Chapitre 4

Le protocole sécurisé SSL

Les trois systèmes de sécurisation SSL, SSH et IPSec présentés dans un chapitre précédent reposent toutes sur le même principe théorique : cryptage des données et transmission de la clé à distance. Nous allons détailler dans ce chapitre les principes théoriques sur lesquels ces solutions reposent puis plus en détail un de ces protocoles, à savoir SSL.

4.1 Les principes théoriques d'un protocole sécurisé

4.1.1 Le chiffrement pour assurer la confidentialité

Dans les protocoles réseau que nous avons vus jusqu'à maintenant, le texte est transmis en clair et peut donc être récupéré par une personne curieuse ou mal intentionnée (qu'il est devenu traditionnel d'appeler **attaquant** ; *attacker* en anglais). Comme nous l'avons vu, ce problème est connu depuis la plus haute Antiquité, époque à laquelle on a commencé à essayer de brouiller le message, par exemple en remplaçant une lettre par la troisième lettre qui suit dans l'alphabet.

Le premier problème à régler est donc celui de la **confidentialité** (*confidentiality* en anglais) : *les données brutes ne doivent pas pouvoir être lues par une personne autre que l'expéditeur et les destinataires.*

Ce problème est résolu, de façon relativement satisfaisante, par le chiffrement des données, dont nous avons déjà parlé.

4.1.2 Le problème de la transmission de la clé

Nous avons crypté un texte, en utilisant telle méthode et telle clé. Très bien. On peut maintenant transmettre le texte crypté à travers le réseau. Le destinataire a besoin de connaître la méthode de cryptage et la clé pour décrypter.

On considère en général que la méthode de cryptage n'est pas un renseignement sensible, de toute façon il n'en existe qu'un nombre restreint. Par conséquent les protocoles soit reposent sur une seule méthode de cryptage, soit l'un des champs de son en-tête précise en clair la méthode de cryptage choisie.

Il en est tout autrement de la clé. Si Alice et Bob[1] ont la possibilité de se rencontrer physiquement, ils peuvent s'échanger la clé à cette occasion. Ceci ne sera pas le cas cependant si vous faites des achats sur Internet. De toute façon on a intérêt à changer la clé de temps en temps et même souvent. On est donc confronter au **problème de transmission de la clé** (*the key management problem* en anglais).

Le problème a été résolu en 1976 à Stanford par Whitfield Diffie et Martin Hellman dans [D-F–76]. Ils y ont suggéré ce qui est maintenant appelé la **cryptographie à clé publique** (*public key cryptography* en anglais ; par opposition les anciennes méthodes s'appellent maintenant **cryptographie à clé secrète**). L'idée fondamentale est que les fonctions de cryptage et de décryptage utilisent des clés différentes. La clé de cryptage (la **clé publique**) peut être publiée mais la clé de décryptage (**clé privée**) est gardée secrète. Bien entendu les deux clés ont un lien (mathématique) mais il est très difficile (pour ne pas dire impossible en pratique) de déterminer la clé privée à partir de la clé privée.

La cryptographie à clé publique est aussi appelée **cryptographie assymétrique** et la cryptographie à clé secrète **cryptographie symétrique** pour des raisons évidentes.

4.1.3 Identification des extrémités et certification

Il est très important d'identifier l'expéditeur et le destinataire (*endpoint authentication* en anglais) ou tout au moins l'un des deux suivant le cas pour pouvoir contrer l'attaque dite du troisième homme. Avant d'envoyer son numéro de carte bancaire, on a intérêt à être sûr qu'on l'envoie au bon endroit.

[1] Il s'agit des noms devenus traditionnels, plus charmants que A et B.

Attaque du troisième homme.- En effet un type d'attaque possible lorsque les clés sont publiées électroniquement ou si elles sont échangées au début d'une communication est connu sous le nom d'**attaque du troisième homme** (*man-in-the middle attack* en anglais).

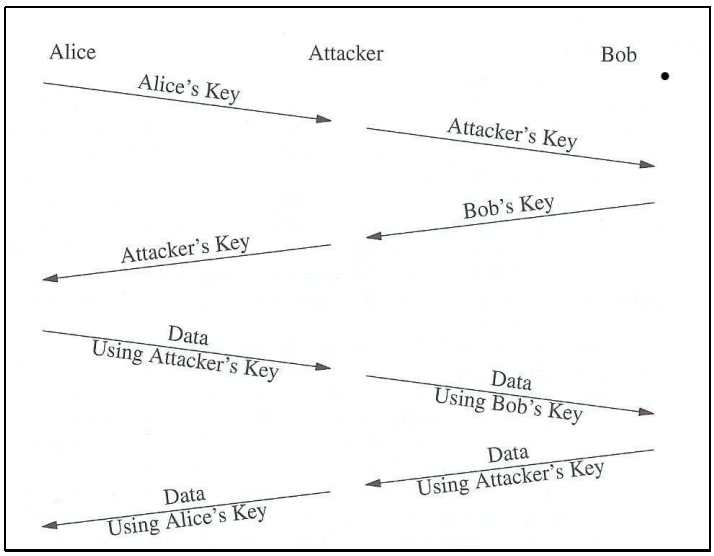

FIG. 4.1 – Attaque du troisième homme

L'attaquant se place entre Alice et Bob et intercepte leurs messages. Lorsqu'Alice envoie sa clé publique, il la garde pour lui et envoie sa propre clé à Bob à la place. Lorsque Bob répond en envoyant sa clé publique, l'attaquant la garde pour lui et envoie sa propre clé à Alice. Lorsque Alice envoie ensuite des données, l'attaquant peut les décrypter, les changer et les envoyer à Bob. Il en est de même lorsque Bob envoie des données (figure 4.1).

Certification.- La solution à ce problème est de faire appel à un tiers en qui on a toute confiance, appelé **autorité de certification** (**CA** pour l'anglais *Certificate Authority*). Celle-ci publie sous forme papier ou sur CD-ROM une liste de **certificats**, constitués sous sa forme la plus simple d'un nom et de sa clé publique.

4.1.4 Intégrité du message et fonction de hachage

Un autre problème concernant la sécurité est de s'assurer que le message que l'on reçoit est celui envoyé par l'expéditeur dans son intégralité, autrement dit qu'une partie de celui-ci n'a pas été altérée (volontairement).

Ceci est facile à vérifier dans le cas de la transmission d'un numéro de carte bancaire, qui comprend un nombre de chiffres fixe, mais cela n'est pas toujours le cas. La théorie montre que la résolution complète de ce problème n'est pas possible ; en pratique on utilise les *fonctions de hachage* conviennent.

Une **fonction de hachage** *(*hash function *en anglais) prend le message (de longueur arbitraire) comme entrée et renvoie un texte de longueur fixe, appelé le* **condensé du message** *(*message digest *en anglais).*

Bien entendu une telle fonction ne peut pas être injective. Le condensé du message doit être suffisamment caractéristique du message et on ne doit pas pouvoir reconstituer le message à

partir de son condensé. Elle doit également être le plus possible résistante aux collisions (*collision-resistance* en anglais), c'est-à-dire que deux messages différents ayant un sens doivent produire deux condensés différents.

L'étude des fonctions de hachage fait l'objet d'un autre cours. Retenons seulement ici que les deux méthodes les plus utilisées sont **MD5** (pour *Message Digest 5*), conçue en 1992, et **SHA-1** (pour *Secure Hash Algorithm 1*), conçue en 1994.

4.1.5 Un système simple d'envoi de message sécurisé

Bien entendu un condensé ne suffit pas puisqu'un attaquant peut lui-même utiliser la fonction de hachage. Nous allons donc montrer comment construire un protocole sécurisé. Commençons par le cas d'envoi de message (un e-mail par exemple). Seul l'expéditeur a besoin d'être identifié.

4.1.5.1 Envoi du message

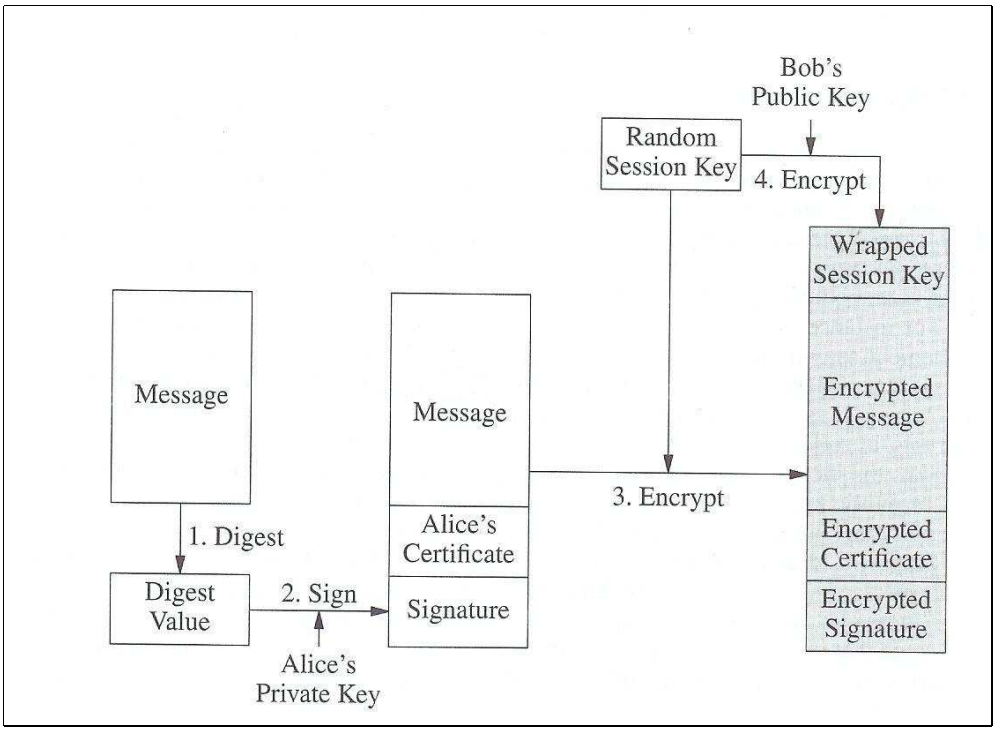

FIG. 4.2 – Envoi d'un message

Alice veut envoyer un message à Bob :

1. Elle calcule le condensé du message, de longueur connue.
2. Elle calcule la **signature numérique** du message en cryptant le condensé avec sa clé privée, signature qui aura également une longueur connue. Elle obtient alors un **message signé**, constitué du message initial, du certificat d'Alice (de longueur connue) et de la signature numérique du message.
3. Elle produit une **clé de session** de façon aléatoire et crypte le message signé avec cette clé.

4. Elle crypte la clé de session avec la clé publique de Bob et l'attache au message signé crypté, en tant que champ de longueur connue.

Elle envoie ce message final à Bob.

