الجـــــــــــــــــــمـهـوريـــــة الجـــــــــــــزائـريـة الديمـــــــقـراطيـة الـشـــــــــعبيـة

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التـــــــــــــــــــــــــــعـليـم العـــــــالي و البـــــــــحث العـــــلـمـــي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جـــــــــــــــامعة أبي بكـــر بـلـقـايد – تـلمســـــــــــــان –

Université Aboubakr Belkaïd – Tlemcen –
Faculté de TECHNOLOGIE

MEMOIRE

Présenté pour l'obtention du **diplôme** de **MASTER**

En : Télécommunications

Spécialité : Réseaux & Télécommunications

Par :

MEZOUARI ILHEM & ABBES AMEL

Sujet

Sécurisation du service HTTP par SSL/TLS (authentification mutuelle par certificats)

Soutenu publiquement, en juin 2018, devant le jury composé de :

Mr BOUACHA Abdelhafid	Maitre de Conférences	Université de Tlemcen	Président
Mr SLIMANI Hicham	Maitre de Conférences	Université de Tlemcen	Examinateur
Mr ABDELMALEK Abdelhafid	Maitre de Conférences	Université de Tlemcen	Encadreur
Mme SLIMANE Zohra	Maitre de Conférences	Centre Univ. Ain Temouchent	Co- Encadreur

Remerciements

En préambule à ce mémoire nous remercions ALLAH qui nous a aidé et nous a donné la patience et le courage durant ces langues années d'étude.

Nous souhaitons adresser nos remerciements les plus sincères aux personnes qui nous ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire ainsi qu'à la réussite de cette formidable année universitaire.

Ces remerciements vont tout d'abord au corps professoral et administratif de la Faculté des Sciences et technologie, pour la richesse et la qualité de leur enseignement et qui déploient de grands efforts pour assurer à leurs étudiants une formation actualisée.

Nous tenons à remercier sincèrement Monsieur Abdelmalek Abdelhafid Maitre de conférences à l'université de Tlemcen et Slimane Zohra Maitre de conférences au centre universitaire d'Ain Temouchent qui, en tant que Directeurs de mémoire, se sont toujours montrés à l'écoute et très disponible tout au long de la réalisation de ce mémoire, pour l'inspiration, l'aide et le temps qu'ils ont bien voulu nous consacrer et sans leurs soutien ce mémoire n'aurait jamais vu le jour.

Nous exprimons notre gratitude à Monsieur Bouacha Abdelhafid, Maitre de conférences à l'université de Tlemcen, pour l'honneur qu'il nous fait en présidant notre Jury, ainsi qu'à Monsieur Slimani Hicham Maitre de conférences à l'université de Tlemcen, pour l'honneur qu'il nous fait en participant à notre jury. Nous les remercions sincèrement pour le temps qu'ils ont consacré à la lecture et à l'évaluation de notre travail.

Nous n'oublions pas nos parents pour leur contribution, leur soutien et leur patience. Enfin, nous adressons nos plus sincères remerciements à tous nos proches et amis, qui nous ont toujours encouragées au cours de la réalisation de ce mémoire.

Merci à tous et à toutes.

Résumé

L'infrastructure à clés publiques associée à la cryptographie asymétrique est la technologie sous-jacente qui assure la sécurité des protocoles Secure Sockets Layer (SSL) et HyperText Transfer Protocol Secure Sockets (HTTPS), qui sont largement utilisés pour tenir compte de l'échelle des transactions sur Internet en assurant l'authentification, la confidentialité et l'intégrité.

C'est dans ce contexte que s'inscrit ce projet de fin d'études qui vise à mettre en pratique une telle solution, qui se résume précisément dans la sécurisation du protocole http par SSL/TLS avec la prise en charge de l'authentification forte des deux parties de la transaction.

Nous présentons les éléments et les moyens nécessaires de compréhension de la problématique, d'installations et de configurations requises. Nous donnons ensuite les étapes de mise en œuvre suivies, ainsi que les résultats de validation.

Mots clés : sécurité, certificat, cryptographie asymétrique, authentification forte, SSL, TLS, HTTPS, PKI, Openssl, Wireshalk

Abstract

The public key infrastructure associated with asymmetric cryptography is the underlying technology that ensures the security of Secure Sockets Layer (SSL) and Hyper Text Transfer Protocol Secure Sockets (HTTPS) protocols, which are widely used to account for scale Internet transactions by providing authentication, confidentiality and integrity.
It is in this context that this end-of-studies project aims to put into practice such a solution, which can be summed up precisely in the SSL / TLS security of the HTTP protocol with the support of strong authentication of the two parts of the transaction.
We present the necessary elements and means of understanding the problem, the required installations and configurations. We then give the followed implementation steps, as well as the validation results.

Keywords : security, certificate, asymmetric cryptography, strong authentication, SSL, TLS, HTTPS, PKI, Openssl, Wireshalk

Table des matières

Chapitre III

Mise en pratique de la sécurisation du service http par TLS avec authentification mutuelle

Liste des figures

Liste des abréviations

ASN1	Abstract Syntax Notation 1
AES	Advanced Encryption Standard
Base64	Chiffrement Base64
BER	Basic Encoding Rules
CA	Certificate Autority Cert Certificate
CRL	Certificate Revocation List
CRT	Certificate
CSR	Certificate Signing Request
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DN	Distinguished Name
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
MAC	Message Authentification Code
MD5	Message Digest 5
NIST	Institute of Standards and Technology
NSA	National Security Agency
PEM	Privacy-Enhanced Mail
PKI	Public Key Infrastructure
RA	Registration Authority
RFC	Request for Comment
RSA	Rivest Shamir Adelman, Encryption and Signature Standard
SHA	Secure Hash Algorithm
SSL	Secure Socket Layer
TLS	Transport Layer Security

Introduction générale

Le besoin de sécurisation des échanges sur Internet n'est pas le même qu'il y a dix ans de cela. Aujourd'hui, il est très courant, de faire toute sorte de transactions bancaires, ou d'échanger des données sensibles professionnelles sur le réseau.

La cryptographie, en tant que moyen de sécurité, est devenue aujourd'hui une science à part entière, les banques font partie des premiers utilisateurs de systèmes cryptographiques, puis les navigateurs, ou browsers, tels que Mozilla Firefox ou Internet Explorer, utilisent le protocole de sécurité SSL (Secure Sockets Layers) basé sur des mécanismes cryptographiques. SSL signifie Secure Sockets Layer, une technologie standard destinée à la sécurité au niveau session de la connexion Internet et à la protection des données sensibles qui sont transmises entre deux systèmes. La version 3.1 du protocole SSL a été renommée en TLS pour Transport Layer Security (Sécurité de la Couche Transport).

L'objectif du travail de ce mémoire est la mise en pratique de la sécurité du protocole http via TLS avec authentification mutuelle forte (par certificats) des deux parties. Nous nous appuyons sur une infrastructure à clé publique qui sera créé au moyen de l'API Openssl. La plateforme de test comporte une autorité de certification, un serveur web Apache et un client (navigateur). Ce travail est conduit par émulation sur deux machines connectées par WLAN, ou l'une d'elles jouera un double rôle de d'autorité de certification et de serveur Web, et l'autre le client.

Le mémoire est organisé comme suit :
Dans le premier chapitre, nous allons passer en revue les spécifications du protocole TLS/SSL en analysant les formats des paquets et le fonctionnement de ses différents sous protocoles : TLS Handshake, TLS Change Cipher Spec, TLS Alert et TLS Record. Ceci, nous permettra de comprendre et vérifier le bon fonctionnement du protocole par un analyseur de réseau tel que Wireshark.

Dans le second chapitre, nous allons présenter le principe de fonctionnement d'une infrastructure à clés publiques (PKI), le format des certificats électroniques ainsi que les

formats de codage de ces certificats; ces notions sont d'une grande utilité dans la réalisation du travail pratique demandé.

Le troisième chapitre est une présentation du travail pratique effectué au cours de ce mémoire. Nous présenterons d'abord l'API Openssl, qui va être utilisée pour créer notre PKI et générer les certificats. Nous détaillerons son fonctionnement, ensuite nous présenterons de façon générale quelques commandes et manipulations.

Ensuite, il sera question des étapes que nous avons suivi pour la mise en pratique de la solution de sécurité visée, en l'occurrence la sécurisation du service http par TLS/SSL en mettant en évidence une authentification mutuelle forte entre les deux entités : le serveur Web (Apache) et le client (Browser). Nous allons créer des paires de clés et des certificats pour le CA, le serveur WEB et le client. Finalement, nous installerons et configurerons le serveur Apache avec les modules SSL appropriés qui permettront l'utilisation du protocole TLS/SSL pour effectuer des authentifications fortes coté serveur et coté client avec des échanges sécurisés. Le test de validation et l'analyse des paquets avec Wireshalk vont confirmer l'aboutissement du travail demandé.

Enfin, nous clôturerons ce mémoire avec une conclusion et des perspectives.

CHAPITRE I
Protocole SSL/TLS

I.1 Introduction

TLS est un protocole cryptographique conçu pour sécuriser une conversation qui consiste un arbitraire nombre de messages entre deux entités.

Le principal objectif de ce protocole est de fournir la confidentialité et l'intégrité des données entre deux machines communicantes authentifiées.

Le protocole SSL a été développé à Netscape, à l'époque où Netscape Navigator régnait sur Internet. La première version du protocole n'a jamais vu le jour, mais la prochaine version 2 était publiée en novembre 1994. Le premier déploiement a eu lieu dans Netscape Navigator 1.1, qui publié en mars 1995.Développé avec peu ou pas de consultation avec des experts en sécurité en dehors de Netscape, SSL 2 a pris fin être un mauvais protocole avec de graves faiblesses. Cela a forcé Netscape à travailler sur SSL 3, qui a été publié à la fin de 1995. Malgré le partage le nom avec les versions antérieures de protocole, SSL 3 était un tout nouveau protocole qui a établi le design que nous connaissons aujourd'hui.

En mai 1996, le groupe de travail TLS a été formé pour migrer SSL de Netscape vers l'IETF (TLS Groupe de travail).

TLS 1.0 a finalement été publié en janvier 1999, comme RFC 2246. La version suivante, TLS 1.1, n'a été publiée qu'en avril 2006 et ne contenait essentiellement que correctifs de sécurité.

TLS 1.2 a été publié en août 2008. Il a ajouté la prise en charge du chiffrement authentifié.

I.2 Présentation du TLS 1.2

Le protocole TLS se décompose en quatre modules : [11], [12]

- Le protocole TLS handshake
- Le protocole TLS Change Cipher Spec
- Le protocole TLS Alert

- Le protocole TLS Record

La sécurité n'est pas le seul objectif de TLS. Il a en fait quatre objectifs principaux, énumérés ici dans l'ordre de priorité :

a) Sécurité cryptographique :

Est le problème principal : permettre une communication sécurisée entre les deux parties qui souhaitent échanger des informations.

b) Interopérabilité :

Les programmeurs indépendants devraient être en mesure de développer des programmes et des bibliothèques qui sont capables de communiquer entre eux en utilisant des paramètres cryptographiques communs.

c) Extensibilité :

LS est effectivement un cadre pour le développement et le déploiement de protocoles cryptographiques. Son objectif important est d'être indépendant de la réalité des primitives cryptographiques (par exemple, des chiffrements et des fonctions de hachage) utilisées, permettant la migration d'une primitive à l'autre sans avoir besoin de créer de nouveaux protocoles.

d) Efficacité :

L'objectif final est d'atteindre tous les objectifs précédents à un coût de performance acceptable, réduire les opérations cryptographiques coûteuses au minimum et fournir une session système de mise en cache pour les éviter lors de connexions ultérieures.

I.3 Sous protocoles TLS 1.2

I.3.1 Protocole TLS Handshake (protocole de négociation)

La poignée de main (handshake) est la partie la plus élitiste du protocole TLS, au cours de laquelle les parties négocient les paramètres de connexion et effectuent l'authentification. Cette phase nécessite généralement six à dix messages, selon les fonctionnalités utilisées. Il peut y avoir beaucoup de variations dans l'échange, selon la configuration et les extensions de protocole supportées. En pratique, nous voyons trois cas : [11]

(1) Handshake complète avec l'authentification du serveur

(2) abrégé une prise de contact qui reprend une session antérieure

(3) une prise de contact avec l'authentification client et serveur.

Les messages de protocole Handshake échangés entre le client et le serveur contiennent trois champs :

Figure I.1 – Champs de TLS handshake

- Type : indique l'objet du message
- Length : indique la longueur du message
- Body (Content) : contient les données transmises

I.3.1.1 Handshake complète (full handshake)

Chaque connexion TLS commence par une poignée de main. Si le client n'a pas préalablement établi session avec le serveur, les deux parties exécuteront une poignée de main complète afin de négocier une session TLS. Pendant cette négociation, le client et le serveur effectueront quatre activités principales : [12], [14]

1. Échanger des capacités et se mettre d'accord sur les paramètres de connexion souhaités.

2. Validez le (s) certificat (s) présenté (s) ou authentifiez-vous en utilisant d'autres moyens.

3. Convenez d'un secret principal partagé qui sera utilisé pour protéger la session.

4. Vérifiez que les messages de prise de contact n'ont pas été modifiés par un tiers

Dans cette section, nous discutons de la prise de contact TLS la plus courante, entre un client qui n'est pas authentifié et un serveur. Le cas de l'authentification du client sera considéré dans le chapitre 3.

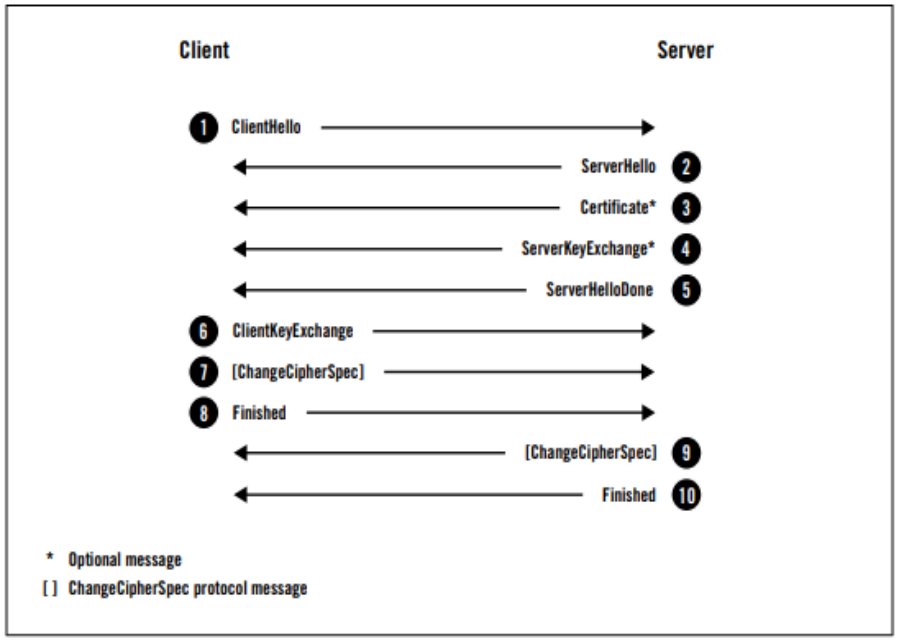

Figure I.2 – Les différents messages du protocole Handshake

1. Le client commence une nouvelle négociation et soumet ses capacités au serveur.

2. Le serveur sélectionne les paramètres de connexion.

3. Le serveur envoie sa chaîne de certificats (uniquement si l'authentification du serveur est requise).

4. Selon l'échange de clés sélectionné, le serveur envoie des informations supplémentaires requis pour générer le secret principal.

5. Le serveur indique l'achèvement de son côté de la négociation.

6. Le client envoie des informations supplémentaires requises pour générer le secret principal.

7. Le client passe au cryptage et informe le serveur.

8. Le client envoie un MAC des messages d'établissement de liaison qu'il a envoyés et reçus.

9. Le serveur passe au cryptage et informe le client.

10. Le serveur envoie un MAC des messages de prise de contact qu'il a reçus et envoyés.

À ce stade-en supposant qu'il n'y avait pas d'erreurs-la connexion est établie et les parties peuvent commencer à envoyer des données d'application.

Maintenant regardons les messages de poignée de main plus en détail.

a) ClientHello : Lorsqu'un client se connecte pour la première fois à un serveur, il doit envoyer le ClientHello comme premier message. Le client peut également envoyer un ClientHelloen réponse à une HelloRequest ou de sa propre initiative afin de renégocier les paramètres de sécurité dans une existante connexion

La structure du message ClientHello :

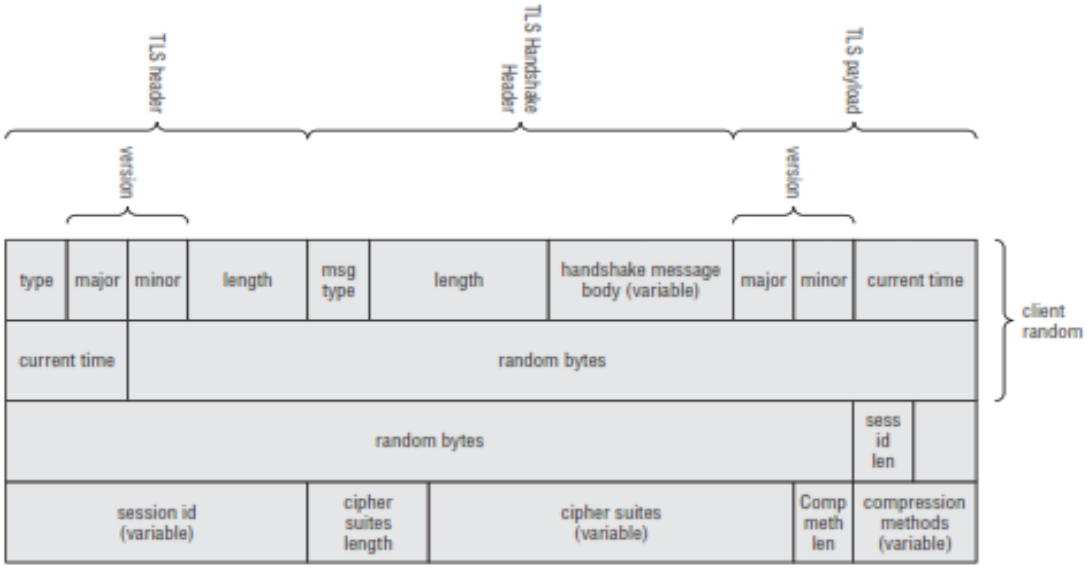

Figure I.3 – Structure de message client hello

Ce message contient les données suivantes :

1. Client_version : La version du protocole indique la meilleure version de protocole prise en charge par le client

2. Random : Le champ Random ou le champ aléatoire contient 32 octets de données. Parmi ceux-ci, 28 octets sont générés de manière aléatoire, les quatre octets restants portent des informations supplémentaires influencées par l'horloge du client (timestamp).

Le client et le serveur fournissent des données aléatoires au cours de la négociation. L'aléatoire rend chaque poignée de main (handshake) unique et joue un rôle clé dans

l'authentification en empêchant la relecture des attaques et vérifier l'intégrité de l'échange de données initial.

3. Session_id : c'est l'ID d'une session que le client souhaite utiliser pour cette connexion. Ce champ est vide si aucun session_id n'est disponible ou si le client souhaite générer de nouveaux paramètres de sécurité.

4. Cipher_suites : Le bloc de suite de chiffrement, il s'agit d'une liste des options cryptographiques supportées par le client, en commençant par la première préférence du client. Si le champ **session_id** n'est pas vide (ce qui implique une demande de reprise de session), ce vecteur doit inclure au moins la suite de chiffrement de cette session.

5. Compression_methods : Ceci est une liste des méthodes de compression supportées par le client, triées par préférence client. Si le champ session_id n'est pas vide (ce qui implique une demande de reprise de session), il doit inclure Compression_methods de cette session. Ce vecteur doit contenir, et toutes les implémentations doivent prendre en charge, Compression_Method.null.

Ainsi, un client et un serveur pourront toujours se mettre d'accord sur la méthode de compression.

