Université de Technologie de Belfort Montbéliard

AC20

Étude du *Forward Secrecy*

Étudiant : Jérôme Boursier

Enseignant : M Frédéric Holweck

16 juin 2014

Ce document ainsi que l'intégralité des recherches ayant permis sa rédaction sont disponibles sur mon site : https://git.fr33tux.org/ac20.git [1]

1. Il est possible de cloner le dépôt via a même adresse :

```
$ git clone http://git.fr33tux.org/ac20.git
```

Table des matières

Table des figures

Liste des Algorithmes

Introduction

Suite aux révélations d'Edward Snowden [2], la sécurisation des échanges numériques est mise sur le devant de la scène. En effet, l'interception des communications par les agences de renseignement du monde entier est devenue monnaie courante [3] et la nécessité de s'en protéger s'avère essentielle [4].

La sécurisation des échanges passe actuellement en partie par le chiffrement des données, qui est considéré comme un moyen sûr de protéger le contenu de ses communications si tant est qu'il est correctement implémenté et mis en place :

> *Encryption works. Properly implemented strong crypto systems are one of the few things that you can rely on.* [5]

La cryptographie s'applique à de nombreux champs, mais je vais ici m'intéresser plus particulièrement aux connexions basées sur les protocoles SSL/TLS.
Ces protocoles permettent à deux machines de communiquer entre elles de manière chiffrée grâce à plusieurs mécanismes :

— permettant l'échange de clés ;
— assurant l'authentification ;
— réalisant le chiffrement ;
— vérifiant l'intégrité.

Je vais y étudier plus précisément l'un d'entre eux, Diffie-Hellman afin, ensuite, de comprendre l'intérêt de la propriété appelée *Forward Secrecy*.
Enfin, nous verrons un exemple de mise en place concrète sur deux serveurs web différents.

2. Ancien employé de Booz Allen Hamilton, un contractant de la National Security Agency
3. The Guardian, 21 Juin 2013 : *GCHQ taps fibre-optic cables for secret access to world's communications*
4. InternetActu, 21 Mai 2010, article de Jean-Marc Manach
5. Réponse donnée par Edward Snowden lors d'une séance de questions-réponses organisée par le journal The Guardian en Juin 2013

1 Le SSL/TLS, qu'est-ce que c'est ?

1.1 Un peu d'histoire...

1.1.1 Secure Socket Layer

Dans les années 1990, le numérique n'était pas encore grand public. Cependant, beaucoup d'entreprises ont perçu le potentiel qu'Internet leur offrait et ont dès lors voulu mettre en place des systèmes d'achat en ligne. La problématique de la sécurisation des échanges est donc devenue primordiale pour le développement de ce nouveau marché.

La société Netscape a ainsi développé la première version du protocole SSL [6]. Cependant, cette version ne fût jamais publiée.

En 1995, Netscape publie la seconde version du protocole SSL. Le SSLv2 contient également de nombreuses failles de sécurité incitant à éviter l'usage de cette version, parmi lesquelles :

- le client choisit le cipher [7] avec les informations transmises par le serveur. Or aucun mécanisme d'authentification vient sécuriser ce choix : une personne tierce peut alors facilement imposer un choix de cipher plus facilement déchiffrable, alors que le client en supporte des plus « sûrs » ;
- des attaques par troncature sont possibles : une connexion est achevée en mettant fin à la session TCP sous-jacente sans préciser d'indications sur la taille des données transférées (ce qui permet de détecter une telle troncature) ;
- la même clé est utilisée pour le chiffrement et l'intégrité. De plus, le SSLv2 ne supporte que le MD5 qui est aujourd'hui obsolète car sujet aux attaques par collisions [8] ;
- la taille du padding est transmise en clair entre le client et le serveur. Cela facilite l'analyse du traffic pour une personne tierce car elle connaît facilement la taille exacte des blocs de données transmis.

Du fait de ces problèmes, il fût décidé de revoir intégralement le protocole et d'en publier une troisième version, exempt de tous ces défauts.

Le développement [9] de la troisième version du protocole SSL permis de corriger les problèmes présents dans les versions précédentes. Ce protocole est aujourd'hui

6. Acronyme de *Secure Socket Layer*.
7. Algorithme de chiffrement et ou déchiffrement.
8. T.Xie et D.Feng *Fast Collision Attack on MD5* : http://eprint.iacr.org/2013/170.pdf
9. Mené par Paul Kocher (cryptographe), Phil Karlton et Alan Freier (Netscape).

toujours utilisé et il fait l'objet d'une RFC [10] (pour intérêt historique uniquement) publiée par l'IETF [11] en août 2011.

Cette nouvelle version apporte donc plusieurs améliorations :
— les messages de fin de chaque échange sont authentifiés et contiennent des informations sur les précédents messages. Si l'intégrité ou l'authentification de l'un de ces messages échoue, la connexion n'est pas réalisée ;
— afin de pallier aux attaques par troncature, des alertes de fin de connexion sont présents. Ces alertes sont authentifiées et permettent d'informer chaque partie si la conversation a pris fin de façon inattendue (et donc potentiellement provoquée par un tiers) ou voulue ;
— l'échange de clés Diffie-Hellman est désormais possible [12] ;
— l'usage combiné des fonctions de hashage MD5 et SHA-1 remplace l'usage de la fonction MD5 simple.