4.1.5.2 Réception du message

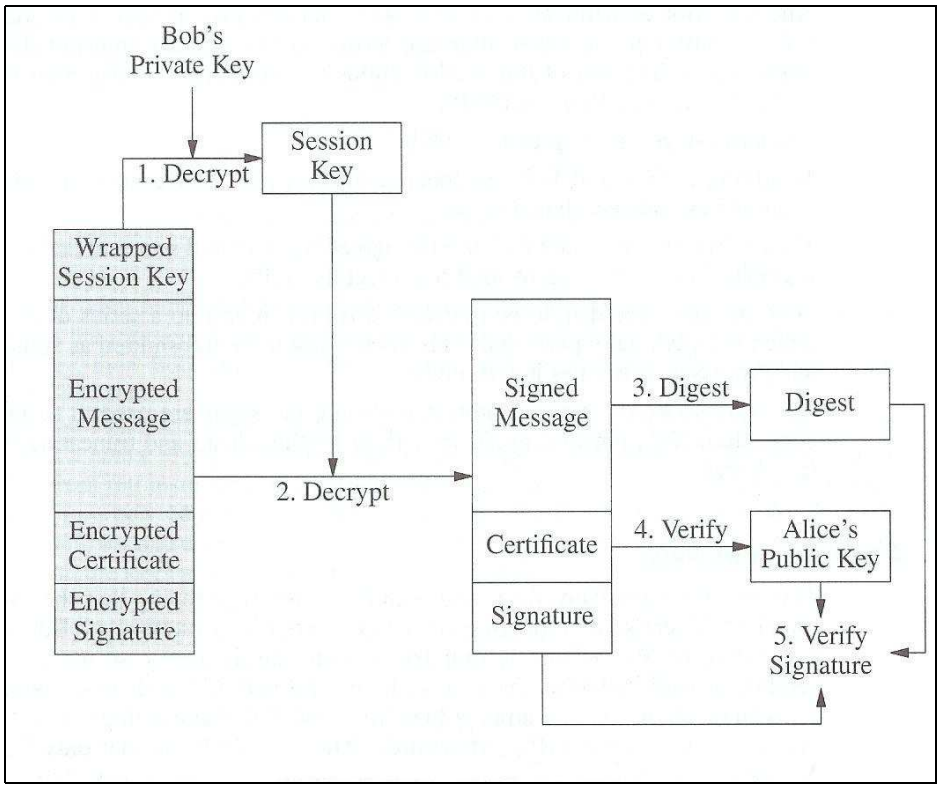

FIG. 4.3 – Réception d'un message

Lorsqu'il reçoit le message :

1. Bob utilise sa clé privée pour décrypter le champ clé de session crypté et obtenir la clé de session obtenue par Alice.
2. Il utilise alors cette clé de session pour décrypter le message signé et crypté et obtenir le message signé.
3. Il peut alors récupérer le message (champ de longueur connue) et en calculer lui-même le condensé.
4. Il vérifie le certificat d'Alice auprès du CA et en extrait la clé publique d'Alice.
5. Il utilise la clé publique d'Alice pour décrypter la signature numérique. Si le résultat obtenu est égal au condensé qu'il a calculé, il fera confiance au message.

4.1.6 Un canal de communication sécurisé

Le système que nous venons de décrire peut évidemment s'appliquer à un canal de communication (qui serait alors sécurisé) mais on perdrait alors beaucoup de temps à calculer à chaque fois une clé de session et les coûts en temps de calcul pour chaque paquet seraient prohibitifs. On préfére disposer d'un ensemble de clés secrètes disponibles pour la session dans son intégralité.

4.1.6.1 Protocole interactif

Dans un **protocole interactif**, on utilise un même ensemble de clés durant toute la session et les certificats sont envoyés (et non récupérés ou vérifiés auprès d'un tiers). Un tel protocole comprend quatre étapes :

1. Une phase de **négociation** (*handshake*, pour poignée de main, en anglais) permettent à Alice et Bob d'utiliser leurs certificats et leurs clés privées pour s'identifier et échanger un nombre appelé **secret partagé** (**MS** pour l'anglais *Master Secret*).
2. Alice et Bob utilisent ce secret commun MS pour **déterminer des clés** qui seront utilisées lors du transfert des données.
3. Les données à transmettre sont divisées en **enregistrements** (*record* en anglais), chacun d'eux étant protégé individuellement.
4. Des messages spéciaux protégés sont utilisés pour la **fermeture de la connexion**. L'existence de cette phase empêche un attaquant de forcer la fermeture et ainsi de tronquer le message.

4.1.6.2 Une négociation simple

Dans le cas où seul Bob a besoin de s'identifier, par exemple pour qu'Alice puisse lui envoyer son numéro de carte de crédit, la négociation peut s'effectuer de la manière suivante :

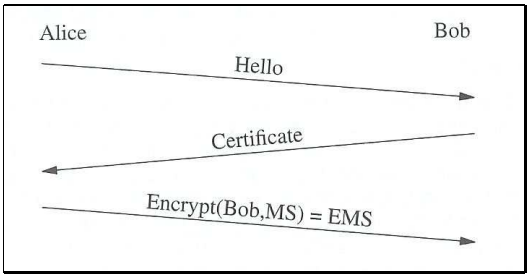

FIG. 4.4 – Identification dans un seul sens

1. Alice envoie un message à Bob pour lui dire qu'elle cherche à communiquer. On parle souvent d'étape d'« Hello ».
2. Bob répond en lui envoyant son certificat.
3. Alice envoie alors le secret à partager MS, bien entendu non en clair mais crypté avec la clé publique de Bob (**EMS** pour l'anglais *Encrypted Master Secret*).

Dans le cas où Bob et Alice doivent tous les deux s'identifier, la négociation peut s'effectuer de la manière suivante :

1. Alice envoie un « Hello » à Bob.

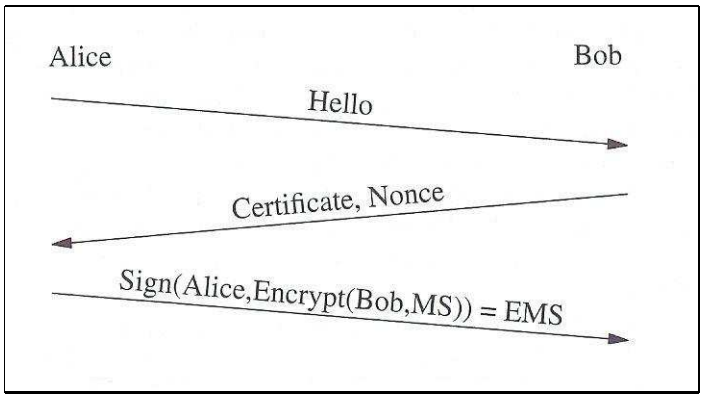

FIG. 4.5 – Identification dans les deux sens

2. Bob répond en lui envoyant son certificat et un nombre aléatoire, appelé **nonce** (pour *Number used ONCE*) ; il y en aura un par session. Un attaquant ne pourra donc pas, plus tard, envoyer le message (3) ci-dessous pour faire croire qu'il s'agit d'Alice (ce qui déjoue ce qui est appelé *replay attack* en anglais).
3. Alice envoie alors le secret à partager MS sous forme d'un EMS qui tient compte de MS, de la clé publique de Bob mais aussi du nonce reçu.

4.1.6.3 Un protocole de transfert simple

On peut associer intégrité du message et identification des extrémités avec ce qu'on appelle un **code d'identification de message** (**MAC** pour l'anglais *Message Authentication Code*) : il s'agit d'un condensé qui dépend à la fois du message et d'une clé.

Si nous envoyons toutes les données (cryptées) suivies du MAC, il faut attendre la fin de la session avant de s'assurer de l'intégrité du message. Nous ne pouvons pas commencer à traiter le début de ce qui a déjà été reçu. Il vaut donc mieux diviser le message en petites unités, chacune ayant son propre MAC, appelées **enregistrements**.

Les enregistrements peuvent avoir la structure suuivante :

Longueur	Numéro	Type	Données	MAC

- Nous pourrions envoyer des enregistrements de longueur fixe mais ceci n'est pas très efficace. Il vaut mieux des enregistrements de longueur variable. Un champ doit donc spécifier la longueur de l'enregistrement.
- Le second champ permet de numéroter les enregistrements au moment de l'envoi. Le destinataire pourra ainsi vérifier qu'il a bien reçu tous les enregistrements.
- Le type permet de distinguer les messages de données des messages de contrôle, en particulier des messages de fermeture de la connexion. Dans le cas le plus simple, on aura le type 0 pour les messages de données et 1 pour les messages de fermeture.
- Le champ données est constitué des données sous forme cryptée.

Puisque ce n'est pas une bonne idée d'utiliser la même clé pour plusieurs opérations de cryptage, lorsqu'un expéditeur veut envoyer des données, il commence par calculer deux clés : une clé E de cryptage des données et une clé MAC M. Lors d'une connexion, quatre clés sont utilisées : E_{CS} et M_{CS} pour le client ainsi que E_{SC} et M_{SC} pour le serveur.

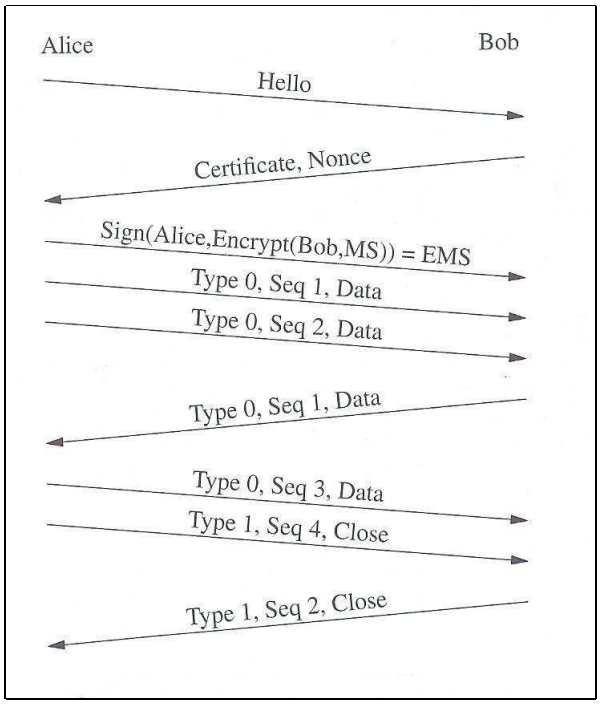

FIG. 4.6 – Protocole de transfert simple

Lorsque le client, par exemple, veut crypter un bloc de données D de longueur L, de type t dont le numéro de séquence est n :

1. Il calcule un MAC de $n||t||D$, où $||$ désigne la concaténation, en utilisant la clé M_{CS}, noté M.
2. Il crypte D en utilisant la clé E_{CS}, obtenant C.
3. Il envoie $L||n||t||C||M$, où t est le type du message.

Lorsque le serveur veut lire le bloc de données :

1. Il commence par lire L.
2. Il peut en déduire n, t, C et M.
3. Il décrypte C pour obtenir D en utilisant la clé E_{SC}.
4. Il calcule M' à partir de $n||t||D$ en utilisant la clé M_{SC}.
5. Si $M' = M$ alors il fait confiance au message.

4.2 Introduction à SSL

Bien entendu il reste beaucoup de paramètres à préciser pour obtenir un protocole à partir de nos principes théoriques. Nous allons voir ce qu'il en est dans le cas de SSLv3.

4.2.1 Généralités sur SSL

4.2.1.1 Historique

SSL (pour l'anglais *Secure Socket Layer*) fut conçu par Netscape pour sécuriser les transactions commerciales sur son navigateur *Netscape Navigator 1.1*, alors le leader du marché en 1994, en remplaçant `http` par `https` (avec 's' pour *secure*). La première version, maintenant connue sous le nom de SSLv1, de 1994 n'a pas été diffusée. La seconde version, **SSLv2**, a été diffusée en novembre 1994 [HIC–95] et une implémentation a été intégrée dans le navigateur en mars 1995.

Wagner et Goldberg [GW–96] ont trouvé une faille de sécurité, due au générateur de nombres pseudo-aléatoires utilisé, qui permettait de casser une connexion SSL en moins d'une heure.

Netscape recruta un consultant en sécurité bien connu, Paul Kocker, pour travailler avec Allan Freier et Phil Karlton afin de développer une nouvelle version de SSL. Cette troisième version, **SSLv3** [FKK–96], est sortie fin 1995.

Microsoft a conçu également un protocole analogue, **PCT** pour *Private Communicatio Technology*, également en 1995 [BLSSY–95].