6. extensions : Les clients peuvent demander des fonctionnalités étendues à partir des serveurs en envoyant des données dans le champ extensions. Dans le cas où un client demande des fonctionnalités supplémentaires en utilisant extensions, et ces fonctionnalités ne sont pas fournies par le serveur, le client peut abandonner la poignée de main.

Un serveur doit accepter ClientHello messages à la fois avec et sans le champ extensions, et (comme pour tous autres messages) il doit vérifier que la quantité de données dans le message correspond précisément à l'un des formats d'extension ; sinon, il doit envoyer une fatale "decode_error" alerte.

b) ServerHello : Le but du message ServerHello est que le serveur communique les paramètres de connexion sélectionnés au client. Le message est de structure similaire à ClientHello mais ne contient qu'une seule option par champ : Le serveur n'est pas requis pour supporter la même meilleure version supportée par le client. Si c'est non, il offre une autre version du protocole dans l'espoir que le client l'acceptera.

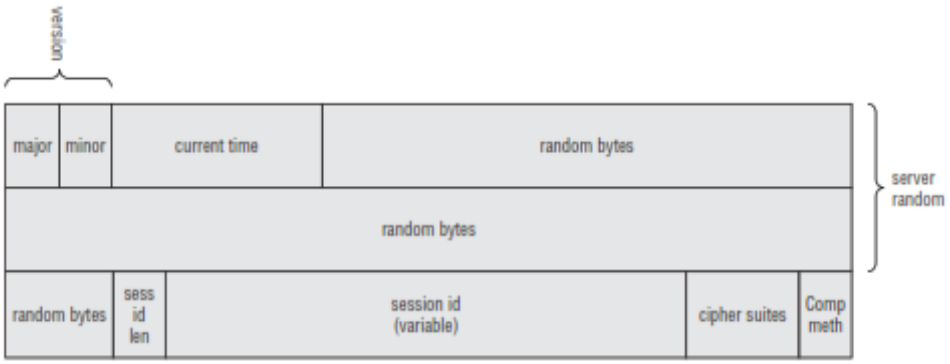

Figure I.4 – Structure du message Server hello

Ce message contient les données suivantes :

1. Server_version : Ce champ contiendra le plus bas de celui recommandé par le client dans le client bonjour et le plus haut supporté par le serveur.

2. Random : Cette structure est générée par le serveur et doit être généré indépendamment à partir de ClientHello. Random.

3. session_id : C'est l'identité de la session correspondant à cette connexion. Si le ClientHello. Session_id était non vide, le serveur cherchera dans son cache de session une correspondance. Si un match est trouvé et le serveur est prêt à établir la nouvelle connexion en utilisant l'état de session spécifié, le serveur répondra avec la même valeur que celle fournie par le client. Cela indique une reprise de la session et dicte que les parties doivent procéder directement aux messages finis. Sinon, ce champ contient une valeur différente identifiant la nouvelle session.
Le serveur peut renvoyer un ID de session vide pour indiquer que la session ne pas être mis en cache et ne peut donc pas être repris. Si une session est reprise, elle doit être reprise en utilisant la même suite de chiffrement initialement négociée.

4. Cipher_suites : La suite de chiffrement unique sélectionnée par le serveur à partir de la liste ClientHello. Cipher_suites. Pour les sessions reprises, ce champ est la valeur de l'état de la session en cours de reprise.

5. Compression_methods : L'algorithme de compression unique sélectionné par le serveur à partir de la liste dans ClientHello. Compression_methods. Pour les sessions reprises, Ce champ a la valeur de l'état de la session reprise.

6. Extensions : Une liste d'extensions. Seules les extensions proposées par le client peuvent apparaître dans la liste du serveur.

c) Certificat (Certificate) : Le serveur doit envoyer un message de certificat chaque fois que la méthode d'échange de clés utilise des certificats pour l'authentification. Ce message suivra toujours immédiatement le message ServerHello.

Ce message transmet la chaîne de certificats du serveur au client. Le certificat doit être approprié pour le chiffrement négocié, l'algorithme d'échange de clés de la suite et toutes les extensions négociées.

d) ServerKeyExchange (Échange de clés du serveur) : Ce message sera envoyé immédiatement après le certificat du serveur. Ce message transmet la clé publique du serveur utilisée par le client pour chiffrer les informations de clé de session. Le message ServerKeyExchange est envoyé par le serveur uniquement lorsque le message de certificat du serveur (si envoyé) ne contient pas assez de données pour permettre au client d'échanger un secret (encrypted pre-master key).

e) Certificate Request (Demande de certificat) : Un serveur non anonyme peut éventuellement demander un certificat du client, le cas échéant pour la suite de chiffrement sélectionnée. Ce message, s'il est envoyé, suivra immédiatement le message ServerKeyExchange (s'il est envoyé, sinon, ce message suit le message de certificat du serveur).

f) Server Hello Done : Le message ServerHelloDone est envoyé par le serveur pour indiquer fin du ServerHello et des messages associés. Après l'envoie du message, le serveur attendra une réponse du client qui peut procéder avec sa phase d'échange de clés. A la réception du message ServerHelloDone, le client devrait vérifier que le serveur a fourni un

certificat valide, si nécessaire, et vérifier que les paramètres du serveur Hello sont acceptables.

g) ClientKeyExchange : Ce message est toujours envoyé par le client. Cela doit immédiatement suivre le message du certificat client, s'il est envoyé. Autrement, il doit être le premier message envoyé par le client après qu'il reçoit le message ServerHelloDone. Avec ce message, le secret pre-master key est fixé, soit directement par transmission du secret chiffré par RSA ou par la transmission des paramètres Diffie-Hellman qui permettront à chaque partie de se mettre d'accord sur le même secret pre-master key.

h) Finished : Le message Finished est le signal que la prise de contact est terminée. Son contenu est crypté, qui permet aux deux parties d'échanger en toute sécurité les données nécessaires pour vérifier l'intégrité de la poignée de main entière.

I.3.1.2 Échange de clés (Key exchange)

L'échange de clés est la partie la plus intéressante de la poignée de main. En TLS, la sécurité de la session dépend d'une clé partagée de 48 octets appelée master key. L'objectif de l'échange de clés est de générer une autre valeur, le secret pre-master key, qui est la valeur à partir de laquelle la clé (master key) est construite.

TLS prend en charge de nombreux algorithmes d'échange de clés afin de prendre en charge divers types de certificats, algorithmes à clé publique et protocoles d'établissement de clé.

I.3.1.2.1 Échange de clés RSA (RSAKey exchange)

L'échange de clés RSA est assez simple ; le client génère un nombre aléatoire, le crypte avec la clé publique du serveur et l'envoie dans le message ClientKeyExchange. Pour obtenir le secret pre-master key, le serveur doit uniquement déchiffrer le message.

La simplicité de l'échange de clés RSA est aussi sa principale faiblesse. Le secret pre-master key est crypté avec la clé publique du serveur, qui reste généralement utilisée pendant plusieurs années. Toute personne ayant accès à la clé privée correspondante peut récupérer le secret pre-master key et construire le même secret maître et compromettre la sécurité de la session.

L'attaque ne doit pas se produire en temps réel. Un adversaire puissant pourrait établir un long terme opération pour enregistrer tout le trafic crypté et attendre patiemment jusqu'à ce qu'il obtienne la clé.

Les autres mécanismes d'échange de clés communs utilisés dans TLS ne souffrent pas de ce problème et sont censés soutenir le secret à terme.

I.3.1.2.2 Échange de clés avec Diffie-Hellman

L'échange de clés Diffie-Hellman (DH) est un protocole d'accord clé qui permet à deux parties d'établir un secret partagé sur un canal de communication non sécurisé. Le protocole utilise une fonction à sens unique basé sur le logarithme discret.

I.3.1.3 Suites de chiffrement (Cipher Suites)

TLS permet une grande flexibilité dans la mise en œuvre de la sécurité souhaitée. C'est effectivement un cadre pour créer des protocoles cryptographiques réels. Bien que les versions précédentes codent en dur certaines primitives cryptographiques dans le protocole, TLS1.2 est entièrement configurable. Une suite de chiffrement est une sélection de primitives cryptographiques et d'autres paramètres qui définissent exactement comment la sécurité sera implémentée. Une suite est définie à peu près par les attributs suivants :

- Méthode d'authentification
- Méthode d'échange de clés
- Algorithme de chiffrement
- Taille de la clé de chiffrement
- Mode de chiffrement
- Algorithme MAC
- PRF (uniquement TLS 1.2-dépend du protocole)
- Fonction de hachage utilisée pour le message Finished (TLS 1.2)
- Longueur de la structure Verify_data (TLS 1.2)

Les noms des suites de chiffrement ont tendance à être longs et descriptifs et assez cohérents : ils sont faits à partir des noms de la méthode d'échange de clés, de la méthode d'authentification, de la définition de chiffrement et algorithme MAC ou PRF optionnel.

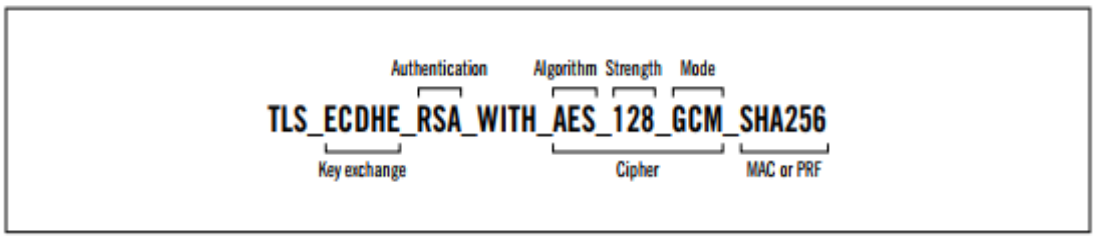

Figure I.5 – construction du nom de la suite de chiffrement

Le tableau suivant représente des exemples de noms de suites de chiffrement et de leurs propriétés de sécurité :

Cipher Suite Name	Auth	KX	Cipher	MAC	PRF
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	RSA	ECDHE	AES-128-GCM	-	SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDSA	ECDHE	AES-256-GCM	-	SHA384
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	RSA	DHE	3DES-EDE-CBC	SHA1	Protocol
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	RSA	AES-128-CBC	SHA1	Protocol
TLS_ECDHE_ECDSA_WITH_AES_128_CCM	ECDSA	ECDHE	AES-128-CCM	-	SHA256

Figure I.6 – Exemples de noms de suites de chiffrement et de leurs propriétés de sécurité

I.3.2. Le protocole TLS Change Cipher Spec

ChangeCipherSpec est un protocole TLS qui n'est pas réellement un message d'établissement de liaison TLS. Ce message de changement de spécification de chiffrement est un message de marqueur, tout comme le Server Done, cela n'inclut aucune donnée, et qui indique le passage du serveur en mode chiffré avec la clé master. Ce protocole sert donc à modifier la stratégie de chiffrement utilisée dans une session.

I.3.3. Le protocole TLS Alert

Les alertes ont pour but d'utiliser un simple mécanisme de notification pour informer l'autre partie de circonstances exceptionnelles. Ils sont généralement utilisés pour les messages d'erreur, à l'exception de close_notify, qui est utilisé lors de l'arrêt de la connexion. Les alertes sont très simples et ne contiennent que deux champs (un octet chacun):

Le champ **AlertLevel** porte la gravité de l'alerte, qui peut être soit d'avertissement soit fatale.

AlertDescription est simplement un code d'alerte. Les messages fatals entraînent la fin immédiate de la connexion et de l'invalidation de la session en cours alors que l'envoi d'une notification d'avertissement ne met pas fin à la connexion, mais le côté récepteur est libre de réagir à l'avertissement en envoyant une alerte fatale.

Le protocole d'Alerte sert donc à signaler une erreur dans la communication, comme une erreur dans le calcul du MAC, ou une impossibilité de s'authentifier pour le client.

Voici quelques codes d'alertes :

Alert Code	Alert Message
0	close_notify
10	unexpected_message
20	bad_record_mac
21	decryption_failed
22	record_overflow
30	decompression_failure
40	handshake_failure
42	bad_certificate
43	unsupported_certificate
44	certificate_revoked
45	certificate_expired
46	certificate_unknown
47	illegal_parameter
48	unknown_ca
49	access_denied
50	decode_error
51	decrypt_error
60	export_restriction
70	protocol_version
71	insufficient_security
80	internal_error
90	user_cancelled
100	no_renegotiation
255	unsupported_extension

I.3.4 Protocole TLS Record

La couche basse de SSL le « Record Layer » ou « Record Protocol » est la couche encapsulant tous les types de données SSL : données brutes (application par ex. http) et données de gestion de la connexion. Le protocole TLS Record est en charge du transport et éventuellement le chiffrement de tous les messages de niveau inférieur échangés sur une connexion. Chaque enregistrement TLS commence par un en-tête court contenant des

informations sur l'enregistrement type de contenu (ou sous-protocole), version du protocole et longueur. Les données de message suivent l'en-tête.

Le traitement que subit un segment arrivant de la couche haute du SSL est comme suit :

- Fragmentation
- Compression (Optionelle)
- Signature (MAC - Message Authentication Code)
- Chiffrement

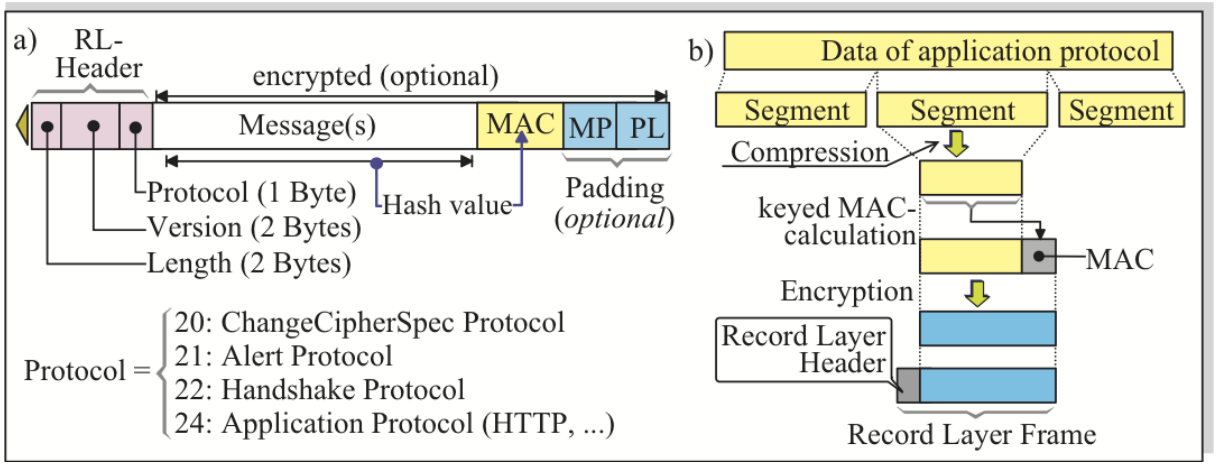

Figure I.7 – Format des messages du protocole TLS record

Voici le détail des opérations du protocole TLS Record :

- **Fragmentation**

Lorsqu'un segment de données provient de la couche haute, il est fragmenté en segments de taille inférieure à 2^{14} = 16384 octets.

- **Compression**

La compression est optionnelle. En fait, elle est toujours active, mais sans effet par défaut. Elle est inutile lorsque le taux de transfert est élevé, où elle crée un surcout de traitement.

- **Signature MAC - Message Authentication Code**

Le MAC est une signature de chaque segment de donnée qui passe dans la connexion. Le MAC sert à prouver qu'un message n'a pas été corrompu durant le transfert (il assure donc

l'intégrité du message). Il s'agit d'une signature HMAC (0, 16 ou 20 octets) générée à l'aide d'une fonction de hachage et des clés produites lors de la négociation.

- **Chiffrement**

Le chiffrement peut être de deux grands types : chiffrement par bloc, tel que AES, ou chiffrement de flux, tel que RC4. L'algorithme de chiffrement étant défini au début de la connexion, le chiffrement consiste à chiffrer le message accompagné du MAC, pour assurer la confidentialité du message.

- **Entête**

Un en-tête de 5 octets est ajouté. Le champ "Protocol" (1 octet) de cet en-tête définit le type du protocole de niveau supérieur au Record Protocol. Les types sont les suivants:

0x20 – Paquet de type Change Cipher Spec
0x21 – Paquet de type Alert
0x22 – Paquet de type Handshake
0x23 – Paquet de type Application Data : ce type correspond aux données effectives de la transaction SSL.

- **Décapsulation**

Lors de la remontée des données dans la pile des protocoles, les opérations sont menées inversement : déchiffrement, contrôle d'intégrité, décompression, concaténation des messages, puis envoi à la couche haute SSL.

I .4 Conclusion

Dans ce chapitre nous avons présenté en détail le protocole TLS avec ses différents sous-protocoles (TLS Handshake, TLS Change Cipher Spec, TLS Alert et TLS Record). Cette synthèse s'avère très utile voire indispensable pour la mise en pratique de la solution de sécurité proposée dans ce mémoire. Dans le chapitre suivant, nous allons examiner les notions de certificats et d'infrastructure à clé publique associées à la cryptographie asymétrique pièce maîtresse du protocole TLS.

CHAPITRE II

Infrastructure à clé publique (PKI) et Certificat électronique

II.1 Introduction

Une Infrastructure à clé publique (Public Key Infrastructure : PKI) permet aux utilisateurs d'un réseau public non sécurisé comme l'Internet d'échanger des données en toute sécurité et de façon privée au moyen d'une paire de clés cryptographiques publiques et privées obtenues et partagées par une autorité de confiance. La PKI prévoit des certificats numériques qui permettent d'identifier les personnes ou les organisations et des services d'annuaire qui peuvent les stocker et, au besoin, les révoquer. [1] - [3]

La PKI est la technologie sous-jacente qui assure la sécurité des protocoles Secure Sockets Layer (SSL) et HyperText Transfer Protocol Secure Sockets (HTTPS), qui sont largement utilisés pour mener des affaires électroniques sécurisées sur Internet. La PKI suppose l'utilisation de la cryptographie à clé publique, qui est la méthode la plus courante sur Internet pour l'authentification d'un expéditeur de message ou le cryptage d'un message. La cryptographie traditionnelle implique la création et le partage d'une clé secrète pour le chiffrement et le déchiffrement des messages.

Ce système de clé secrète présente l'inconvénient important que si la clé est découverte ou interceptée par quelqu'un d'autre, les messages peuvent facilement être décryptés. Pour cette raison, la cryptographie à clé publique et l'infrastructure à clé publique est l'approche privilégiée sur Internet.

II.2 Présentation de l'Infrastructure à clé publique

Une infrastructure à clés publiques est un ensemble de composants physiques (ordinateurs, des équipements cryptographiques logiciels ou matériels type HSM : Hardware Security Module ou encore des cartes à puces), de procédures humaines (vérifications, validation) et de logiciels (système et application) destiné à gérer les clés publiques des utilisateurs d'un système. [6]

Une PKI permet de lier des clés publiques à des identités (comme des noms d'utilisateurs ou d'organisations). Elle fournit des garanties permettant de faire a priori confiance à une clé publique obtenue par son biais.

Des normes internationales décrivent les éléments fonctionnels de base d'une PKI : l'autorité d'enregistrement, l'autorité de certification (Certification Authority : CA) et un service de publication (annuaire). [15]

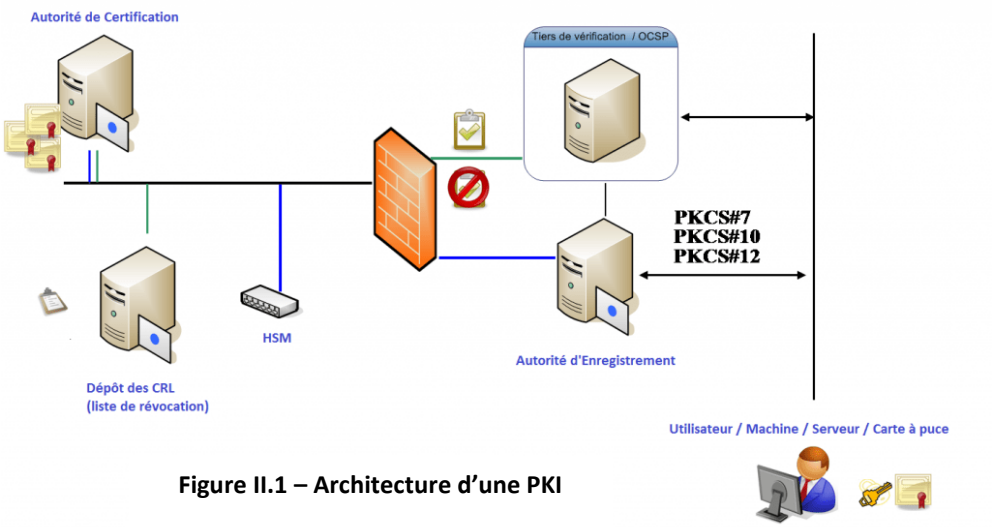

Figure II.1 – Architecture d'une PKI

II.2.1 L'autorité d'enregistrement (AE)

L'autorité d'enregistrement (AE) ou Registration Authority (RA) vérifie l'identité du demandeur, s'assure que celui-ci possède bien un couple de clés privée-publique et récupère la clé publique du demandeur. Elle transmet ensuite ces informations (informations d'identité du demandeur ainsi que sa clé publique) à l'autorité de certification. Une autorité d'enregistrement peut être une personne habilitée, dans un secrétariat.

