client, celui-ci commence par envoyer:

- son certificat dans un **message certificate** si il en possède un
- un **message no_certificate** s'il n'en possède pas.

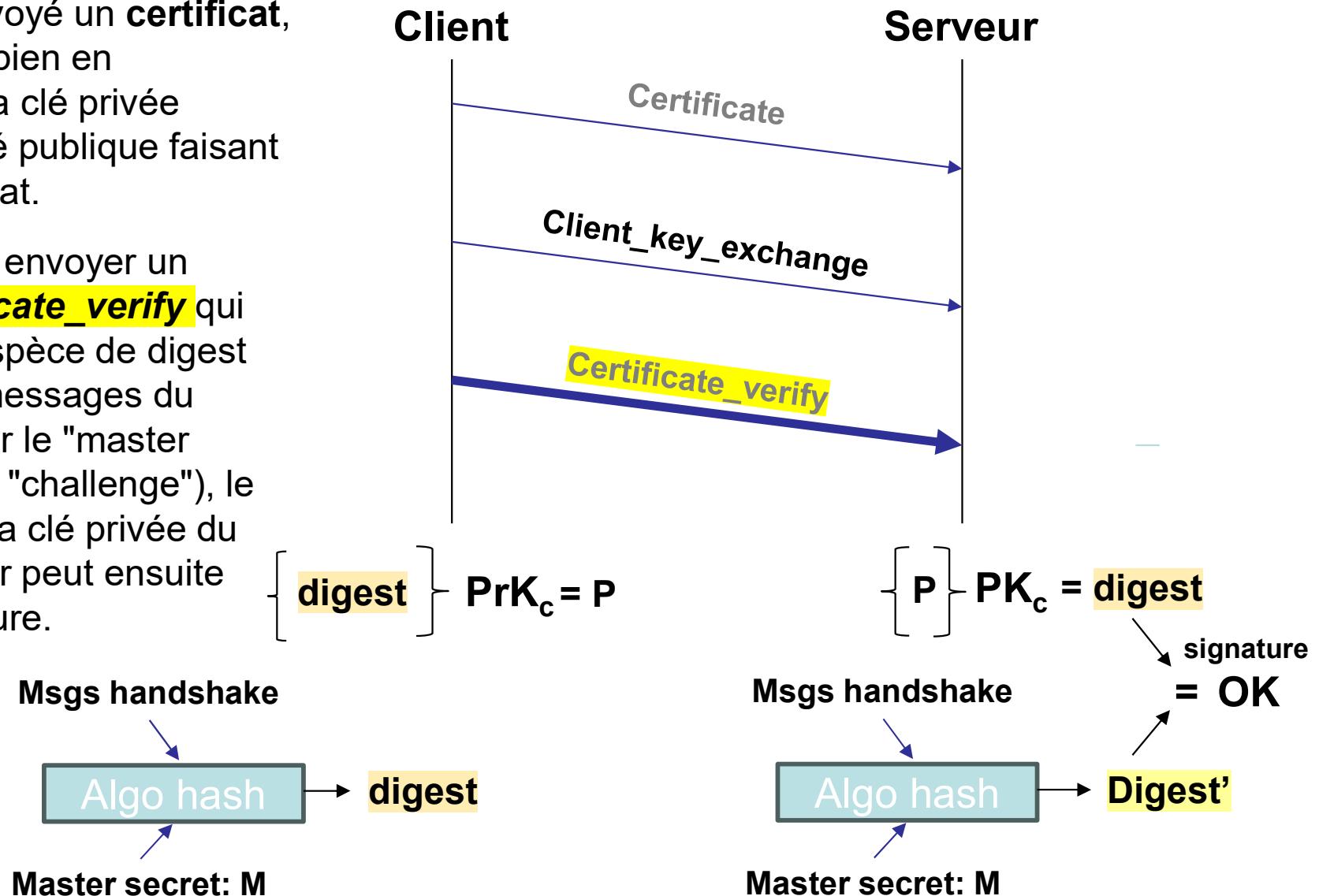
=> Warning ! (SSL_Alert)



On a donc le choix de continuer ou d'arrêter le handshake
=> dans ce dernier cas, pas de session SSL/TLS créée.

2.11 Phase 3: Les certificats éventuels (optionnel): « Certificate_verify »

- Si le client a envoyé un **certificat**, on test qu'il est bien en possession de la clé privée associée à la clé publique faisant l'objet du certificat.
- le client va donc envoyer un message **certificate_verify** qui comporte une espèce de digest calculé sur les messages du handshake et sur le "master secret" (donc un "challenge"), le tout signé avec la clé privée du client. Le serveur peut ensuite vérifier la signature.

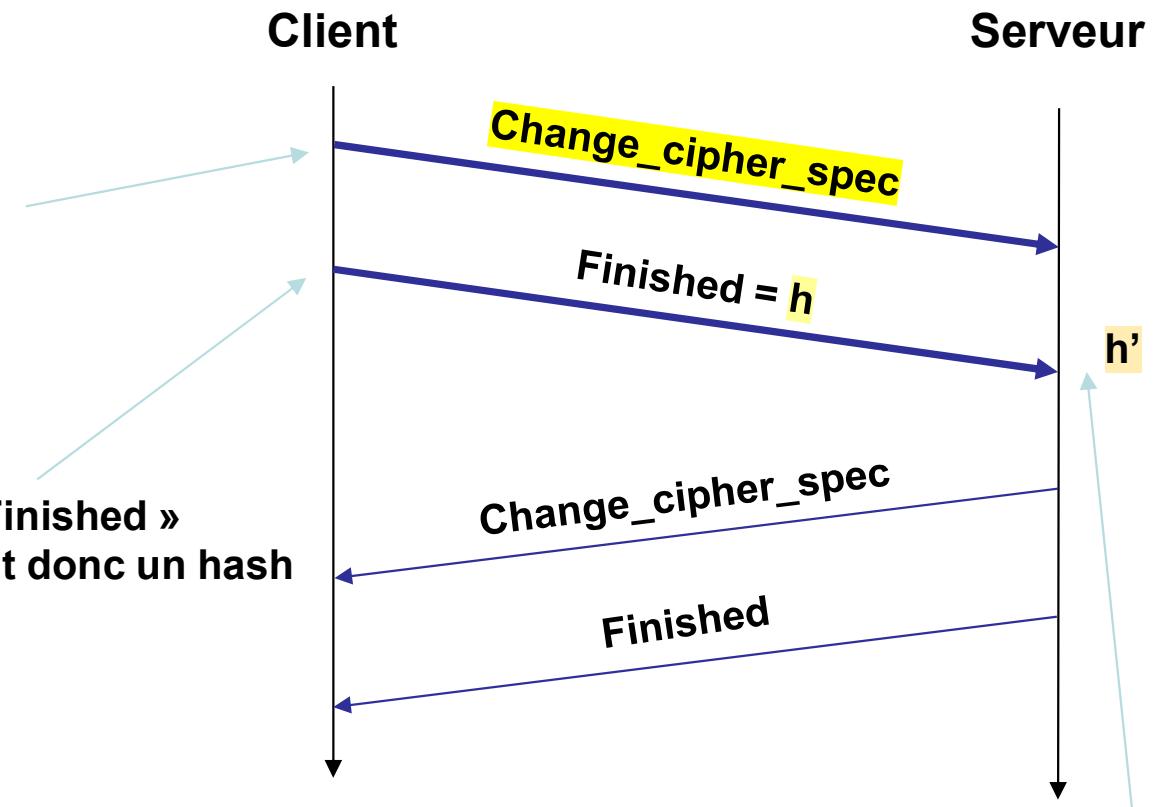
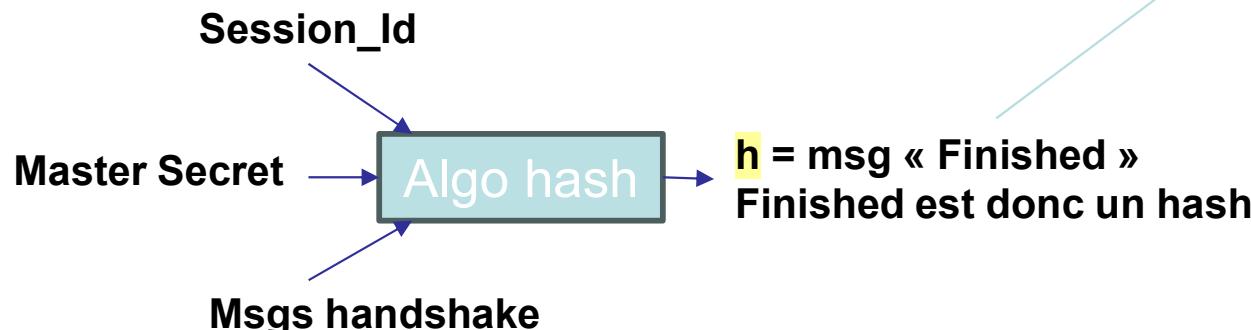


2.12 Phase 4: L'enregistrement et la validation des spécifications négociées

Les clés (3) sont maintenant en mémoire des deux côtés mais il faut:

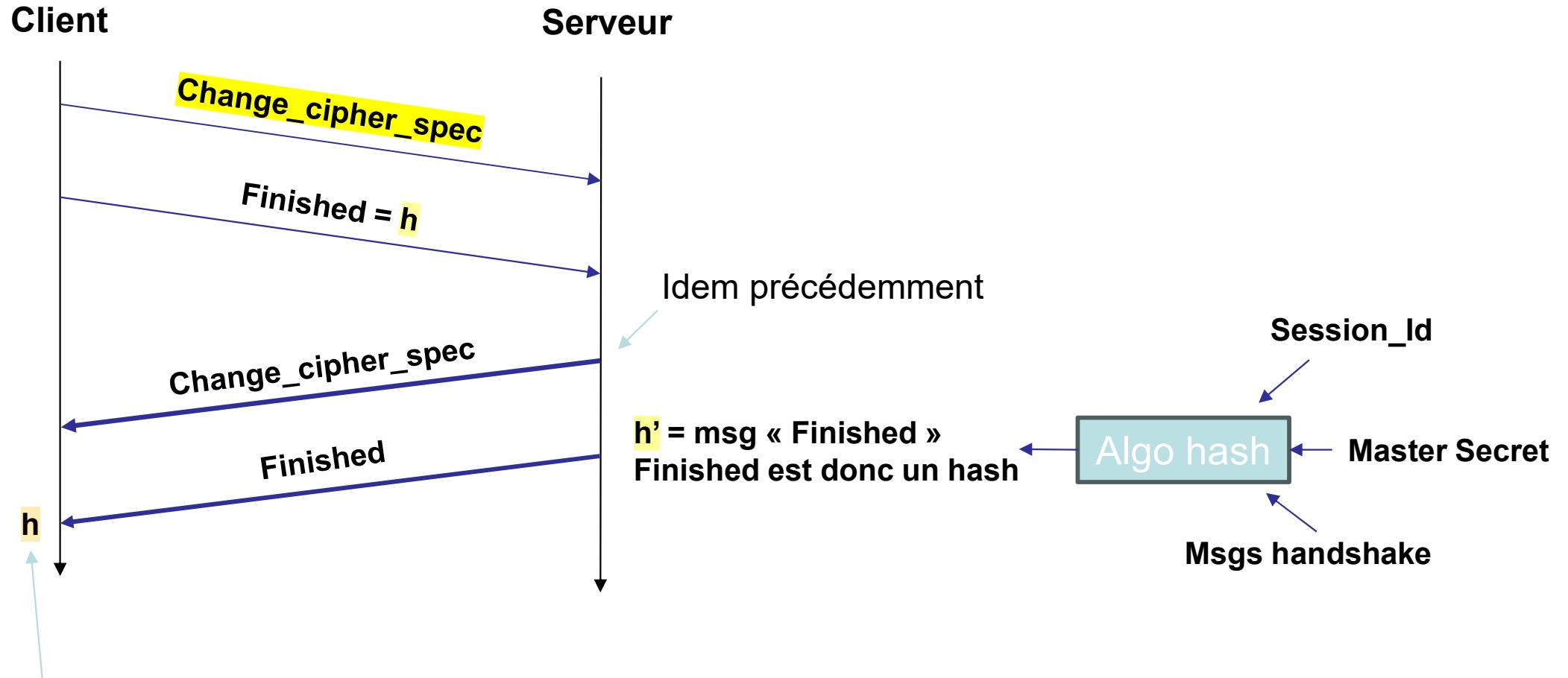
- leur assurer une pérennité => **enregistrement de ces clés**
- être certain des 2 côtés que les 3 clés ont **bien été créées**

Le client envoie un message **change_cipher_spec**, dont le but est d'enregistrer la stratégie de sécurité temporaire en stratégie courante.



Du côté serveur: on fabrique également le hash (**h'**) et si **h' = h** alors **OK. Le serveur est donc sûr que c'est bien le client du début**

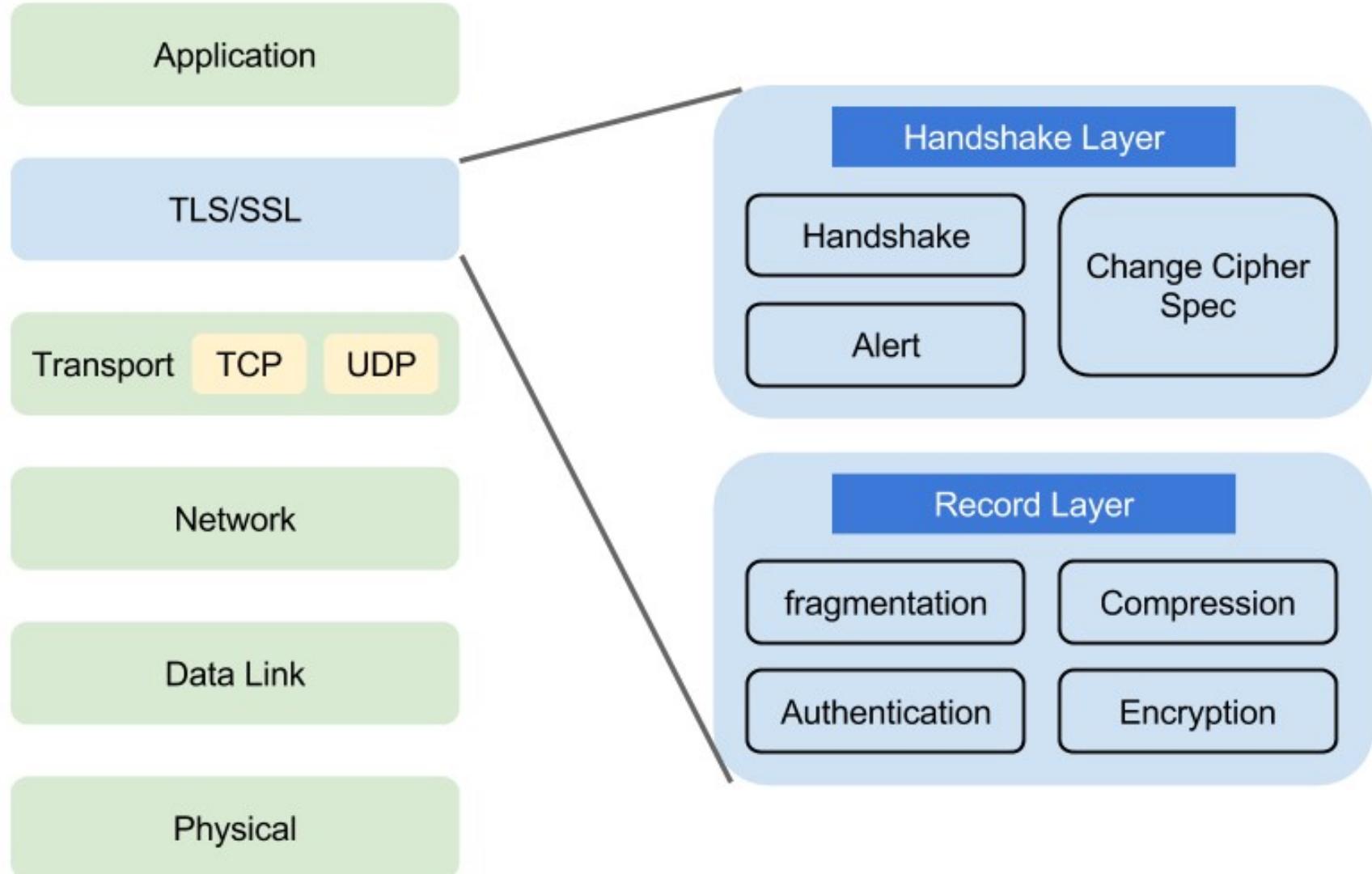
2.13 Phase 4: L'enregistrement et la validation des spécifications négociées