En mai 1996, l'*Internet Engineering Task Force* (IETF) a mis en place un groupe de travail *Transport Layer Security* (**TLS**) pour standardiser les protocoles du type SSL. TLS a été publié comme [RFC 2246] en janvier 1999, avec peu de différences par rapport à SSLv3.

4.2.1.2 Place du protocole dans la suite TCP/IP

SSL/TLS est un protocole qui prend place entre le protocole de la couche de transport (dans les faits uniquement TCP) et la couche application, comme le montre la figure 4.7.

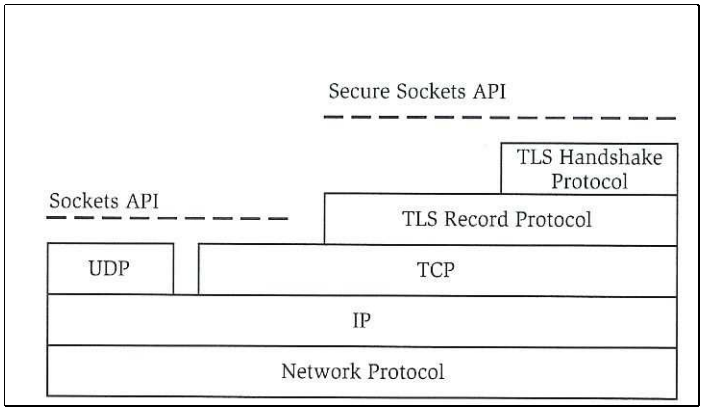

Fig. 4.7 – Place du protocole SSL dans la suite TCP/IP

4.2.1.3 Web sécurisé : le cas de HTTPS

Comme nous l'avons déjà dit, SSL est une suite complète de protocoles sécurisés. Cependant elle est surtout implémentée pour sécuriser les navigateurs Web.

L'utilisateur commencera son URL par `https://` au lieu du `http://` habituel. Le navigateur (sur lequel SSL est implémenté) fera alors appel au numéro de port 443 au lieu du numéro de port 80 usuel. Ceci signifie qu'il faut intercaler SSL/TLS entre TCP et HTTP. C'est l'implémentation qui s'occupe de tout.

4.2.2 Les enregistrements SSL

Le flux des données est transmis sous la forme d'une série de **fragments**, chaque fragment étant protégé et transmis de façon individuelle. Pour transmettre un fragment, on commence par calculer son MAC. La concaténation du fragment et de son MAC est crypté, obtenant ainsi une **charge cryptée** (*encrypted payload* en anglais). On lui attache un en-tête, obtenant ainsi un **enregistrement SSL**.

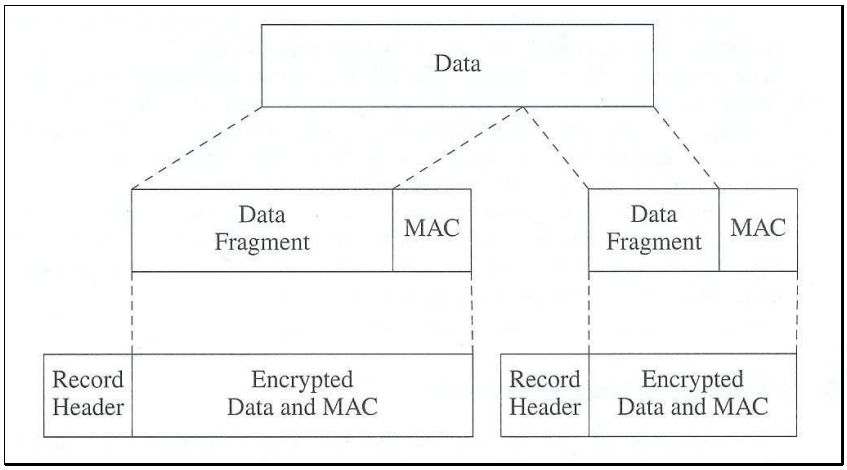

FIG. 4.8 – Fragmentation des données SSL

La structure d'un en-tête d'un enregistrement SSL est montré sur la figure suivante[2] :

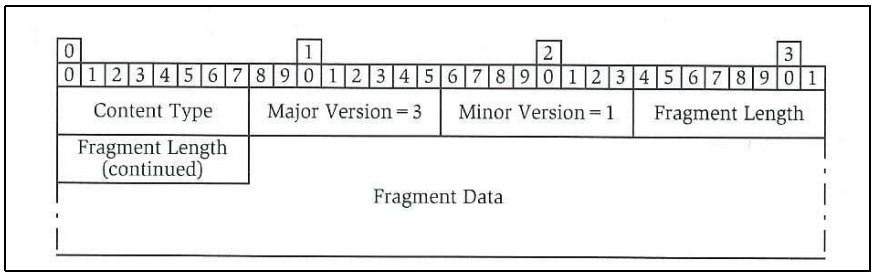

FIG. 4.9 – En-tête d'un enregistrement SSL

[2]Ces structures ne sont pas définies par rapport à des octets mais de façon analogue à des structures du langage C. Nous les présentons de cette façon pour plus de lisibilité.

- Le **type de contenu** occupe le premier octet. Il indique si l'enregistrement comporte des données ou des messages de contrôle. Il a quatre valeurs possibles :
  - 20, ou `change_cipher_spec`, pour un changement de spécification de chiffrement ;
  - 21, ou `alert`, pour une alerte SSL/TLS, permettant soit de signaler une erreur, soit de clore la session ;
  - 22, ou `handshake`, pour un message de négociation ;
  - 23, ou `application_data`, pour des données.
- Le deuxième et le troisième octets précisent le nombre majeur et le nombre mineur du numéro de version de SSL/TLS : 2.0 pour SSLv2, 3.0 pour SSLv3 et 3.1 pour TLS.
- Les deux octets suivants spécifient la longueur du fragment, chaque fragment devant être d'une taille inférieure à 2^{14} octets après décompression.

4.3 Le protocole de négociation

Le protocole de négociation est utilisé pour déterminer les algorithmes (de cryptage et de compression) à utiliser, pour établir les clés qui seront utilisées pour ces algorithmes et, éventuellement, pour identifier le client.

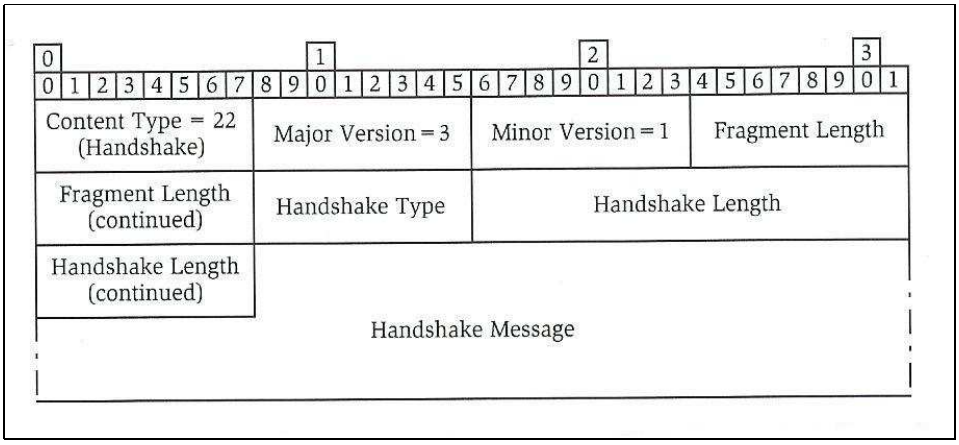

FIG. 4.10 – Enregistrement de négociation SSL

Les messages de négociation sont transmis à travers des enregistrements qui commencent comme ce qui est indiqué à la figure 4.10 : l'octet de type de l'en-tête SSL est 22 ; cet en-tête est suivi d'un octet de type de négociation, de trois octets de longueur puis du message de négociation prorprement dit.

4.3.1 Les étapes de la négociation

La négociation s'effectue de la façon suivante (voir figure 4.11) :

1. La phase de négociation est initiée par le client qui envoie un message du type « `Client Hello` » (type 1 de négociation). Ceci indique que le client désire établir une session de sécurité sur sa connexion TCP. Il envoie un nombre aléatoire (qui sera utilisé pour la détermination des clés), un identificateur de session et la liste des algorithmes de sécurité et de compression qu'il est capable d'utiliser.

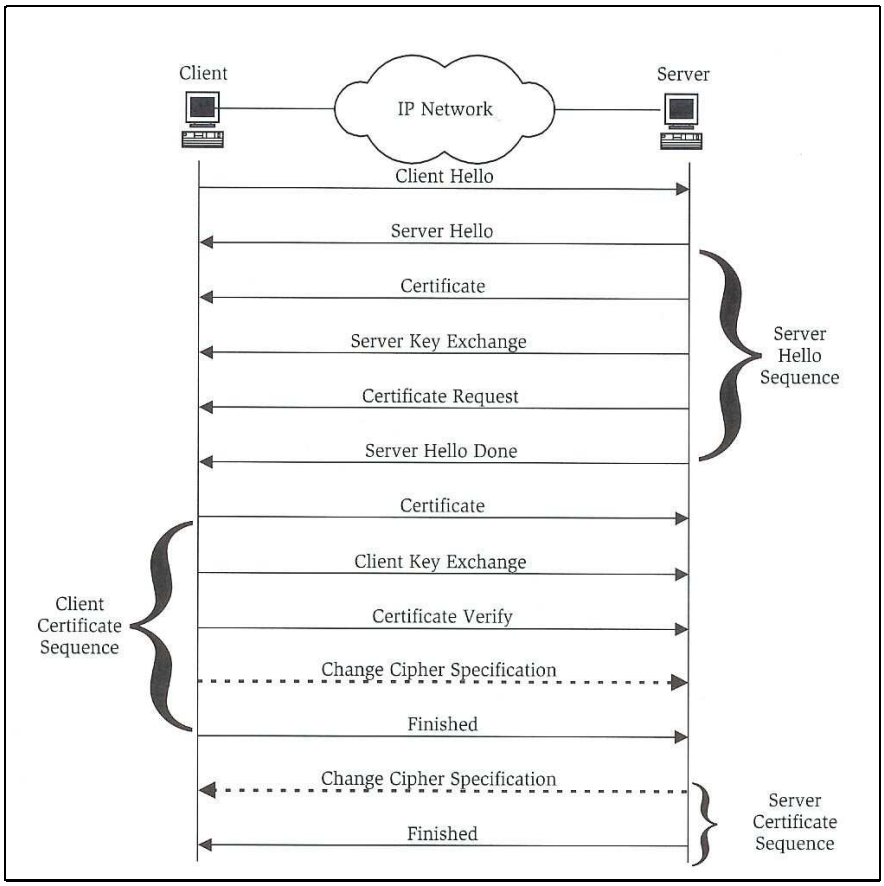

FIG. 4.11 – Protocole de négociation SSL

Les trois paramètres négociables sont la version de SSL/TLS, les algorithmes de sécurité et la méthode de compression.

2. Le serveur répond par une série des messages qui définissent les paramètres de sécurité du serveur :
   - Il commence par un message du type « `Server Hello` » (type 2 de négociation). Celui-ci fait savoir au client que le serveur a bien reçu son « Client Hello » (en envoyant le même numéro de session), envoie également un nombre aléatoire et indique l'algorithme de compression et l'algorithme de sécurité qu'il a choisi parmi ceux proposés par le client.
   - Le serveur envoie ensuite un message du type « `Certificate` » (type 11 de négociation) pour s'identifier si le client l'a demandé.
   - Le serveur envoie alors un message du type « `Server Key Exchange` » (type 12 de négociation) pour transmettre les clés de sécurité si le client l'a demandé.
   - Le serveur envoie éventuellement un message du type « `Certificate Request` » (type 13 de négociation) s'il désire que le client s'identifie.
   - Le serveur indique enfin qu'il a terminé sa série de messages en envoyant un message du type « `Server Hello Done` » (type 14 de négociation). Ceci permet au client de savoir qu'il n'a pas à attendre un message du type « `Certificate Request` ».

