Une introduction à SSL

Felip Manyé i Ballester

6 juin 2009

Plan

Plan

Buts et enjeux de SSL

But

Sécuriser les échanges de données sur Internet (sur un réseau TCP/IP) entre un client et un serveur.

Fonctionnalités

- *Authentification* du serveur voire du client
- *Chiffrement* des données
- *Non-répudiation*
- *Intégrité* des données

Chiffrement et déchiffrement des messages

Principe

On souhaite transformer un message « en clair » M en un message inintelligible, « chiffré » C. On se donne

- une fonction de chiffrement E, telle que $C = E(M)$
- une fonction de déchiffrement D, telle que $M = D(C)$
- l'identité suivante doit donc être vérifiée : $D(E(M)) = M$

Algorithmes et clefs

- Algorithme cryptographique : méthode de chiffrement
- Clef cryptographique : donnée secrète, au moins en partie, dans un espace de cardinalité importante, sur laquelle se base un algorithme

Chiffrement symétrique

Principe

La même clef est utilisée par les deux parties, pour le chiffrement et le déchiffrement.

Avantages

- Rapidité du calcul
- Authentification
- Confidentialité

Inconvénients

- Échange de la clef ?

Exemples d'utilisation

- Chiffrement d'une clef privée (« passphrase »)
- Chiffrement au sein d'une session SSL

Quelques algorithmes symétriques ou à clef secrète (par blocs)

- DES (Data Encryption Standard) : seulement 2^{56} clefs ; craqué en 1999 en 22h15min
- AES (Advanced Encryption Standard) : remplaçant d'AES. 128 bits, 192 bits, 256 bits
- Triple DES : construit sur DES. 112 bits, 168 bits
- IDEA (International Data Encryption Algorithm). Clef de 128 bits
- Blowfish : peut remplacer DES. Pas de brevet ni licence. Éprouvé. 32 bits à 448 bits.
- RC4, RC5 (Ron's Code our Rivest's Cipher)

Chiffrement asymétrique

Principe

- La clef de chiffrement est différente de la clef de déchiffrement
- Clef de chiffrement : clef *privée*, **secrète**
- Clef de déchiffrement : clef *publique*, divulguée

Avantages

- Gestion des clefs facilitée

Inconvénients

- 1000 fois plus coûteux

Exemples d'utilisation

- Authentification, signature
- Chiffrement de messages *courts*
- Échange de clefs

Quelques algorithmes asymétriques (à clef publique)

- RSA (Rivest-Shamir-Adleman) : signature, chiffrement. Le brevet a expiré en 2000. Clefs : 1024 bits à 2048 bits (utilisateurs)
- DSA (Digital Signature Algorithm) : signature uniquement
- Diffie-Hellman : échange de clefs

Fonction de hachage à sens unique

Définition

Fonction f :

- qui convertit une suite de bits de taille quelconque en une suite de bits de taille fixe (« empreinte numérique »)
- telle qu'il est très difficile de trouver $x_1 \neq x_2$ vérifiant $f(x_1) = f(x_2)$ (« collision »)

En pratique

- Si l'on change un seul bit de l'information en entrée, le résultat est drastiquement différent
- Une empreinte numérique de 128 bits est un *minimum* (cassable)

Signature numérique

Principe

Envoyer :

- un message en clair M
- le même message chiffré avec sa clef privée $C = E_{priv}(M)$: la signature

Le destinataire du message calcule $M' = D_{pub}(C)$ avec la clef publique. Si $M = M'$, la signature est vérifiée.

En Pratique

On envoie un « code d'identification de message »(MAC), c'est-à-dire *l'empreinte numérique chiffrée avec la clef privée.*

Fiabilité des algorithmes et sécurité

- Ne pas utiliser de méthodes « maison », leur préférer des algorithmes éprouvés par les cryptanalistes
- Conserver jalousement les clefs secrètes ou privées
- Choisir un bon compromis en termes de *longueur de clef*
  - temps de calcul
  - aspects légaux
- L'entropie doit être suffisante lors de la génération de la clef
  - Proscrire les sources d'entropie « maison » : utiliser /dev/urandom sous *nix
  - On doit pouvoir faire confiance à OpenSSL dans ce domaine
- Évidemment, mettre les systèmes à jour (cf. faille Debian en mai 2008)

Aspects légaux de la cryptographie en 2008

Utilisation

« En vertu de l'article 30-I de la loi 2004-575 du 21 juin 2004, l'*utilisation* de moyens de cryptologie est libre »

Autres circonstances

« En revanche, la fourniture, l'importation et l'exportation des ces moyens sont réglementés en France. Ces opérations sont soumises soit au régime de la déclaration, soit au régime de l'autorisation. »

Source :
http ://www.ssi.gouv.fr/fr/reglementation/index.html#crypto

Plan

Certificats X.509

Un certificat :

- est le support de la clef publique d'une entité
- contient des données sur ce dernier

Trois champs sont obligatoires :

- le TBS (To Be Signed) qui sera signé et contient les informations du certificat
- l'algorithme asymétrique et la fonction de hachage utilisée pour la signature
- la signature de l'empreinte numérique du TBS

Description ASN.1 simplifiée d'un certificat X.509

```
Certificate  : :=  SEQUENCE  {
        tbsCertificate        TBSCertificate,
        signatureAlgorithm    AlgorithmIdentifier,
        signatureValue        BIT STRING }

   TBSCertificate  : :=  SEQUENCE  {
        version          [0]    EXPLICIT Version DEFAULT v1,
        serialNumber            CertificateSerialNumber,
        signature               AlgorithmIdentifier,
        issuer                  Name,
        validity                Validity,
        subject                 Name,
        subjectPublicKeyInfo SubjectPublicKeyInfo,
        issuerUniqueID [1] IMPLICIT UniqueIdentifier OPTIONAL,
                                - If present, version MUST be v2 or v3
        subjectUniqueID [2] IMPLICIT UniqueIdentifier OPTIONAL,
                                - If present, version MUST be v2 or v3
        extensions      [3] EXPLICIT Extensions OPTIONAL
                                - If present, version MUST be v3}
```