IV. LES 4 SOUS-PROTOCOLES DE SSL/TLS

- 1) SSL Handshake
- 2) SSL Record
- 3) SSL Change cipher spec
- 4) SSL Alert

1. Description (couches)



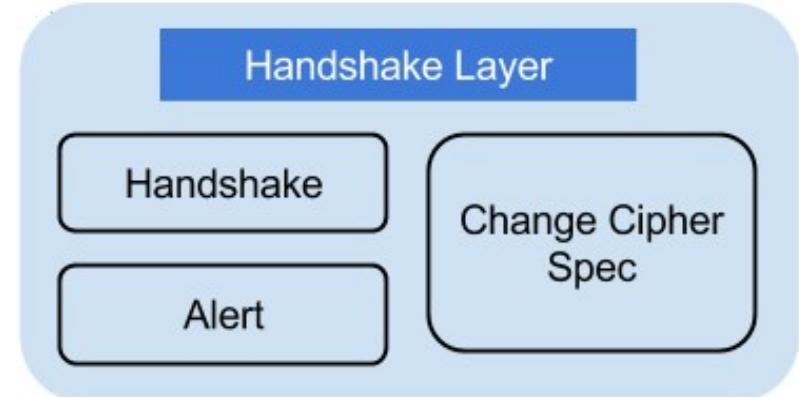
2. Description (fonctions / couches)

Handshake layer

Le sous-protocole

SSL handshake

- gère l'établissement de la connexion sécurisée
- effectue la négociation des paramètres de la session qui est susceptible de lancer:
 - **le sous-protocole SSL Change Cipher SPEC** » qui permet au serveur et au client de s'accorder sur les algorithmes de chiffrements asymétrique (en pratique, par exemple, RSA) et symétrique (par exemple, DES) ainsi que sur l'algorithme d'authentification (par exemple, SHA-1, MD5, SHA 256, SHA 384, ...)
 - **le sous-protocole SSL Alert** » gère les messages d'erreur entre le client et le serveur



3. Description (suite)

Record layer

Le sous-protocole **SSL record**

- Cette couche définit la façon dont sont traitées les données à chiffrer:
 - le chiffrement au moyen de la clé secrète négociée dans le "**Handshake protocol**", chiffrement précédé de la compression du message [**confidentialité**];
 - La création d'un HMAC utilisant la clé en question [**intégrité + authentication**].
En réalité, le message est au préalable fragmenté et ce traitement est appliqué à chaque bloc.

Il est susceptible de lancer le sous protocole "**SSL Alert**".

4. SSL record (Suite)

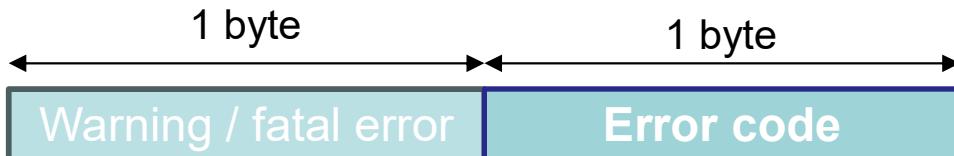
- Les données à transmettre sont découpées en bloc de taille inférieure à 16 Ko
- Chaque bloc est ensuite éventuellement compressé – si c'est le cas, ce doit être sans pertes et ne doit pas augmenter la taille du contenu de plus de 1024 bytes
- On calcule un HMAC pour chaque bloc
- Chaque bloc avec son HMAC, est crypté selon le cryptage symétrique négocié précédemment
- un header SSL est ajouté devant chaque bloc. Ce dernier contient:
 - un Content type (1 byte) : le protocole utilisé pour manipuler le fragment (soit change_cipher_spec, alert ou handshake, soit un protocole applicatif comme http ou ftp).
 - une Major version (1 byte) : version sslv3/TLS1.3
 - une Minor version (1 byte) : si sslv3 : valeur = 0
 - une Compressed length (2 bytes) : longueur du fragment de texte clair.Schématiquement, ce qui est transmis à la couche de transport est donc :

1B	1B	1B	2B	nB
Content type	Major version	Minor version	Length	(Bloc de donnée + HMAC) cryptés

5. Description: SSL_Alert

Le sous protocole **SSL Alert**

Il gère les messages d'erreur entre les deux entités: client/serveur, serveur/client



En cas d'erreur fatale, la connexion est fermée après l'envoi du message. Les autres connexions de la session concernée ne sont pas coupées, mais il n'est plus possible d'en créer de nouvelles.

VALEUR	NOM	ERREUR FATALE	COMMENTAIRE
0	Close_Notify	Non	L'émetteur indique qu'il va terminer la connexion
10	Unexpect_Message	Oui	L'émetteur indique qu'il a reçu un message qu'il ne peut interpréter
20	Bad_Record_Mac	Oui	L'émetteur indique que le message reçu semble être altéré (la somme de contrôle n'est pas la bonne)
30	Decompression_Failure	Oui	L'émetteur indique qu'il a reçu des données qu'il ne peut décompresser
40	HandShake_Failure	Oui	L'émetteur indique qu'il n'a pas été capable de négocier un jeu de sécurité qui lui convient pour cette session
41	No_Certificate	Non	Le client indique qu'il ne dispose d'aucun certificat pouvant répondre aux attentes du serveur
42	Bad_Certificate	Non	L'émetteur indique qu'il a reçu un certificat corrompu
43	Unsupported_Certificate	Non	L'émetteur indique qu'il ne peut gérer le type de certificat reçu
44	Certificate_Revoked	Non	L'émetteur indique que le certificat reçu a été révoqué par l'organisme de certification
45	Certificate_Expired	Non	L'émetteur indique que le certificat a expiré.
46	Certificate_Unknown	Non	L'émetteur indique qu'il y a eu un problème avec le certificat reçu
47	Illegal_Parameter	Non	L'émetteur indique qu'il a reçu un handshake contenant des valeurs de paramètre non conformes

V. LES VARIABLES D'ÉTATS SSL/TLS

- 1) Variables de session**
- 2) Variable de connexion**

1. introduction

- L'existence de ces deux types de variable provient du fait qu'il faut **éviter** au client de devoir **refaire tout le handshake** à chaque fois qu'il désire se reconnecter (dans un intervalle de temps défini par exemple: la journée).
- La session (**variables de session**) existera donc durant cet intervalle de temps (**ces variables ne changent pas durant toute la session**)
- il y aura éventuellement des nouvelles connexions (**variables de connexion**). **Ces variables sont modifiées à chaque nouvelle connexion.**

2. Variables de session

Ne changent pas durant la session

Session SSL	
Session identifier	numéro de session fixé par le serveur (1 byte)
Peer certificate(s)	certificat du serveur + certificat éventuel du client
Cipher spec	algorithme de chiffrement symétrique, de digest, CBC (avec IV) ou stream
Master secret	séquence de bits de 48 bytes, base de la clé de session utilisée pour le chiffrement et de la clé secrète utilisée pour les HMACs
Compression method	méthode de compression
Is resumable	pour indiquer si de nouvelles connexions peuvent être créées sur base de cette session

3. Variables de connexion

Changent à chaque connexion/reconnexion

Connexion SSL	
server et client random	les nombres aléatoires échangés lors du handshake
server et client write HMAC secret	la clé secrète utilisée par le serveur et le client pour générer les HMACs
server et client write key	la clé symétrique utilisée par le serveur et le client pour chiffrer leurs messages
initialization vector	pour les chiffrements CBC
sequence number	numéro de séquence

Les 2 clés pour le HMAC + la clé de chiffrement = 3 clés
(ces 3 clés sont des variables de connexion)

Elles changent donc lors de chaque nouvelle connexion !

4. Remarque importante sur les 3 clés

- Dans le tableau des variables de session (constante durant la session)

Master secret	séquence de bits de 48 bytes, base de la clé de session utilisée pour le chiffrement et de la clé secrète utilisée pour les HMACs
---------------	---

- Dans le tableau des variables de connexions (changent lors de nouvelles connexion)

server et client write HMAC secret	la clé secrète utilisée par le serveur et le client pour générer les HMACs
server et client write key	la clé symétrique utilisée par le serveur et le client pour chiffrer leurs messages

Or le master secret =



= constante (session)

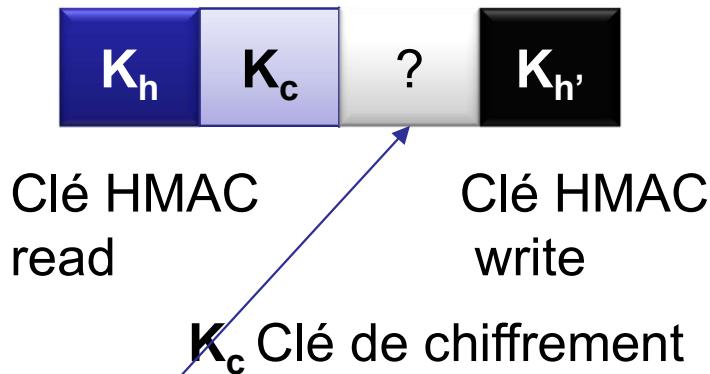
Les 3 clés
(connexions)
=> variables !!!

Clé HMAC
read

Clé HMAC
write

K_c Clé de chiffrement

5. Remarque suite



Le master secret est bien constant mais **contient 4 blocs**. Les 3 clés vont donc tourner dans les 4 blocs

⇒ **attribution des blocs différents par rapport aux 3 clés lors de chaque connexion / reconnexion pour une même session**
(permutations / arrangements)



.....

VI. FAIBLESSES DE SSL/TLS

1. Analyse des risques

Si lors de l'utilisation d'une connexion SSL/TLS, tout le monde suit les règles du jeu, ce protocole est robuste.

Risques:

- l'authentification du client est optionnelle, si bien que celui-ci peut utiliser, par exemple, un numéro de carte de crédit dérobée
- le serveur peut très bien utiliser les informations fournies par le client (par exemple, encore une fois, son numéro de carte de crédit) à des fins malhonnêtes
- le client a souvent tendance à ignorer les messages d'avertissement concernant un certificat invalide (autorité de certification non reconnue, validité dépassée).
- si un client s'authentifie à l'aide d'un certificat, il enregistrera probablement ses paramètres de reconnaissance sur son ordinateur : la cible rêvée pour un cheval de Troie ...

2. Fraudes / remèdes

Fraudes:

En fait, les fraudes constatées dans le domaine de l'e-commerce trouvent souvent leur explication dans le fait que les **serveurs auxquels les commandes sont envoyées reçoivent et mémorisent les informations de paiement de leurs clients** (comme le numéro de carte de crédit et sa date de validité). En effet, ils deviennent ainsi la cible privilégiée des hackers qui tenteront d'accéder à ces données.

Remèdes:

Il faut séparer:

- la commande (le contenu du caddie virtuel – OI : Order Information) qui est toujours envoyée au serveur marchand.
- les informations de paiement (PI : Payment Information) qui sont envoyées à un serveur de l'institution financière concernée.

Par exemple: 3D-Secure



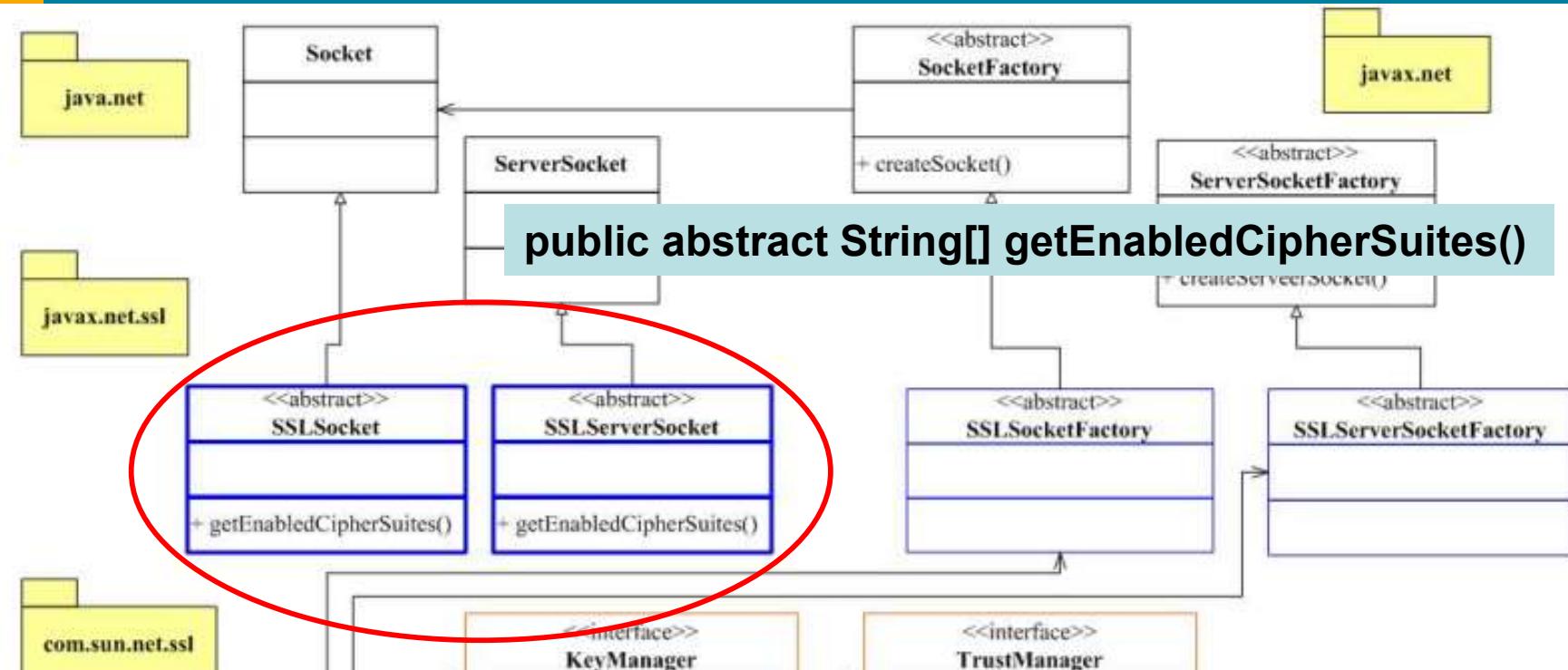
VII. IMPLÉMENTATION

SSLSocket

1. Introduction

- Le but de SSI est d'assurer un niveau de sécurité lors d'échanges d'informations entre un client et un serveur.
- Ces derniers pour communiquer utilisent des sockets, il suffit dès lors de remplacer ces sockets par des sockets SSL (SocketSSL et SSLServerSocket).
- Il ne faut donc rien changer à la programmation classique des clients/serveurs déjà programmés.
- Il faudra donc programmer une dizaine de lignes au début de votre application, rien d'autre n'est à changer (intégration simplement des lignes de configuration propre à SSL dans votre code entre les parties applicative et transport).