II.2.2 CA : L'autorité de certification (Certification Authority)

Celle-ci reçoit les demandes de création de certificats venant des autorités d'enregistrement. Elle crée les certificats, signe ces certificats en utilisant sa clé privée et les transmet aux utilisateurs et au service de publication. La clé privée d'une autorité de certification est un secret qui nécessite une protection maximale. Un tiers qui prendrait connaissance de cette clé, pourrait générer des faux certificats. Une méthode pour assurer cette protection est de

stocker ce secret sur un ordinateur dédié, enfermé dans un coffre, jamais connecté sur un réseau, que l'on utilise uniquement pour créer les certificats.

II.2.3 Service de publication (annuaire)

Celui-ci rend disponible les certificats émis par l'autorité de certification. Il publie aussi la liste des certificats valides et des certificats révoqués. Concrètement ce service peut être rendu par un annuaire électronique LDAP (Lightweight Directory Access Protocol) ou un serveur web accessibles par l'internet.

II.2.4 CRL, Certificate Revocation List

Lorsqu'un certificat est créé, il contient sa date de création et une date de fin de validité. Passée cette date aucune application ne l'acceptera. Mais cette date n'est pas suffisante pour invalider un certificat dans certains cas. En effet, une personne peut quitter une entreprise ou changer de service ou se faire dérober sa clé secrète. Dans ce cas il faut invalider son certificat courant. La méthode est la même que pour les cartes bancaires. Chaque autorité de certification publie régulièrement la liste des certificats révoqués, qui ne sont plus valides. Cette liste est généralement dans un annuaire LDAP, accessible par le web. Pour garantir son origine et son intégrité, elle est signée par l'autorité de certification (CA).

II.3 Services de sécurité offerts par une PKI

Les cinq exigences en matière de cyber sécurité sont la non-répudiation, la confidentialité, l'intégrité, la responsabilité et la confiance. Une mise en œuvre réussie de la PKI joue un rôle important dans la satisfaction de ces cinq exigences. [7]

- **Non-Répudiation**

Pour qu'une transaction commerciale soit valable, aucune des parties ne peut plus tard nier l'existence ou l'exécution de cette transaction. La PKI utilise des signatures numériques pour lier l'identité d'une partie à la transaction de sorte que la connaissance de la transaction ne peut plus être refusée par la suite.

- **Confidentialité**

La PKI offre la protection de la vie privée par le biais du chiffrement à clé publique et privée.

Ce système permet à des parties non liées de mener leurs activités en toute sécurité sur un réseau non protégé.

- **Intégrité**

La PKI offre l'intégrité grâce aux signatures numériques, qui peuvent être utilisées pour prouver que les données n'ont pas été altérées en transit. C'est important en soi, mais aussi pour la non-répudiation.

- **Responsabilité**

La PKI permet de rendre des comptes en vérifiant l'identité des utilisateurs au moyen de signatures numériques. Comme les signatures numériques sont plus sûres que les combinaisons nom d'utilisateur/mot de passe, les utilisateurs sont plus susceptibles d'être tenus responsables de leurs actions.

- **Confiance**

Tout le concept d'une PKI repose sur la confiance. Vous faites confiance à l'autorité de certification émettrice (CA). Si vous ne faites pas confiance à l'AC qui a délivré le certificat, vous ne pouvez pas faire confiance aux certificats qu'ils ont délivrés, ni aux organismes auxquels ils ont été délivrés. Ce n'est pas la faute des organisations, mais de l'AC elle-même.

II.4 Certification

La certification est la fonction fondamentale de toutes les PKI. Les certificats fournissent un moyen sûr de publier les clés publiques, de sorte que leur validité peut être fiable, c’est donc une preuve de la correspondance entre une clé publique et un ensemble d'informations (nom, prénom, email,...)

II.4.1 Certification du sujet

Les procédures initiales utilisées pour certifier les clés publiques de l'utilisateur sont les plus importantes, car une tentative de mascarade réussie pendant cette phase peut être difficile à détecter ultérieurement. Cela peut être fait automatiquement par des PKI "internes" (dirigées par une entreprise pour ses employés), mais dans d'autres cas, elles peuvent exiger

une authentification par la CA ou la RA, et même une validation de documents réels comme des passeports ou des cartes d'identité.

II.4.2 Format de certificat

L'autorité de certification crée un certificat en signant une collection d'informations sur l'entité. Ces informations comprennent la clé publique et le nom distinctif de l'entité et peuvent inclure un identificateur unique facultatif qui contient des informations supplémentaires sur l'entité. Le format standard d'un certificat est défini selon la norme X.509 (actuellement X509v3) : [8]

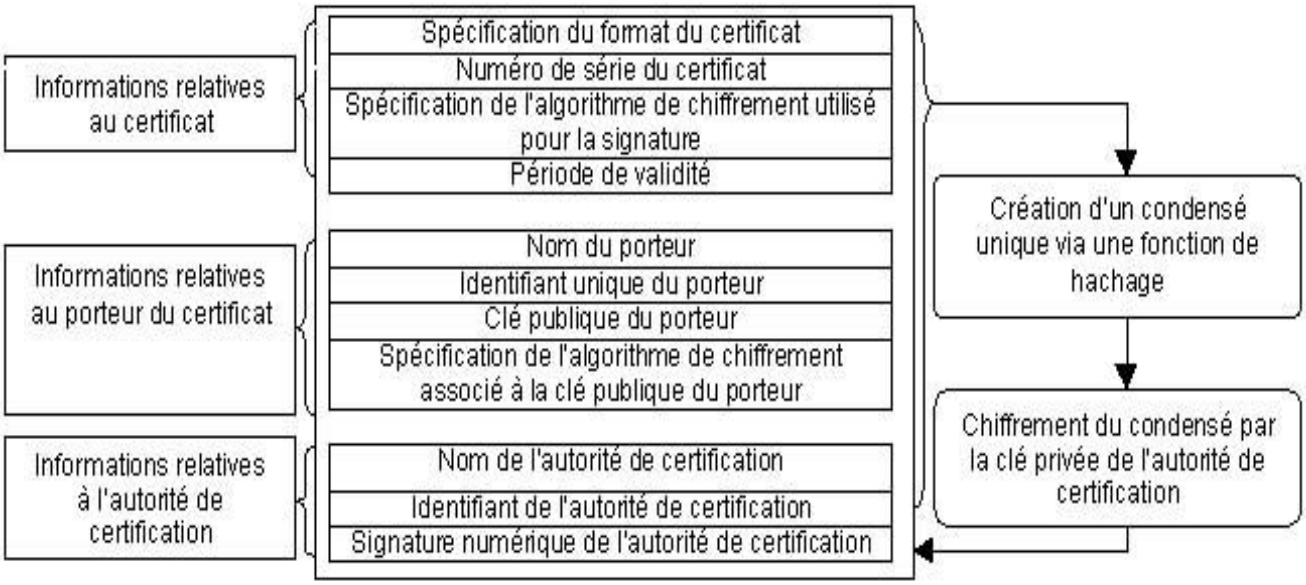

Figure II.2 – Format d'un certificat X.509

L'authentification par certificats X.509 est utilisée sur support physique (carte à puce ou Token USB) ou support logiciel. C'est une authentification forte qui fait intervenir une signature d'un challenge avec la clé privée de l'utilisateur. Le serveur d'authentification vérifie l''identité de l'utilisateur grâce au certificat, et il l'authentifie en vérifiant sa signature avec la clé publique présente dans le certificat.

II.4.3 Extensions des fichiers X.509

PEM Format : C'est le format le plus commun utilisé pour les certificats. La plupart des serveurs s'attendent à ce que les certificats et la clé privée se trouvent dans des fichiers séparés

- Généralement, ils sont des fichiers ASCII codés Base64
- Les extensions utilisées pour les certificats PEM sont .cer, .crt, .pem, .key files
- Apache et un serveur similaire utilisent des certificats de format PEM

DER Format : Le format DER est la forme binaire du Certificat. Tous les types de certificats et clés privées peuvent être encodés au format DER. Les certificats au format DER ne contiennent pas les déclarations "BEGIN CERTIFICATE / END CERTIFICATE"

- Les certificats au format DER utilisent le plus souvent les extensions '.cer' et '.der'

P7B/PKCS#7 Format : Le format PKCS # 7 ou P7B est stocké au format Base64 ASCII et possède une extension de fichier de .p7b ou .p7c

- Un fichier P7B ne contient que des certificats et des certificats de chaîne (CA intermédiaires), et non la clé privée

PFX/P12/PKCS#12 Format : Le format PKCS#12 ou PFX/P12 est un format binaire pour stocker le certificat du serveur, les certificats intermédiaires et la clé privée dans un fichier chiffré.

- Ces fichiers ont généralement des extensions telles que .pfx et .p12
- Ils sont généralement utilisés sur les machines Windows pour importer et exporter des certificats et des clés privées

OpenSSL peut être utilisé pour convertir le certificat au format approprié.

II.5 Conclusion

Dans ce chapitre nous avons présenté le principe de fonctionnement d'une infrastructure à clés publiques, le format des certificats électroniques ainsi que les formats de codage de ces certificats; ces notions sont d'une grande utilité dans ce qui suit.

Chapitre III

Mise en pratique de la sécurisation du service http par TLS avec authentification mutuelle

III.1 Introduction

Dans ce chapitre, nous allons présenter les étapes que nous avons suivies dans l'implémentation du protocole TLS 1.2 pour sécuriser un site Web [9], [10]. L'objectifs est le suivant : des connexions Web sécurisées https avec authentification mutuelle par certificat X.509, et du serveur et du client. Pour se faire, nous avons utilisé l'outil Openssl.

Pour émuler, une connexion http sur Internet, nous avons utilisé deux machines connectées par un LAN : la première jouera deux rôles, celui du CA et du serveur Web (Windows10) et la seconde le client. Des paires de clés et des certificats seront créés pour le CA , le serveur WEB et le client.

III.2 Présentation d'Openssl

OpenSSL est une implémentation open source du protocole SSL et TSL pour une communication sécurisée, fournissant de nombreuses opérations cryptographiques comme le chiffrement et le déchiffrement des données, la création et la vérification de signatures, le calcul des paires de clés publiques et privées et le traitement des certificats. [4], [5]

Openssl est disponible pour presque tous les systèmes d'exploitation et ses propres fonctions API sont disponibles pour le développement d'applications sécurisées. Openssl offre :

1. une bibliothèque de programmation en C permettant de réaliser des applications client/serveur sécurisées s'appuyant sur SSL/TLS.

2. une commande en ligne (openssl) permettant :

- la création de clés RSA, DSA (signature)
- la création de certificats X509
- le calcul d'empreintes (MD5, SHA,...)
- le chiffrement et déchiffrement (DES, IDEA, RC2, RC4, Blowfish, ...)

- la réalisation de tests de clients et serveurs SSL/TLS

III.3 Installation d'Openssl sur Windows 10

Pour effectuer certaines opérations de cryptographie (création d'une clé privée, génération d'un CSR, conversion d'un certificat...) sur un poste Windows 10 nous avons installé Openssl 1.0.2k. Voici les étapes d'installation :

1. Télécharger le fichier d'installation d'openssl « Win32OpenSSL_Light-1_0_2k » disponible sur le site : [16]
   On choisit suivant les besoins la version Windows 32 bits ou 64 bits

2. Exécuter le fichier Win32OpenSSL_Light-1_0_2k

3. Cliquer sur Next (Suivant).

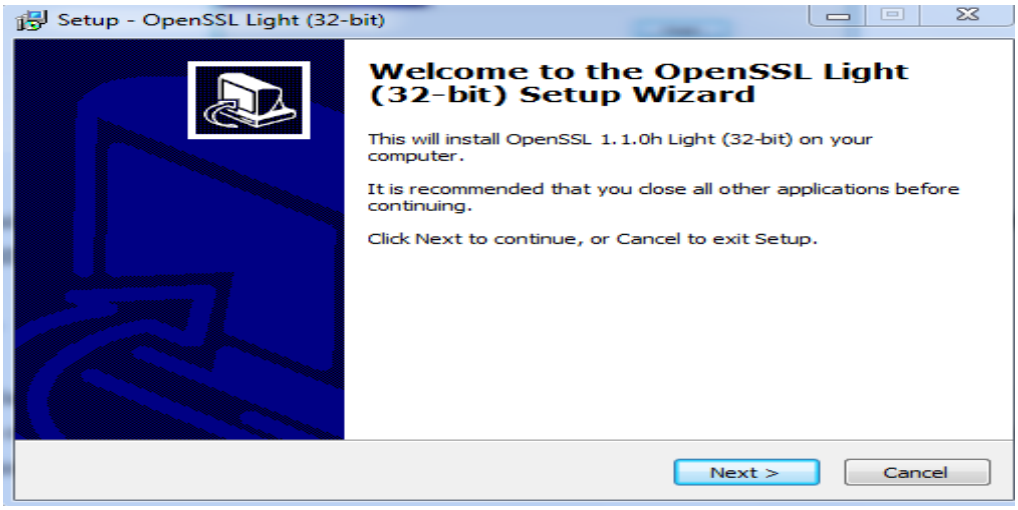

Figure III.1 - Première étape de l'installation d'openssl

4. Sélectionner « accept the terms in the License Agreement » (J'accepte les termes du contrat de licence), puis cliquez sur Next (Suivant)

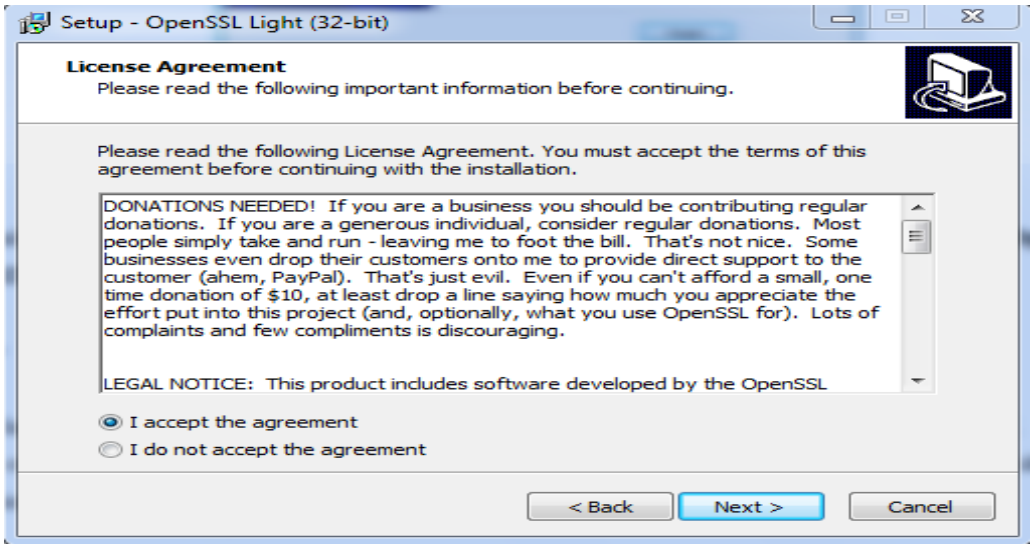

Figure III.2 - Deuxième étape de l'installation d'openssl

5. Indiquer le chemin d'accès pour installer openssl, puis cliquez sur Next (suivant).

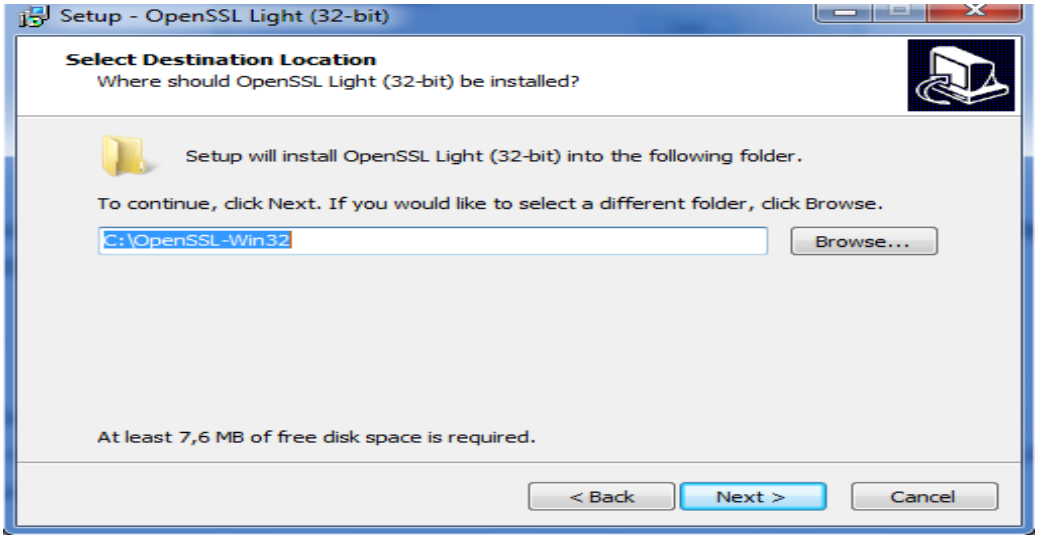

Figure III.3 - Troisième étape de l'installation d'openssl

6. Copier les DLLs Openssl dans le dossier d'installation du binaire Openssl

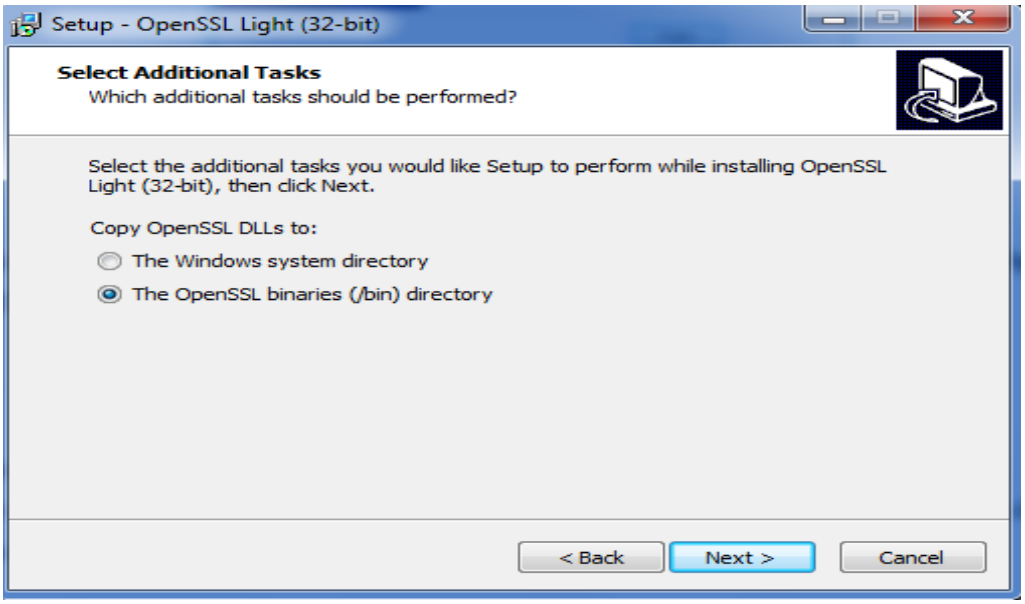

Figure III.4 - Quatrième étape de l'installation d'openssl

7. Lorsque la boîte de dialogue suivante apparaît, cliquer sur Next (suivant).

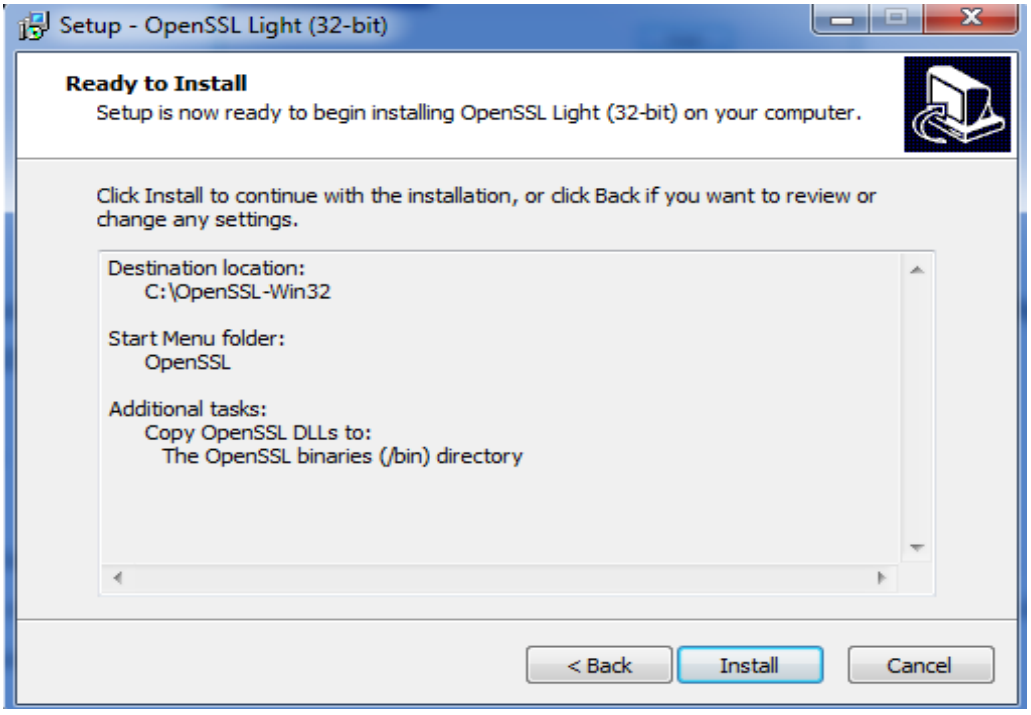

Figure III.5 - Cinquième étape de l'installation d'openssl

9. Quand l'installation et terminer, cliquer sur finish (terminer)

Figure III.6 - Sixième étape de l'installation d'openssl

L'installation standard d'Openssl sur un poste Windows est effectuée sur "C:\OpenSSLWin32" et l'exécutable est situé dans le sous répertoire "bin".