Chaîne de certificats

- Un certificat signé par une autorité de certification (AC) « mère » peut à son tour signer un autre certificat
- On parle de « chaîne de certificats »
- Pour contrôler cette chaîne, on utilise les extensions basicConstraints (valeurs CA :false ou CA :true) et keyCertSign (pour en limiter la longueur)
- On valide un certificat fils de manière récursive avec la clef publique du parent, jusqu'à la CA racine dont le certificat est « auto-signé »

Vérification d'une signature de certificat

Pour vérifier la validité de la signature d'un certificat :

- on calcule l'empreinte M du TBS
- on récupère la clef publique de l'autorité de certification dans le certificat racine pré-installé
- ce qui permet de déchiffrer le contenu du champ signature S : $M' = D(S)$
- on s'assure que $M = M'$, ce qui prouve que le certificat a bien été signé par cette autorité de certification, et qu'il appartient à la bonne personne (authentification), mais aussi qu'il n'a pas été contrefait (intégrité)

Champ d'utilisation d'un certificat

Les extensions de SSLv3 permettent de destiner le certificat à un usage particulier :

- keyUsage : signature (digitalSignature, nonRepudiation), chiffrement (keyEncipherment), délégation AC fille(keyCertSign)
- nsCertType : serveur (server), utilisateur (clientAuth, emailProtection)

Validation d'un certificat

Certificat serveur

- période de validité
- AC de confiance ?
- vérification de la signature
- selon applications, vérifier le nom de domaine

Certificat client

- période de validité
- AC de confiance ?
- vérifier la signature (voir CertificateVerify) ou comparer à copie du certificat installer localement
- le DN est-il connue (stocké dans un LDAP...) ?
- vérifier les permissions

Formats de certificats

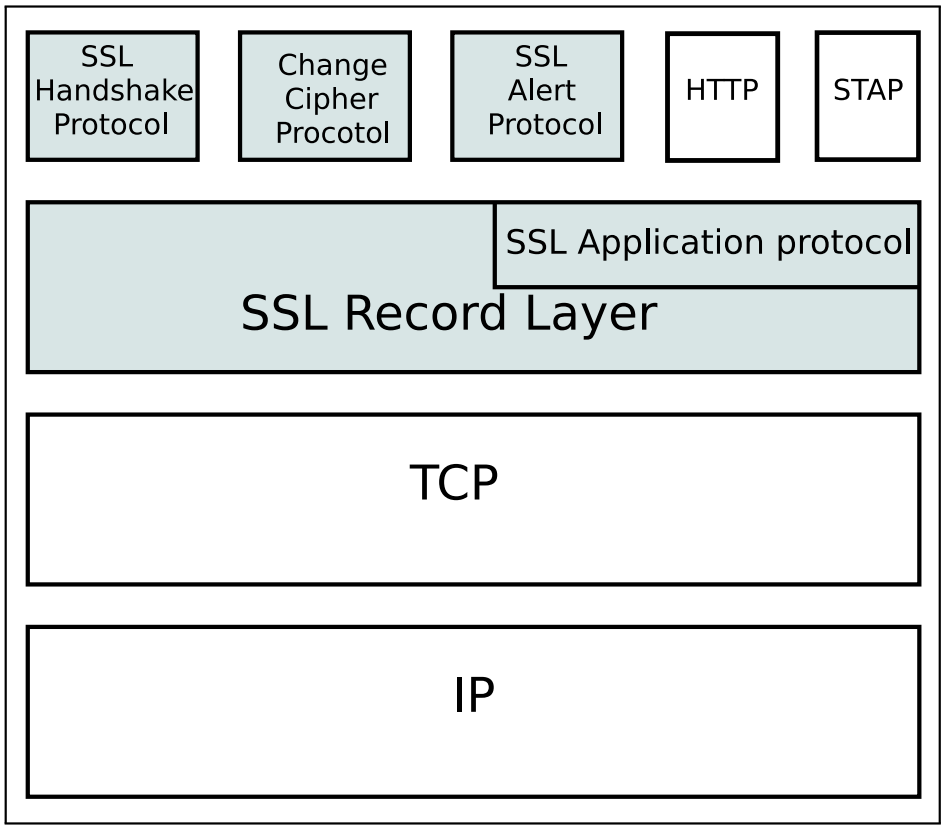
- DER (« Distinguished Encoding Rules ») : certificat binaire encodé selon règles ASN.1
- PEM (« Privacy Enhanced Mail ») : certificat DER encodé à son tour en base 64
- PKCS#12 (« Public-Key Cryptography Standard #12 ») : rassemble en un seul fichier un biclef, le certificat correspondant et toute la chaîne de certificats AC. Cf Sagem
- PKCS#10 : demande de certificat (Lyra)

Plan

Historique

- *Secure Socket Layer* : développé par Netscape, qui équipe ses navigateurs de la version 2.0
- Actuellement, SSLv3
- Brevet US racheté par l'IETF en 2001 $\Rightarrow$ description de TLSv1 (Transport Layer Security) dans la RFC 2246
- SSLv3 et TLSv1 subtilement incompatibles, mais exactement le même principe

Principe



- Au-dessus d'un protocole de transport *connecté* (TCP)
- Une phase de négociation de clef de session : cryptographie *asymétrique*
- Pendant la session : cryptographie *symétrique*