3. Le client s'embarque alors dans une série de messages qui définissent les paramètres de sécurité du client :
   - Le message « `Certificate` » (type 11 de négociation, rappelons-le) identifie le client, uniquement en réponse à un « `Certificate Request` ».
   - Le message « `Client Key exchange` » (type 16 de négociation) permet de transmettre les clés de sécurité.
   - Le client confirme qu'il a accepté le certificat envoyé par le serveur (si c'est le cas) en envoyant un message du type « `Certificate Verify` » (type 15 de négociation, qui devrait bien sûr plutôt s'appeler « `Certificate Verified` »).
   - Le client envoie alors un message du type « `Change Cipher Specification` » (type 20 des enregistrements, rappelons-le) pour indiquer que les messages suivants seront cryptés.
   - Le client termine par un message (crypté) du type « `Finished` » (type 20 de négociation).
4. Le serveur répond en envoyant :
   - un message du type « Change Cipher Specification » ;
   - suivi d'un message du type « `Finished` ».

L'ordre des messages est imposé. La suite de message montrée à la figure 4.11 peut être réinitialisée durant la session TCP sécurisée pour renégocier les paramètres de sécurité.

Maintenant les données peuvent être échangées.

4.3.2 Client Hello

4.3.2.1 Structure d'un message `Client Hello`

La structure d'un message `Client Hello` est représentée à la figure 4.12.

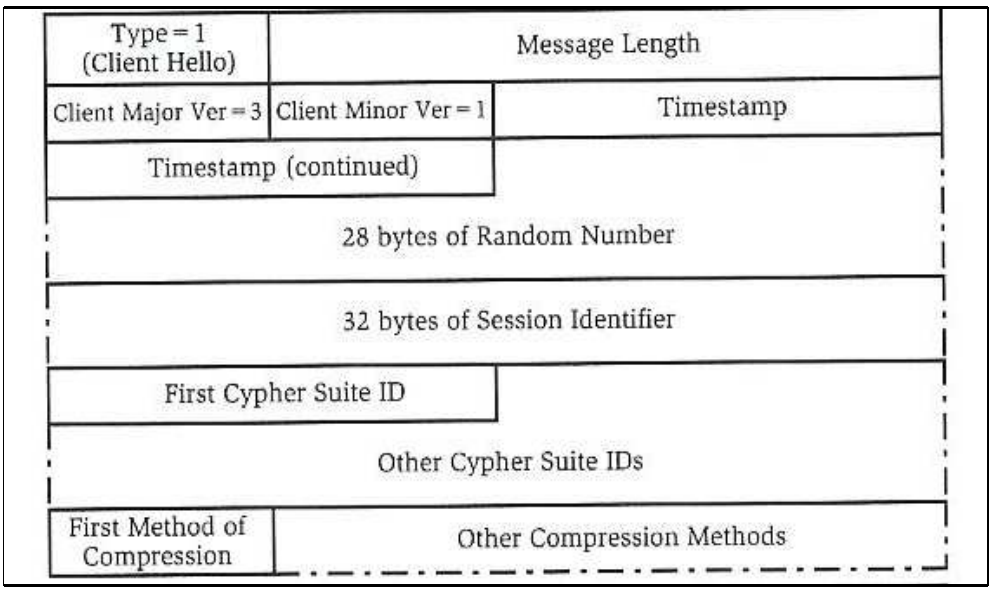

FIG. 4.12 – Structure d'un message de négociation `Client Hello` de SSL

On a l'en-tête SSLv3, suivi de l'en-tête des messages de négociation avec le type égal à 1, suivi de neuf champs :

Cipher Suite	Auth	Key Exchange	Encryption	Digest	Number
TLS_RSA_WITH_NULL_MD5	RSA	RSA	NULL	MD5	0x0001
TLS_RSA_WITH_NULL_SHA	RSA	RSA	NULL	SHA	0x0002
TLS_RSA_EXPORT_WITH_RC4_40_MD5	RSA	RSA_EXPORT	RC4_40	MD5	0x0003
TLS_RSA_WITH_RC4_128_MD5	RSA	RSA	RC4_128	MD5	0x0004
TLS_RSA_WITH_RC4_128_SHA	RSA	RSA	RC4_128	SHA	0x0005
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA	RSA_EXPORT	RC2_40_CBC	MD5	0x0006
TLS_RSA_WITH_IDEA_CBC_SHA	RSA	RSA	IDEA_CBC	SHA	0x0007
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA	RSA_EXPORT	DES40_CBC	SHA	0x0008
TLS_RSA_WITH_DES_CBC_SHA	RSA	RSA	DES_CBC	SHA	0x0009
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	RSA	3DES_EDE_CBC	SHA	0x000A
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	RSA	DH_DSS_EXPORT	DES_40_CBC	SHA	0x000B
TLS_DH_DSS_WITH_DES_CBC_SHA	DSS	DH	DES_CBC	SHA	0x000C
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DSS	DH	3DES_EDE_CBC	SHA	0x000D
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA	DH_EXPORT	DES_40_CBC	SHA	0x000E
TLS_DH_RSA_WITH_DES_CBC_SHA	RSA	DH	DES_CBC	SHA	0x000F
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	RSA	DH	3DES_EDE_CBC	SHA	0x0010
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	DSS	DHE_EXPORT	DES_40_CBC	SHA	0x0011
TLS_DHE_DSS_WITH_DES_CBC_SHA	DSS	DHE	DES_CBC	SHA	0x0012
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DSS	DHE	3DES_EDE_CBC	SHA	0x0013
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA	DHE_EXPORT	DES_40_CBC	SHA	0x0014
TLS_DHE_RSA_WITH_DES_CBC_SHA	RSA	DHE	DES_CBC	SHA	0x0015
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	RSA	DHE	3DES_EDE_CBC	SHA	0x0016
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5	-	DH_EXPORT	RC4_40	MD5	0x0017
TLS_DH_anon_WITH_RC4_128_MD5	-	DH	RC4_128	MD5	0x0018
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA	-	DH	DES_40_CBC	SHA	0x0019
TLS_DH_anon_WITH_DES_CBC_SHA	-	DH	DES_CBC	SHA	0x001A
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	-	DH	3DES_EDE_CBC	SHA	0x001B
TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA †	RSA	RSA	DES_CBC	SHA	0x0062
TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA †	RSA	RSA	DES_CBC	SHA	0x0063
TLS_RSA_EXPORT1024_WITH_RC4_56_SHA †	RSA	RSA	RC4_56	SHA	0x0064
TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA †	RSA	RSA	RC4_56	SHA	0x0065
TLS_DHE_DSS_WITH_RC4_128_SHA †	RSA	RSA	RC4_56	SHA	0x0066

FIG. 4.13 – Algorithmes de sécurisation

– Les deux premiers octets spécifient le plus haut numéro de version que le client est capable de mettre en œuvre.

On suppose implicitement que lorsque le client donne un numéro de version, il peut également mettre en œuvre toutes les versions antérieures.

– Le second champ, en fait sous-champ constitué de quatre octets, indique l'heure sous la forme UNIX, en secondes depuis le 1^{er} janvier 1970, zéro heure GMT.

– Le troisième champ, en fait sous-champ constitué de 28 octets, est un nombre engendré de façon aléatoire par le client. Ces deux sous-champs constituent le champ appelé `random`.

– Le quatrième champ, constitué d'un octet, précise la longueur (en octets) du champ suivant.

– Le cinquième champ est constitué d'un mot pouvant aller jusqu'à 32 octets, représentant l'**identificateur de session**.

Le client précise un identificateur de session lorsqu'il désire réutiliser les clés d'une connexion précédente plutôt que d'en engendrer de nouvelles (puisque le calcul de ces clés prend beaucoup de temps).

– Les deux octets suivant précisent la longueur de la liste des algorithmes de sécurité.

– Vient ensuite la liste des algorithmes de sécurité. Chacun de ceux-ci est reconnu par un numéro (de deux octets), suivant le tableau de la figure 4.13 (dans laquelle les algorithmes marqués d'un † ne font pas partie du standard TLS). Cette liste est donnée dans l'ordre décroissant de préférence du client.

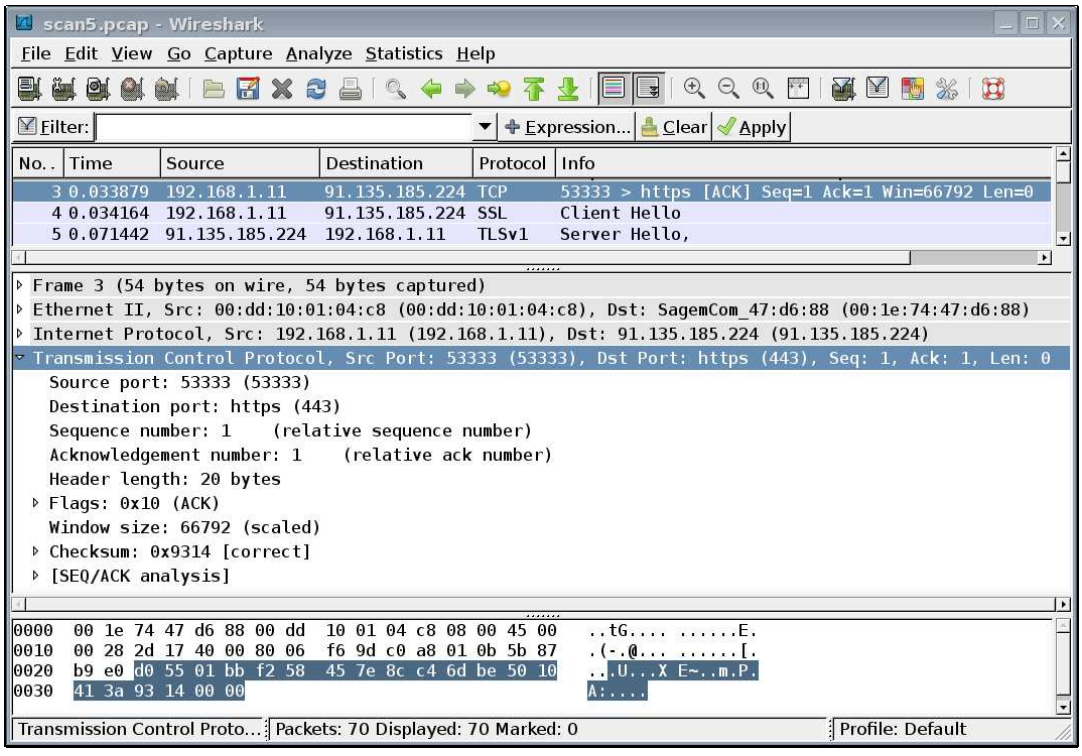

FIG. 4.14 – Appel à HTTPS

– L'octet qui suit la liste des algorithmes de sécurité indique la longueur de la suite des algorithmes de compression que comprend le client.

– Vient ensuite la liste des algorithmes de compression.

Les données cryptées sont aléatoires et donc les méthodes de compression ne donnent pas grand chose[3]. Si on veut compresser les données, il faut donc le faire avant le cryptage et donc dans le protocole SSL et non après, par exemple au momemt de la transmission sur le modem.

En fait ni SSLv3, ni TLS ne définissent d'algorithmes de compression. La compression n'est donc presque jamais utilisée avec SSL : une seule « méthode de compression » est proposée, la méthode `NULL`. L'implémentation OpenSSL propose des méthodes de compression.

4.3.2.2 Analyse d'une trame SSL

Nous allons analyser les trames SSL pour vérifier, autant que faire se peut, la structure des messages SSL et que nous ne pouvons pas facilement récupérer le mot de passe (ou autres informations critiques).