Pour lancer l'invite de commande, aller dans le menu démarrer, puis exécuter "cmd". Pour exécuter le programme dans "l'invite de commandes" Windows il faudra donner le chemin complet : >C:\OpenSSL-Win32\bin\openssl.exe

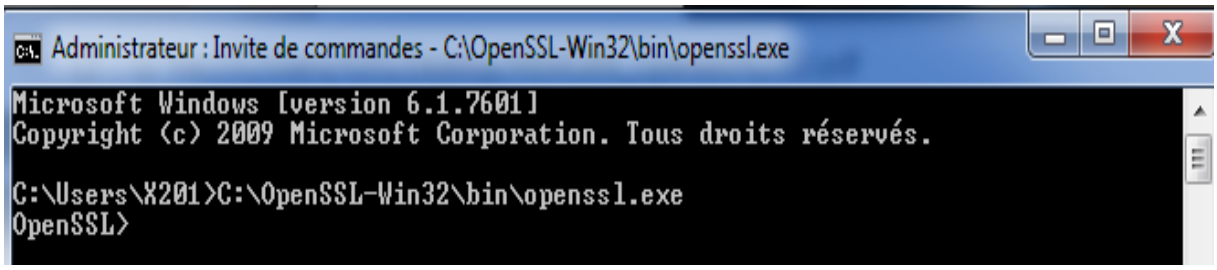

Figure III.7 - Lancement d'openssl

L'API Openssl fournit une variété de commandes dont chacune possède de nombreuses options et arguments. Les commandes-pseudo commandes-standard-liste, commandes-signature-message-liste, et commande-chiffrement-liste génèrent respectivement une liste

des noms de toutes es commandes standards, commandes de signature de messages ou commandes de chiffrement, respectivement, qui sont disponible dans l'utilitaire Openssl.

La syntaxe générale de la commande openssl est :

Openssl <commande> <options >

Voici une liste non exhaustive des commandes les plus utilisées : [5],[13]

• openssl genrsa -out <fichier_rsa.priv> <size>

Cette commande génère la clé privée RSA de taille size. Les valeurs possibles pour size sont : 512, 1024, etc.

• openssl rsa -in <fichier_rsa.priv> -des3 -out <fichier.pem>

Cette commande chiffre la clé privée RSA avec l'algorithme DES3. Nous pouvons utiliser DES, 3DES, AES, etc.

• Openssl rsa -in <fichier_rsa.priv> -pubout -out <fichier_rsa.pub>

Cette commande stocke la partie publique dans un fichier à part (création de la clé publique associée à la clé privée).

• openssl enc <-algo> -in <File.txt> -out <Filechiff.enc>

Cette commande permet de chiffrer un texte clair File.txt avec l'algorithme spécifié (openssl enc --help pour avoir la liste des possibilités ou bien openssl List-cipher-commandes) dans un fichier Filechiff.enc.

• openssl enc <-algo> -in < Filechiff.enc > -d -out < File.txt >

pour le déchiffrement.

• Openssl dgst <-algo> -out <sortie> <entrée>

Cette commande est utilisée pour hacher un fichier. L'option <-algo> est le choix de l'algorithme de hachage (MD5, SHA1, SHA2, …..).

• Openssl rsautl –encrypt -inkey <rsa.pub> -in < File.txt > -out < Filechiff.enc >

Cette commande permet de chiffrer File.txt avec RSA en utilisant la clé publique rsa.pub.

• Openssl rsautl -decrypt -inkey <rsa.priv> -in < Filechiff.enc > -out < File.txt >

Cette commande sert à déchiffrer Filechiff.enc

• Openssl rsautl -sign -inkey <rsa.priv> -in <File.txt> -out <Filesign.sig>

Cette commande permet de générer une signature.

• Openssl rsautl -verify -pubin -inkey <rsa.pub> -in Filesign.sig

Cette commande sert à vérifier une signature

• pour afficher la clé au format pem : Openssl rsa -in <rsa.priv>

• pour afficher la clé au format text : Openssl rsa -in <rsa.priv> -text –nout

• pour afficher la clé publique : Openssl rsa -pubin –in fichier_rsa.pub -text -noout

Les commandes nous permettant de générer les CSR et Certificats seront vues dans la suite de ce chapitre.

III.4 Apache 2.4.33 sur Windows 10

Apache est un serveur web ; un serveur web est un logiciel permettant à des clients d'accéder à des pages web, c'est-à-dire en réalité des fichiers au format HTML à partir d'un navigateur installé sur un ordinateur distant ; il est capable d'interpréter les requêtes http arrivant sur le port associé (par défaut le port 80).

Apache est le serveur web le plus répandu sur internet , téléchargeable du site : https://httpd.apache.org/download.cgihttps://www. Apache lounge.com

Une fois installé, on peut démarrer le service :

Figure III.8 - Lancement du serveur apache

Pour vérifier le bon fonctionnement de serveur apache, depuis un navigateur web (exemple Mozilla Firefox) , on introduit les indéfférement URL suivantes :

http://localhost

ou http://127.0.0.1

ou http://192.168.1.2 : adresse IP de notre serveur

ou http://serverweb : le nom de notre machine hébergeant notre serveur

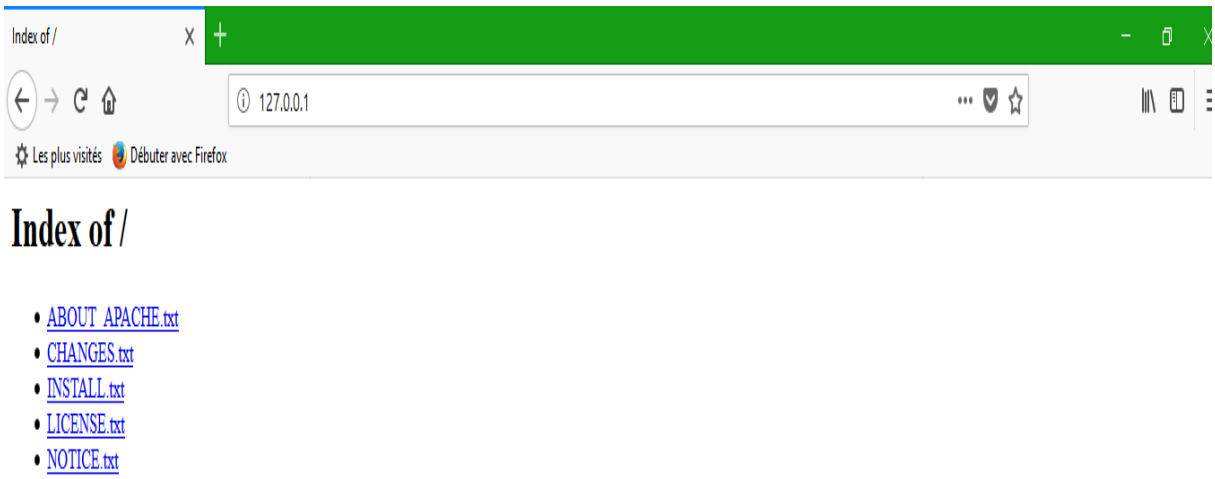

Figure III.9- Fonctionnement de http sur W10

III.5 Etapes de mise en pratique de la solution de sécurité proposée

III.5.1 Création du réseau entre deux ordinateurs

Tout d'abord, nous créons un réseau WLAN pour connecter le serveur et le client, et cela en utilisant les deux commandes suivants :

Pour autoriser le mode réseau hébergé on utilise la commande :

- netsh wlan set hostednetwork mode=allow ssid=RESEAUPC key=ilhemmez

et puis pour démarrer ce réseau on utilise la commande suivante

- netsh wlan start hostednetwork

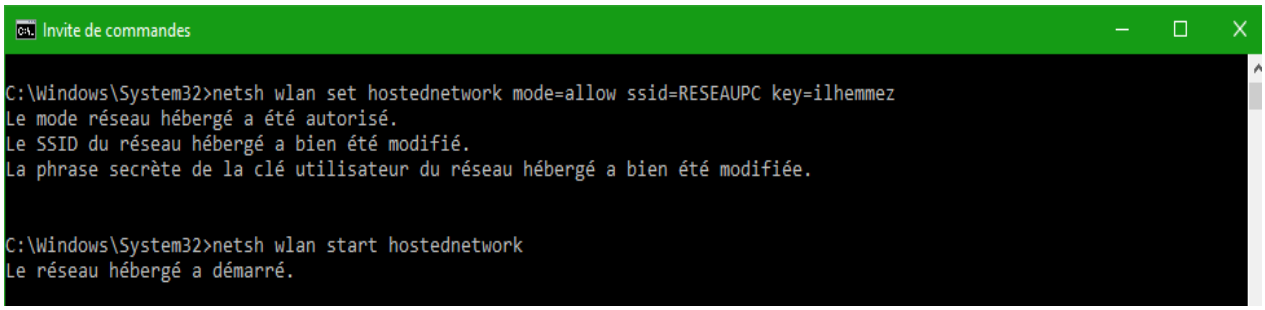

Figure III.10– création du réseau entre deux pc (serveur et client)

Ensuite on va attribuer les addresse IP : 192.168.1.2 comme adresse du serveur et l'adresse 192.168.1.4 comme adresse du client ; finallement, nous faisons un test de la connectivité:

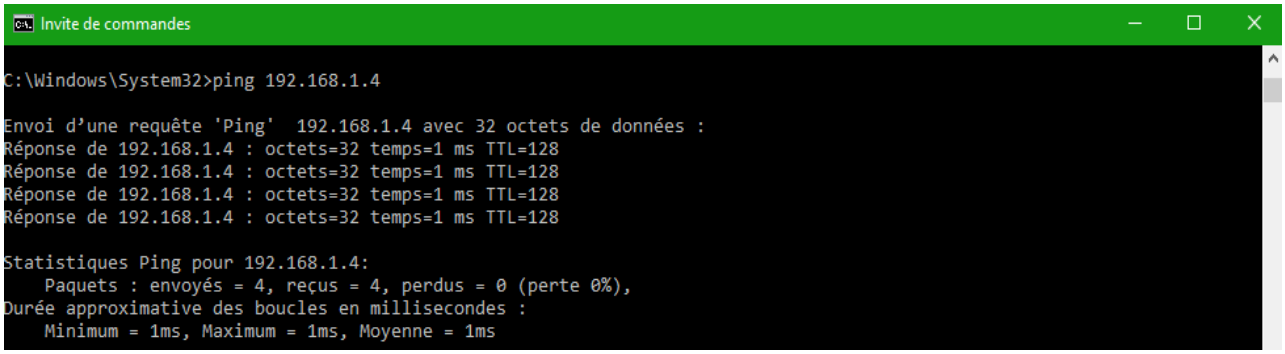

Figure III.11 - Capture d'écran Ping sur w10

III.5.2 Création du CA : Autorité de Certification

Notre machine W10 va faire office de CA avec un certificat x509 (auto signé). Ce certificat racine doit être protégé par un mot de passe et disposer d'une durée de validité non nulle.

- Génération d'une paire de clés privée/publique

  On utilise la commande : genrsa -out cleCA.key 2048

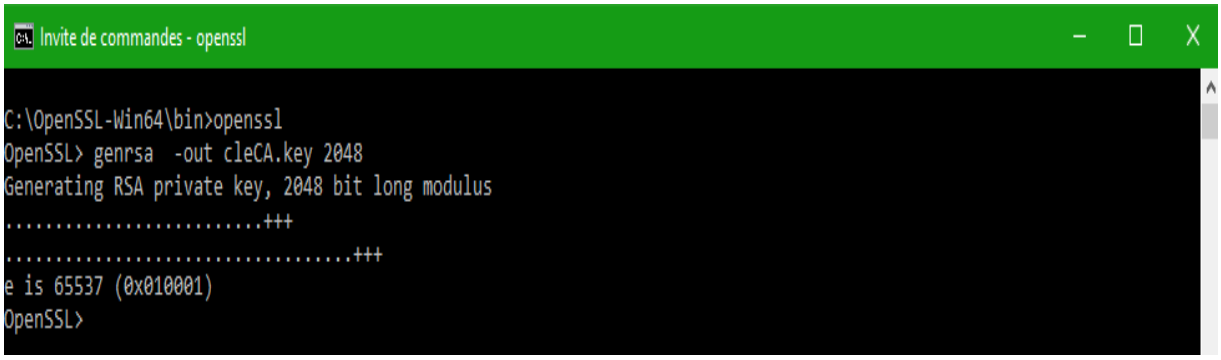

Figure III.12 - Capture d'écran présentant la création d'une paire de clé du CA

- Pour afficher la clé au format PEM :

  On utilise la commande : rsa -in cleCA.key

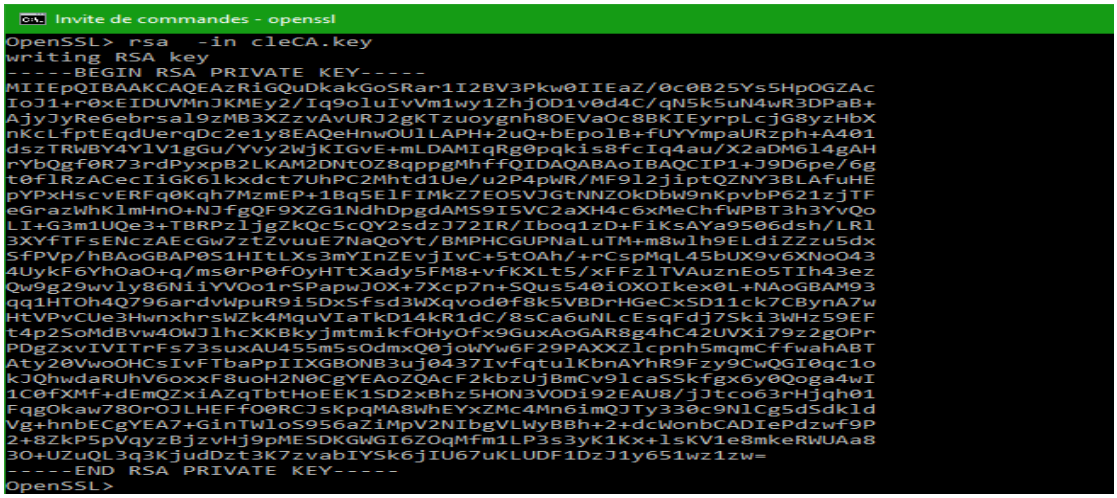

Figure III.13 - Capture d'écran présentant la clé du CA au format PEM

- Pour afficher la clé au format text on utilise cette commande :

  rsa -in cleCA.key -text –noout

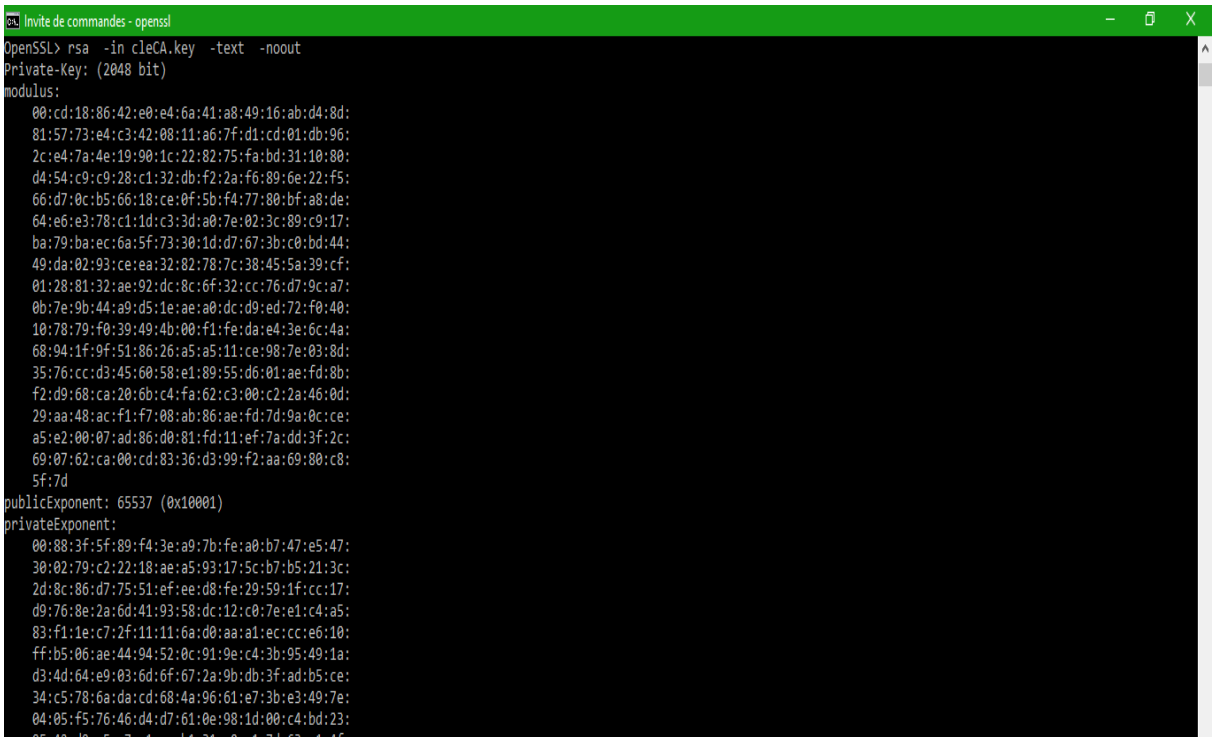

Figure III.14 - Capture d'écran présentant la clé du CA au format text

- Pour extraire la clé publique dans un fichier « CA_clepublique.key », on utilise :