Algorithmes

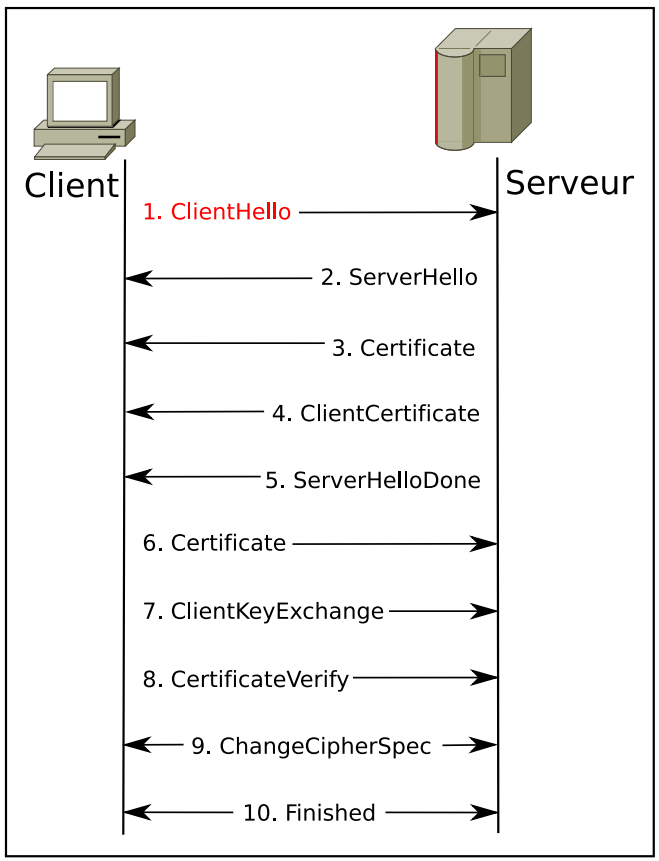
Algorithmes utilisés

- un algorithme asymétrique pour l'authentification : RSA, DSA
- un algorithme asymétrique pour l'échange de clefs secrètes : RSA, Diffie-Hellman
- un algorithme à clef secrète pour le chiffrement du trafic
- une fonction de hachage à sens unique pour calculer les codes d'authentification de messages (MAC)

« Handshake » SSL

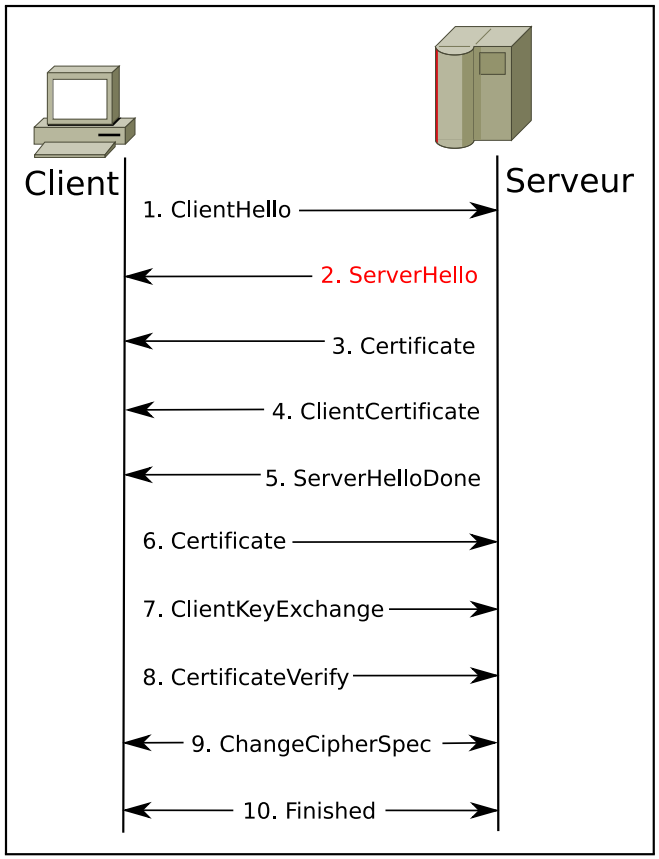
- Authentification des parties
- Négociation des algorithmes cryptographiques
- Négociation de la clef de session pour SSL Record : MasterKey

« Handshake » SSL 1/10



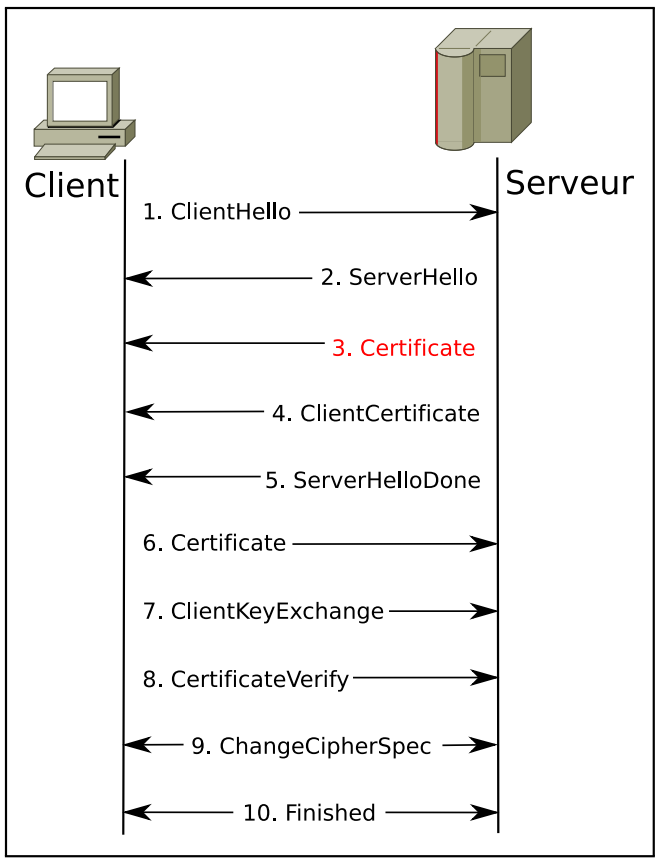
- Demande de connexion au serveur
- Initiation du dialogue SSL
- Le client envoie au serveur :
  - la version du protocole SSL
  - un identificateur de session (session_identifier)
  - la liste des algorithmes cryptographiques supportés (cipher_suite)
  - la liste des méthodes de compression
  - un nombre aléatoire *client_random* sur 32 octets