Capture des trames.- Vous utilisez certainement un service Web qui donne accès à un certain moment à un accès sécurisé (consultation de votre compte bancaire, commande en ligne, etc.). Accédez à ce service, préparez-vous à passer à l'accès sécurisé en indiquant votre identifiant et votre mot de passe, faites démarrer la capture de trames `Wireshark` et appuyez sur le bouton qui donne accès au service. Fermez la capture des trames (il y en a déjà une centaine en général) dès que vous avez accès au service.

Recherchez la première trame SSL : c'est facile grâce aux indications de `Wireshark`; ce le serait un peu moins si on devait le faire à la main. Il s'agit évidemment d'un `Client Hello`. Remarquez qu'avant cette trame, on transmet une requête à un serveur HTTPS (figure 4.14) : il s'agit d'un segment TCP dont le numéro de port est 443 (et non 80, comme pour HTTP). Après acquittement du serveur puis du client, le client envoie un `Client Hello` au serveur.

Analyse du message `Client Hello`.- La seule façon de savoir qu'il s'agit d'une trame SSL est d'une part que le numéro de port du segment est 443 (numéro « bien connu » pour HTTPS) et, d'autre part, qu'il y a des données (contrairement à la trame précédente, par exemple). Analysons le premier message SSL (figure 4.15) :

– Le premier octet, celui de type, est égal `16h` soit 22, ce qui montre qu'il s'agit d'un message de négociation.

– Les deux octets suivants (`03h` et `01h`) montrent qu'il s'agit de la version 3.1, c'est-à-dire de TLSv1.0.

– Les deux octets suivants, (`00 9Eh`), spécifient la longueur des données, ici 158 octets, ce qui nous conduit jusqu'à la fin de la trame.

– Puisqu'il s'agit d'un message de négociation, le premier octet de données, ici `01h`, spécifie le type de message de négociation : il s'agit d'un `Client Hello`.

– Les trois octets suivants, ici `00 00 9Ah`, spécifient la longueur des données, soit 154 octets, ce qui nous amène jusqu'à la fin de la trame.

– Les deux octets suivants (`03h` et `01h`) montrent que la plus haute version de SSL acceptée par le client est 3.1.

– Les quatre octets suivants, ici `4A C5 B9 32h`, indiquent l'heure (vérifiez ce qui est donné par l'analyseur de trames à titre d'exercice, ici 2 octobre 2009, 10h 26 mn 26 s).

[3] Rappelons qu'il n'existe pas de méthode de compression universelle. Il faut connaître plus ou moins la structure des données pour lui appliquer une méthode de compression efficace.

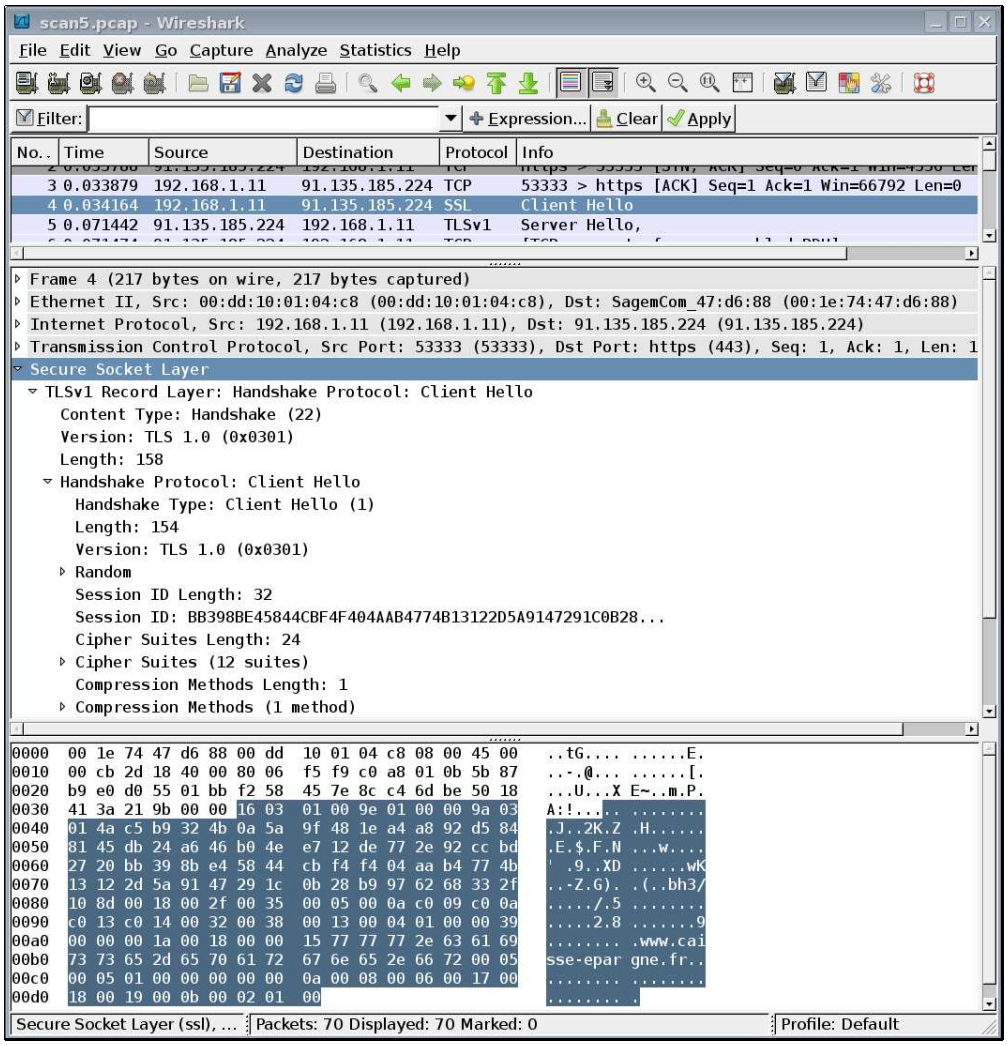

FIG. 4.15 – Client Hello

- Les 28 octets suivants contituent le nombre aléatoire engendré par le client et transmis au serveur. Il n'y a pas grand chose à en dire.
- L'octet suivant, ici `20h`, spécifie la longueur de l'identificateur de session. On a ici 32 octets donc un identificateur de session : le client désire réutiliser les clés d'une session antérieure.
- Les 32 octets suivants spécifient alors l'identificateur de sessions, sur lequel il n'y a pas grand chose à dire.
- Les deux octets suivants indiquent la longueur de la liste des algorithmes de sécurité que peut utiliser le client. On a ici `00 18h`, soit 24 octets et donc 12 algorithmes (chacun occupant deux octets).
- Les 24 octets suivants spécifient donc ces algorithmes de sécurité (figure 4.16). On pourra comparer ce que propose l'analyseur de trame avec le tableau de la figure 4.13.
- L'octet qui suit la liste des algorithmes de sécurité indique la longueur de la suite des

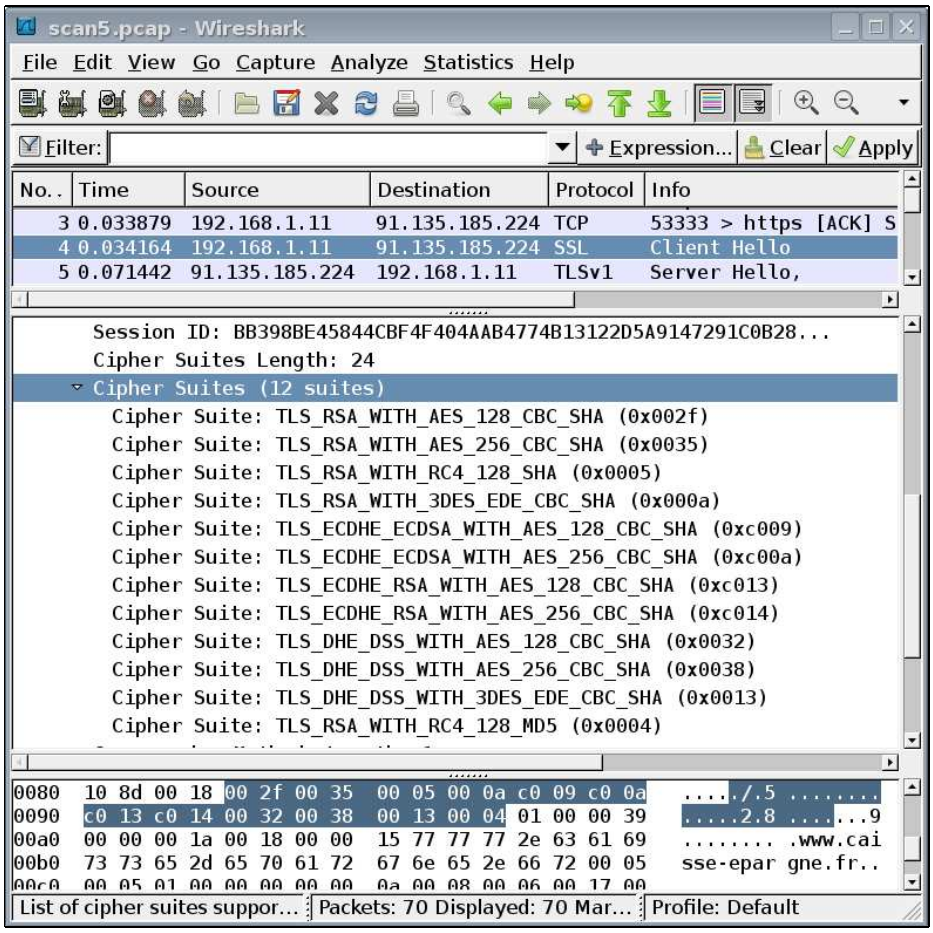

FIG. 4.16 – Liste des algorithmes de sécurité d'un Client Hello

algorithmes de compression que comprend le client, qui est ici égal à `01h`, comme dans presque tous les cas.

– Le dernier octet précise cette méthode de compression, à savoir `00h` pour la méthode `null`, c'est-à-dire qu'il n'y a pas de compression.

– Les deux octets suivants (ici `00 39h`) précisent la longueur de l'extension, soit 57 octets, qui ne nous occupera pas ici. On est parvenu à la fin de la trame.