  rsa -in cleCA.key -pubout –out CA_clepublique.key

- Pour afficher la clé publique, on utilise :

  rsa -pubin –in CA_clepublique.key

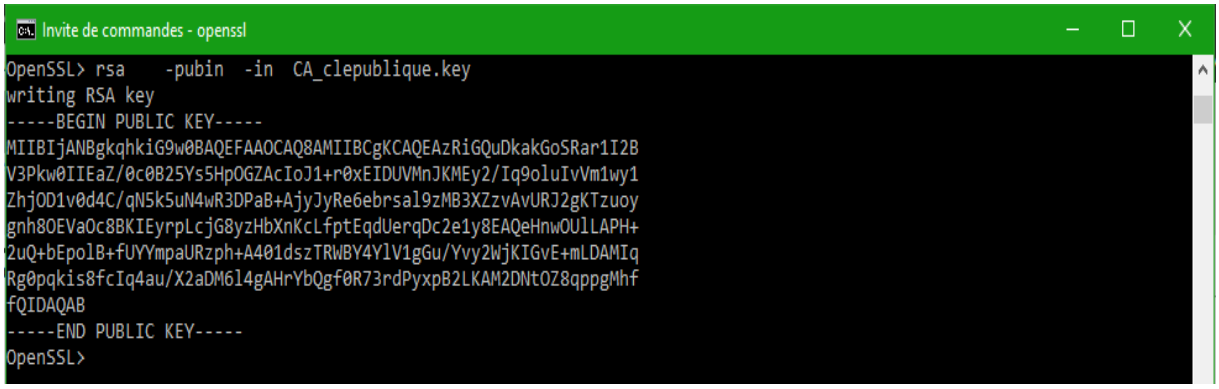

Figure III.15 - Capture d'écran présentant la clé publique du CA

- Pour afficher la clé publique au format text, on utilisant la commande :

  rsa -pubin –in CA_clepublique.key -text -noout

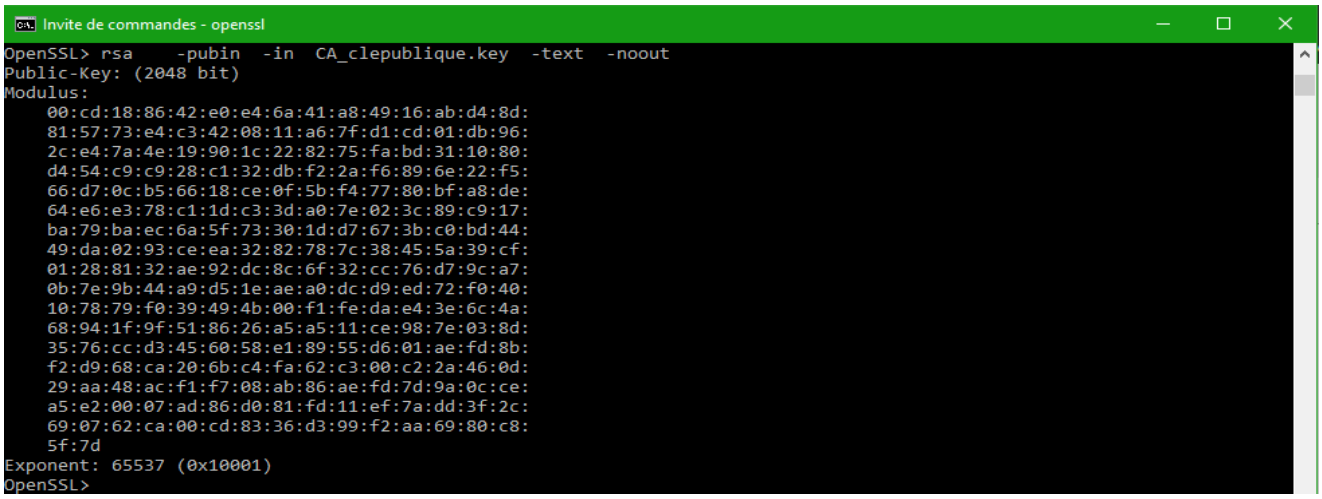

Figure III.16 - Capture d'écran présentant la clé publique du CA au format text

- Génération d'un certificat auto-signé pour le CA :

  On va créer un certificat X509 pour une durée de validité de 10 ans auto-signé ; on utilise la commande :

  req –new –x509 –days 3650 –key cleCA.key –out CA_cert.cert -config c:\OpenSSL-Win64\bin\openssl.cfg

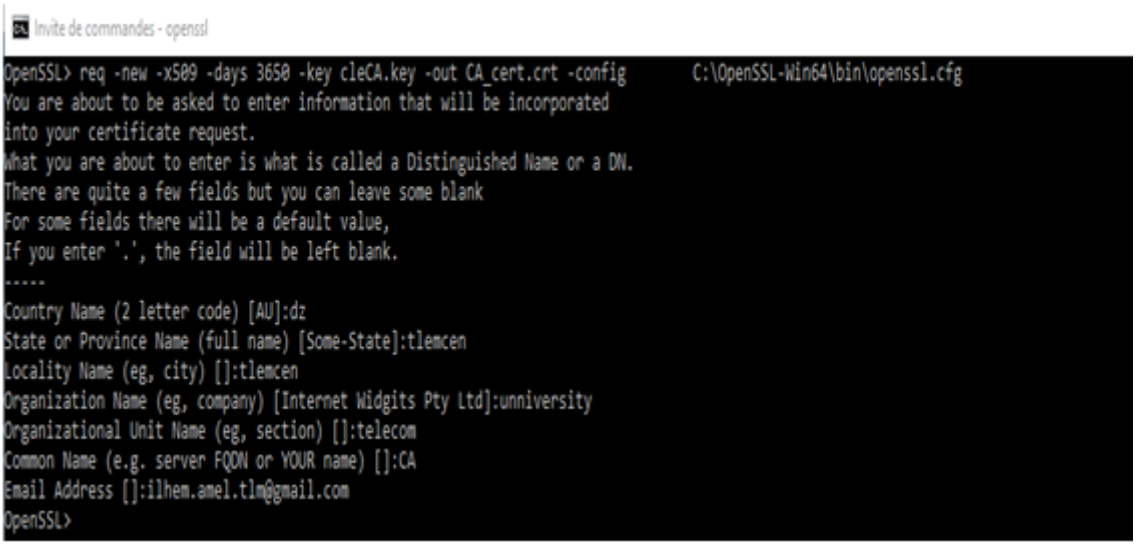

Figure III.17 - Capture d'écran présentant la signature du certificat du CA

- Pour afficher ce certificat, on utilise la commande suivante :

  x509 –in CA_cert.crt –text

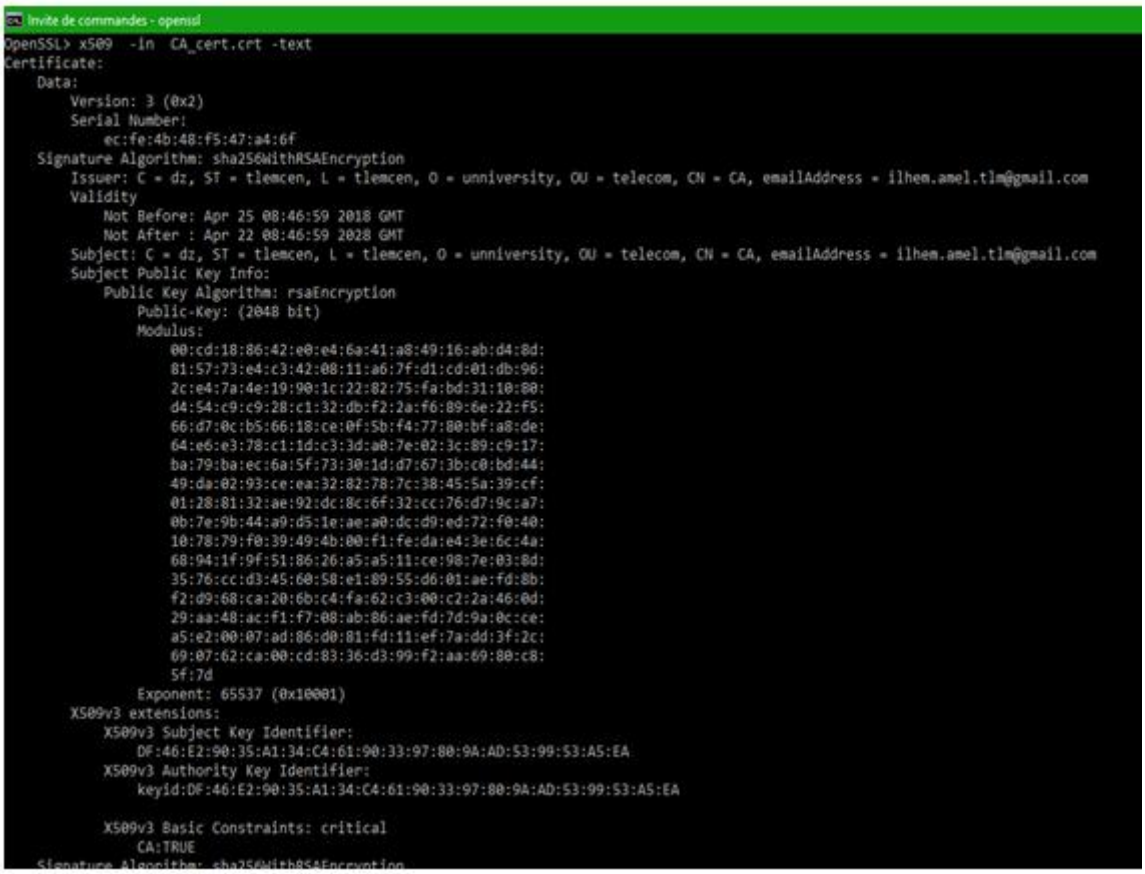

Figure III.18 - Capture d'écran présentant le certificat du CA au format text

III.5.3 Installation du certificat du CA sur le client

Maintenant on va importer le certificat auto-signé du CA sur le navigateur client (Mozilla Firefox).

Ceci permettra au client de valider le certificat transmis par le serveur web lors de la requête http :

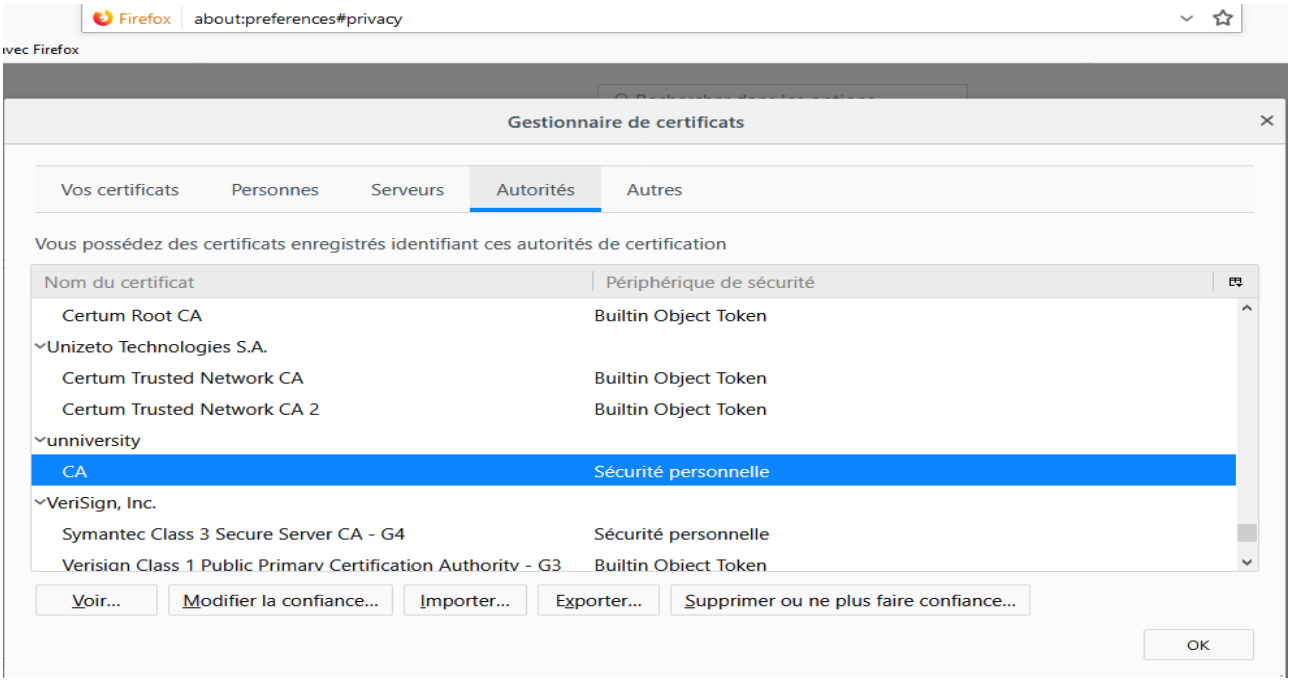

Figure III.19 - Capture d'écran présentant l'installation du certificat du CA sur Firefox (client)

Pour afficher les détails du certificat, on clique sur le bouton **voir** :

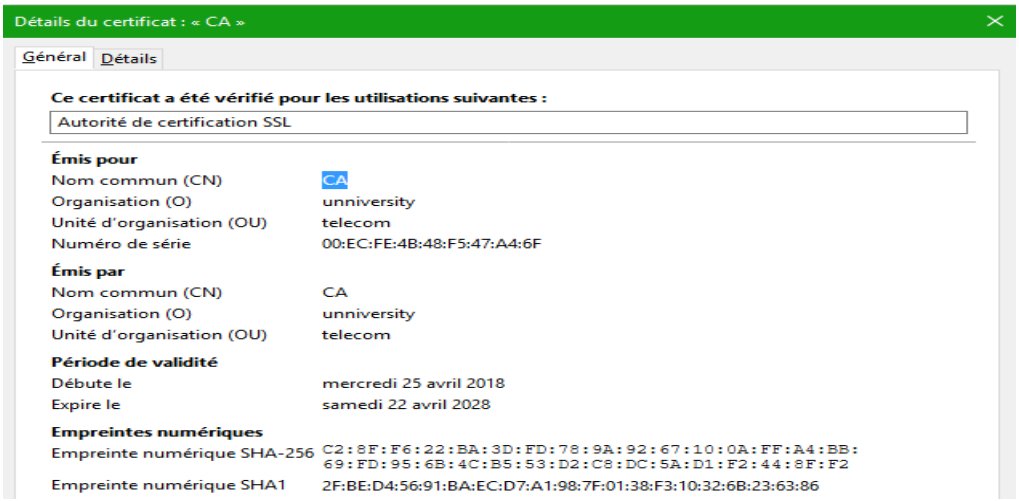

Figure III.20 - Capture d'écran présentant les détails du certificat

III.5.4 Création et signature du certificat du serveur Apache

Maintenant que nous disposons de notre autorité de certification nous allons pouvoir créer et signer les certificats de notre serveur web :

- Génération d'une paire de clés privée/publique :

genrsa –out serveurwebcle.key 2048

Figure III.21 - Capture d'écran présentant la création d'une paire de clés pour le serveur web

- Génération d'une demande de signature du certificat :

  req –new –key serveurwebcle.key –out serveurwebreq.csr –config C:\OpenSSL-Win64\bin\openssl.cfg

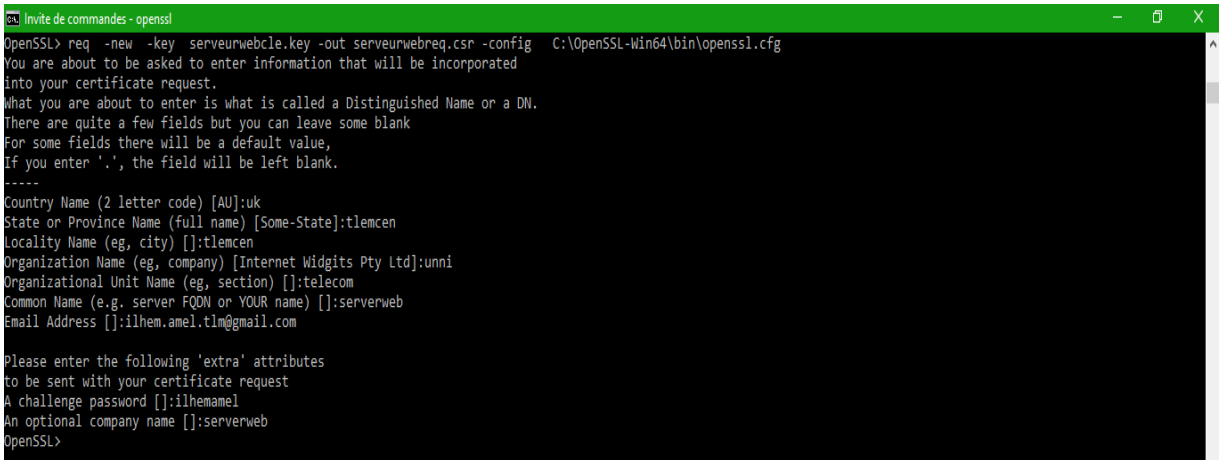

Figure III.22 - Capture d'écran présentant la demande CSR pour le serveur web

- Signature du certificat du serveur web par la CA

  Pour signer le certificat du serveur web par la CA, on utilise cette commande :

  x509 -req -in serveurwebreq.csr -out serveurweb.crt -CA CA_cert.crt -CAkey cleCA.key -CAcreateserial -days 365

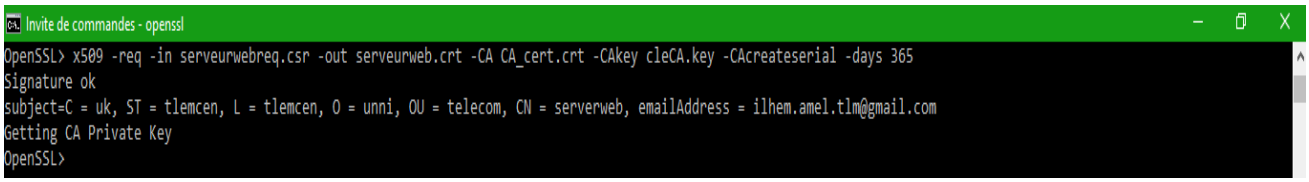

Figure III.23 - Capture d'écran présentant la signature du certificat du serveur web par CA

III.5.5 Configuration du serveur Apache

- Modification de httpd :

On active le module SSL : LoadModule ssl_module modules/mod_ssl.so

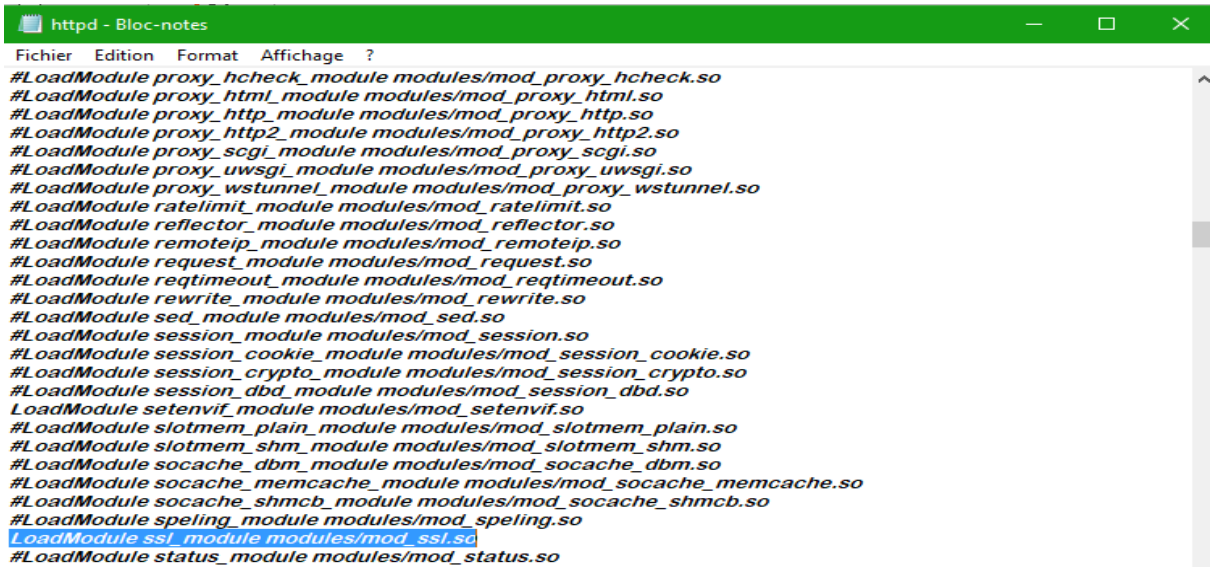

Figure III.24 - Capture d'écran présentant l'activation du module SSL sur Apache

- On donne le chemin du fichier de configuration de ssl :