« Handshake » SSL 2/10



- Le serveur envoie au client :
  - la version du protocole SSL
  - l'identificateur de session session_identifier
  - les algorithmes cryptographiques sélectionnés
  - un nombre aléatoire *server_random sur 32 octets*
- Si les listes d'algorithmes ne concordent pas, renvoi de l'erreur handshake_failure.

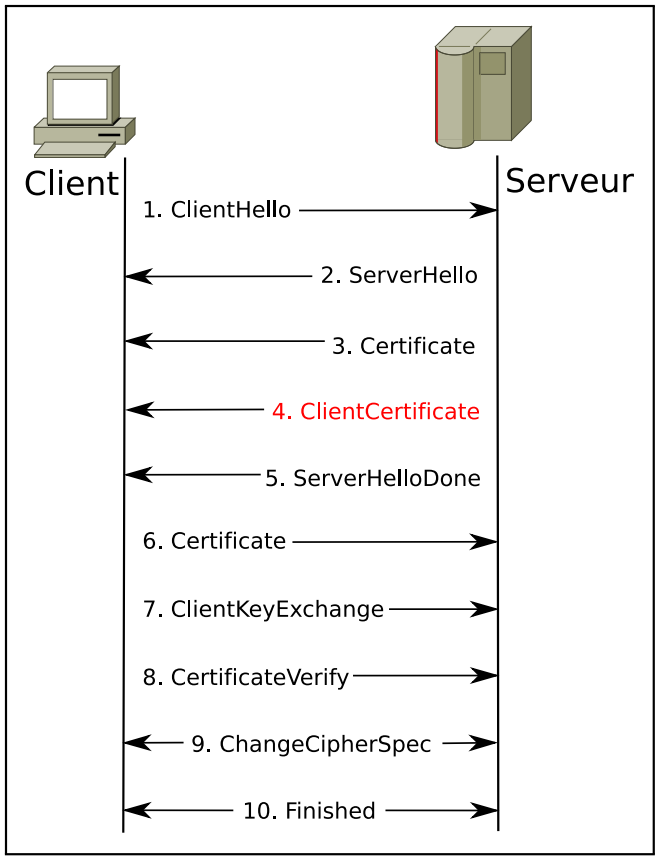
« Handshake » SSL 3/10



- Le serveur envoie son certificat X.509
- Le client le vérifie. Si erreur, la connexion ne sera pas établie

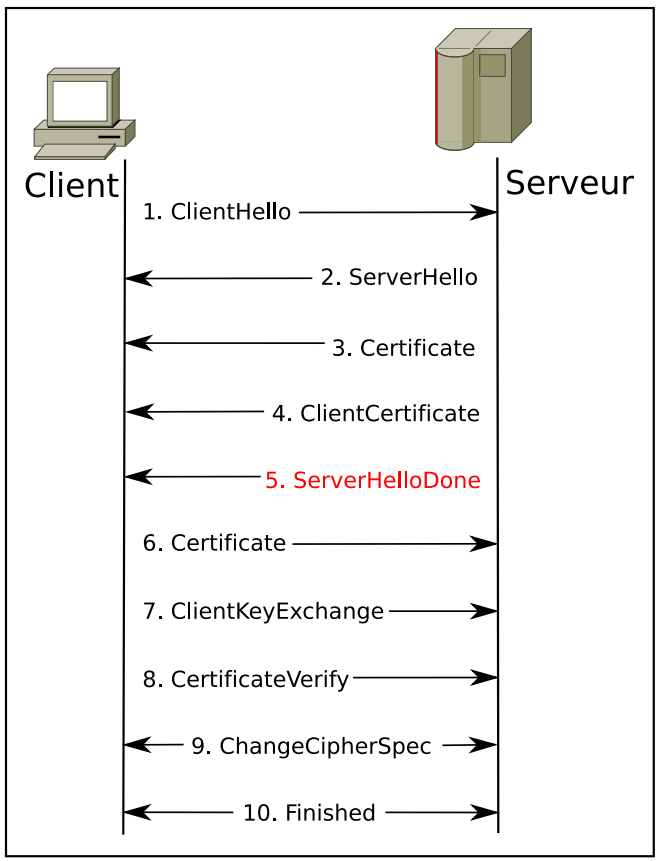
Éventuellement, le serveur peut aussi envoyer un message *ServerKeyExchange* pour initier l'échange de clef avec Diffie-Hellman.