2. Les packages Java: les sockets SSL

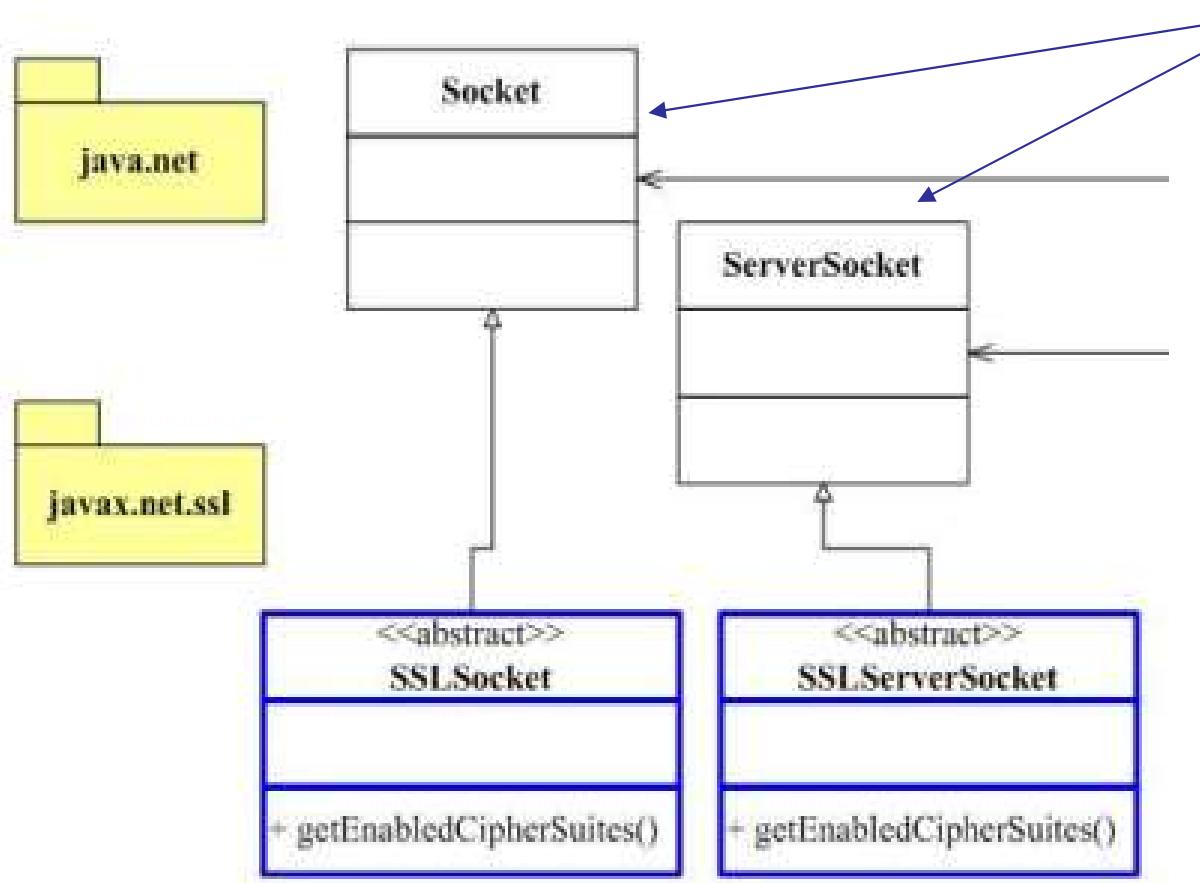


Ces 2 classes Interviennent dans le Handshake vu précédemment:
La méthode « **getEnabledCipherSuite()** » donne les algorithmes
qui ont été retenus lors du handshake.



Ces 2 classes sont **abstraites** => le retour des **classes factories**:
l'implémentation de ces classes est donc fournie par des Providers

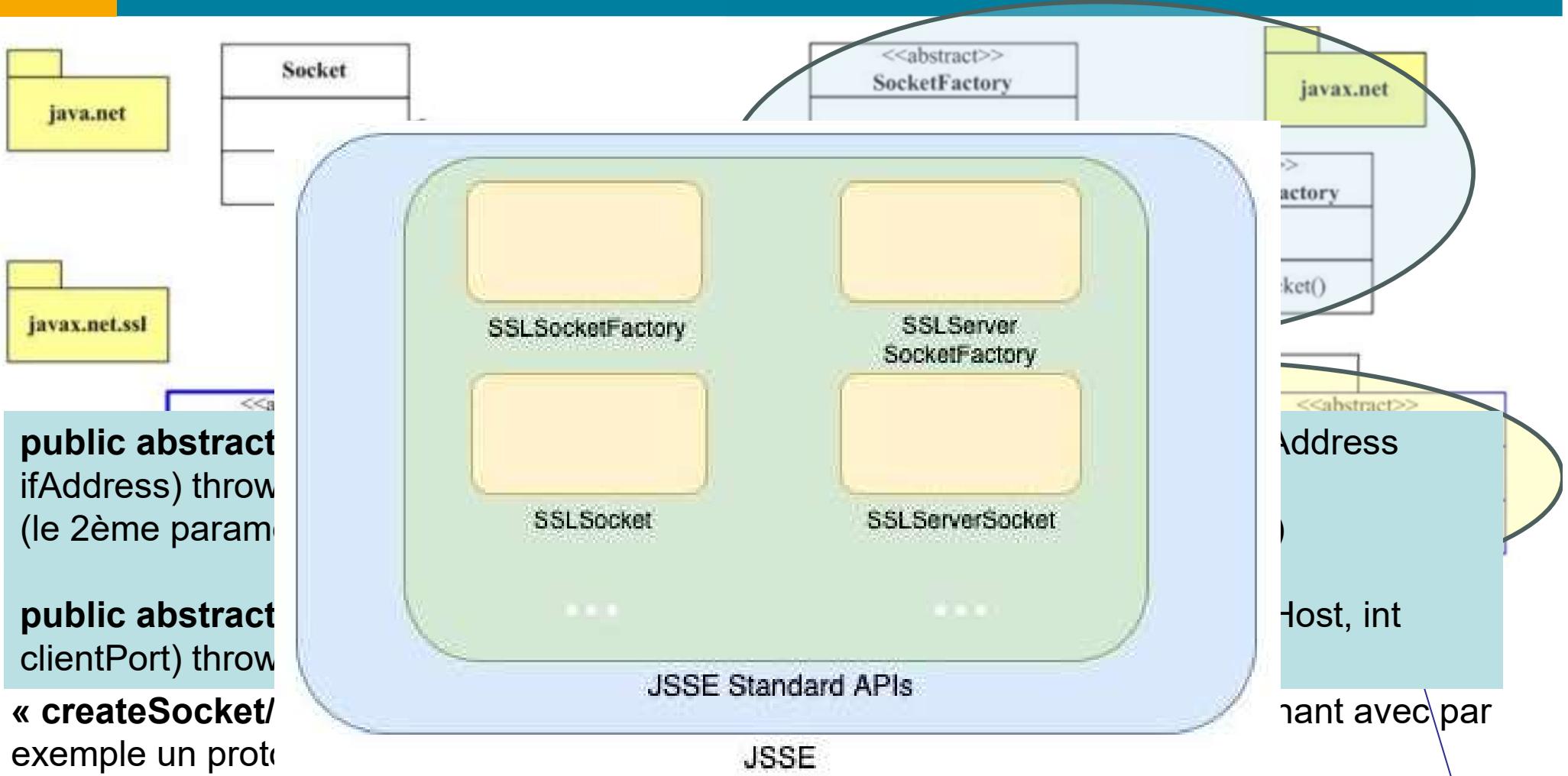
3. les sockets SSL: classes abstraites



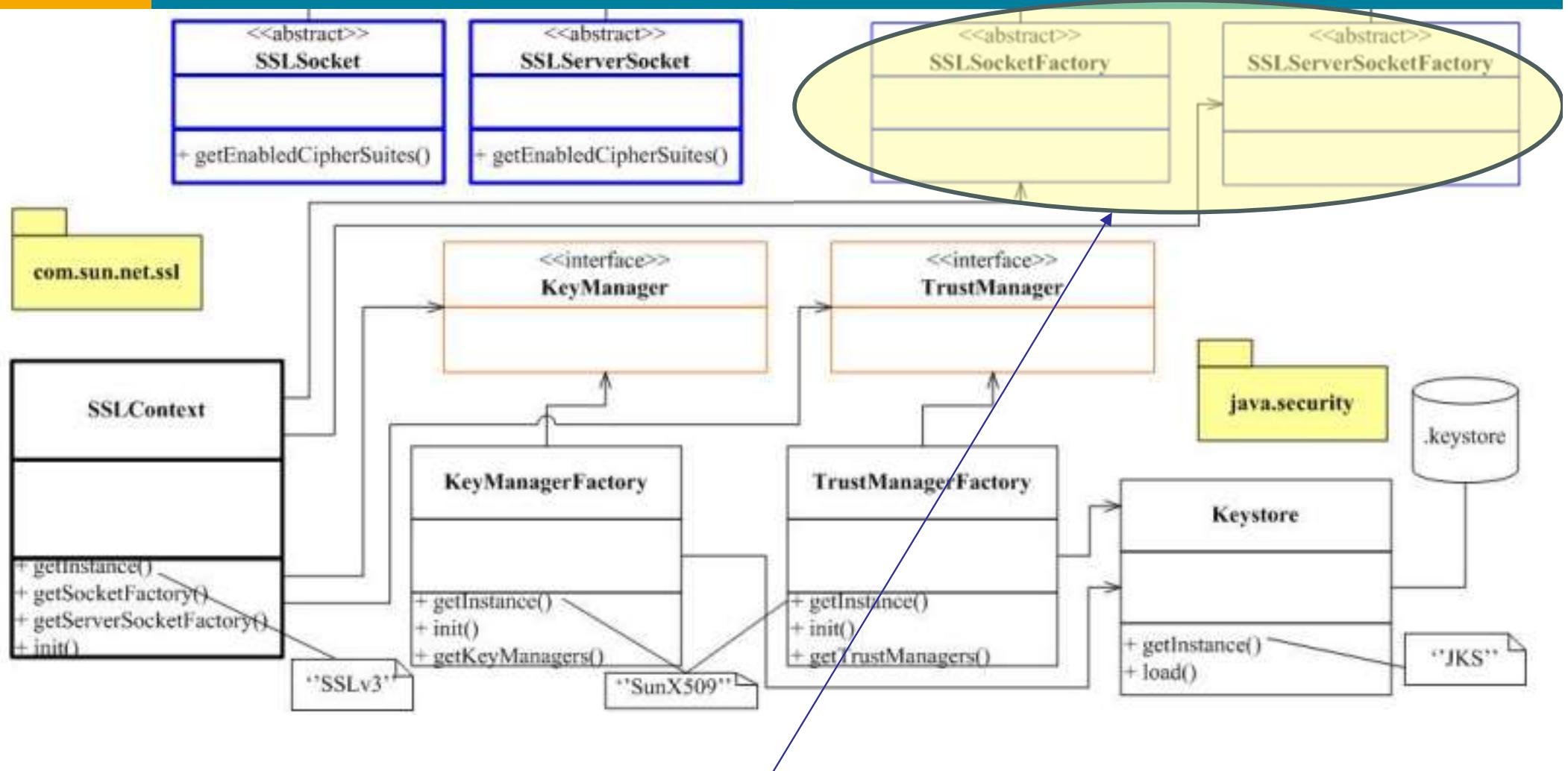
Les classes abstraites héritent des classes normales de Socket du package `java.net`:

- Ces sockets peuvent être
 - Instanciées
 - Paramétrées.
- On pourrait désirer en fabriquer des nouvelles en confectionnant par exemple: notre propre protocole

4. les sockets SSL: classes factories



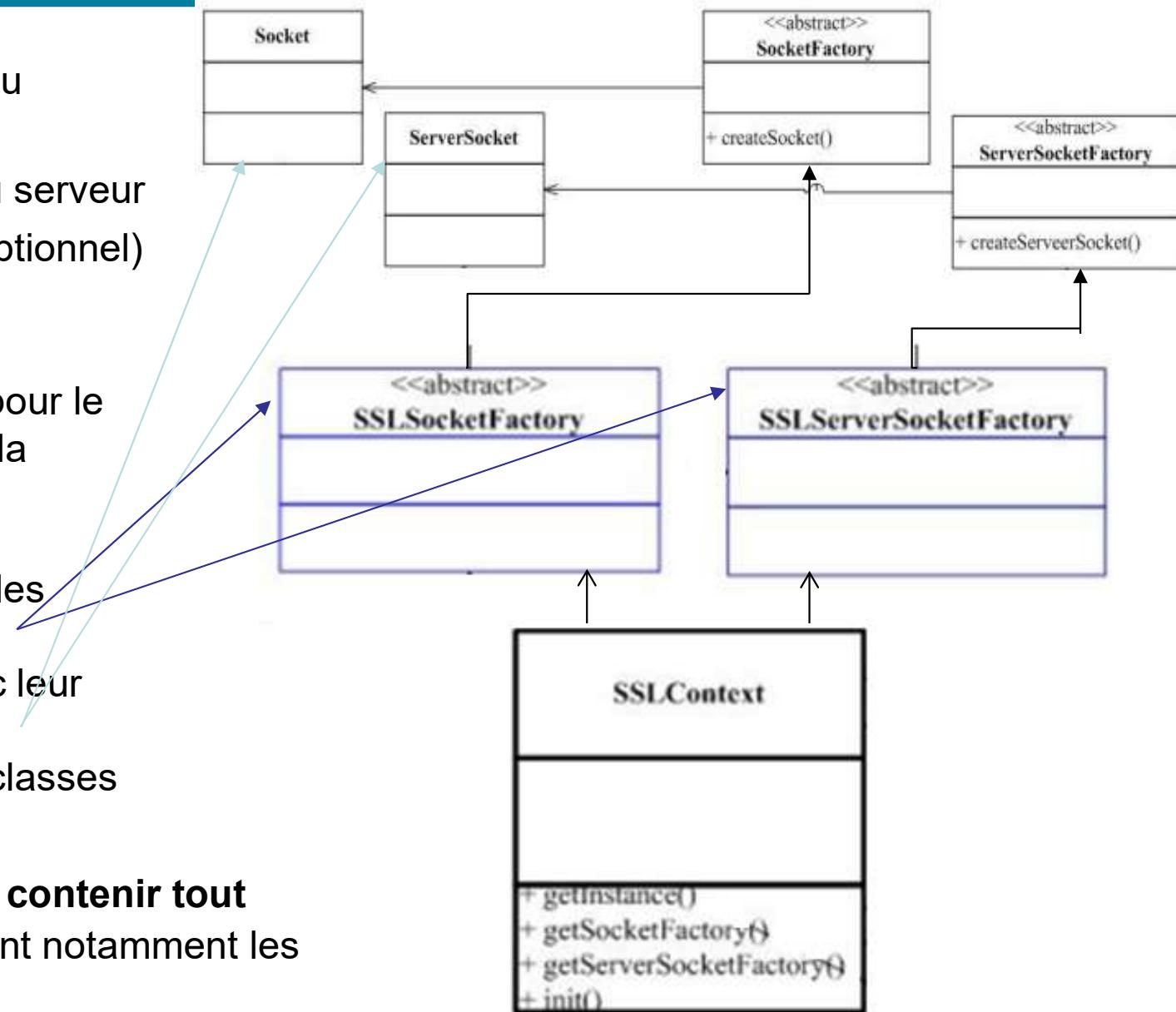
5. les sockets SSL: SSLContext



On ne peut pas instancier ces deux classes (abstraites). Il faut en fait avant toute chose se préoccuper du **Handshake**

6. les sockets SSL: SSLContext (suite)

- Pour faire le Handshake au minimum, il faut:
 - Au moins le certificat du serveur
 - Le certificat du client (optionnel)
 - Vérifier les signatures
- Tout ce qui est nécessaire pour le Handshake est fourni dans la classe **SSLContext**.
- Cette dernière fournit alors les deux classes factories qui fournissent à leur tour (avec leur méthode `createSocket()` / `createServerSocket()`) les classes sockets.
- **Le SSLContext doit aussi contenir tout résultat du Handshake** dont notamment les certificats



7. La classe SSLContext

- Cette classe SSLContext s'instancie par une nouvelle factory :

```
public static SSLContext getInstance (String protocol, Provider provider)  
throws NoSuchAlgorithmException, NoSuchProviderException
```

Le premier paramètre est par contre incontournable, il désigne le protocole et surtout la version utilisée.