  Include conf/extra/httpd-ssl.conf

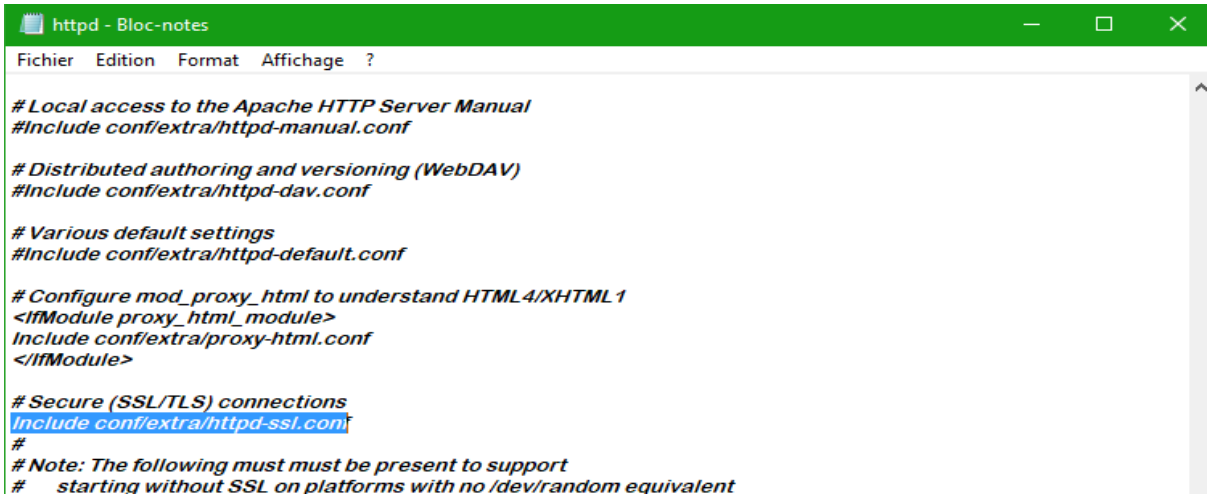

Figure III.25 - Capture d'écran présentant le chemin du fichier de configuration de SSL

- Modification de httpd-ssl :

  - On donne le chemin du certificat du serveur web :

SSLCertificateKeyFile"C: /Program Files (x86)/Apache24/conf/serveurweb cle et certificat/serveurwebcle.key"

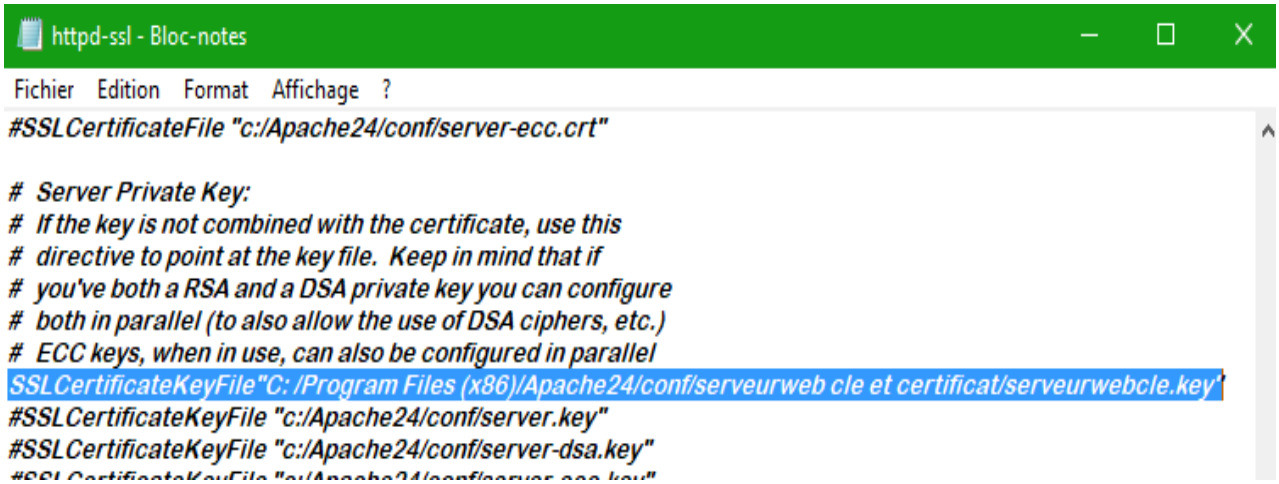

Figure III.26 - Capture d'écran présentant le chemin du certificat de serveur web

- On donne le chemin de la clé du serveur web :

SSLCertificateFile"C:/Program Files (x86) /Apache24/conf/serveurweb cle et certificat/serveurweb.crt"

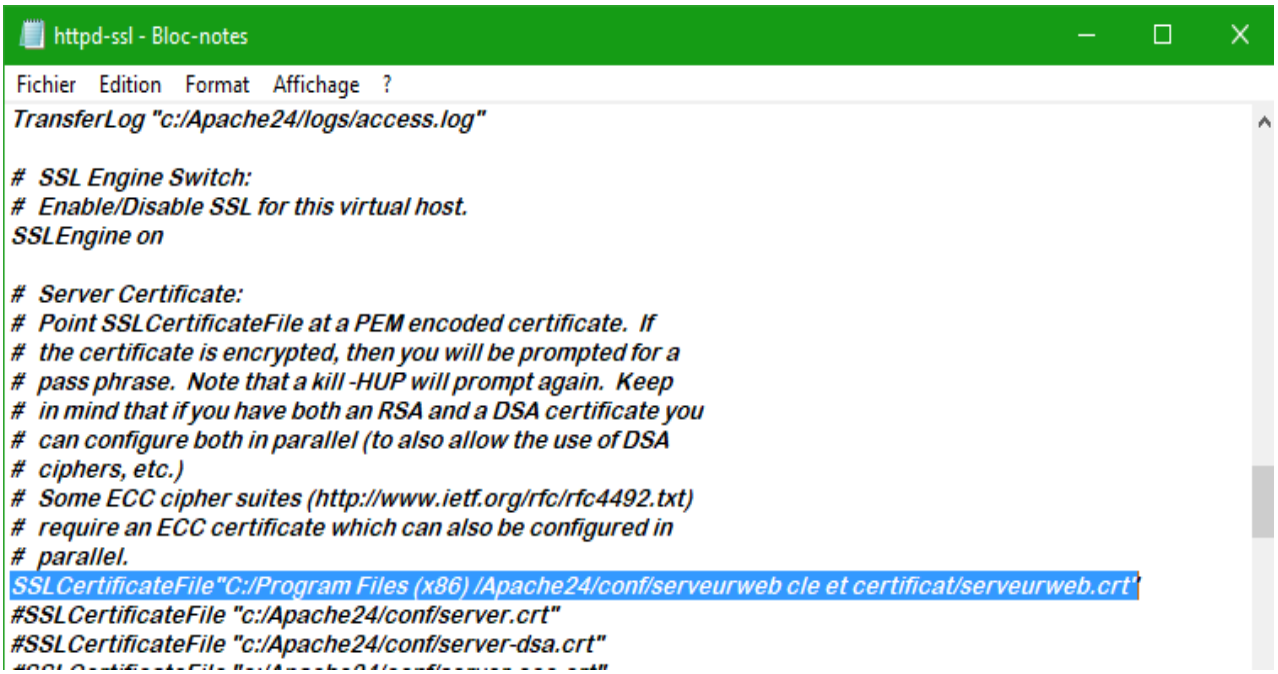

Figure III.27 - Capture d'écran présentant le chemin du la clé de serveur web

Pour éviter le fonctionnement en http (non sécurisé), il faut ajouter un commentaire # devant listen 80, ce dernier qui existe dans le dossier httpd.conf.

On peut vérifier le bon fonctionnement de https :

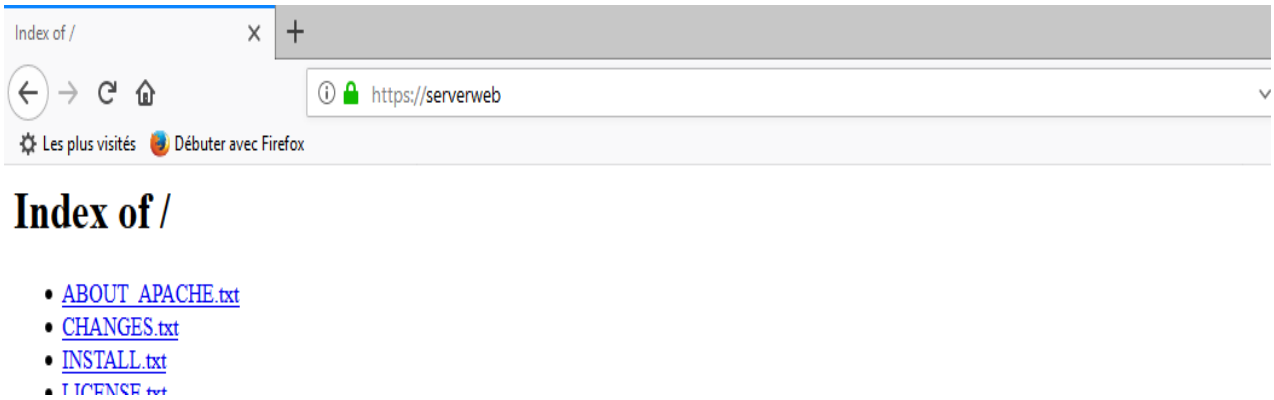

Figure III.28 - Capture d'écran présentant le fonctionnement de https sur le client

III.6 Prise en charge de l'authentification du client

III.6.1 Création et signature du certificat client

Génération d'une paire de clés privées/publiques :

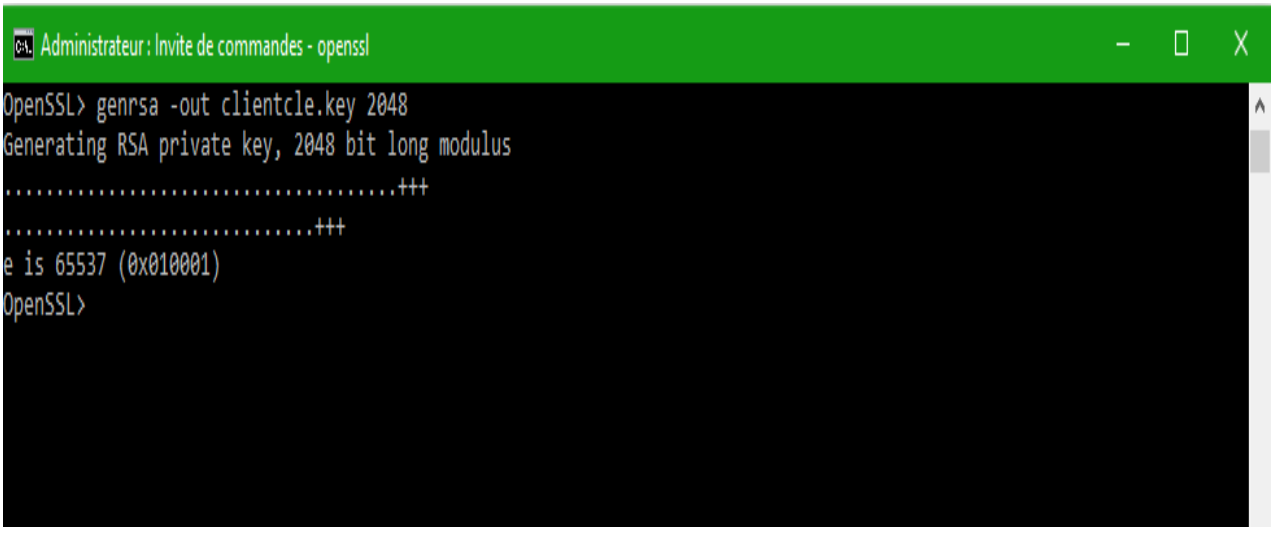

Figure III.29- Capture d'écran présentant la création d'une paire de clés pour le client

- Génération d'une demande de signature du certificat du client :

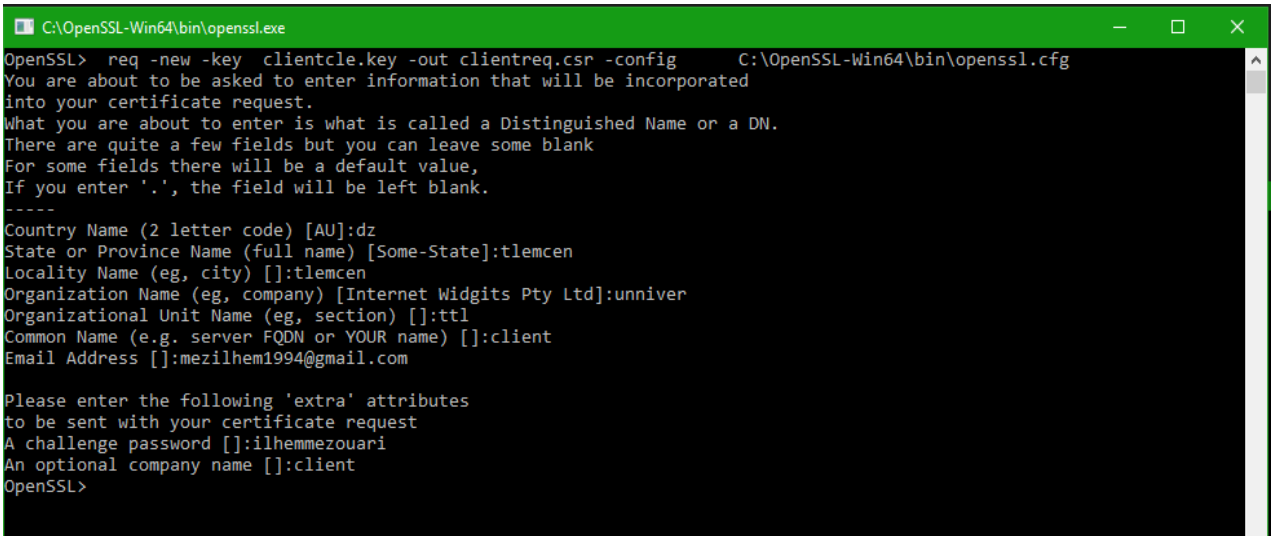

Figure III.30 - Capture d'écran présentant la demande CSR pour le client

- Signature du certificat du client par la CA :

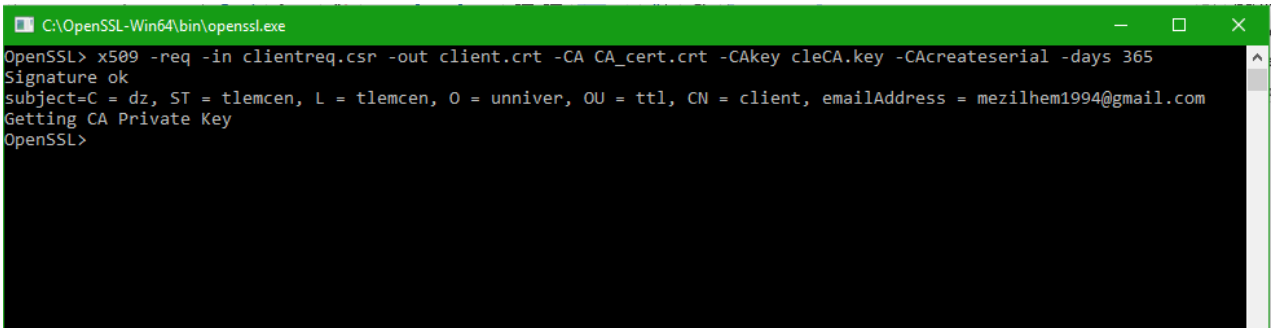

Figure III.31 - Capture d'écran présentant la signature du certificat du client par CA

III.6.2 Configuration du serveur Apache

Activation de SSLVerifyClient et SSLVerifyDepth :

- SSLVerifyClient

  La valeur require permet d'imposer au client la présentation d'un certificat X509 lors de la phase de négociation SSL.

- SSLVerifyDepth

  Indique le nombre maximum de niveau que le serveur va analyser pour décider s'il juge le certificat client acceptable.

  On va activer les deux SSLVerifyClient et SSLVerifyDepth en supprimant le # devant :

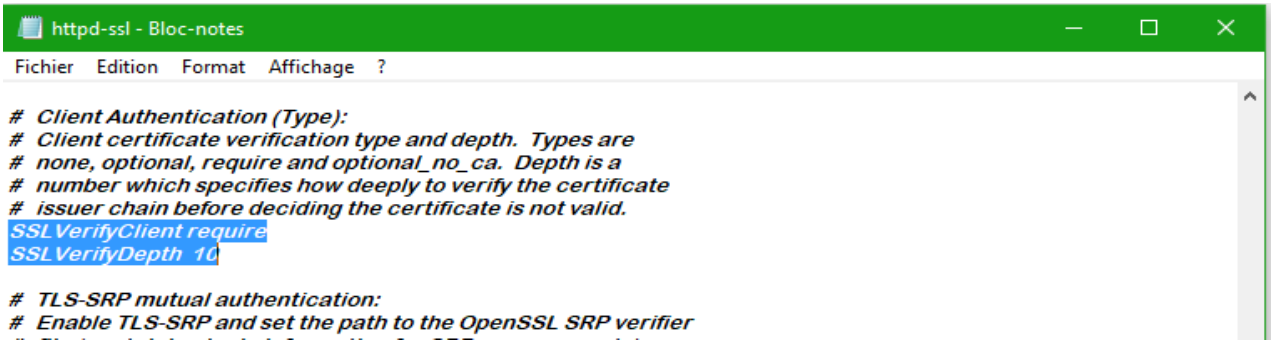

Figure III.32- Capture d'écran présentant activation du SSLVerifyClient et SSLVerifyDepth

- SSLCACertificateFile

Cette directive permet de spécifier le certificat de l'autorité de certification reconnue par le serveur. Seuls les certificats clients signés par l'autorité de certification identifiée par ce certificat seront acceptés. On donne le chemin du fichier :

SSLCACertificateFile "c:/Program Files (x86)/Apache24/conf/serveurweb cle et certificat/CA_cert.cert »

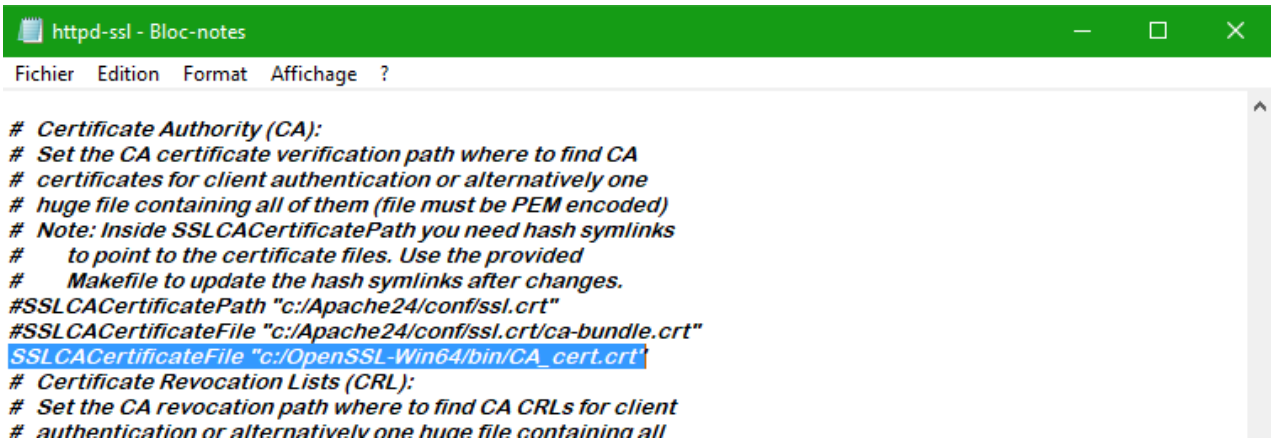

Figure III.33 - Capture d'écran présentant le chemindu CA

Il faut d'abord installer le certificat client sur le navigateur Firefox (client)