4.3.3 Server Hello

4.3.3.1 Structure d'un message `Server Hello`

La structure d'un message `Server Hello` est représentée à la figure 4.17. Il comprend l'en-tête SSL, suivi de l'en-tête des message de négociation avec le type égal à 2, suivi de six champs :

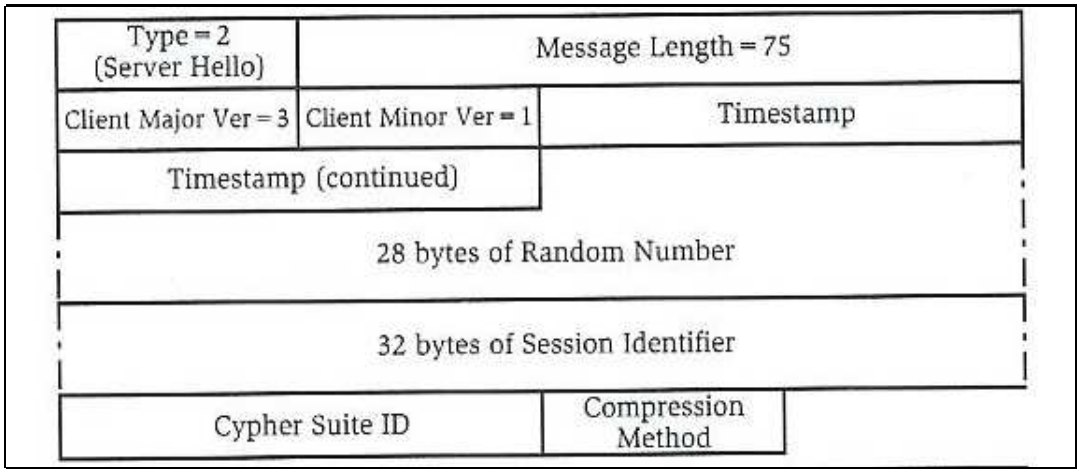

FIG. 4.17 – Structure d'un message de négociation `Server Hello`

- Les deux premiers octets spécifient la version, comme dans le cas du `Client Hello`.
- Le second champ, en fait un sous-champ constitué de quatre octets, indique l'heure sous la forme UNIX, en secondes depuis le 1^{er} janvier 1970, zéro heure GMT.
- Le troisième champ, en fait un sous-champ constitué de 28 octets, est un nombre engendré de façon aléatoire par le serveur. Ces deux sous-champs constituent, comme dans le cas du client, le champ appelé `random`.
- Le quatrième champ, constitué d'un octet, précise la longueur (en octets) du champ suivant.
- Le cinquième champ est constitué d'un mot pouvant aller jusqu'à 32 octets, représentant l'**identificateur de session**. Celui-ci pourra être réutilisé plus tard par le client pour démarrer une nouvelle session en utilisant les clés d'une session antérieure. Le serveur ne propose pas d'identificateur de session s'il ne veut pas qu'il en soit ainsi.
- Les deux octets suivant spécifie l'algorithme de sécurité retenu par le serveur dans la liste que lui a envoyé le client.
- Le dernier octets spécifie l'algorithme de compression retenu par le serveur dans la liste que lui a envoyé le client.

4.3.3.2 Analyse d'un message `Server Hello`

Dans notre exemple, la trame suivante est une trame contenant un message SSL du type `Server Hello` (figure 4.18) :

- On sait qu'il s'agit d'un message SSL car le port source du segment TCP est 443 (pour HTTPS).
- Le premier octet, celui de type, est égal `16h` soit 22, ce qui montre qu'il s'agit d'un message de négociation.

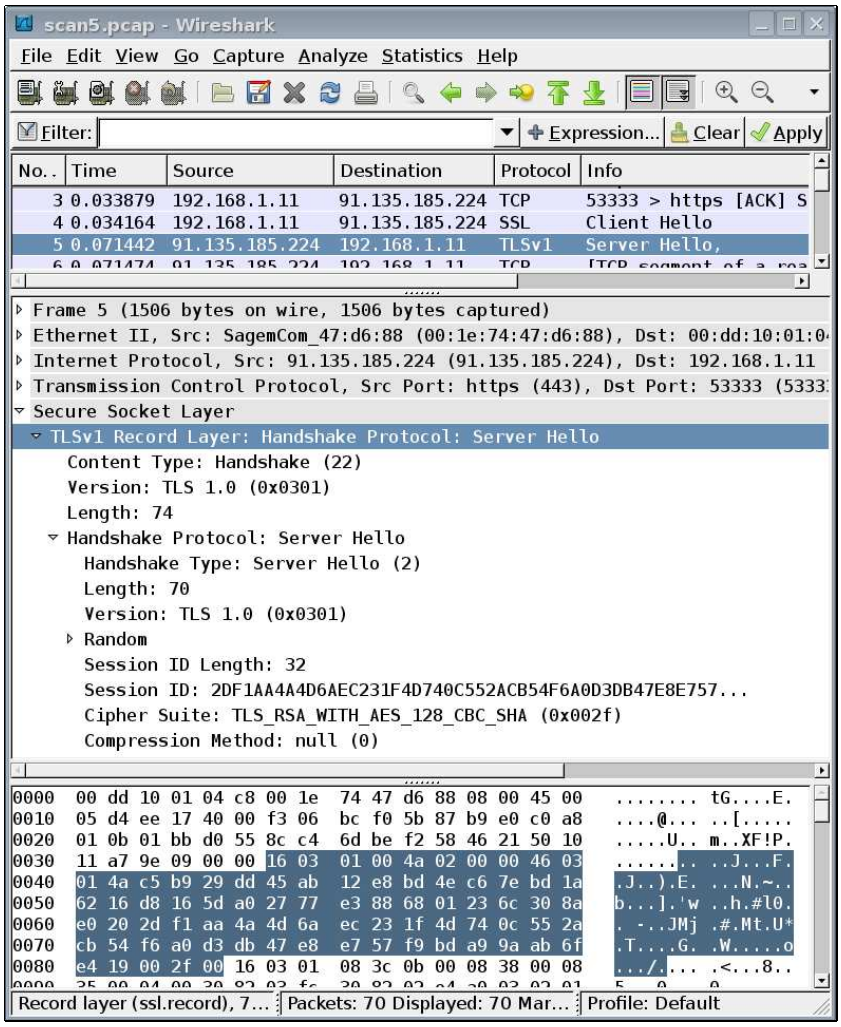

FIG. 4.18 – Server Hello

- Les deux octets suivants (`03h` et `01h`) montrent qu'il s'agit de la version 1.0 de TLS.
- Les deux octets suivants, (`00 4Ah`), spécifient la longueur des données, ici 74 octets, ce qui ne nous conduit pas jusqu'à la fin de la trame. La trame contient donc au moins un second message SSL.

  On remarquera qu'il n'est pas fait explicitement mention de ce second message SSL par l'analyseur de trames, si ce n'est par une virgule. D'où l'intérêt de savoir analyser les trames à la main.
- Puisqu'il s'agit d'un message de négociation, le premier octet de données, ici `02h`, spécifie le type de message de négociation : il s'agit d'un `Server Hello`.
- Les trois octets suivants, ici `00 00 46h`, spécifient la longueur des données, soit 70 octets, ce qui nous amène jusqu'à la fin du premier message SSL.
- Les deux octets suivants (`03h` et `01h`) montrent que le serveur a choisi la version 1.0 de TLS, la plus haute version proposée par le client.

- Les quatre octets suivants, ici `4A C5 B9 29h`, indiquent l'heure (vérifiez ce qui est donné par l'analyseur de trames à titre d'exercice, ici 2 octobre 2009, 10h 26mn 17s).

  On remarquera que l'heure d'envoi de la réponse est antérieure à celle d'envoi de la requête. Ceci est évidemment dû au fait que l'un des ordinateurs, pour ne pas dire les deux, ne sont pas à l'heure. Ce n'est pas grave puisque ce champ n'est pas utilisé comme estampille temporelle. Seul le champ `random` est utilisé pour générer les clés.
- Les 28 octets suivants contituent le nombre aléatoire engendré par le serveur et transmis au client. Il n'y a pas plus à en dire que dans le cas du client.
- L'octet suivant, ici `20h` soit 32, spécifie la longueur de l'identificateur de session.
- Les trente deux octets suivant spécifient donc dans notre cas l'identificateur de session, sur lequel il n'y a pas grand chose à dire.
- Les deux octets suivant, ici `00 2Fh`, spécifient les algorithmes de sécurité choisis par le serveur dans la liste proposée par le client. On a donc RSA pour l'identification, AES_128 pour l'échange de clés, CBC pour le cryptage et SHA pour les condensés.

  On remarquera que le serveur a choisi le premier système d'algorithmes de sécurité proposé par le client.
- Le dernier octet spécifie la méthode de compression choisie. On ne peut qu'avoir zéro (pour pas de méthode de compression) puisque le client n'en proposait pas. C'est bien le cas.

4.3.4 Message Certificate

4.3.4.1 Structure d'un message `Certificate`

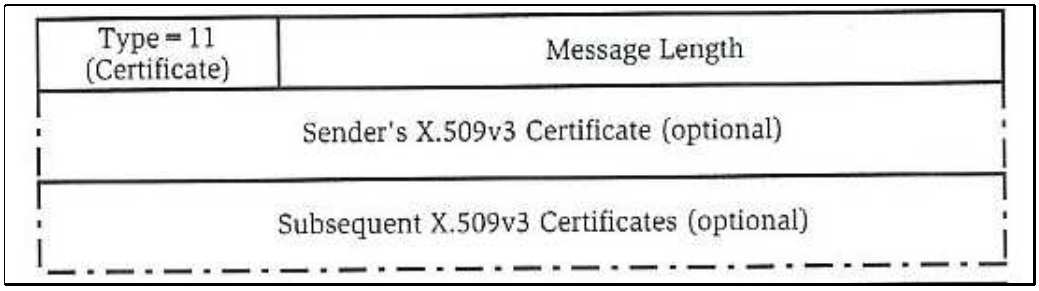

FIG. 4.19 – Structure d'un message de négociation `Certificate`

La structure d'un message `Certificate` est représentée à la figure 4.19. Il comprend l'en-tête SSL, suivi de l'en-tête des messages de négociation dont le type est égal à 11, suivi d'une suite de certificats X.509.

Nous n'entrerons pas ici dans la structure de tels certificats, définie par l'ITU en 1988 [ITU–1988] puis comme [RFC 2459].

4.3.4.2 Analyse d'un message `Certificate`

Intéressons-nous maintenant aux deuxième message SSL de la trame (figure 4.20) :

- Le premier octet, celui de type, est égal `16h` soit 22, ce qui montre qu'il s'agit d'un message de négociation.
- Les deux octets suivants (`03h` et `01h`) montrent qu'il s'agit de la version 1.0 de TLS.

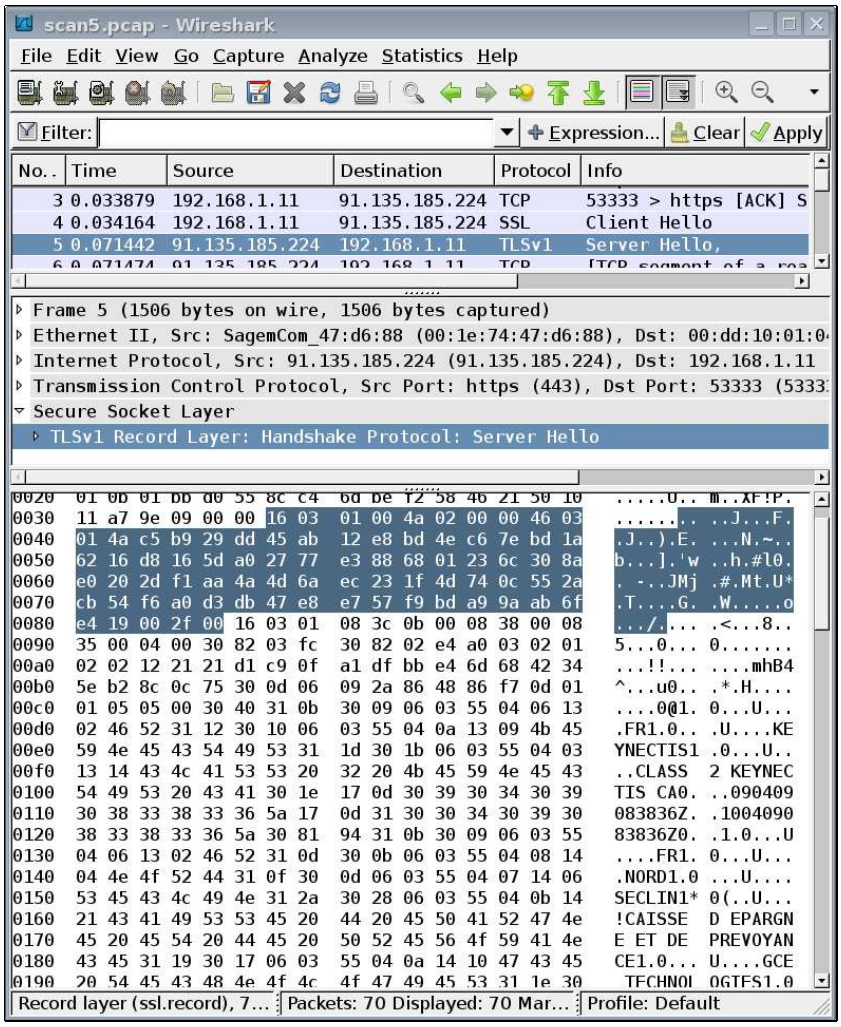

FIG. 4.20 – Début d'un message SSL « Certificate »

– Les deux octets suivants, (`08 3Ch`), spécifient la longueur des données, ici 2 108 octets, ce qui nous conduit au-delà de la fin de la trame.