« Handshake » SSL 4/10



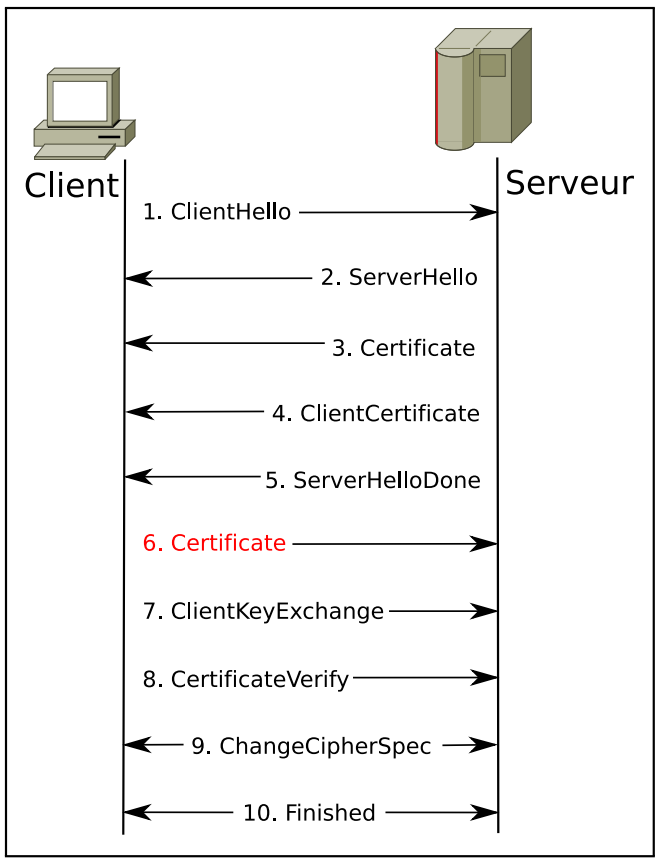
- Cette étape est optionnelle ; elle est utilisée pour la *double authentification*. Le serveur demande le certificat du client.

« Handshake » SSL 5/10



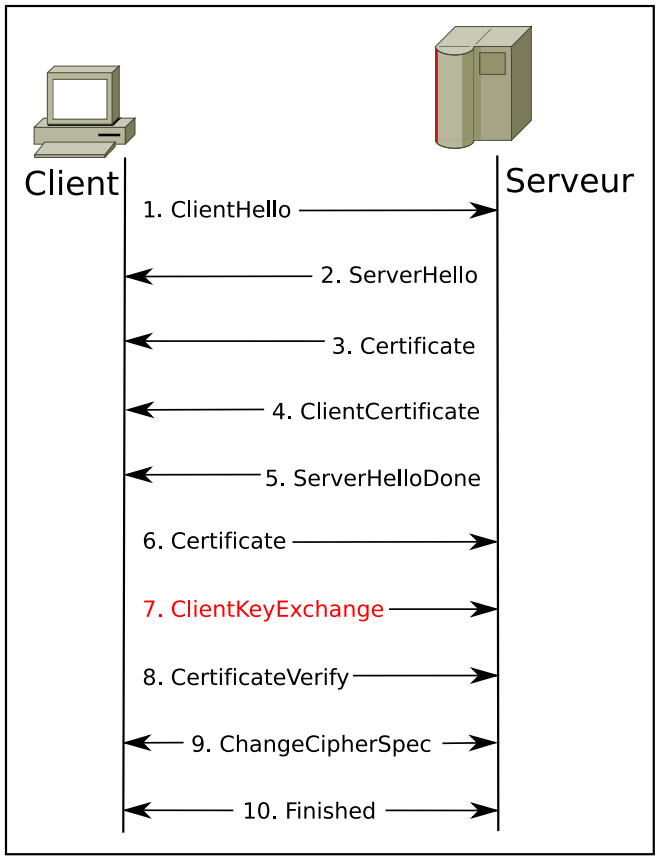
- Le serveur a envoyé tous ses messages et se met en attente d'un réponse.

« Handshake » SSL 6/10



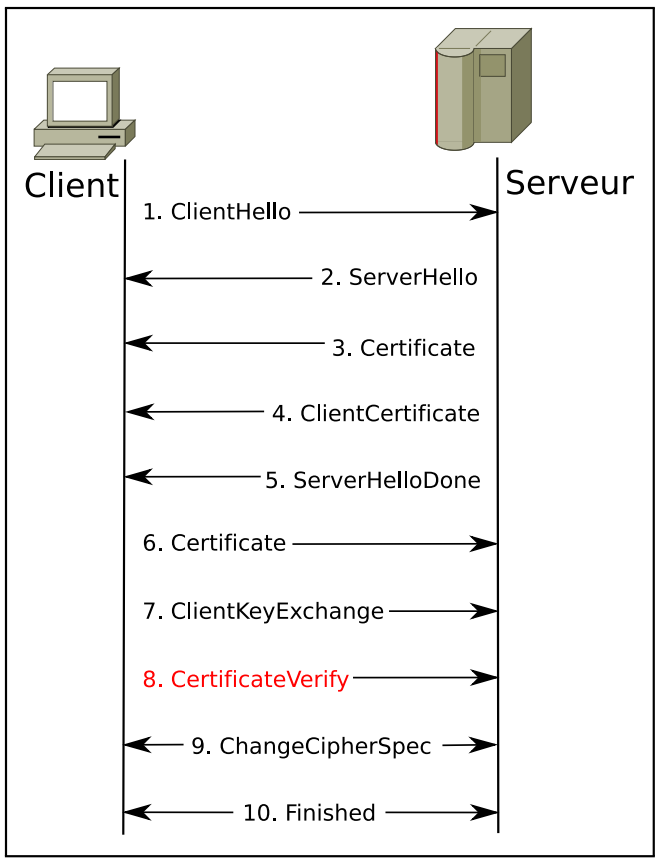
- Si le serveur l'a demandé, le client envoie son certificat.
- S'il n'en a pas, il envoie le message d'alerte no_certificate. C'est alors au serveur de décider s'il poursuit la communication.