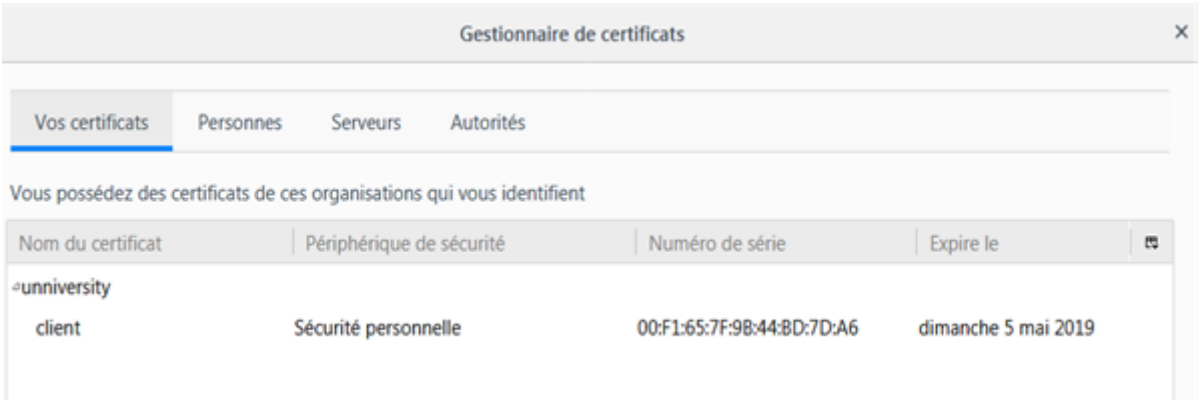

Figure III.34- Capture d'écran présentant l'installation du certificat client sur Firefox (client)

Puis nous testons l'accès à la page web et le fonctionnement du HTTPS : le serveur envoie la requête ci-dessous dans laquelle il demande l'identification du client avec un certificat de sécurité :

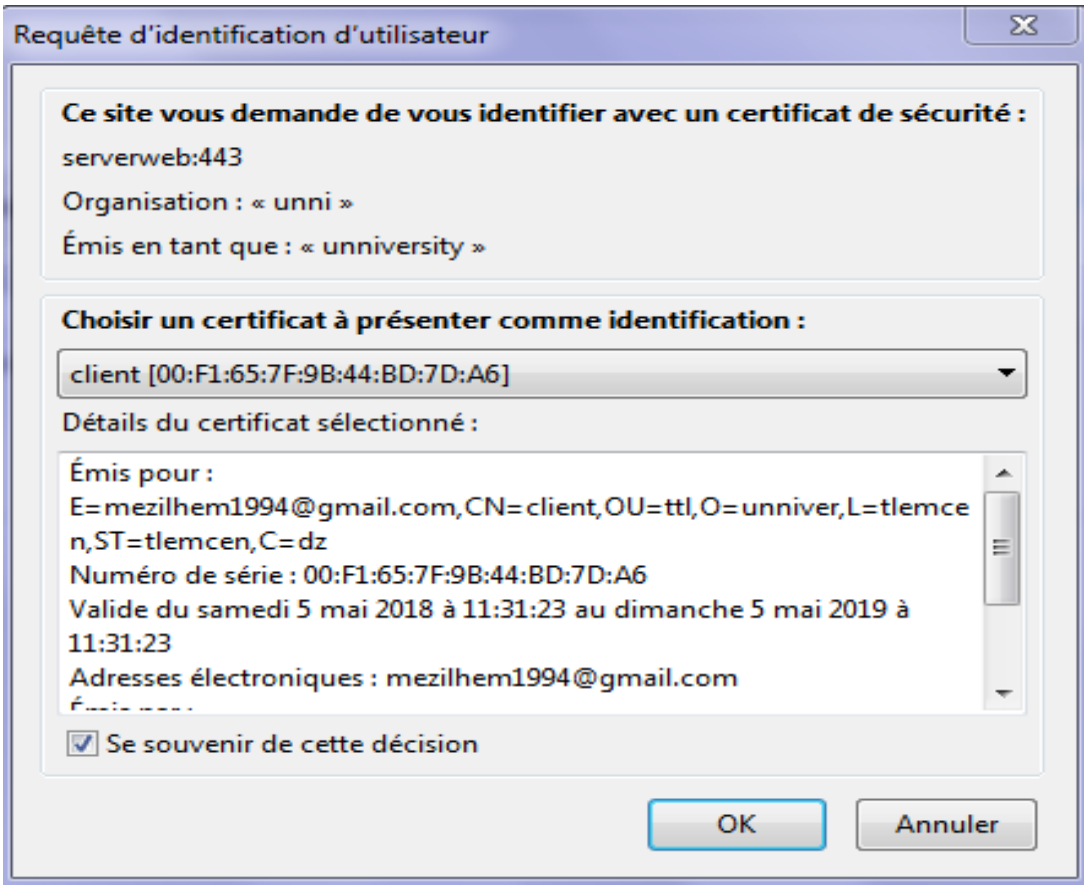

Figure III.35 - Capture d'écran présentant la demande de l'identification du client (par certificat)

Le résultat obtenu est une authentification mutuelle forte.

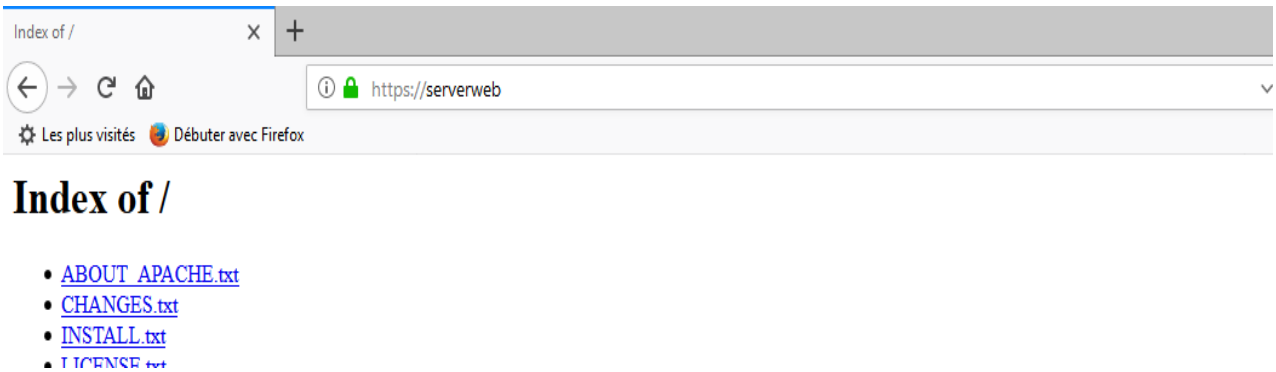

Figure III.36- Capture d'écran présentant le fonctionnement de https sur le client

(Authentification mutuelle forte)

III.7 Analyse du traffic TLS avec Wireshark

Les captures avec l'analyseur de réseau Wireshark que nous avons effectuées portent sur le fonctionnement normal du protocole TLS 1.2, notamment la négociation Handshake.

Capture 1 : le client envoie au serveur le message **Client Hello** du protocole TLS 1.2

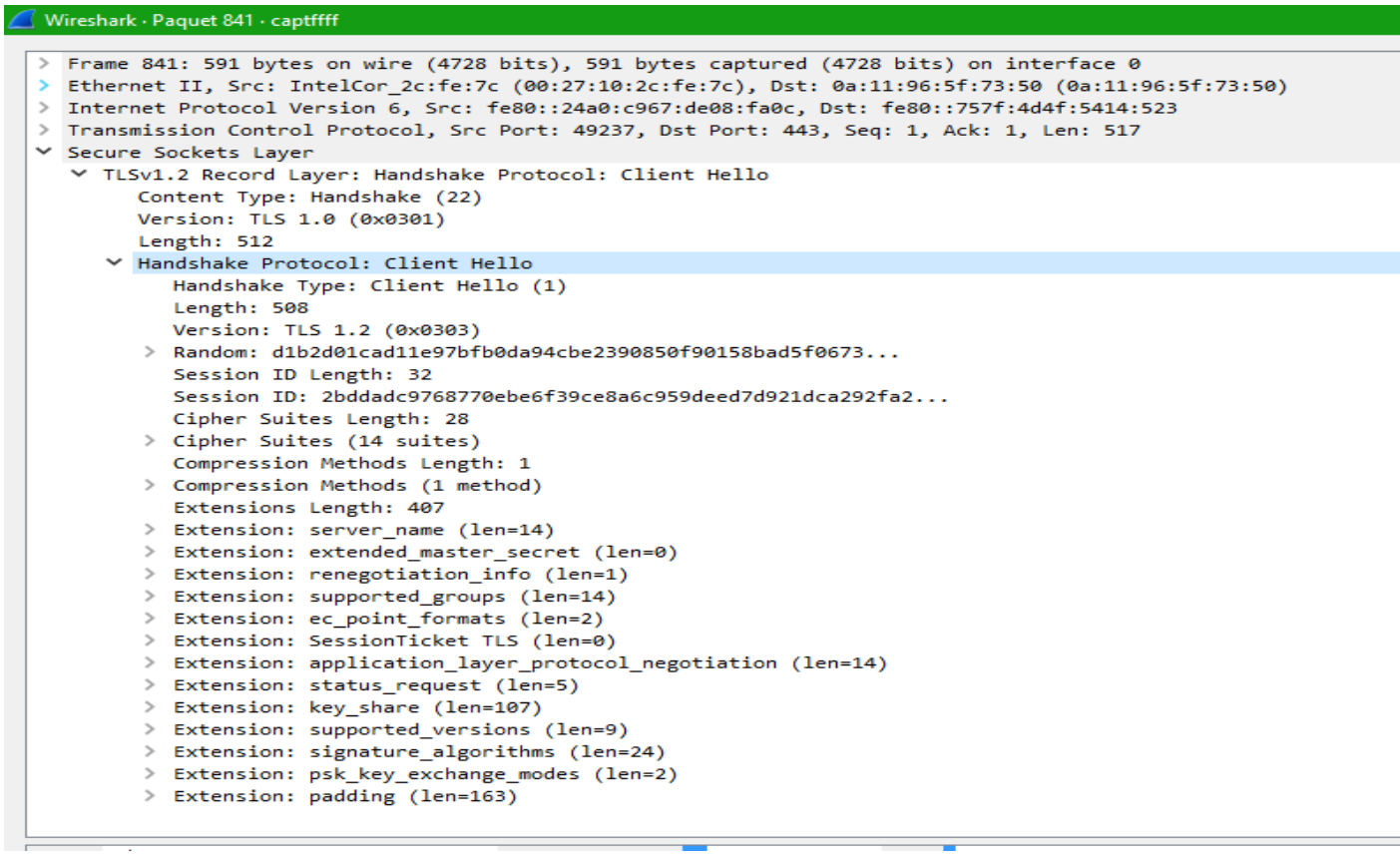

Figure III.37 - Capture d'écran présentant le message Client Hello

Capture 2 : le serveur envoie au client le message handshake **Server Hello**

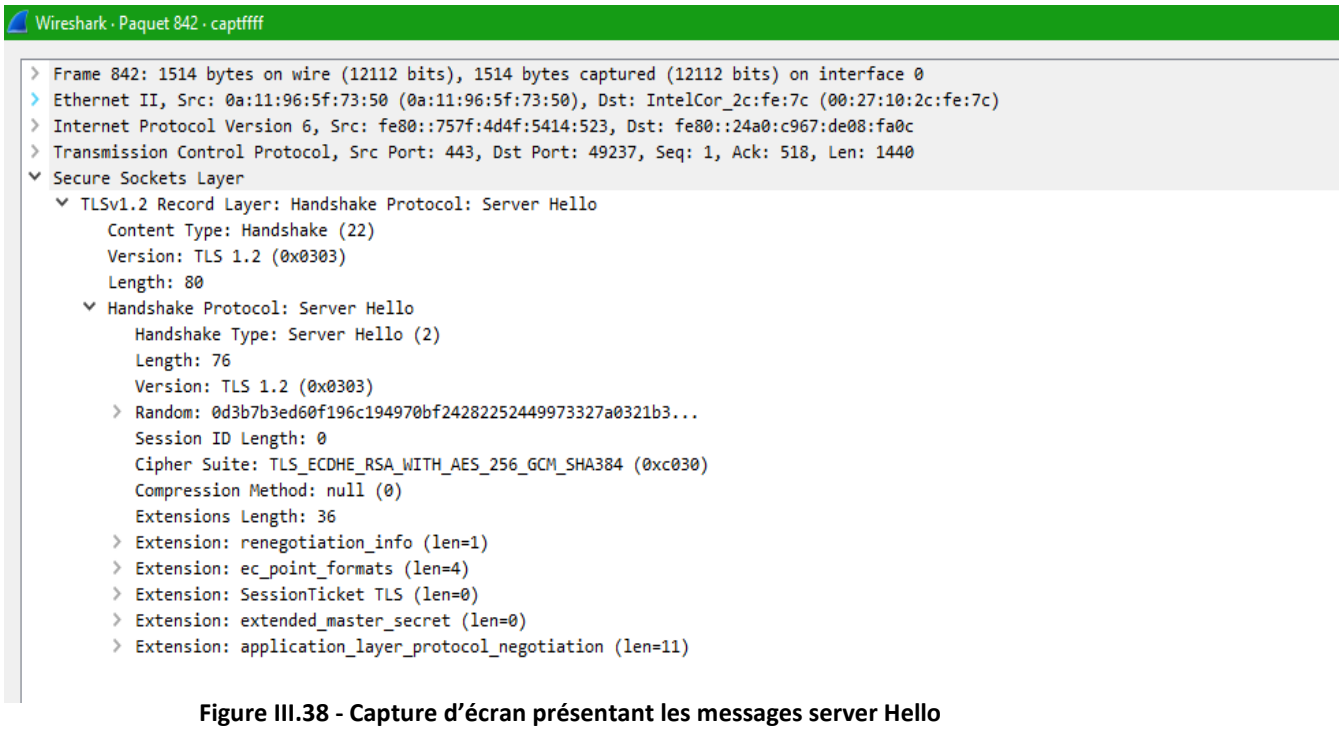

Figure III.38 - Capture d'écran présentant les messages server Hello

Capture 3 : Le serveur envoie son certificat au client par le message **Certificate**

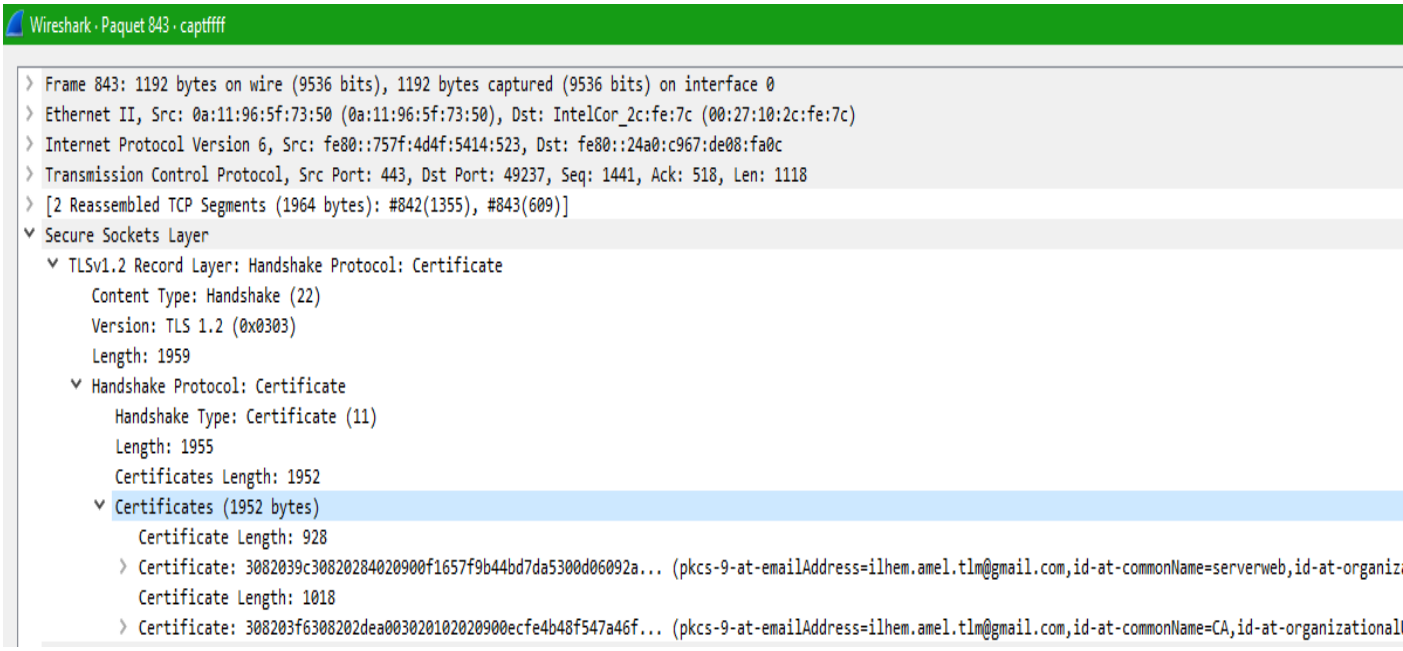

Figure III.39 - Capture d'écran présentant le message Certificate

Capture 4 : le serveur envoie le message **Server Key Exchange**

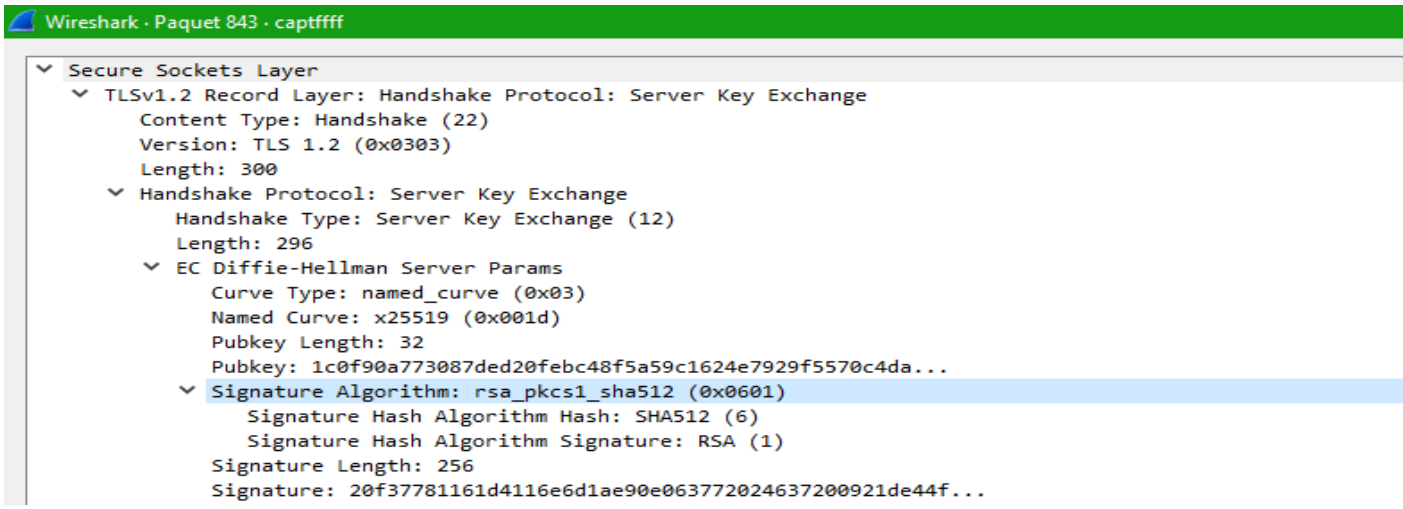

Figure III.40 - Capture d'écran présentant le message Server Key Exchange

Capture 5 : le serveur envoie le message **Certificate Request**, il demande le certificat du client