Dans notre cas:

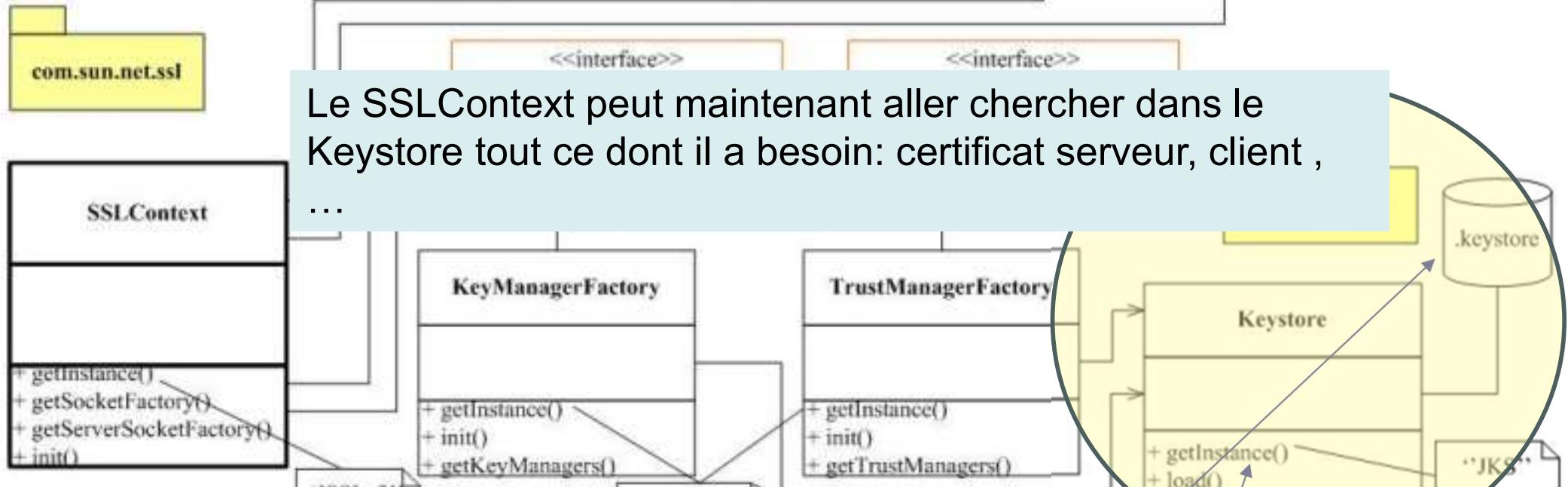
```
SSLContext SsIC = null;  
SsIC = SSLContext.getInstance("SSLv3"); ("TLSv1.2"); ou ("TLSv1.3")
```

- On a besoin des certificats (au moins celui du serveur avec des clés) il faut donc un **KeyStore** (magasin de clés composé de plusieurs « compartiments » comme les:
 - **trusted certificate entries** qui contiennent juste un certificat
 - **key entries** qui contiennent une paire de clés (privée/publique) dans un certificat.

Il faut donc un **objet KeyStore** (en mémoire) qui a besoin d'un **fichier « keystore »** construit avec un outil par exemple comme: **Keytool ou KeyToolGui**.

On n'instancie pas l'objet Keystore, on le charge en mémoire avec un getInstance() car il peut exister plusieurs types de keystore (JKS (java), PK12(windows), bouncy castle).

8. La classe Keystore



Remarque: il existe aussi des méthodes de type **setXXXX** pour « charger » cet objet.:

```
private static final String[] protocols = new String[]{"TLSv1.3"};
private static final String[] cipher_suites = new String[]{"TLS_AES_128_GCM_SHA256"};
```

```
serverSocket.setEnabledProtocols(protocols)
serverSocket.setEnabledCipherSuites(cipher_suites)
```

load() est très bien !

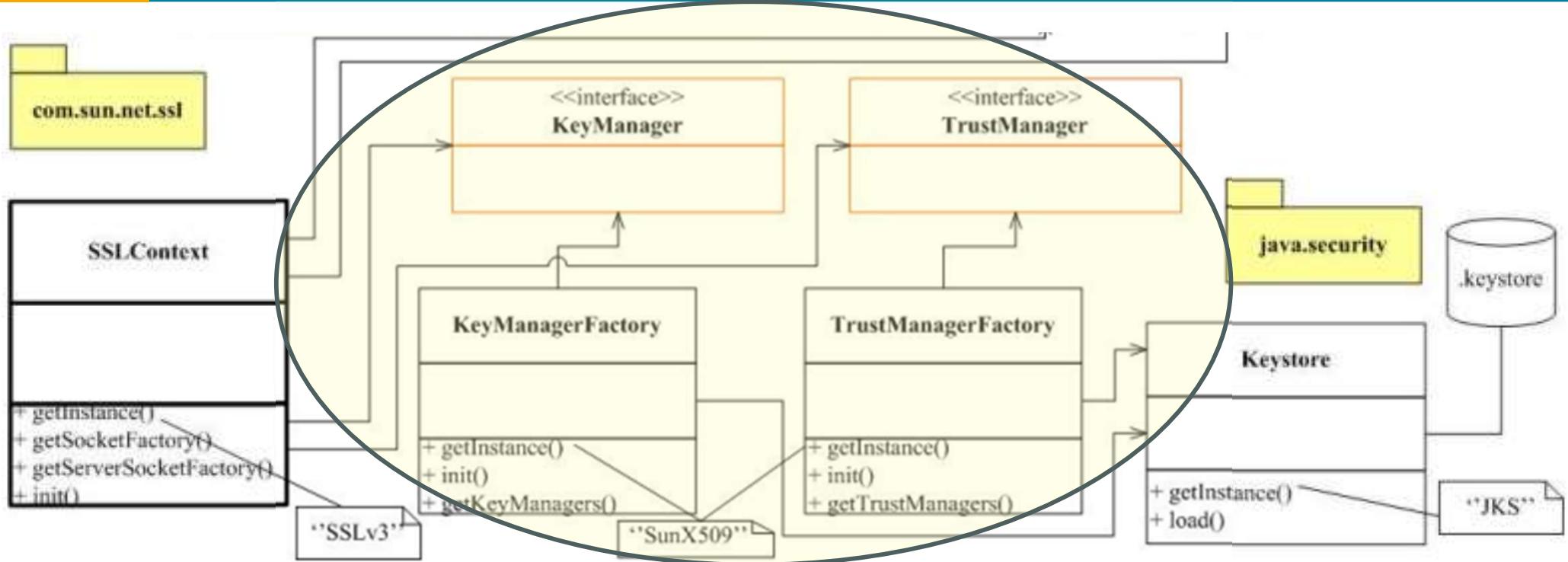
9. Programmation: objet Keystore

- **public final void load(InputStream stream, char[] password)** throws IOException, NoSuchAlgorithmException, CertificateException

Cette méthode vérifiera l'intégrité du keystore avant de charger effectivement l'objet en mémoire (ce qui explique le 3ème type d'exception potentiel).

- KeyStore ServerKs = KeyStore.getInstance("JKS");
- String FICHIER_KEYSTORE = "c:\\makecert\\serveur_keystore";
- char[] PASSWD_KEYSTORE = "beaugosser".toCharArray();
- FileInputStream ServerFK = new FileInputStream (FICHIER_KEYSTORE);
- **ServerKs.load(ServerFK, PASSWD_KEYSTORE);**

10. KeyManager/TrustManager: généricité

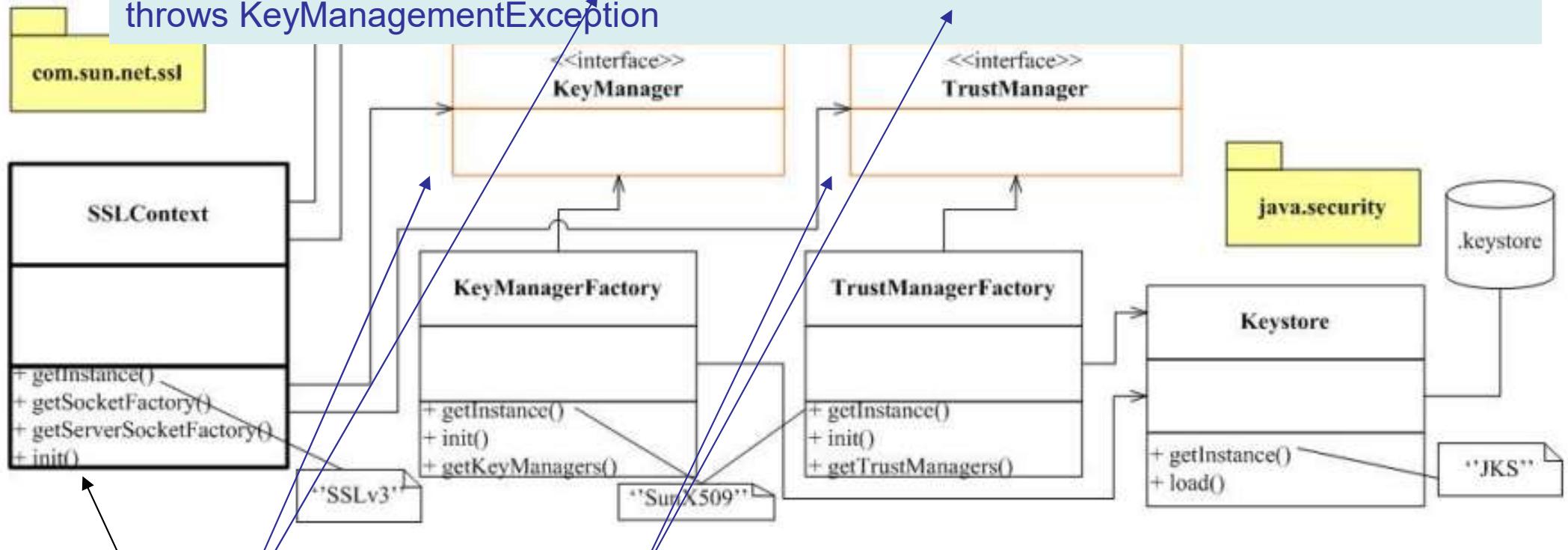


La partie centrale permet de rendre générique l'opération de la **méthode init()** du **SSLContext** car tout ce dont a besoin cette classe ne se trouve pas nécessairement dans un **KeyStore**. Les informations pourraient être stockées sur :

- un serveur (un fournisseur de certificats, un serveur d'authentification)
- Une carte à puce (carte d'identité belge par exemple, attention les clés privées ne sont pas extractables).

11. KeyManager/TrustManager: leur contenu

```
public final void init (KeyManager[] km, TrustManager[] tm, SecureRandom random)  
throws KeyManagementException
```



- La méthode **init()** de la classe **SSLContext** demande comme paramètres **deux tableaux**: un de **KeyManager[]** et un de **TrustManager[]**. Comme ce sont des **interfaces**, ça peut être des **choses de natures différentes**.
 - KeyManager** = **un couple de clés privée/publique** (par exemple pour le plan B). Correspond à une key entry.
 - TrustManager** = moyen pour récupérer une **clé publique certifiée** (dans la version la plus simple, c'est **un certificat**)

12. KeyManager/TrustManager: obtenir les tableaux

- On obtient ces tableaux d'objets au moyen de **factories** (vu que ce qu'on attend n'est connu que sous forme d'interface) appartenant au même package:
 - **KeyManagerFactory**
 - **TrustManagerFactory**
- On n'instancie pas directement les 2 classes sitées pci-dessus. On va obtenir des instances de ces types grâce à la méthode: **getInstance()** car il existe plusieurs moyens de représenter les concepts de clés publiques/privées. Dans notre cas « **SunX509** »
 - `public static final KeyManagerFactory getInstance(String algorithm, String provider)`
throws NoSuchAlgorithmException, NoSuchProviderException
 - `public static final TrustManagerFactory getInstance(String algorithm, String provider)`
throws NoSuchAlgorithmException, NoSuchProviderException

13. KeyManager/TrustManager: getInstance()

Utilisation du package: **javax.net.ssl**

```
SSLContext SslC = SSLContext.getInstance("SSLv3");
```

```
KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
char[] PASSWD_KEY = "sexyser".toCharArray();
kmf.init(ServerKs, PASSWD_KEY);
```

```
TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
tmf.init(ServerKs);
```

```
SslC.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
```

Objet SSLContext

14. La socket SSL

Côté serveur:

```
SSLServerSocketFactory SslSFac= SsIC.getServerSocketFactory();
SslSSocket = (SSLServerSocket) SslSFac.createServerSocket(6000);
// Version non sécurisée
// CSocket = SSocket.accept();
// *** Version sécurisée ***
SslSocket = (SSLSocket)SslSSocket.accept();
```

Num du port

Côté client:

```
SSLSocketFactory SslSFac= SsIC.getSocketFactory();
Socket SslSocket = (SSLSocket) SslSFac.createSocket("nomHôte", 6000);
```

NomHôte si résolution DNS
possible ou Adresse IP de l'hôte

15. Les certificats: deux keystores (1 pour le serveur/un pour le client)

Côté serveur:

Type de fichier de clés : jks

Fournisseur de fichier de clés : SUN

Certificat qui permet de vérifier les signatures

Votre fichier de clés d'accès contient 2 entrées

christophe, 14 juil. 2023, trustedCertEntry,

Empreinte du certificat (SHA1) : 38:27:CD:7D:AC:7A:B3:69:D6:21:3D:63:43:21:1B:4F:F9:55:DF:2C

serveurssl, 14 juil. 2023, PrivateKeyEntry,

Empreinte du certificat (SHA1) : 6F:7A:2A:88:F4:EA:02:12:54:DE:AD:AD:5B:38:38:4B:94:B7:99:C9

Côté client:

Type de fichier de clés : jks

Fournisseur de fichier de clés : SUN

Contient la paire de clés publique/privée
pour réaliser les signatures

Votre fichier de clés d'accès contient 2 entrées

christophe, 14 juil. 2023, trustedCertEntry,

Empreinte du certificat (SHA1) : 38:27:CD:7D:AC:7A:B3:69:D6:21:3D:63:43:21:1B:4F:F9:55:DF:2C

clientssl, 14 juil. 2023, PrivateKeyEntry,

Empreinte du certificat (SHA1) : E8:C3:C6:CD:51:C4:96:09:3E:9B:86:5F:86:10:97:86:2C:22:E5:0A

15. CLI (keytool) suite 1

C:\CertSSL>keytool

Outil de gestion de certificats et de clés

Commandes :

- certreq Génère une demande de certificat
- changealias Modifie l'alias d'une entrée
- delete Supprime une entrée
- exportcert Exporte le certificat
- genkeypair Génère une paire de clés
- genseckeypair Génère une clé secrète
- gencert Génère le certificat à partir d'une demande de certificat
- importcert Importe un certificat ou une chaîne de certificat
- importpass Importe un mot de passe
- importkeystore Importe une entrée ou la totalité des entrées depuis un autre fichier de clés
- keypasswd Modifie le mot de passe de clé d'une entrée
- list Répertorie les entrées d'un fichier de clés**
- printcert Imprime le contenu d'un certificat
- printcertreq Imprime le contenu d'une demande de certificat
- printcrl Imprime le contenu d'un fichier de liste des certificats révoqués (CRL)
- storepasswd Modifie le mot de passe de banque d'un fichier de clés

Utiliser "keytool -command_name -help" pour la syntaxe de command_name

15. CLI (keytool) suite 2

```
C:\CertSSL>keytool -list /?
```

```
Option non admise : /?
```

```
keytool -list [OPTION]...
```

Répertorie les entrées d'un fichier de clés

Options :