« Handshake » SSL 7/10



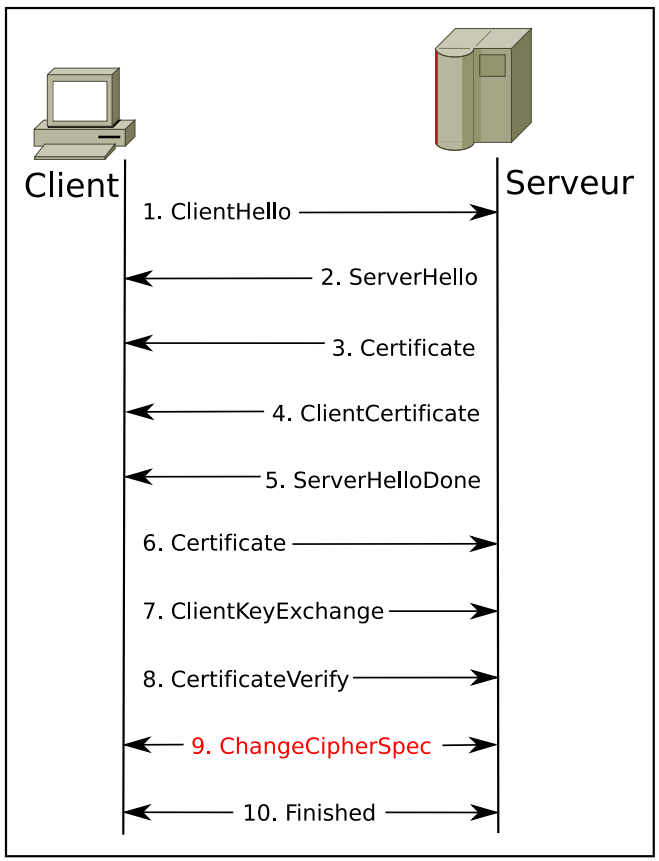
- Le client envoie une pré-clef secrète PreMasterKey chiffrée avec la clef publique du serveur :
  - Dans le cas de RSA, suite aléatoire de 48 octets
  - Dans le cas de Diffie-Hellman, elle a déjà été échangée
- À partir de *PreMasterKey*, *client_random* et *server_random*, et à l'aide de fonctions de hachage, les deux parties calculent la clef de session *MasterKey*

« Handshake » SSL 8/10



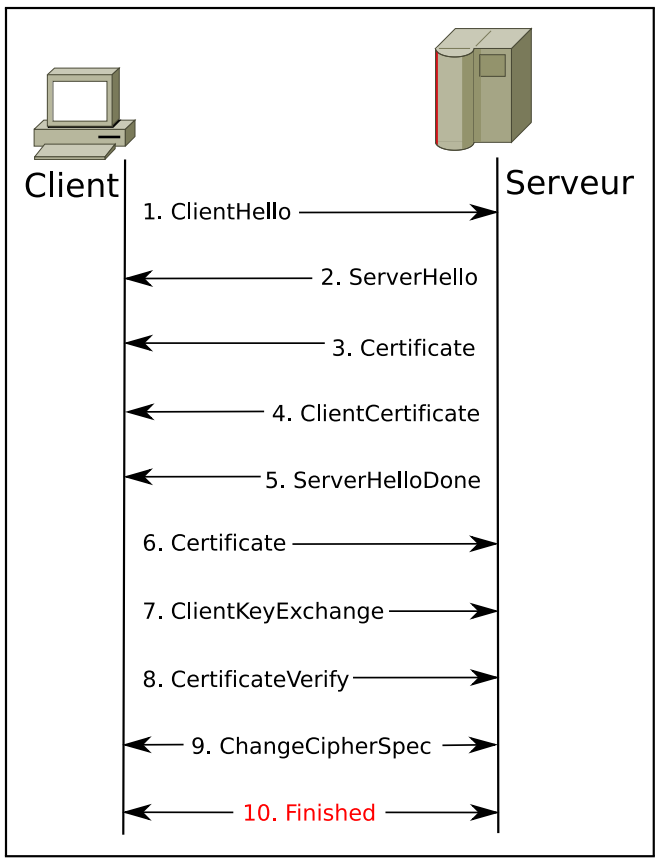
- En double authentification, le client prouve qu'il est en possession de la clef privée correspondant à son certificat en envoyant des données déjà échangées *chiffrées avec sa clef privée*

« Handshake » SSL 9/10



- Les algorithmes ont été acceptés

« Handshake » SSL 10/10



- Fin de la négociation et du « Handshake »
- Début de la session SSL
- Dorénavant, les communications sont chiffrées de manière *symétrique* avec la clef de session *MasterKey* négociée précédemment

Plan

Infrastructure à clefs publiques

- une PKI (Public Key Infrastructure) ou ICP (infrastructure à clefs publiques) met à disposition les clefs publiques (donc les certificats) de ses utilisateurs.
- elle gère l'intégralité du cycle de vie d'un certificat, de sa signature à sa mort naturelle (dates de validité) ou sa révocation s'il est compromis.
- un logiciel de PKI permet d'accomplir ces tâches et de garder une trace des certificats clients
- le groupe PKIX de l'IETF est à l'origine d'un effort de normalisation des spécifications et protocoles de PKI

Autorités de certification et d'enregistrement

- L'autorité de certification (AC ou CA) est une entité représentant le capital de la confiance de la PKI. C'est elle qui signe les demandes de certificat avec sa clef privée, dont la confidentialité est primordiale
- L'autorité d'enregistrement (AE ou RA) est chargée de recevoir les demandes de signature de certificat et de s'assurer de leur validité et de l'identité du demandeur. Elle peut être incluse à l'AE

Entité d'enrôlement

- l'entité d'enrôlement (EE) peut être accessible directement par l'utilisateur, ou uniquement par un opérateur de l'AE.
- elle permet d'effectuer une demande de certificat, par exemple au travers d'une interface web.
- l'utilisateur peut fournir ses données ou un certificat PKCS#10
- la fonctionnalité SPKAC des navigateurs (Signed Public Key And Challenge) permet d'effectuer une demande de manière décentralisée

Mais que faire des certificats compromis ?