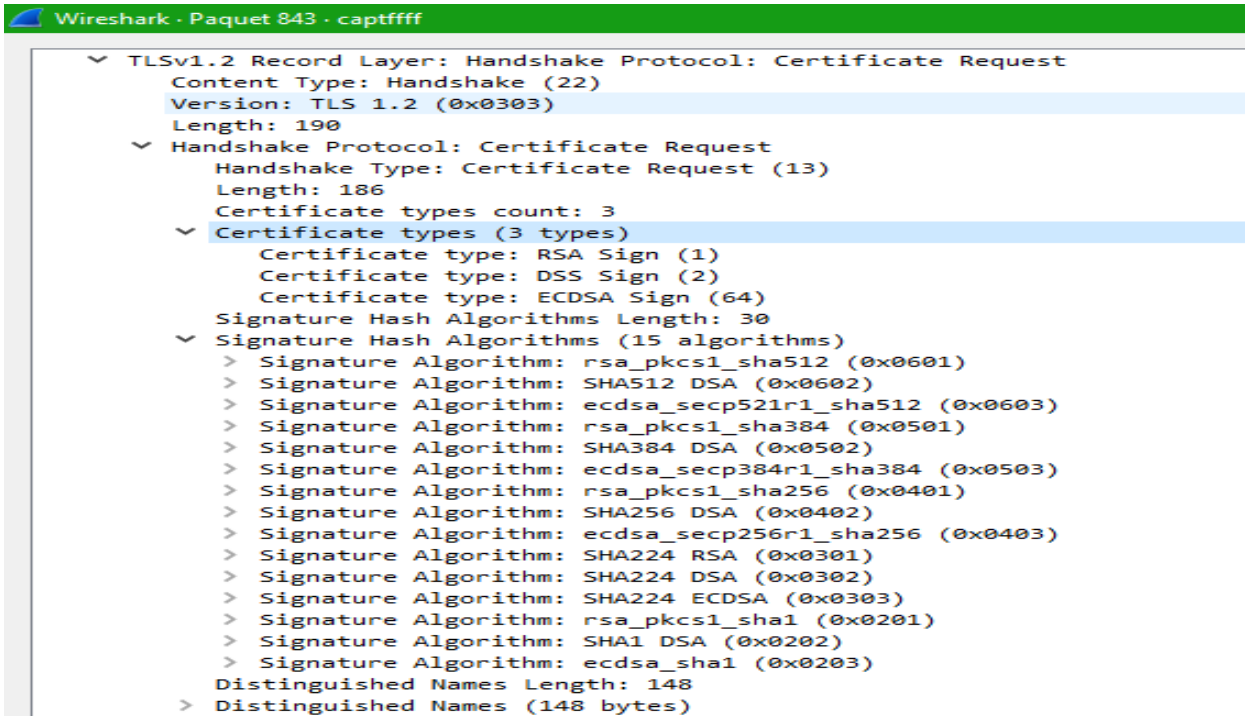

Figure III.41 - Capture d'écran présentant le message Certificate Request

Capture 6 : le serveur envoie au client le message **Hello Done**

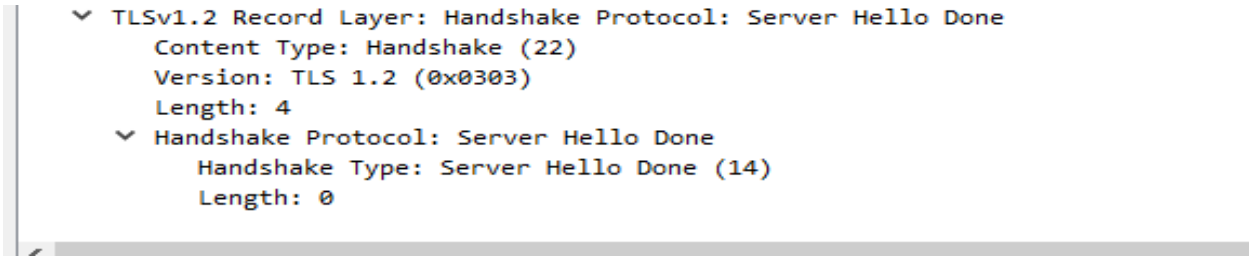

Figure III.42 - Capture d'écran présentant le message Hello Done

Capture 7 : le client répond au serveur par le message **Certificate** : il envoie son propre certificat

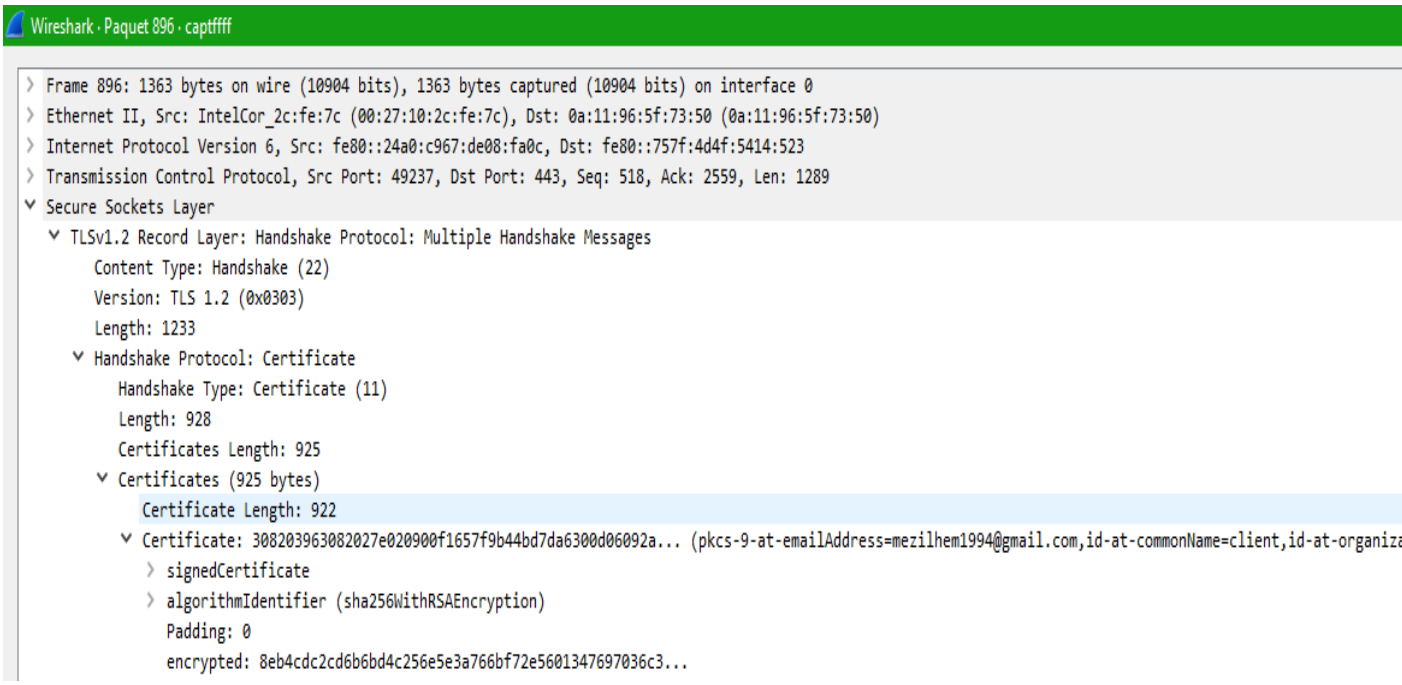

Figure III.43- Capture d'écran présentant le message Certificate envoyé par le client

Capture 8 : le client envoie le message **Client Key Exchange**

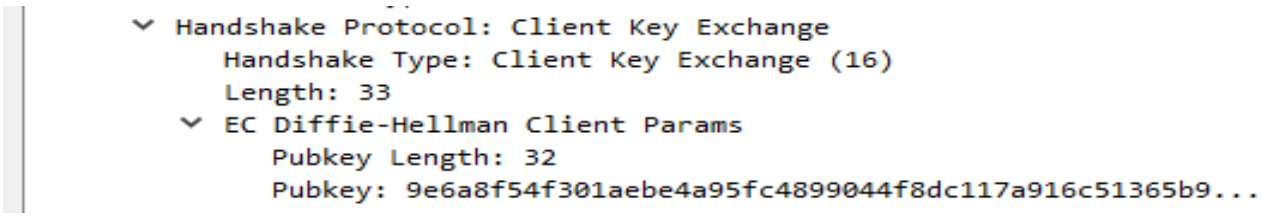

Figure III.44 - Capture d'écran présentant le message Client Key Exchange

Capture 9 : le client confirme que le certificat du serveur est valide à partir du CA (**Certificate Verify**)

```
Handshake Protocol: Certificate Verify
    Handshake Type: Certificate Verify (15)
    Length: 260
  Signature Algorithm: rsa_pkcs1_sha256 (0x0401)
      Signature Hash Algorithm Hash: SHA256 (4)
      Signature Hash Algorithm Signature: RSA (1)
    Signature length: 256
    Signature: bc458420dd1bd6039af00d7b007e1fa2e7336dbc07f70ec0...
```

Figure III.45 - Capture d'écran présentant le message Certificate verify

Capture 10 : le client envoie le message du protocole **Change Cipher Spec** et **Encrypted Handshake** message

```
TLSv1.2 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
TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 40
    Handshake Protocol: Encrypted Handshake Message
```

Figure III.46 - Capture d'écran présentant Change Cipher Spec et Encrypted Handshake message

Capture 11 : le client envoie le message **Encrypted Application Data (HTTP over TLS)**

Wireshark · Paquet 897 · captffff

```
Frame 897: 442 bytes on wire (3536 bits), 442 bytes captured (3536 bits) on interface 0
Ethernet II, Src: IntelCor_2c:fe:7c (00:27:10:2c:fe:7c), Dst: 0a:11:96:5f:73:50 (0a:11:96:5f:73:50)
Internet Protocol Version 6, Src: fe80::24a0:c967:de08:fa0c, Dst: fe80::757f:4d4f:5414:523
Transmission Control Protocol, Src Port: 49237, Dst Port: 443, Seq: 1807, Ack: 2559, Len: 368
Secure Sockets Layer
  TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 363
      Encrypted Application Data: 00000000000000017f7558109220ee7f70378285d5ade5dd...
```

Figure III.47 - Capture d'écran présentant le message Application Data (chiffré)

Capture 12 : le serveur envoie au client les messages New session ticket, change Cipher Spec et Encrypted handshake message

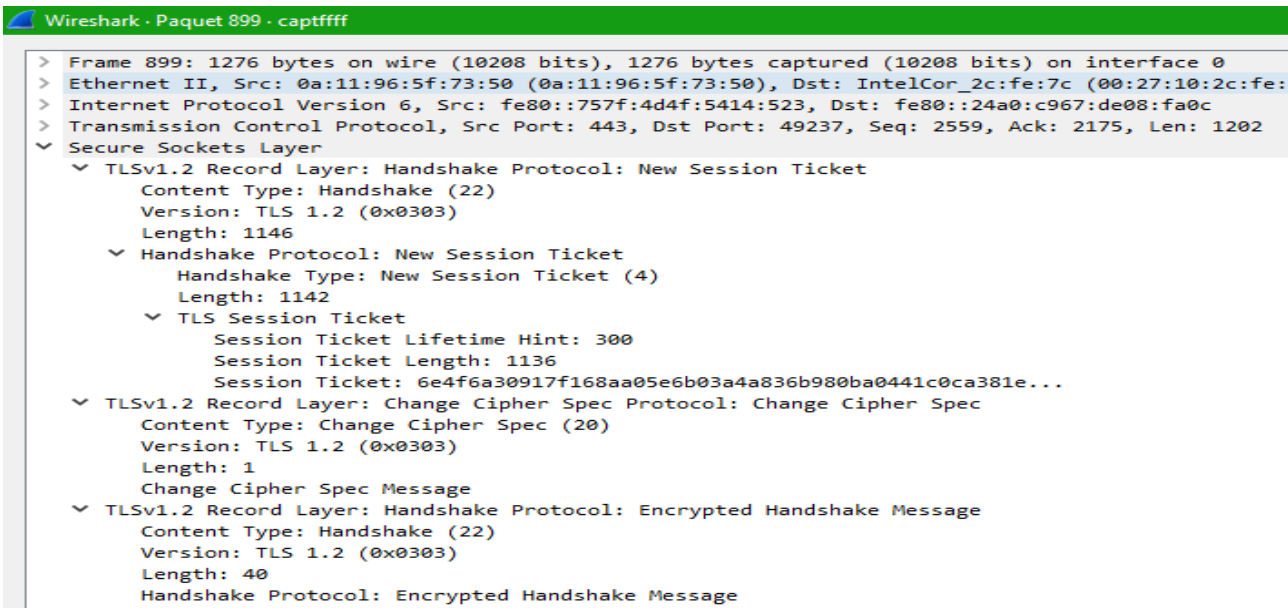

Figure III.48- Capture d'écran présentant les messages New session ticket, change Cipher Spec et Encrypted handshake message

Capture 13 : le serveur envoie le message **Encrypted Application Data**

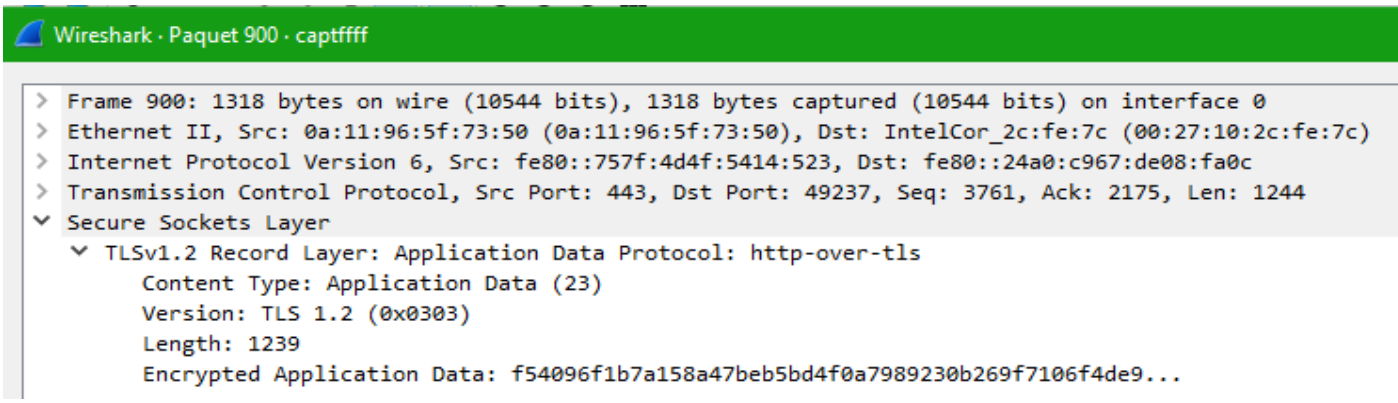

Figure III.49 - Capture d'écran présentant le message Application Data envoyé par le serveur

III.8 Conclusion

Ce chapitre a présenté en détail les étapes de mise en œuvre d'une infrastructure à clés publiques en utilisant l'API Openssl. Les installations et les configurations ont été réalisées avec succès compte tenu du test de validation appliqué à la sécurisation du serveur Apache. On a pu réaliser une connexion http sécurisée coté serveur et faire de même coté client afin d'avoir une authentification mutuelle forte (par certificats).

Conclusion générale

La cryptographie symétrique consiste à chiffrer puis déchiffrer un message en utilisant la même clé et le même algorithme. La distribution des clés a été le point faible des systèmes de cryptographie symétrique, d'où la proposition des algorithmes à clés publiques (algorithmes asymétriques).

La cryptographie asymétrique (à clés publiques) exige que chacun des correspondants possède une clé publiée dans un annuaire utilisée par tout le monde pour chiffrer des messages destinés à un individu particulier, et l'autre privée que cet individu est seul à détenir et qui lui permet de déchiffrer les messages qu'il reçoit.

Bien que très efficace, la cryptographie asymétrique comporte cependant un enjeu majeur ; qui consiste en la gestion des clés publiques. En effet, l'efficacité de ce mécanisme de sécurité dépend du niveau de certitude que détient l'utilisateur d'une clé publique quant à l'identité de son propriétaire légitime.

Pour s'assurer de cette identité, des certificats sont introduits, ceux-ci sont générés par des infrastructures à clés publiques (PKI).

Au cours de ce projet de fin d'études, nous avons mis en œuvre une solution de sécurisation du service http au niveau TLS via une infrastructure à clés publiques (PKI) en utilisant Openssl. Ce travail a exigé une étude théorique et pratique très poussée.

En effet, vu la récence de l'approche de certification dans le monde de la sécurité informatique, il nous a été indispensable d'effectuer une étude de l'existant à travers une documentation riche et variée.

Nous avons, dans un premier temps, présenté en détail le protocole TLS. En suite, nous avons donné les caractéristiques de base de la cryptographie asymétrique en se focalisant sur les PKI tout en essayant de simplifier au maximum leur compréhension. Enfin, nous avons présenté les démarches que nous avons suivies pour la mise en œuvre de la solution. Le projet a aboutit et les résultats de validation sont exposés en fin de mémoire.

Comme continuité à ce travail, il serait intéressent d'examiner les vulnérabilités d'implémentation de TLS 1.2 et d'analyser les attaques possibles induites contre le protocole https, enfin voir les améliorations apportées dans la version toute récente TLS 1.3.

Bibliographie

[1] Klaus schemeh. « Cryptography and Public Key Infrastructure on the Internet » Gesellsschaft fur IT –Sicherheit AG Boshum, Germany, 2001

[2] John R. Vacca. « Public Key Infrastructure Building Trusted Applications and Web Services », AUERBACH PUBLICATIONS A CRC Press Company Boca Raton London New York Washington, D.C, 2004

[3] Andrew Nash, William Duane, Celia Joseph, and Derek Brink. « PKI : Implementing and Managing E-Security »,Osborne/McGraw-Hill New York Chicago San Francisco Lisbon London Madrid Mexico City Milan New Delhi San Juan Seoul Singapore Sydney Toronto, 2001

[4] Ivan Ristic. « Openssl Cookbook : a guide to the most frequently used openssl features and commands », Feisty Duck, Nov 2013

[5] Vittorio Giovara. « Openssl-User Manuel and Data Format », December , 2007

[6] Kapil Raina. «PKI Security Solutions for the Enterprise : Solving HIPAA, E-Paper Act, And Other Compliance Issues »,Wiley Publishing, Inc. 2003

[7] Marc Fischlin, Johannes Buchmann, Mark Manulis (Eds.). « Public Key Cryptography-PKC 2012 » ,15th International Conference on Practice and Théory in Public Key Cryptography Darmstadt, Germany, May 2012, Proceedings.

[8] Stig F . Mjolsnes Sjouke Mauw, Sokratis, K.Katsikas (Eds.). « Public Key Infrastructure ».5th European PKI Workshop : Theory and Practice, EuroPKI 2008 Trondheim, Norway, June 16-17 ,2008 Proceedings

[9] Ivan Ristíc, « Bulltproof SSL and TLS Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications ».Feisty Duck London Copyright March 2015.

[10] Joshua Davies « Implementing SSL /TLS Using Cryptography and PKI». By Wiley Publishing, Inc, Indianapolis, Indiana, 2011

[11] Rolf Oppliger, Ph .D. « SSL and TLS Theory and practice ». eSECURITY Technologies Beephovenstrasse 10 CH-3073 Gumligen Switzerland, August 11,2009

[12] Stephen A .Thomas « SSL and TLS Essentials Securing the Web ». Wiley Computer Publishing, John Wiley & Sons, Enc 2000

[13] Pravir Chandra, Matt Messier, John Viega « Network Security with Openssl ». O'Ruilly, June 2002

[14] T. Dierks Independent, E.Rescorla. « The Transport Layer Security (TLS) Protocol Version ».RTFM, Inc. August 2008.

[15] https://blog.squad.fr/cyber-securite/pki-complexe-plein-de-perspectives.html

[16] https://slproweb.com/products/Win32OpenSSL.html