-rfc	sortie au style RFC
-alias <alias>	nom d'alias de l'entrée à traiter
-keystore <keystore>	nom du fichier de clés
-storepass <arg>	mot de passe du fichier de clés
-storetype <storetype>	type du fichier de clés
-providername <providername>	nom du fournisseur
-providerclass <providerclass>	nom de la classe de fournisseur
-providerarg <arg>	argument du fournisseur
-providerpath <pathlist>	variable d'environnement CLASSPATH du fournisseur
-v	sortie en mode verbose
-protected	mot de passe via mécanisme protégé

Utiliser "keytool -help" pour toutes les commandes disponibles

15. CLI (keytool) suite 3

```
C:\CertSSL>keytool -keystore serveurKeyStore.jks -list
```

Entrez le mot de passe du fichier de clés :

Type de fichier de clés : jks

Fournisseur de fichier de clés : SUN

Votre fichier de clés d'accès contient 2 entrées

```
christophe, 14 juil. 2023, trustedCertEntry,
```

```
Empreinte du certificat (SHA1) : 38:27:CD:7D:AC:7A:B3:69:D6:21:3D:63:43:21:1B:4F:F9:55:DF:2C
```

```
serveurssl, 14 juil. 2023, PrivateKeyEntry,
```

```
Empreinte du certificat (SHA1) : 6F:7A:2A:88:F4:EA:02:12:54:DE:AD:AD:5B:38:38:4B:94:B7:99:C9
```

Warning:

Le fichier de clés JKS utilise un format propriétaire. Il est recommandé de migrer vers PKCS12, qui est un format standard de l'industrie en utilisant

```
"keytool -importkeystore -srckeystore serveurKeyStore.jks -destkeystore serveurKeyStore.jks  
-deststoretype pkcs12".
```

15. CLI (keytool) suite 4

```
C:\CertSSL>keytool -keystore serveurKeyStore.jks -list -v
```

Entrez le mot de passe du fichier de clés :

Type de fichier de clés : jks

Fournisseur de fichier de clés : SUN

Votre fichier de clés d'accès contient 2 entrées

Nom d'alias : christophe

Date de création : 14 juil. 2023

Type d'entrée : trustedCertEntry

Propriétaire : CN=christophe, OU=HEPL, O=moi, L=seraing, ST=liege, C=BE

Emetteur : CN=christophe, OU=HEPL, O=moi, L=seraing, ST=liege, C=BE

Numéro de série : 64b10ba5

Valide du : Fri Jul 14 10:47:33 CEST 2023 au : Thu Jul 14 10:47:33 CEST 2033

Empreintes du certificat :

MD5 : 08:A1:FC:D9:EB:49:40:81:3E:3F:0E:C8:D4:76:27:ED

SHA1 : 38:27:CD:7D:AC:7A:B3:69:D6:21:3D:63:43:21:1B:4F:F9:55:DF:2C

SHA256 :

BD:57:B1:46:34:E3:EC:CB:C7:C0:9A:A7:6D:8C:91:87:C6:54:D6:59:59:40:E3:AA:8E:7F:1B:75:16:2E:82:5A

Nom de l'algorithme de signature : SHA256withRSA

Algorithme de clé publique du sujet : Clé RSA 2048 bits

Version : 3

Extensions :

#1: ObjectId: 2.5.29.19 Criticality=false

BasicConstraints:[

 CA:true

 PathLen:2147483647

]

15. CLI (keytool) suite 5

```
*****
```

Nom d'alias : serveurssl

Date de création : 14 juil. 2023

Type d'entrée : PrivateKeyEntry

Longueur de chaîne du certificat : 1

Certificat[1]:

Propriétaire : CN=serveurSSL, OU=HEPL, O=moi, L=seraing, ST=liege, C=BE

Emetteur : CN=christophe, OU=HEPL, O=moi, L=seraing, ST=liege, C=BE

Numéro de série : 64b10f32

Valide du : Fri Jul 14 11:02:42 CEST 2023 au : Mon Jul 14 11:02:42 CEST 2031

Empreintes du certificat :

MD5 : CE:26:23:DB:F9:5B:98:68:9E:CC:04:F5:40:B8:4C:06

SHA1 : 6F:7A:2A:88:F4:EA:02:12:54:DE:AD:AD:5B:38:38:4B:94:B7:99:C9

SHA256 :

B8:8F:F9:FE:77:6C:99:9C:5A:E5:50:FB:62:2D:33:E7:BF:8D:3C:DD:5D:C5:6F:F8:A7:23:7F:5C:F0:98:37:28

Nom de l'algorithme de signature : SHA256withRSA

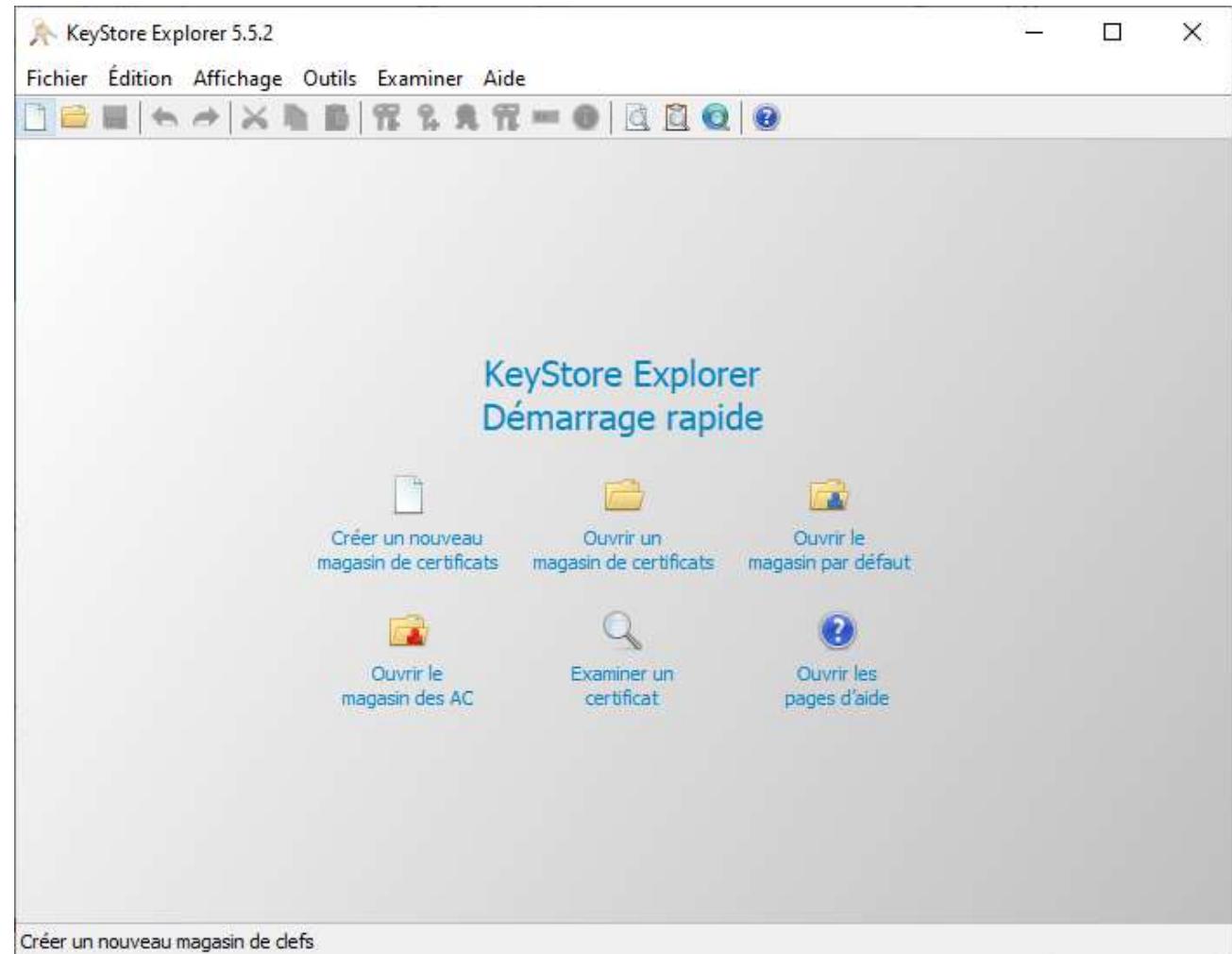
Algorithme de clé publique du sujet : Clé RSA 2048 bits