  C'est la raison pour laquelle l'analyseur de trames ne s'est pas occupé de ce message. Le message SSL a été fragmenté par TCP. On retrouve le message SSL réassemblé comme (pseudo-)trame 8.

Intéressons-nous donc plutôt à la pseudo-trame 8 et plus exactement au premier message SSL qu'elle contient (figure 4.21) :

– Le premier octet, celui de type, est égal `16h` soit 22, ce qui montre qu'il s'agit d'un message de négociation.

– Les deux octets suivants (`03h` et `01h`) montrent qu'il s'agit de la version 1.0 de TLS.

– Les deux octets suivants, (`08 3Ch`), spécifient la longueur des données. On retrouve 2 108 octets, c'est-à-dire que le second message SSL de la trame précédemment étudiée se retrouve ici (comme premier message SSL). On ne vas pas jusqu'au bout de la trame, il y a donc au

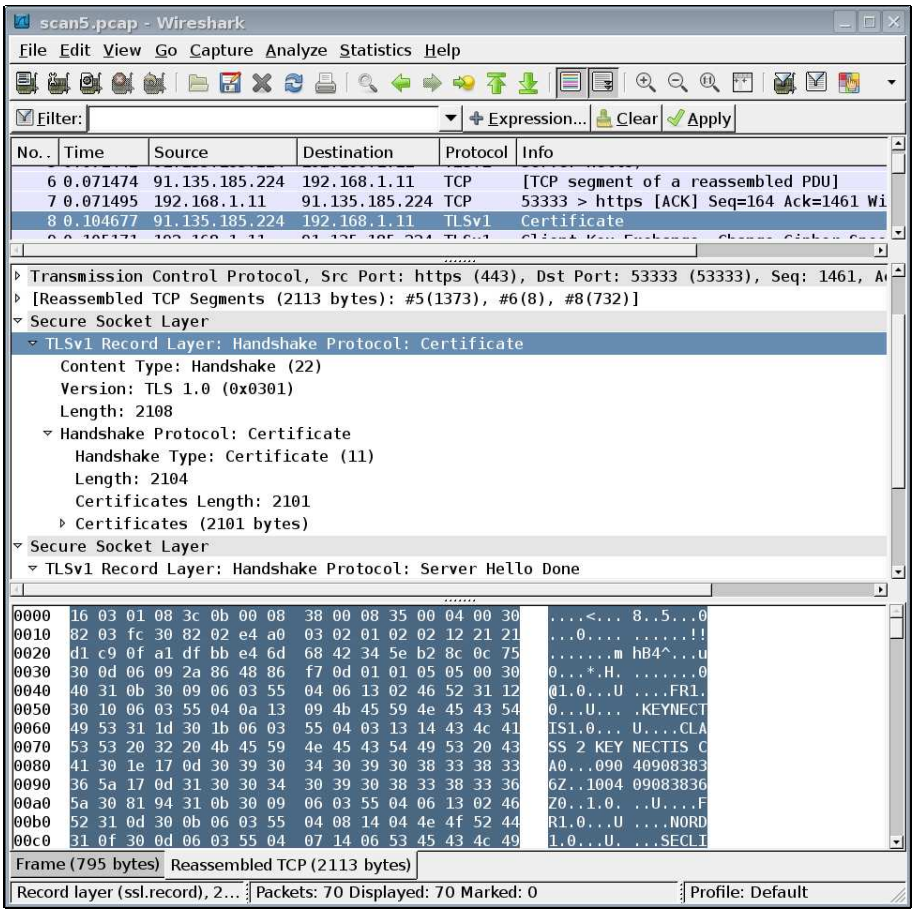

FIG. 4.21 – Message SSL « Certificate »

moins un autre message SSL.

- Puisqu'il s'agit d'un message de négociation, le premier octet de données, ici `0Bh`, soit 11, spécifie le type de message de négociation : il s'agit d'un `Certificate`.
- Les trois octets suivants, ici `00 08 38h`, spécifient la longueur des données, soit 2 104 octets, ce qui nous amène jusqu'à la fin du message SSL.
- Les octets suivants concernent le certificat et ne nous intéresseront pas ici.

4.3.5 Message « Server Hello Done »

4.3.5.1 Structure d'un message `Server Hello Done`

La structure d'un message `Server Hello Done` est représentée à la figure 4.22. Il comprend l'en-tête SSL, suivi de l'en-tête des messages de négociation, dont le type est égal à 14 et la longueur du message toujours égale à 0.

FIG. 4.22 – Structure d'un message de négociation `Server Hello Done`

4.3.5.2 Analyse d'un message `Server Hello Done`

Intéressons-nous maintenant aux deuxième message SSL de la pseudo-trame (figure 4.23). Il est un peu plus difficile de voir où commence ce message à cause de la fragmentation des messages effectuée par TCP mais on le peut :

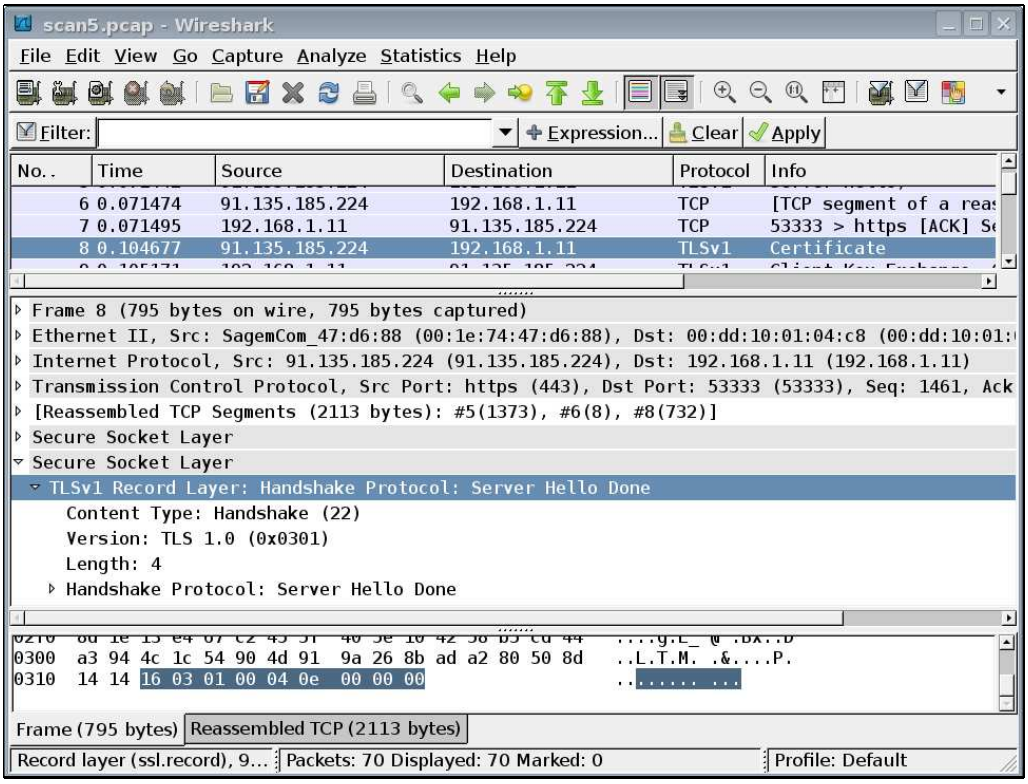

FIG. 4.23 – Server Hello Done

- Le premier octet, celui de type, est égal `16h` soit 22, ce qui montre qu'il s'agit d'un message de négociation.
- Les deux octets suivants (`03h` et `01h`) montrent qu'il s'agit de la version 1.0 de TLS.
- Les deux octets suivants, (`00 04h`), spécifient la longueur des données, on retrouve bien quatre octets, ce qui nous conduit jusqu'à la fin de la trame.
- Puisqu'il s'agit d'un message de négociation, le premier octet de données, ici `0Eh`, soit 14, spécifie le type de message de négociation. On a donc ici un `ServerHelloDone`.
- Les trois octets suivants, ici `00 00 00h`, spécifient la longueur des données, et on retrouve bien soit zéro octet. On a donc terminé.

4.3.6 Message « Key Exchange »

4.3.6.1 Structure d'un message `Key Exchange`

La structure d'un message `Key Exchange` est représentée à la figure 4.24. Il comprend l'en-tête SSL, suivi de l'en-tête des messages de négociation, dont le type est égal à 16 pour un client et à 12 pour un serveur.

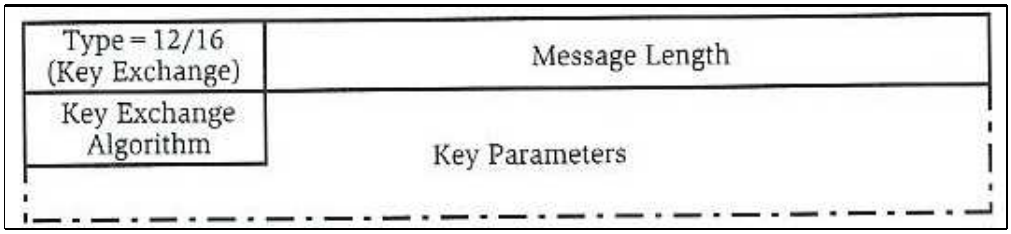

FIG. 4.24 – Structure d'un message de négociation `Key Exchange`

Les données dépendent de l'algorithme de cryptage choisi. Dans le cas de RSA, par exemple, le client engendre de façon aléatoire un entier ayant le nombre de bits décidé, appelé *PreMaster-Secret* en anglais, qui apparaît sous forme cryptée en utilisant la clé du client.