- le point faible des PKI
- le préjudice d'une compromission aggravé par le comportement des clients courants (navigateurs. . ., qui ignorent les protocoles existants
- les CRL (Certification Revocation List), « normalisées » par la RFC 3280 :
  - moyen de mise à disposition ? (cf. Verisign...). LDAP, HTTP,. . .
  - rythme d'actualisation ? (fenêtre de temps)
  - taille et stockage ?
  - consultation par les clients ?
- OCSP (Online Certificate Status Protocol), RFC 2560 :
  - déclare valide un certificat qui n'a pas été révoqué
  - sujet aux attaques : messages signés en cache, messages d'erreur non signés

Plan

Pourquoi l'accélération SSL ?

- les calculs de cryptographie asymétrique peuvent mettre un serveur à genoux
- pour dissocier totalement la couche SSL de STAP

Accélération logicielle : utilisation de stunnel

Stunnel est capable de gérer la couche SSL indépendamment de l'application. Suivant les versions, paramétrable en ligne de commandes ou avec un fichier de configuration.

Exemple en ligne de commande

stunnel -f -D 7 -d 9501 -r afsollx :9500 -o log -p stunnel.pem -P ~/.stunnel/pidfile -v 3 -a ~/.stunnel/cert_clients

- -d et -r : IP/port source et destination. Exécuter stunnel sur une autre machine permet de monter une installation type proxy SSL
- -p : fichier PEM contenant le certificat *et* la clef privée du serveur
- -v : vérification du certificat. 2 : validité, 3 : comparaison avec fichiers installés localement (-a).

Dispositifs d'accélération matérielle

Différents types

- cartes jouant le rôle de co-processeur (CryptoNetX chez Broadcom) : obsolète
- cartes réseau contenant les biclefs et gérant le flux SSL (nCipher, Sun).
- boîtiers indépendants (nCipher, Kemp). Mêmes fonctionnalités + répartition de charge.

Intégration

- avec la PKI ?
- avec stunnel et/ou OpenSSL ?
- avec l'environnement (drivers fermés) ?

Accélération matérielle : quart d'heure consumériste

FIG. 1: nCipher netHSM

FIG. 2: nCipher nFast et nForce

FIG. 3: Kemp Load Balancer 1500

Plan

Généralités

Un logiciel de PKI permet de gérer les certificats d'une organisation.
Liste d'applications libres ou « open source » :

- tinyCA
- NewPKI
- EJBCA
- IDX-PKI, et sa version "commerciale" OpenTrust PKI
- OpenCA

Outils extérieurs à la PKI

- LDAP : annuaire. Stockage de données concernant l'émission de certificats en plus des données usuelles
- Kerberos : serveur d'authentification

OpenSSL en ligne de commande

Fichier de configuration de l'AC 1/2

```
[ ca ]
default_ca = exampleca
[ exampleca ]
dir = /opt/exampleca
certificate = $dir/cacert.pem
database = $dir/index.txt
new_certs_dir = $dir/certs
private_key = $dir/private/cakey.pem
serial = $dir/serial
default_crl_days = 7
default_days = 365
default_md = md5
policy = exampleca_policy
x509_extensions =
certificate_extensions
```

```
[ exampleca_policy ]
commonName = supplied
stateOrProvinceName = supplied
countryName = supplied
emailAddress = supplied
organizationName = supplied
organizationalUnitName =
optional
[ certificate_extensions ]
basicConstraints = CA :false
[ req ]
default_bits = 2048
default_keyfile =
/opt/exampleca/private/cakey.pem
```

OpenSSL en ligne de commande

Fichier de configuration de l'AC 2/2

```
default_md = sha1
prompt = no
distinguished_name =
root_ca_distinguished_name
x509_extensions =
root_ca_extensions
[ root_ca_distinguished_name ]
commonName = Example CA
stateOrProvinceName = Virginia
countryName = US
emailAddress = ca@exampleca.org
organizationName = Root
Certification Authority
[ root_ca_extensions ]
basicConstraints = CA :true
```

Créer l'AC

- export OPENSSL_CONF = /opt/exempleCA
- openssl req -x509 -newkey rsa -out cacert.pem -outform PEM

OpenSSL en ligne de commande

Création de certificat

Effectuer une demande de certificat (certificate request) :

- unset OPENSSL_CONF
- openssl req -newkey rsa :1024 -keyout testkey.pem -keyform PEM -out testreq.pem

Signer cette demande :

- export OPENSSL_CONF = /opt/exempleCA
- openssl ca -in testreq.pem

Placer un certificat client et la chaîne des AC dans un conteneur PKCS#12 :

- openssl pkcs12 -export -in testreq.pem -inkey testkey.pem -certfile CA.pem -out certificat.p12 -name "Certificat TPE"
- chmod 600 certificat.p12