Version : 3

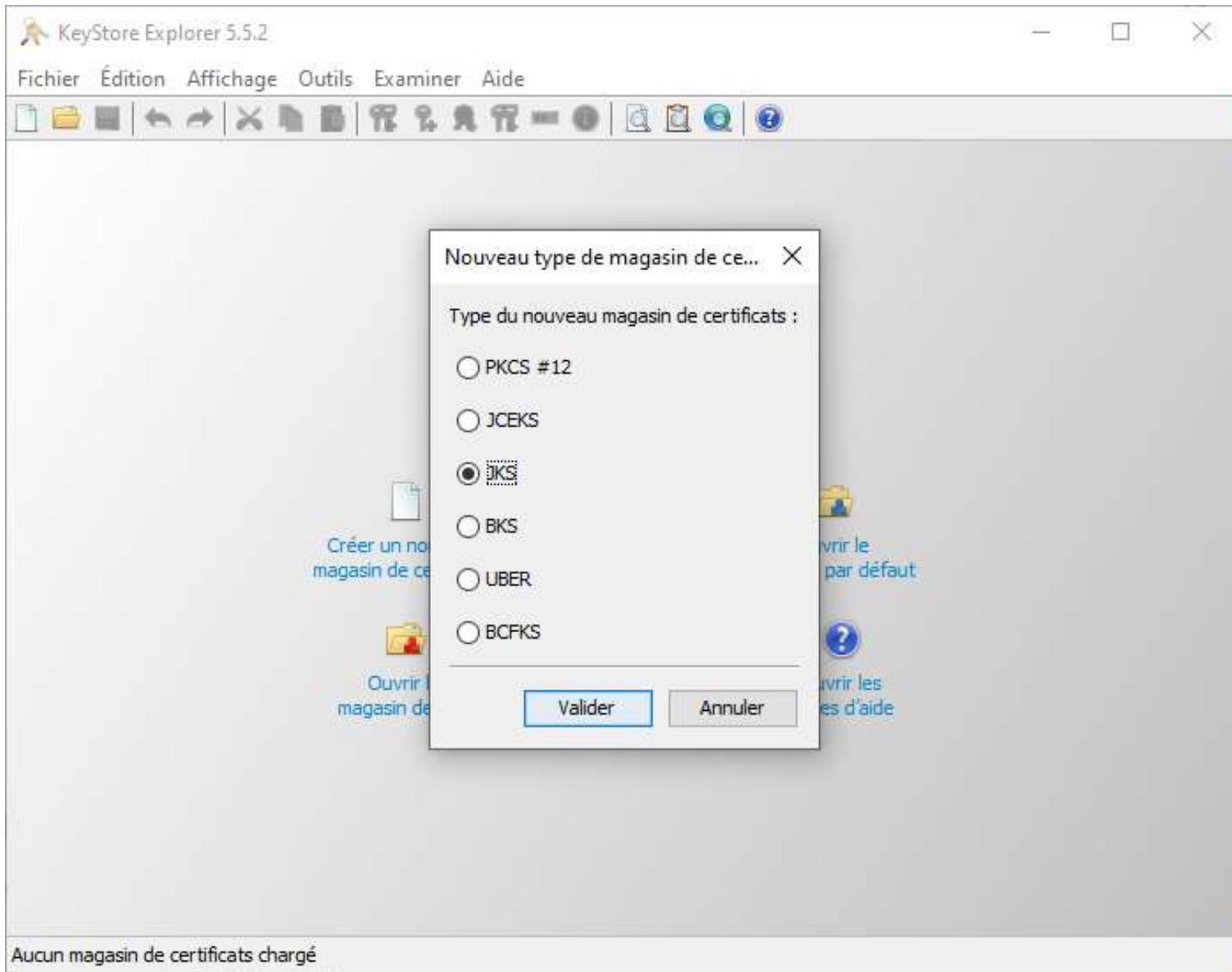
```
*****
```

```
*****
```

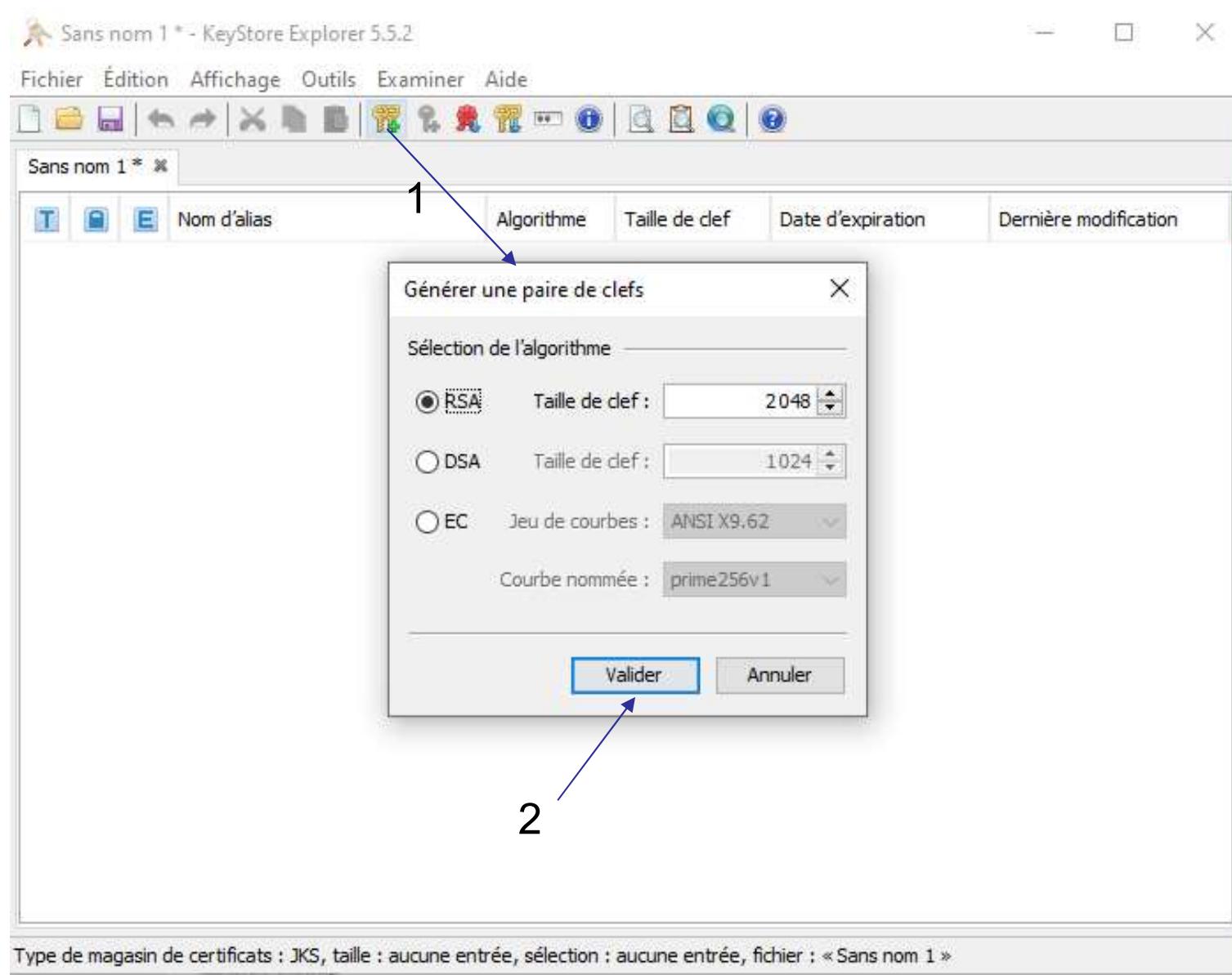
17.1. Outil KeyStore Explorer



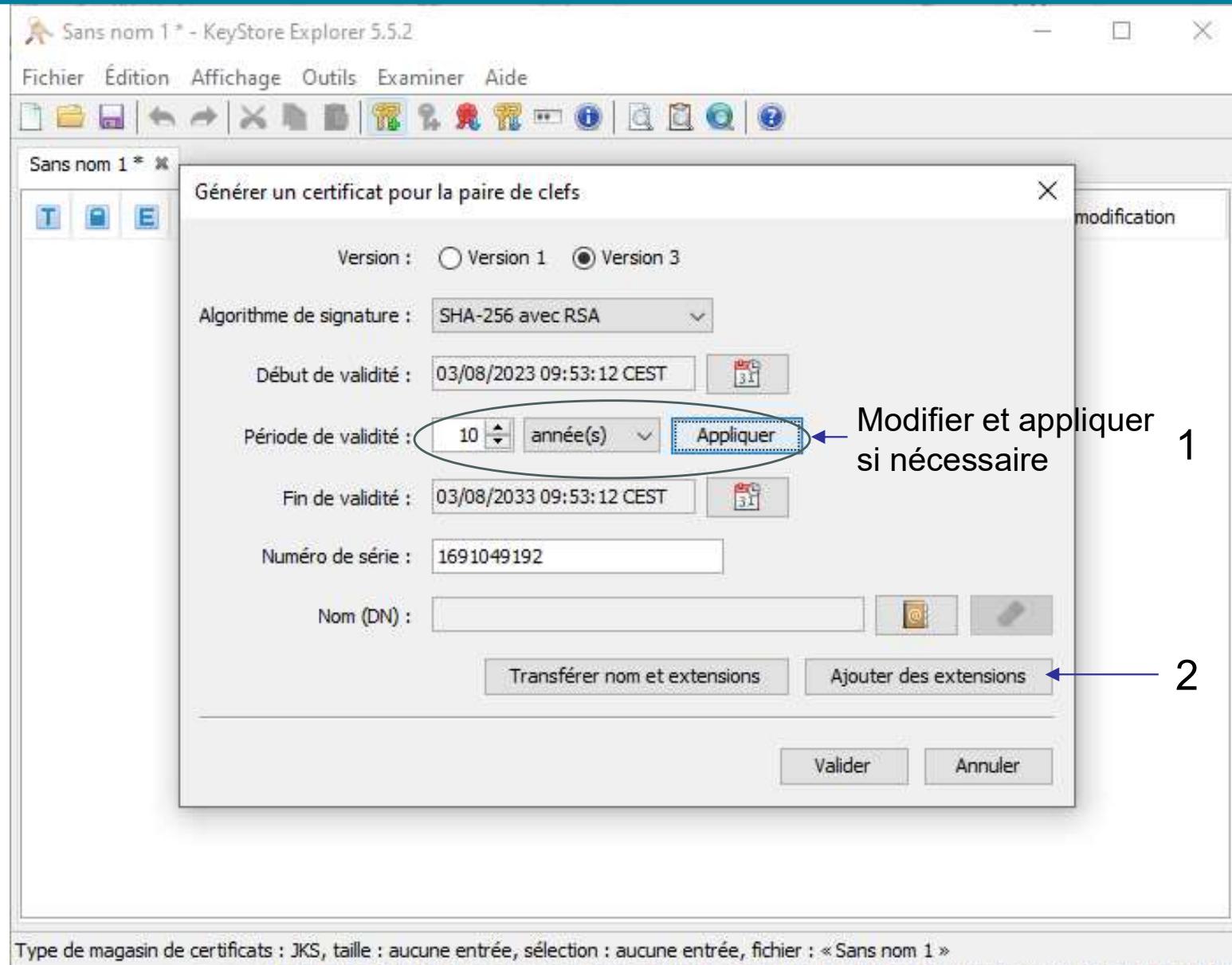
17.1. Outil KeyStore Explorer: choix du type de magasin (keystore)



17.2. Outil KeyStore Explorer: Création d'un CA (non officiel)



17.3. Outil KeyStore Explorer: Création d'un CA (non officiel)



VIII. ANALYSE DES ÉCHANGES

Wireshark

1. Echanges complets (SSLv3 -> TLS 1.2)

No.	Source	Time	Destination	Protocol	Length	Trans	Info
16	127.0.0.1	25.805804	127.0.0.1	TCP	56 ✓		55206 → 55555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
19	127.0.0.1	25.805958	127.0.0.1	TCP	56 ✓		55555 → 55206 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
21	127.0.0.1	25.806473	127.0.0.1	TCP	44 ✓		55206 → 55555 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
22	127.0.0.1	25.912983	127.0.0.1	TLSv1	163 ✓		Client Hello ←
23	127.0.0.1	25.913026	127.0.0.1	TCP	44 ✓		55555 → 55206 [ACK] Seq=1 Ack=120 Win=2619648 Len=0
24	127.0.0.1	25.950228	127.0.0.1	TLSv1	1313 ✓		Server Hello, Certificate, Server Key Exchange, Server Hello Done ←
25	127.0.0.1	25.950299	127.0.0.1	TCP	44 ✓		55206 → 55555 [ACK] Seq=120 Ack=1270 Win=2618368 Len=0
26	127.0.0.1	25.971085	127.0.0.1	TLSv1	119 ✓		Client Key Exchange
27	127.0.0.1	25.971155	127.0.0.1	TCP	44 ✓		55555 → 55206 [ACK] Seq=1270 Ack=195 Win=2619648 Len=0
28	127.0.0.1	25.988013	127.0.0.1	TLSv1	50 ✓		Change Cipher Spec
29	127.0.0.1	25.988077	127.0.0.1	TCP	44 ✓		55555 → 55206 [ACK] Seq=1270 Ack=201 Win=2619648 Len=0
30	127.0.0.1	26.037846	127.0.0.1	TLSv1	97 ✓		Encrypted Handshake Message
31	127.0.0.1	26.037916	127.0.0.1	TCP	44 ✓		55555 → 55206 [ACK] Seq=1270 Ack=254 Win=2619648 Len=0
32	127.0.0.1	26.040379	127.0.0.1	TLSv1	50 ✓		Change Cipher Spec
33	127.0.0.1	26.040437	127.0.0.1	TCP	44 ✓		55206 → 55555 [ACK] Seq=254 Ack=1276 Win=2618368 Len=0
34	127.0.0.1	26.040895	127.0.0.1	TLSv1	97 ✓		Encrypted Handshake Message
35	127.0.0.1	26.040936	127.0.0.1	TCP	44 ✓		55206 → 55555 [ACK] Seq=254 Ack=1329 Win=2618368 Len=0
36	127.0.0.1	26.041201	127.0.0.1	TLSv1	113 ✓		Application Data
37	127.0.0.1	26.041233	127.0.0.1	TCP	44 ✓		55206 → 55555 [ACK] Seq=254 Ack=1398 Win=2618368 Len=0
38	127.0.0.1	26.042423	127.0.0.1	TLSv1	145 ✓		Application Data
39	127.0.0.1	26.042468	127.0.0.1	TCP	44 ✓		55555 → 55206 [ACK] Seq=1398 Ack=355 Win=2619392 Len=0
40	127.0.0.1	26.043482	127.0.0.1	TLSv1	150 ✓		Application Data, Application Data
41	127.0.0.1	26.043545	127.0.0.1	TCP	44 ✓		55206 → 55555 [ACK] Seq=355 Ack=1504 Win=2618112 Len=0
42	127.0.0.1	26.044537	127.0.0.1	TLSv1	150 ✓		Application Data, Application Data
43	127.0.0.1	26.044578	127.0.0.1	TCP	44 ✓		55555 → 55206 [ACK] Seq=1504 Ack=461 Win=2619392 Len=0
44	127.0.0.1	26.044708	127.0.0.1	TLSv1	81 ✓		Encrypted Alert
45	127.0.0.1	26.044734	127.0.0.1	TCP	44 ✓		55555 → 55206 [ACK] Seq=1504 Ack=498 Win=2619392 Len=0
46	127.0.0.1	26.045283	127.0.0.1	TLSv1	81 ✓		Encrypted Alert
47	127.0.0.1	26.045322	127.0.0.1	TCP	44 ✓		55206 → 55555 [ACK] Seq=498 Ack=1541 Win=2618112 Len=0
48	127.0.0.1	26.045568	127.0.0.1	TCP	44 ✓		55206 → 55555 [FIN, ACK] Seq=498 Ack=1541 Win=2618112 Len=0
49	127.0.0.1	26.045606	127.0.0.1	TCP	44 ✓		55555 → 55206 [ACK] Seq=1541 Ack=499 Win=2619392 Len=0
50	127.0.0.1	26.045657	127.0.0.1	TCP	44 ✓		55206 → 55555 [RST, ACK] Seq=499 Ack=1541 Win=0 Len=0

Phase 1

2. Phase 1 : Client Hello

```
> Frame 22: 163 bytes on wire (1304 bits), 163 bytes captured (1304 bits) on interface \Device\Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 55206, Dst Port: 55555, Seq: 1, Ack: 1, Len: 119
< Transport Layer Security
  < TLSv1 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 114
    < Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 110
      Version: TLS 1.0 (0x0301)
      < Random: 64b111a4774c258c59c80eec00db631f6a8990123eef9501f73eb38addd1966
        GMT Unix Time: Jul 14, 2023 11:13:08.000000000 Paris, Madrid (heure d'été)
        Random Bytes: 774c258c59c80eec00db631f6a8990123eef9501f73eb38addd1966
      Session ID Length: 0
      Cipher Suites Length: 28
      < Cipher Suites (14 suites)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
        Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
        Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
        Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
        Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
        Cipher Suite: TLS_DHE_DSS_WITH_AES_256_CBC_SHA (0x0038)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
        Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
        Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xc00e)
        Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
        Cipher Suite: TLS_DHE_DSS_WITH_AES_128_CBC_SHA (0x0032)
      Compression Methods Length: 1
    > Compression Methods (1 method)
```

3. Phase 1: Server Hello

```
> Frame 24: 1313 bytes on wire (10504 bits), 1313 bytes captured (10504 bits) on interface \Device\NPF_
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 55555, Dst Port: 55206, Seq: 1, Ack: 120, Len: 1269
▼ Transport Layer Security
  ▼ TLSv1 Record Layer: Handshake Protocol: Multiple Handshake Messages
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 1264
  ▼ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 81
    Version: TLS 1.0 (0x0301)
  ▼ Random: 64b111a50fd7d48769b08132c4682d4f90595dd3690a524bb118f4b95d073640
    GMT Unix Time: Jul 14, 2023 11:13:09.000000000 Paris, Madrid (heure d'été)
    Random Bytes: 0fd7d48769b08132c4682d4f90595dd3690a524bb118f4b95d073640
    Session ID Length: 32
    Session ID: 64b111a5b6335d7a0f35f7029f47c615bbca6e430121a83cd6f73a56f892fa03
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Compression Method: null (0)
    Extensions Length: 9
    > Extension: renegotiation_info (len=1)
    > Extension: extended_master_secret (len=0)
      [JA3S Fullstring: 769,49172,65281-23]
      [JA3S: 46df52e211001fa8da188599db66e0db]
    > Handshake Protocol: Certificate
    > Handshake Protocol: Server Key Exchange
    > Handshake Protocol: Server Hello Done
```

4. Phase 2: Certificat (S->C)

```
▼ Handshake Protocol: Certificate
  Handshake Type: Certificate (11)
  Length: 840
  Certificates Length: 837
  ▼ Certificates (837 bytes)
    Certificate Length: 834
    ▼ Certificate: 3082033e30820226a003020102020464b10f32300d06092a864886f70d01010b05003061... (id-at-commonName=serveurSSL,id-at-organizationalUnitName=HEPL,
      ▼ signedCertificate
        version: v3 (2)
        serialNumber: 0x64b10f32
      ▼ signature (sha256WithRSAEncryption)
        Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
      ▼ issuer: rdnSequence (0)
        ▼ rdnSequence: 6 items (id-at-commonName=christophe,id-at-organizationalUnitName=HEPL,id-at-organizationName=moi,id-at-localityName=seraing,id-
          > RDNSequence item: 1 item (id-at-countryName=BE)
          > RDNSequence item: 1 item (id-at-stateOrProvinceName=liege)
          > RDNSequence item: 1 item (id-at-localityName=seraing)
          > RDNSequence item: 1 item (id-at-organizationName=moi)
          > RDNSequence item: 1 item (id-at-organizationalUnitName=HEPL)
          > RDNSequence item: 1 item (id-at-commonName=christophe)
      ▼ validity
        ▼ notBefore: utcTime (0)
          utcTime: 2023-07-14 09:02:42 (UTC)
        ▼ notAfter: utcTime (0)
          utcTime: 2031-07-14 09:02:42 (UTC)
      ▼ subject: rdnSequence (0)
        ▼ rdnSequence: 6 items (id-at-commonName=serveurSSL,id-at-organizationalUnitName=HEPL,id-at-organizationName=moi,id-at-localityName=seraing,id-
      ▼ subjectPublicKeyInfo
        ▼ algorithm (rsaEncryption)
        ▼ subjectPublicKey: 3082010a0282010100be7f5ed1725872a3abe0bc0eff18559b2f1b64610b1b05d6c19718...
    ▼ algorithmIdentifier (sha256WithRSAEncryption)
      Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
      Padding: 0
      encrypted: 930f8b3135726145af20a62cc36397033077d738cf47e5893ce122546c91f95215184bf7...
```

5. Phase 2: Server Key Exchange + Server Hello Done

▼ Handshake Protocol: Server Key Exchange

Handshake Type: Server Key Exchange (12)

Length: 327

▼ EC Diffie-Hellman Server Params

Curve Type: named_curve (0x03)

Named Curve: secp256r1 (0x0017)

Pubkey Length: 65

Pubkey: 04506e1bdb5dc1a8fb905bf54ee7c8b2cfbf186b982cea5727647d714c9b303edb699d92...

Signature Length: 256

Signature: bb1e236909862517f0edaalc1cb1d724c4b12a871d0e926393b2e4c3be3f9d909cb52ea4...