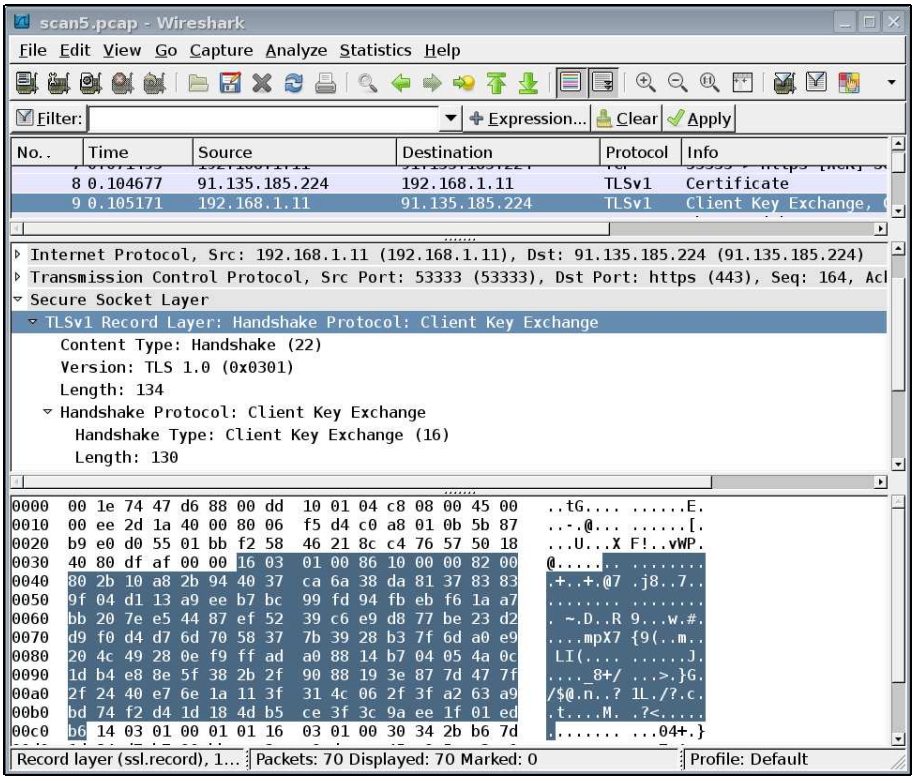

FIG. 4.25 – Client Key Exchange

4.3.6.2 Analyse d'un message `Client Key Exchange`

Dans notre exemple, la trame suivante est une trame qui contient trois messages SSL : `Client Key Exchange`, `Change Cipher Spec` et `Encrypted Handshake Message`. Intéressons-nous au premier de ces messages (figure 4.25) :

- Le premier octet, celui de type, est égal `16h` soit 22, ce qui montre qu'il s'agit d'un message de négociation.
- Les deux octets suivants (`03h` et `01h`) montrent qu'il s'agit de la version 1.0 de TLS.
- Les deux octets suivants, (`00 86h`), spécifient la longueur des données, ici 134 octets, ce qui ne nous conduit pas jusqu'à la fin de la trame. La trame contient donc au moins un second message SSL.
- Puisqu'il s'agit d'un message de négociation, le premier octet de données, ici `10h` soit 16, spécifie le type de message de négociation : il s'agit ici d'un `Client Key Exchange`.
- Les trois octets suivants, ici `00 00 82h`, spécifient la longueur des données, soit 130 octets, ce qui nous amène jusqu'à la fin du premier message SSL.
- Les 130 octets de données correspondent au *PreMasterSecret* chiffré en utilisant la clé privée du client.

  On ne peut pas en dire beaucoup plus et, en particulier, il est difficile de retrouver la valeur du *PreMasterSecret*, ce qui est le but recherché.

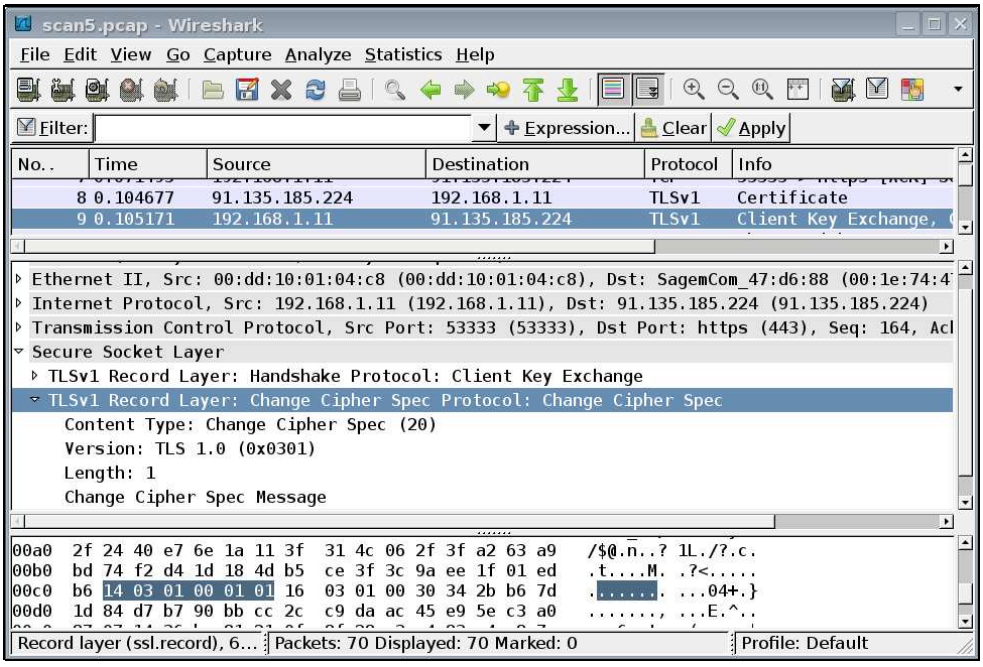

FIG. 4.26 – Change Cipher Spec

4.3.7 Message « Change Cipher Spec »

4.3.7.1 Structure d'un message `Change Cipher Spec`

Un message `Change Cipher Spec` est constitué d'en-tête SSL, dont le type est égal à 20, suivi des données sur un octet, dont la valeur n'a pas d'importance et qui est en général égal à 1.

Ce message indique que l'expéditeur utilisera désormais les paramètres négociés (algorithmes de sécurité, clés et méthode de compression) dans les messages futurs.

4.3.7.2 Analyse d'un message `Change Cipher Spec`

Intéressons-nous maintenant aux deuxième message SSL de la trame (figure 4.26) :

- Le premier octet, celui de type, est égal `14h` soit 20, ce qui montre qu'il s'agit d'un message `Change Cipher Spec`.
- Les deux octets suivants (`03h` et `01h`) montrent qu'il s'agit de la version 1.0 de TLS.
- Les deux octets suivants, (`00 01h`), spécifient la longueur des données, ici un octet, ce qui ne nous conduit pas jusqu'à la fin de la trame. La trame contient donc au moins un troisième message SSL.
- Puisqu'il s'agit d'un message de changement, l'octet des données n'a pas de sens. Il est ici égal à un.

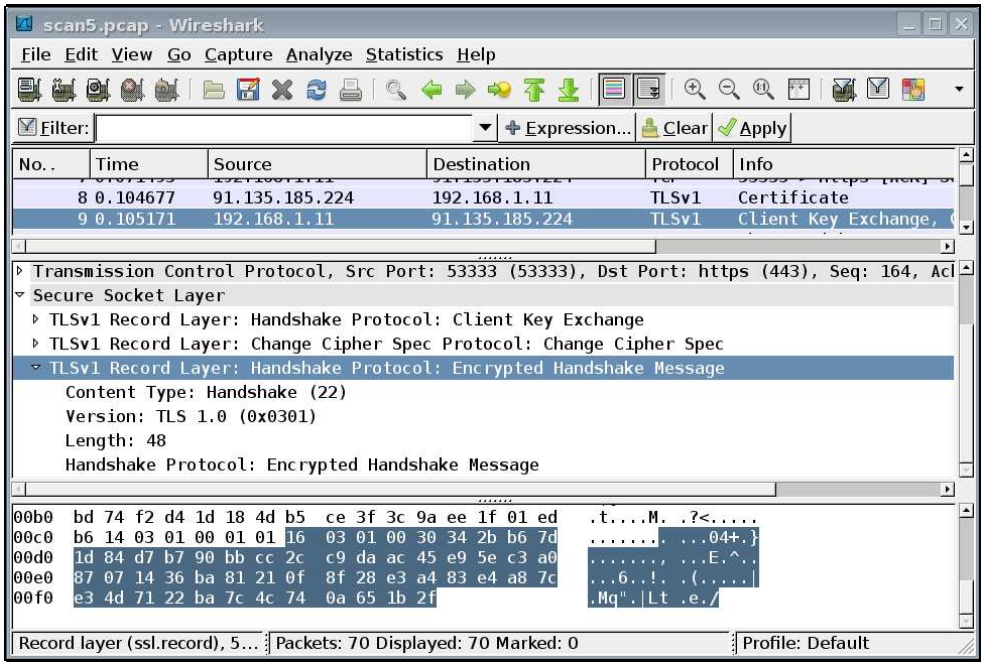

FIG. 4.27 – Finished

4.3.8 Message « Finished »

4.3.8.1 Structure d'un message `Finished`

Le message `Finished` est le premier message qui apparaît sous forme cryptée. Nous ne détaillerons pas la structure des données brutes ici.

4.3.8.2 Analyse d'un message `Finished`

Intéressons-nous maintenant aux troisième message SSL de la trame (figure 4.27) :

- Le premier octet, celui de type, est égal `16h` soit 22, ce qui montre qu'il s'agit d'un message de négociation.
- Les deux octets suivants (`03h` et `01h`) montrent qu'il s'agit de la version 1.0 de TLS.
- Les deux octets suivants, (`00 30h`), spécifient la longueur des données, ici 48 octets, ce qui nous conduit jusqu'à la fin de la trame. La trame contenait donc trois messages SSL.
- Les 48 octets de données sont sous forme cryptée. Il n'est donc pas possible d'en déterminer la structure.

  Tout ce qu'on peut dire est qu'il s'agit certainement d'un message `Finished` vu l'endroit où il est placé et puisque c'est le seul type de message de négociation crypté.

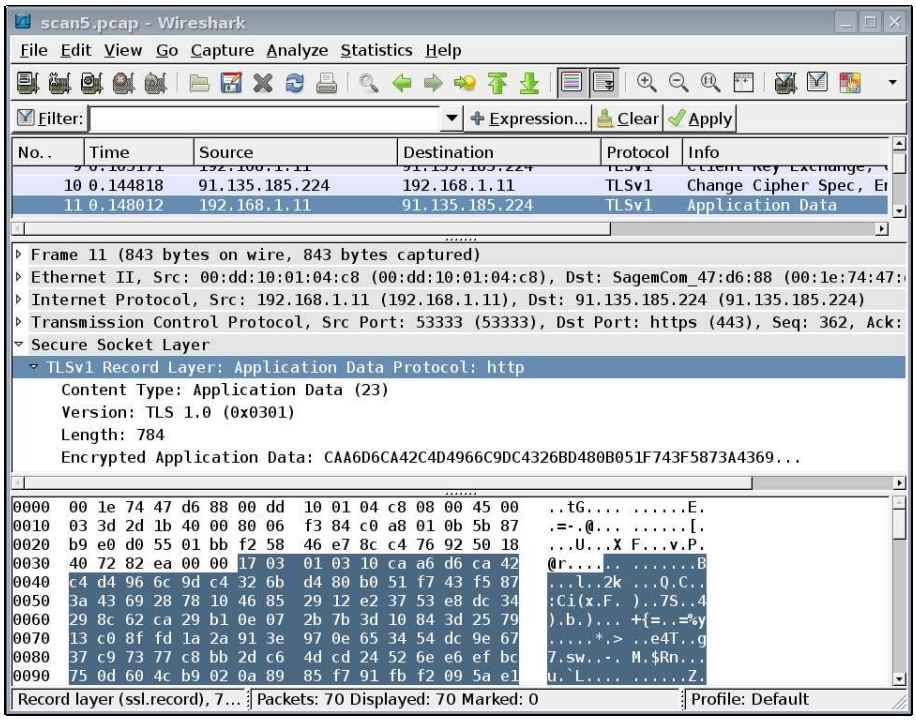

FIG. 4.28 – Données SSL

4.4 Analyse d'une trame de données SSL

La trame suivante (voir figure 4.28) contient deux messages SSL : `Change Cipher Spec` et `Finished` de la part du serveur. Les trames suivantes sont des trames SSL de données (type 23). Elles sont cryptées donc on ne peut pas en soutirer grand chose. Examinons la première d'entre elles :

- Le premier octet, celui de type, est égal `17h` soit 23, ce qui montre qu'il s'agit de données SSL.
- Les deux octets suivants (`03h` et `01h`) montrent qu'il s'agit de la version 1.0 de TLS.
- Les deux octets suivants, (`03 10h`), spécifient la longueur des données, ici 784 octets, ce qui nous conduit jusqu'à la fin de la trame.
- Les 784 octets de données sont sous forme cryptée.

  Il est très difficile de savoir de quoi il s'agit. C'est ce qui est recherché. L'analyseur de trames précise qu'il s'agit de HTTP : il se réfère évidemment au port utilisé (443 pour HTTPS) ; il n'a pas réussi plus que nous à savoir de quoi il s'agit.