TinyCA

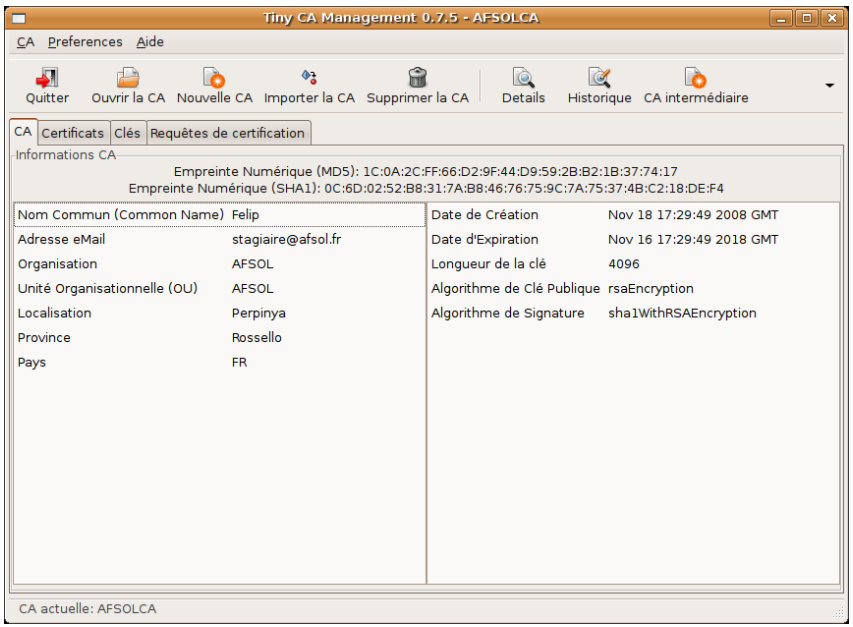


FIG. 4: Interface graphique de TinyCA

Avantages

- très simple
- idéal pour un VPN tel que celui d'AFSOL

Inconvénients

- interface GTK
- rythme de développement ?

NewPKI

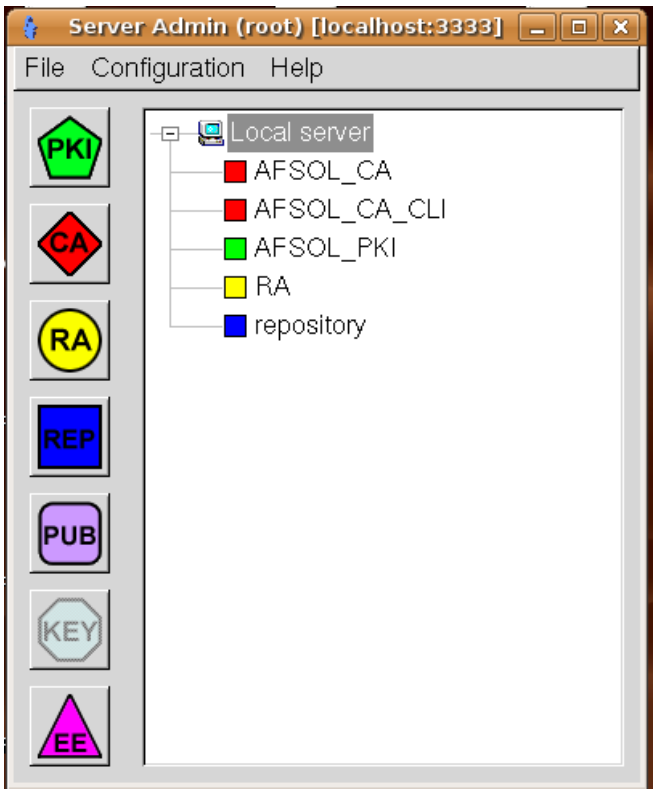


FIG. 5: Interface de newPKI

Avantages

- Très bien structuré en classes réutilisables
- Rajout possible d'une couche en PHP

Inconvénients

- interface GTK
- rythme de développement ?
- compatibilité MySQL 5
- Intégration stunnel/HSM ?

IDX-PKI

- deux versions : une sous licence GPL, une « commerciale » (modules HSM supplémentaires...)
- engagement sur version libre en baisse ?
- haute modularité
- répondeur OCSP
- interface web

EJBCA

- PKI en java (JBOSS) $\Rightarrow$ un peu lourd et éloigné des technologies usuelles de STAP
- développement actif et support HSM
- répondeur OCSP
- interface web
- authentification web par certificat client
- trop orienté client HTTP ?
- moins modulaire qu'OpenCA

OpenCA

FIG. 6: Interface d'OpenCA

- interface web
- très modulable
- support de HSM
- répondeur OCSP
- importation et exportation de données (par exemple pour Stunnel)
- authentification web par certificat client
- un tout petit peu compliqué à mettre en œuvre

Conclusion

- Une technologie complexe, en plein déploiement
- Un aspect désormais important de la monétique
- Intégrer une PKI à STAP pour structures de taille réduite (moins de 200 TPE) : un projet en soi :
  - déploiement technique et simplification d'une application existante (OpenCA pourrait être un bon choix)
  - aspects légaux : AFSOL ne doit pas avoir connaissance de la clef privée de l'AC $\Rightarrow$ génération automatique.
  - nécessité d'une formation minimale pour le personnel
  - normalisation des TPE ? Sagem : « phase de test ».

Bibliographie

- Cachat Ch., Carella D., *PKI Open Source : déploiement et administration*, O'Reilly France, 2003
- Chandra P., Messier M., Viega J., *Network Security with OpenSSL*, O'Reilly, 2002
- Rescorla E., *SSL and TLS : Designing and Building Secure Systems*, Addison-Wesley Professional, 2000
- Menezes A. J., Van Oorschot P. C., Vanstone S.A., *Handbook of Applied Cryptography*, CRC Press, 2001 (5^{e} tirage), disponible en PDF sur : http ://www.cacr.math.uwaterloo.ca/hac/

That's all folks !