▼ Handshake Protocol: Server Hello Done

Handshake Type: Server Hello Done (14)

Length: 0

6. Phase 3: Client Key Exchange

```
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 55206, Dst Port: 55555, Seq: 120, Ack: 1270, Len: 75
▼ Transport Layer Security
  ▼ TLSv1 Record Layer: Handshake Protocol: Client Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 70
  ▼ Handshake Protocol: Client Key Exchange
    Handshake Type: Client Key Exchange (16)
    Length: 66
  ▼ EC Diffie-Hellman Client Params
    Pubkey Length: 65
    Pubkey: 04b43e935119406b2ae018b141405c265f8d758031ab59c10bb0af38ce47b9765051ac90...
```

7. Phase 4: Change Cypher Spec (C -> S)

- > Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- > Transmission Control Protocol, Src Port: 55206, Dst Port: 55555, Seq: 195, Ack: 1270, Len: 6
- ▼ Transport Layer Security
 - ▼ TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
 - Content Type: Change Cipher Spec (20)
 - Version: TLS 1.0 (0x0301)
 - Length: 1
 - Change Cipher Spec Message
- > Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- > Transmission Control Protocol, Src Port: 55206, Dst Port: 55555, Seq: 195, Ack: 1270, Len: 6
- ▼ Transport Layer Security
 - ▼ TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
 - Content Type: Change Cipher Spec (20)
 - Version: TLS 1.0 (0x0301)
 - Length: 1
 - Change Cipher Spec Message

8. Phase 4: Change Cypher Spec (S -> C)

```
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 55555, Dst Port: 55206, Seq: 1270, Ack: 254, Len: 6
▼ Transport Layer Security
  ▼ TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.0 (0x0301)
    Length: 1
    Change Cipher Spec Message
```

```
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 55555, Dst Port: 55206, Seq: 1276, Ack: 254, Len: 53
▼ Transport Layer Security
  ▼ TLSv1 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 48
    Handshake Protocol: Encrypted Handshake Message
```

9. Echanges Data Crytés

```
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 55555, Dst Port: 55206, Seq: 1329, Ack: 254, Len: 69
▼ Transport Layer Security
  ▼ TLSv1 Record Layer: Application Data Protocol: Application Data
    Content Type: Application Data (23)
    Version: TLS 1.0 (0x0301)
    Length: 64
    Encrypted Application Data: 3310f0a6a8e7e87be48abd1b8afaa321c2538a079f66d6838d0b72c329aa2c58d4162a07...
```

0000	02 00 00 00 45 00 00 6d	c9 c5 40 00 80 06 00 00E..m ..@.....
0010	7f 00 00 01 7f 00 00 01	d9 03 d7 a6 35 11 5e fe 5.^.
0020	da 2d 2c 32 50 18 27 f9	f9 eb 00 00 17 03 01 00	.,,2P.'..
0030	40 33 10 f0 a6 a8 e7 e8	7b e4 8a bd 1b 8a fa a3	@3..... {.....
0040	21 c2 53 8a 07 9f 66 d6	83 8d 0b 72 c3 29 aa 2c	!..S...f.r..),,
0050	58 d4 16 2a 07 80 c9 d5	98 1d f1 70 52 e3 84 47	X...*..... pR..G
0060	ac ae 13 fe 44 bc 82 2d	5d ae af c0 50 54 50 7eD...]...PTP~
0070	79		y

10.1. Echange avec certificat client

No.	Source	Time	Destination	Protocol	Length	Trans	Info
44	127.0.0.1	12.943264	127.0.0.1	TCP	56	✓	56648 → 55555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
45	127.0.0.1	12.943423	127.0.0.1	TCP	56	✓	55555 → 56648 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
46	127.0.0.1	12.943748	127.0.0.1	TCP	44	✓	56648 → 55555 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
47	127.0.0.1	12.978780	127.0.0.1	TLSv1.2	315	✓	Client Hello
48	127.0.0.1	12.978846	127.0.0.1	TCP	44	✓	55555 → 56648 [ACK] Seq=1 Ack=272 Win=2619648 Len=0
49	127.0.0.1	13.044962	127.0.0.1	TLSv1.2	138	✓	Server Hello
50	127.0.0.1	13.045033	127.0.0.1	TCP	44	✓	56648 → 55555 [ACK] Seq=272 Ack=95 Win=2619648 Len=0
51	127.0.0.1	13.051501	127.0.0.1	TLSv1.2	894	✓	Certificate
52	127.0.0.1	13.051571	127.0.0.1	TCP	44	✓	56648 → 55555 [ACK] Seq=272 Ack=945 Win=2618624 Len=0
53	127.0.0.1	13.159312	127.0.0.1	TLSv1.2	349	✓	Server Key Exchange
54	127.0.0.1	13.159375	127.0.0.1	TCP	44	✓	56648 → 55555 [ACK] Seq=272 Ack=1250 Win=2618368 Len=0
55	127.0.0.1	13.163099	127.0.0.1	TLSv1.2	300	✓	Certificate Request
56	127.0.0.1	13.163157	127.0.0.1	TCP	44	✓	56648 → 55555 [ACK] Seq=272 Ack=1506 Win=2618112 Len=0
57	127.0.0.1	13.163912	127.0.0.1	TLSv1.2	53	✓	Server Hello Done
58	127.0.0.1	13.163962	127.0.0.1	TCP	44	✓	56648 → 55555 [ACK] Seq=272 Ack=1515 Win=2618112 Len=0
59	127.0.0.1	13.181031	127.0.0.1	TLSv1.2	895	✓	Certificate
60	127.0.0.1	13.181119	127.0.0.1	TCP	44	✓	55555 → 56648 [ACK] Seq=1515 Ack=1123 Win=2618880 Len=0
61	127.0.0.1	13.198035	127.0.0.1	TLSv1.2	86	✓	Client Key Exchange
62	127.0.0.1	13.198092	127.0.0.1	TCP	44	✓	55555 → 56648 [ACK] Seq=1515 Ack=1165 Win=2618880 Len=0
63	127.0.0.1	13.330453	127.0.0.1	TLSv1.2	313	✓	Certificate Verify
64	127.0.0.1	13.330520	127.0.0.1	TCP	44	✓	55555 → 56648 [ACK] Seq=1515 Ack=1434 Win=2618624 Len=0
65	127.0.0.1	13.341725	127.0.0.1	TLSv1.2	50	✓	Change Cipher Spec
66	127.0.0.1	13.341784	127.0.0.1	TCP	44	✓	55555 → 56648 [ACK] Seq=1515 Ack=1440 Win=2618624 Len=0
67	127.0.0.1	13.346848	127.0.0.1	TLSv1.2	89	✓	Encrypted Handshake Message
68	127.0.0.1	13.346920	127.0.0.1	TCP	44	✓	55555 → 56648 [ACK] Seq=1515 Ack=1485 Win=2618368 Len=0
69	127.0.0.1	13.365620	127.0.0.1	TLSv1.2	1991	✓	New Session Ticket
70	127.0.0.1	13.365696	127.0.0.1	TCP	44	✓	56648 → 55555 [ACK] Seq=1485 Ack=3462 Win=2616320 Len=0
71	127.0.0.1	13.367182	127.0.0.1	TLSv1.2	50	✓	Change Cipher Spec
72	127.0.0.1	13.367242	127.0.0.1	TCP	44	✓	56648 → 55555 [ACK] Seq=1485 Ack=3468 Win=2616320 Len=0
73	127.0.0.1	13.368183	127.0.0.1	TLSv1.2	89	✓	Encrypted Handshake Message
74	127.0.0.1	13.368238	127.0.0.1	TCP	44	✓	56648 → 55555 [ACK] Seq=1485 Ack=3513 Win=2616064 Len=0
75	127.0.0.1	13.368977	127.0.0.1	TLSv1.2	105	✓	Application Data
76	127.0.0.1	13.369019	127.0.0.1	TCP	44	✓	56648 → 55555 [ACK] Seq=1485 Ack=3574 Win=2616064 Len=0
77	127.0.0.1	13.372473	127.0.0.1	TLSv1.2	139	✓	Application Data

Phase 1

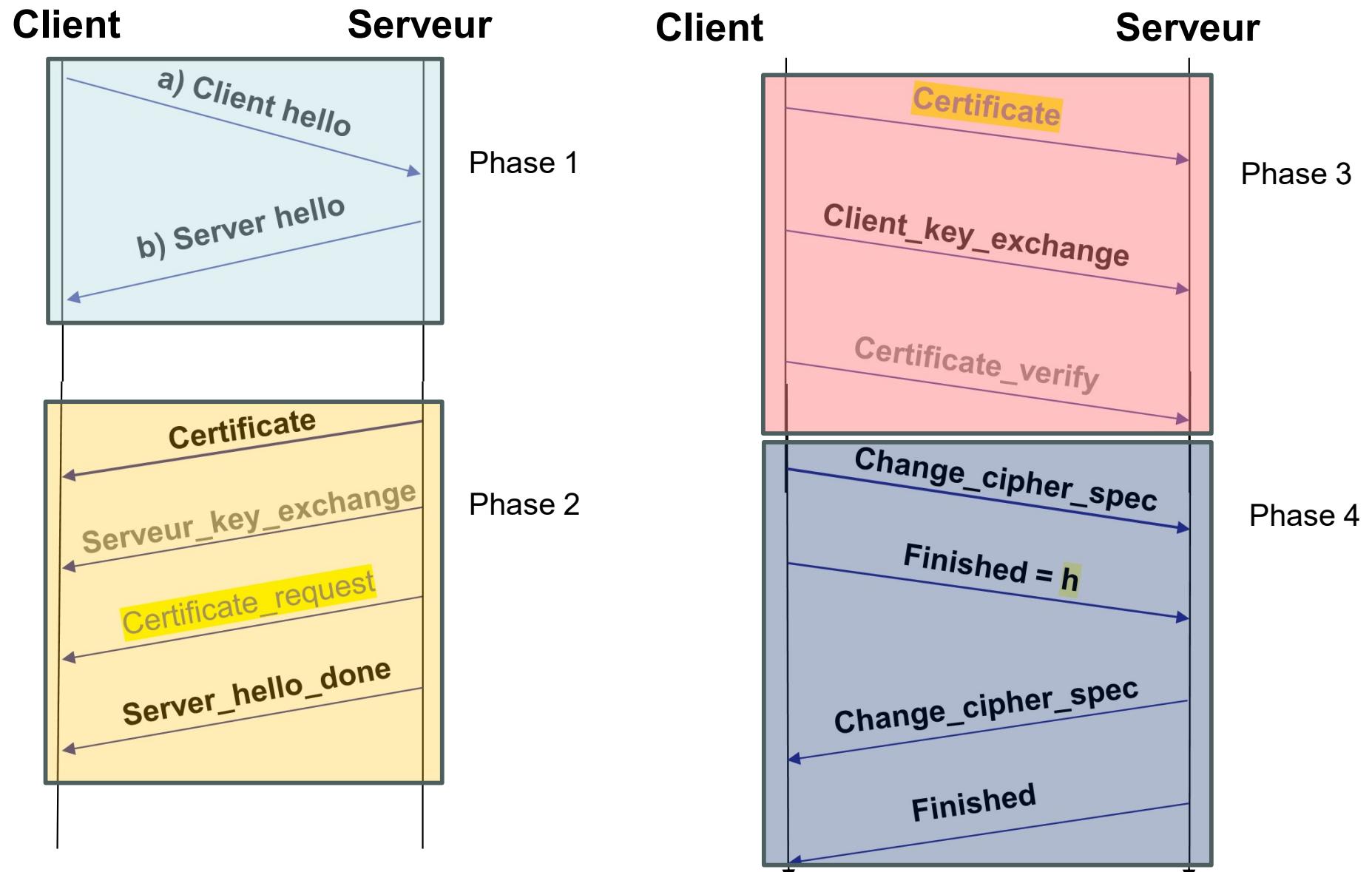
Phase 2

Phase 3

Phase 4

```
SSLServerSocketFactory SslSFac= SslC.getServerSocketFactory();
SslSSocket = (SSLServerSocket) SslSFac.createServerSocket(55555);
SslSSocket.setNeedClientAuth(true);
```

10.2. Echange avec certificat client



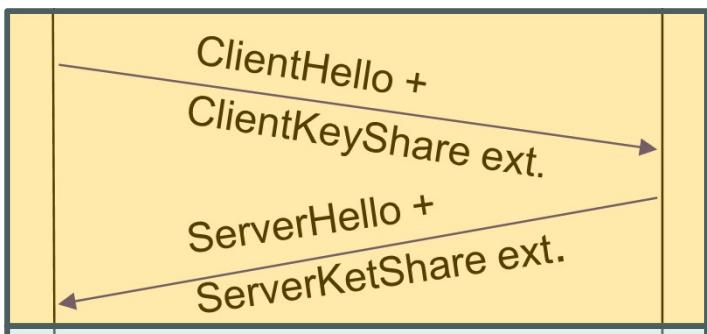
11.1. Echanges pour TLSv1.3 (introduction)

- RFC 8446
- Propose des solutions contre les vulnérabilités des versions précédentes.
Différents types d'attaque:
 - Jusqu'à la version TLS 1.0 le dernier bloc d'un paquet de données était utilisé comme vecteur d'initialisation (IV) en vue du chiffrement du paquet suivant => certain niveau de prédictibilité Si des mêmes données sont chiffrées avec la même clé, le résultat sera identique=> le but est de trouver la bonne combinaison pour obtenir ce même résultat avec le même IV. Ce dernier étant connu (**Beast attack**).
 - Attaque de type « padding oracle » avec l'implémentation du mode CBC (**LUCKY13, POODLE**)
 - Jusque TLS 1.1: afin d'éviter le mode CBC, on pouvait utiliser l'algorithme de chiffrement RC4. Ce dernier présente des failles du point vue de biais statistiques (**BIAISRC4**)
 - Dans TLS 1.2, problème de vulnérabilité de la fonction de hachage MD5 (**SLOTH**)
 - Faiblesse au niveau de certaines primitives asymétriques (**BLEICHENBACHER** sur RSA) (**DROWN, ROBOT**)
 - Les clés RSA (voire Diffie-Hellman) peuvent être de nos jours cassées plus facilement (absence de contrôle sur la taille des paramètres => groupes de grande taille, voir plus tard)
 - Vulnérabilités des mécanismes d'intégrité dans la négociation dues essentiellement par la non utilisation de la signature électronique.
 - Le protocole SSL-TLS étant assez complexe (point de vue implémentation, extensions, ...) mène à une probabilité de failles importantes

11.2. Echanges pour TLSv1.3: Modifications par rapport aux versions précédentes

Client

Serveur



Echanges
en clair

Partie
Crypté.

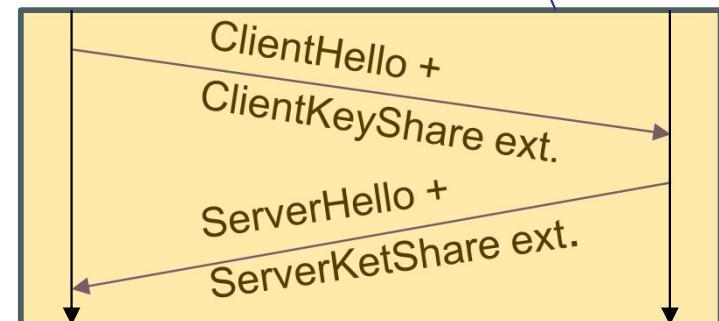
Cette partie (en bleu) est contenue dans le ServerHello

- Un nouvel « handshake » est utilisé : un seul échange (A/R) pour la négociation au lieu de deux précédemment.
- Ces échanges contiennent de suite des données chiffrées dès le premier paquet

Données chiffrées

11.2. Echanges pour TLSv1.3: Analyse des échanges

No.	Source	Time	Destination	Protocol	Length	Trans	Info
598	127.0.0.1	78.376185	127.0.0.1	TCP	56 ✓		55555 → 51692 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
599	127.0.0.1	78.376467	127.0.0.1	TCP	44 ✓		51692 → 55555 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
600	127.0.0.1	78.592902	127.0.0.1	TLSv1.3	472 ✓		Client Hello
601	127.0.0.1	78.592972	127.0.0.1	TCP	44 ✓		55555 → 51692 [ACK] Seq=1 Ack=429 Win=2619648 Len=0
602	127.0.0.1	78.688178	127.0.0.1	TLSv1.3	171 ✓		Server Hello
603	127.0.0.1	78.688248	127.0.0.1	TCP	44 ✓		51692 → 55555 [ACK] Seq=429 Ack=128 Win=2619648 Len=0
604	127.0.0.1	78.713889	127.0.0.1	TLSv1.3	50 ✓		Change Cipher Spec
605	127.0.0.1	78.713948	127.0.0.1	TCP	44 ✓		55555 → 51692 [ACK] Seq=128 Ack=435 Win=2619648 Len=0
606	127.0.0.1	78.743666	127.0.0.1	TLSv1.3	50 ✓		Change Cipher Spec
607	127.0.0.1	78.743738	127.0.0.1	TCP	44 ✓		51692 → 55555 [ACK] Seq=435 Ack=134 Win=2619648 Len=0
608	127.0.0.1	78.752816	127.0.0.1	TLSv1.3	114 ✓		Application Data
609	127.0.0.1	78.752885	127.0.0.1	TCP	44 ✓		51692 → 55555 [ACK] Seq=435 Ack=204 Win=2619392 Len=0
610	127.0.0.1	78.763173	127.0.0.1	TLSv1.3	374 ✓		Application Data
611	127.0.0.1	78.763236	127.0.0.1	TCP	44 ✓		51692 → 55555 [ACK] Seq=435 Ack=534 Win=2619136 Len=0



11.3. Echanges pour TLSv1.3: ClientHello + ClientKeyShare ext.

Transport Layer Security

TLSv1.3 Record Layer: Handshake Protocol: Client Hello

Content Type: Handshake (22)

Version: TLS 1.2 (0x0303)

Length: 423

Handshake Protocol: Client Hello

Handshake Type: Client Hello (1)

Length: 419

Version: TLS 1.2 (0x0303)

Random: fdbf98a3cd57ef3f202f97bf90be4019426cddfc4fc3691ab86fcc06b05bbc99

Session ID Length: 32

Session ID: c4019c55851f8c79c2312fd974f200ce568507a3b390fc9a426270b255a45bb5

Cipher Suites Length: 74

> Cipher Suites (37 suites)

Compression Methods Length: 1

> Compression Methods (1 method)

Extensions Length: 272

> Extension: status_request (len=5)

> Extension: supported_groups (len=22)

> Extension: ec_point_formats (len=2)

> Extension: status_request_v2 (len=9)

> Extension: extended_master_secret (len=0)

> Extension: session_ticket (len=0)

> Extension: signature_algorithms (len=38)

> Extension: supported_versions (len=5)

> Extension: psk_key_exchange_modes (len=2)

> Extension: signature_algorithms_cert (len=38)

> Extension: key_share (len=107)

[JA3 Fullstring [truncated]: 771,4866-4865-4867-49196-49195-52393-49200-52392]

[JA3: 795a08fe385896aee616d3b7236502da]

On commence avec TLSv1.2 car beaucoup de serveur fonctionne toujours avec cette version. On ajuste après en TLSv1.3 si cela possible

Informations habituelles: random, session ID, ...

Négociation des algorithmes de cryptage

Echange de clés non authentifiées

Null: juste pour ressembler à TLSv1.2

Groupes sur lesquelles des échanges de clés sont envisagés (notamment Diffie-Hellman: le p et le n). Ces infos sont dans des extensions

Versions TLS supportées

Extension: supported_versions (len=5)
Type: supported_versions (43)
Length: 5
Supported Versions length: 4
Supported Version: TLS 1.3 (0x0304)
Supported Version: TLS 1.2 (0x0303)

Les groupes (diffie-Hellman)

```
✓ Extension: supported_groups (len=22)
  Type: supported_groups (10)
  Length: 22
  Supported Groups List Length: 20
  ✓ Supported Groups (10 groups) ←
    Supported Group: x25519 (0x001d)
    Supported Group: secp256r1 (0x0017)
    Supported Group: secp384r1 (0x0018)
    Supported Group: secp521r1 (0x0019)
    Supported Group: x448 (0x001e)
    Supported Group: ffdhe2048 (0x0100)
    Supported Group: ffdhe3072 (0x0101)
    Supported Group: ffdhe4096 (0x0102)
    Supported Group: ffdhe6144 (0x0103)
    Supported Group: ffdhe8192 (0x0104)
```

ClientHello

Identification de courbes elliptiques (cryptographie)

ServerHello

```
✓ Key Share extension
  Client Key Share Length: 105
  ✓ Key Share Entry: Group: x25519, Key Exchange length: 32
    Group: x25519 (29)
    Key Exchange Length: 32
    Key Exchange: 86c82dae2a56528eb3a5ba4103e47164766aa6d5b7e0623e7594c7098b49a846
  ✓ Key Share Entry: Group: secp256r1, Key Exchange length: 65
    Group: secp256r1 (23)
    Key Exchange Length: 65
    Key Exchange: 04016da2a100a378a8a857df2d83d5b7516aae94264d057d66ad2780a495f0479
```

11.3. Echanges pour TLSv1.3: ServerHello + ServerKeyShare ext.

```
▼ Transport Layer Security
  ▼ TLSv1.3 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 122
  ▼ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 118
    Version: TLS 1.2 (0x0303)
    Random: 65f2e1bad35bb454181e94eb89fd5306e83edc52c04313686c4e8053632d5e95
    Session ID Length: 32
    Session ID: c4019c55851f8c79c2312fd974f200ce568507a3b390fc9a426270b255a45bb5
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302) ← Choix du serveur
    Compression Method: null (0)
    Extensions Length: 46
  ▼ Extension: supported_versions (len=2)
    Type: supported_versions (43)
    Length: 2
    Supported Version: TLS 1.3 (0x0304) ← Version supportée
  ▼ Extension: key_share (len=36)
    Type: key_share (51)
    Length: 36
  ▼ Key Share extension
    ▼ Key Share Entry: Group: x25519, Key Exchange length: 32
      Group: x25519 (29)
      Key Exchange Length: 32
      Key Exchange: e72d2ee79ccb3c0fdcac965aea18dd6a866ca223f3864ce2f07df23302636d75
      [JA3S Fullstring: /71,4866,43-51]
      [JA3S: 15af977ce25de452b96affa2addb1036]
```

Idem client: random, session ID, ...

Choix du serveur

Version supportée

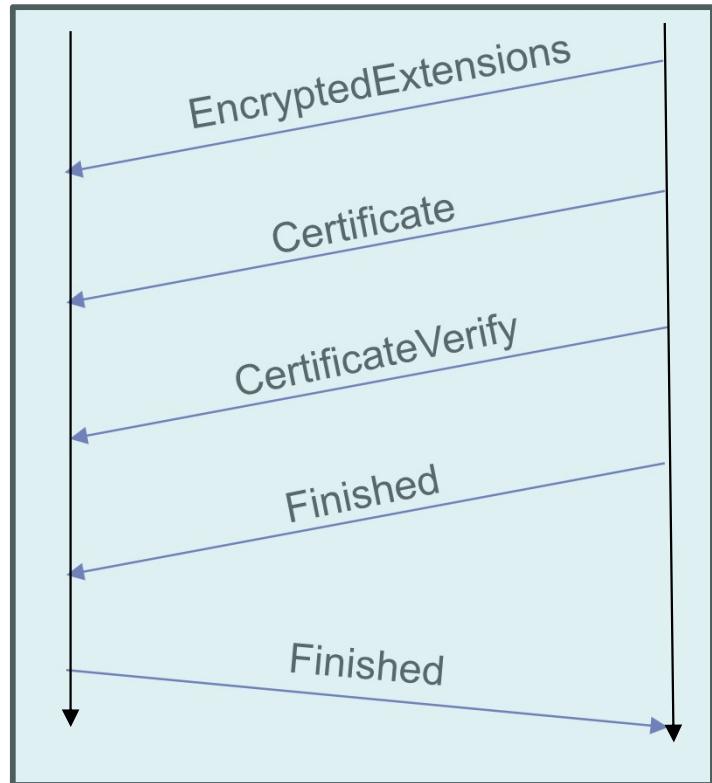
Il est maintenant possible de dériver un premier jeu de clés symétriques. Ces dernières permettront de sécuriser le reste de la négociation

11.4. Echanges pour TLSv1.3: Analyse de ces Hello...

- **Les paquets Hello** échangent des données qui permettent **uniquement la protection symétrique des messages**: algorithmes de chiffrement combinés comme AES-GCM (*Galois/Counter Mode (GCM)* est un mode d'opération de chiffrement par bloc en cryptographie symétrique)ou Chacha20-Poly1035 (*système d'encryption “chacha20” authentifié (Poly1035, hash) avec addition de donnée (AEAD)*)
- **Les autres éléments négociés** dans les versions antérieures de la suite cryptographique **sont désormais traités dans des extensions**:
 - L'extension **supported_groups** reprend l'ensemble des groupes Diffie-Hellman supportés. la liste contient des références à des groupes nommés. Point 11.3
 - l'extension **signature_algorithms** : le client informe le serveur sur les algorithmes de signature que ce dernier peut utiliser pour s'authentifier.

Jusqu'à présent, on a la possibilité de protéger les communications (comme par exemple, le certificat du serveur, le certificat du client n'étant pas obligatoire) mais **sans garantie sur la personne de l'autre côté => Pas d'authentification, pas de signature !**

11.5. Echanges pour TLSv1.3: Authentification du serveur



- Le message « **EncryptedExtensions** » contient des informations complémentaires (cryptées => différents des extensions expliquées précédemment) comme des réponses aux extensions ne servant pas lors de la première phase de négociation: réponse à l'extension « Server Name Indication ».
- L'authentification du serveur est réalisée par l'envoi d'un message « **Certificate** » qui contient le chaîne de certificats du serveur. Il contient également des extensions notamment des informations de révocations.
- Le message « **CertificateVerify** » inclut une signature de tous les messages de négociation précédemment échangés.
- Le message « **Finished** », contient notamment MAC (Message Authentication Code) des messages précédents (assure donc que les messages proviennent bien du bon client et serveur). Une protection en confidentialité et en intégrité est également assurée = **message de**

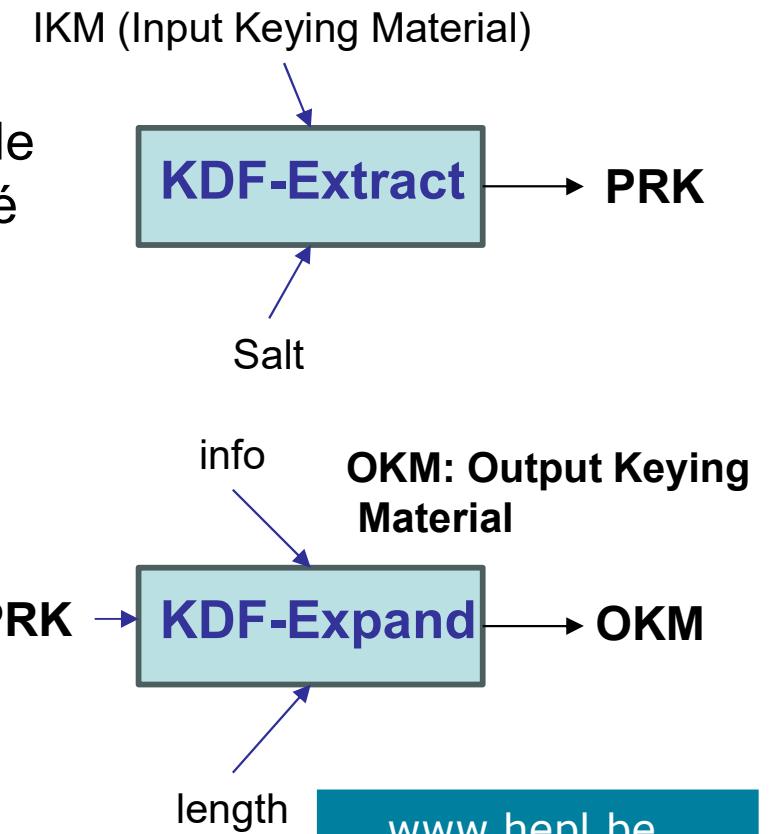
A partir de cette séquence, contrairement aux versions précédentes, tous les messages vont être signés et pas seulement certains de ceux-ci.

11.6. Echanges pour TLSv1.3: Améliorations de la fonction de dérivation des clés

Du point de vue de la dérivation des clés, à la place d'un HMAC utilisé jusqu'à la version TLS 1.2, TLS 1.3 utilise: **HKDF** (*HMAC-based Extract-and-Expand Key Derivation Function*, voir RFC 5869). Il y a deux parties **Extract** et **Expand key**

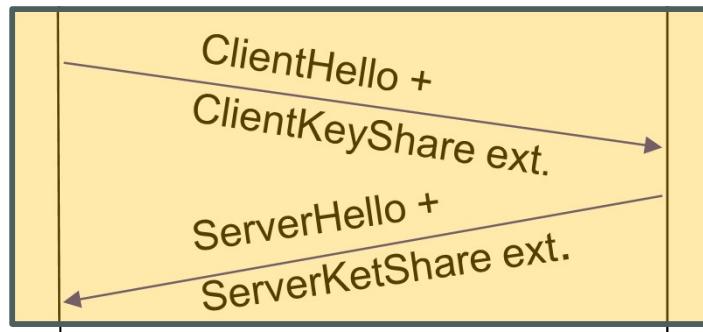
KDF-Extract prend comme entrée un message IKM comme par exemple un secret partagé généré à l'aide de Diffie-Hellman, et un sel facultatif. Il génère alors une clé cryptographique appelée PRK ("clé pseudo-aléatoire").

KDF-Expand prend comme entrée la PRK, des informations complémentaires (contexte et application spécifiques) optionnelles et la longueur en Byte de la sortie. Permet d'obtenir plusieurs clés pseudo-aléatoires à partir de la PRK



11.6. Echanges pour TLSv1.3: Change Cipher Spec

Client Serveur



- Dans les versions précédentes, ce message sollicitait le changement des clés. Au début de TLSv1.3, il avait disparu. Afin d'éviter des éventuelles interruptions par certains équipements, il a fait sa réapparition afin de créer un mode de compatibilité pour que TLSv1.3 ressemble dans un certain sens à TLSv1.2
- Ces messages non signés n'ont aucune signification !

602	127.0.0.1	78.688178	127.0.0.1	TLSv1.3	171 ✓	Server Hello
603	127.0.0.1	78.688248	127.0.0.1	TCP	44 ✓	51692 → 55555 [ACK] Seq=429 Ack=128 Win=2619648 Len=0
604	127.0.0.1	78.713889	127.0.0.1	TLSv1.3	50 ✓	Change Cipher Spec
605	127.0.0.1	78.713948	127.0.0.1	TCP	44 ✓	55555 → 51692 [ACK] Seq=128 Ack=435 Win=2619648 Len=0
606	127.0.0.1	78.743666	127.0.0.1	TLSv1.3	50 ✓	Change Cipher Spec
607	127.0.0.1	78.743738	127.0.0.1	TCP	44 ✓	51692 → 55555 [ACK] Seq=435 Ack=134 Win=2619648 Len=0
608	127.0.0.1	78.752816	127.0.0.1	TLSv1.3	114 ✓	Application Data

Wireshark · Paquet 604 · sauvegardeCertTLS1_3.pcapng

Frame 604: 50 bytes on wire (400 bits), 50 bytes captured (400 bits) on interface \Device\NPF_Loopback, Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 51692, Dst Port: 55555, Seq: 429, Ack: 128, Len: 6
Transport Layer Security
 TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
 Content Type: Change Cipher Spec (20)
 Version: TLS 1.2 (0x0303)
 Length: 1
 Change Cipher Spec Message

12. Authentification du client

- Pas obligatoire
- Le serveur peut demander cette authentification comme dans les versions précédentes en utilisant le message suivant:
 - **CertificateRequest**
- Le client répond alors avec les messages:
 - **Certificate**
 - **CertificateVerify**
- TLSv1.3 permet ces échanges (demandes) à n'importe quel moment après la négociation principale

13. Reprise de session

- Le serveur peut envoyer un message: **NewSessionTicket** afin d'autoriser le client à reprendre une session ultérieure.
- Le client ou le serveur peuvent demander la mise en place de nouvelles clés: **KeyUpdate**
- La reprise de session est fusionnée avec l'authentification par clé partagée (PSK). Le serveur fournit les tickets qui seront utilisés comme PSK => simplification par rapport aux versions précédentes.
- Un nouveau mode de reprise est apparu (autorisation de ce dernier via le ticket). Ce dernier permet l'obtenir dès le premier paquet, une clé commune. Cette dernière permet de chiffré directement les paquets suivants juste après le **ClientHello**. Point faible, on pourrait rejouer ce paquet. Des solutions ont été apportées mais bof ... => on peut le désactiver.

14. conclusions

- Amélioration du point de vue de la cryptographie:
 - Plus de RSA (ne permet pas d'assurer une confidentialité persistante)
 - Utilisation des courbes elliptiques
 - Plus de CBC et RC4
- Diminution du nombre d'échanges lors de la négociation et de l'ouverture de session plus rapide (gain entre 25 et 50 % par rapport aux versions précédentes)
- Simplification de la machine à états: fusion de certaines fonctionnalités comme la reprise de session avec l'authentification par clé partagée
- Les versions précédentes devraient encore co-exister durant quelques années avec la version 1.3 (adaptation lente des applications)