Cette version pose encore plusieurs problèmes, et notamment avec certains algorithmes utilisés pour créer les codes d'authentification de message [13] (voir partie 1.2) qui n'ont pas été soumis à des analyses poussées afin de détecter d'éventuels problèmes.

1.1.2 Transport Layer Security

En 1999, l'IETF (voir note 11) publie la RFC 2246 [14] dans laquelle les spécifications de la première version du protocole TLS [15] sont établies.

Cette nouvelle version se base sur le protocole SSL version 3 et se veut en assurer la continuité [16]. Ce protocole apporte plusieurs nouveaux aspects par rapport au SSL :
— l'usage du MAC est déprécié, et le HMAC [17] est utilisé à la place ;
— l'utilisation de certains ciphers et mécanismes est désormais requis et non facultatif (Diffie-Hellman, 3DES...).

10. RFC 6101 *The Secure Sockets Layer Protocol Version 3.0* : https://tools.ietf.org/html/rfc6101
11. Acronyme de *Internet Task Force*, organisme informel élaborant des RFC (*Request For Comments*), documents techniques dont certains deviennent des standards
12. Voir partie 2.1 à la page 22.
13. Appelés *MAC* pour *Message Authentification Code*. J'utiliserai cette abbréviation par la suite.
14. RFC 2246 *The TLS Protocol Version 1.0* : https://tools.ietf.org/html/rfc2246
15. Acronyme de *Transport Layer Security*.
16. Le TLSv1 est aussi appelé SSLv3.1.
17. *Keyed-Hash Message Authentification Code* est un MAC combiné avec une clé secrète. Voir RFC 2104.

En 2006, l'IETF publie la RFC 4346 [18] et pose les spécifications du TLS version 1.1. Les principales différences avec la version précédente sont :

— le vecteur d'initialisation utilisé dans les algorithmes de chiffrement par bloc (voir partie1.4) est désormais explicite et non plus implicite comme c'était le cas dans les versions précédentes ;

— le message d'erreur en cas de problème avec le padding des blocs (voir 1.4) n'est plus spécifique à cette erreur mais générique aux erreurs de déchiffrement du MAC [19].

En 2008, la dernière version en date est définie avec la RFC 5246 [20] pour le protocole TLS version 1.2. Elle apporte plusieurs nouveautés, parmi lesquelles :

— La fonction de hashage MD5 est dépréciée et remplacée par la fonction de hashage SHA-256 ;

— le serveur et le client peuvent désormais spécifier quels algorithmes de hashage et de signature sont acceptés ainsi que l'ordre de préférence des ciphersuites (voir partie 1.4) ;

— la compatibilité avec le SSLv2 est retirée [21].

Aujourd'hui, la majorité des logiciels récents supportent le TLSv1.2. Cependant, ce dernier n'est pas toujours activé du côté serveur (voir partie 4).

1.2 La connexion

Une connexion client/serveur avec le protocole TLS s'apparente à une conversation ou à une négociation au sein de laquelle plusiers échanges sont nécessaires pour parvenir à un accord.

En effet, l'échange doit pouvoir assurer la confidentialité, l'intégrité, ainsi que l'authentification des données, c'est pourquoi plusieurs étapes sont nécessaires afin d'aboutir à une session sécurisée.

18. RFC 4346 *The Transport Layer Security (TLS) Protocol Version 1.1* : https://tools.ietf.org/html/rfc4346

19. Ce qui permet de parer aux attaques basées sur le timing, telles que décrites par B.Moeller, *Security of CBC CipherSuites in SSL/TLS : Problems and CounterMeasures* : https://www.openssl.org/ bodo/tls-cbc.txt

20. RFC 5246 *The Transport Layer Security (TLS) Protocol Version 1.2* : https://tools.ietf.org/html/rfc5246

21. RFC 6176 *Prohibiting Secure Sockets Layer (SSL) Version 2.0* : https://tools.ietf.org/html/rfc6176

1.2.1 Aperçu

Pour cela, l'établissement d'une connexion TLS débute par « une poignée de main » [22] entre les deux parties qui permet d'échanger les différents paramètres afin d'assurer les trois points précédents :

Client Hello
- Le client envoie au serveur la liste des Ciphers [23] supportés.

Server Hello
- Le serveur envoie au client la Cipher Suite choisie ;
- Le serveur envoie au client son certificat [24].

Server Hello Done
- Le serveur informe le client qu'il a fini de transmettre les informations nécessaires.

Client Key Exchange [25]
- Le client envoie au serveur les paramètres nécessaires à la création de la clé secrète partagée. Ces informations sont chiffrées avec la clé publique du serveur.

Change Cipher Spec
- Le client informe le serveur que son prochain message sera chiffré.

Client Finished
- Le client informe le serveur qu'il a transmis toutes les informations nécessaires. Ce message est chiffré.

Change Cipher Spec
- Le serveur informe à son tour le client que son prochain message sera chiffré.

Server finished
- Le serveur informe le client qu'il a transmis toutes les informations nécessaires. Ce message est chiffré.

Une fois cette négociation achevée avec succès, les données peuvent être envoyées chiffrées.

Au contraire, si une erreur apparaît lors de l'un des échanges, la conversation est interrompue.

1.2.2 Détails

22. Expression généralement peu traduite et appelée « handshake ».
23. Voir partie 1.4
24. Voir partie 1.3
25. Voir parties 2.1 et 3.2

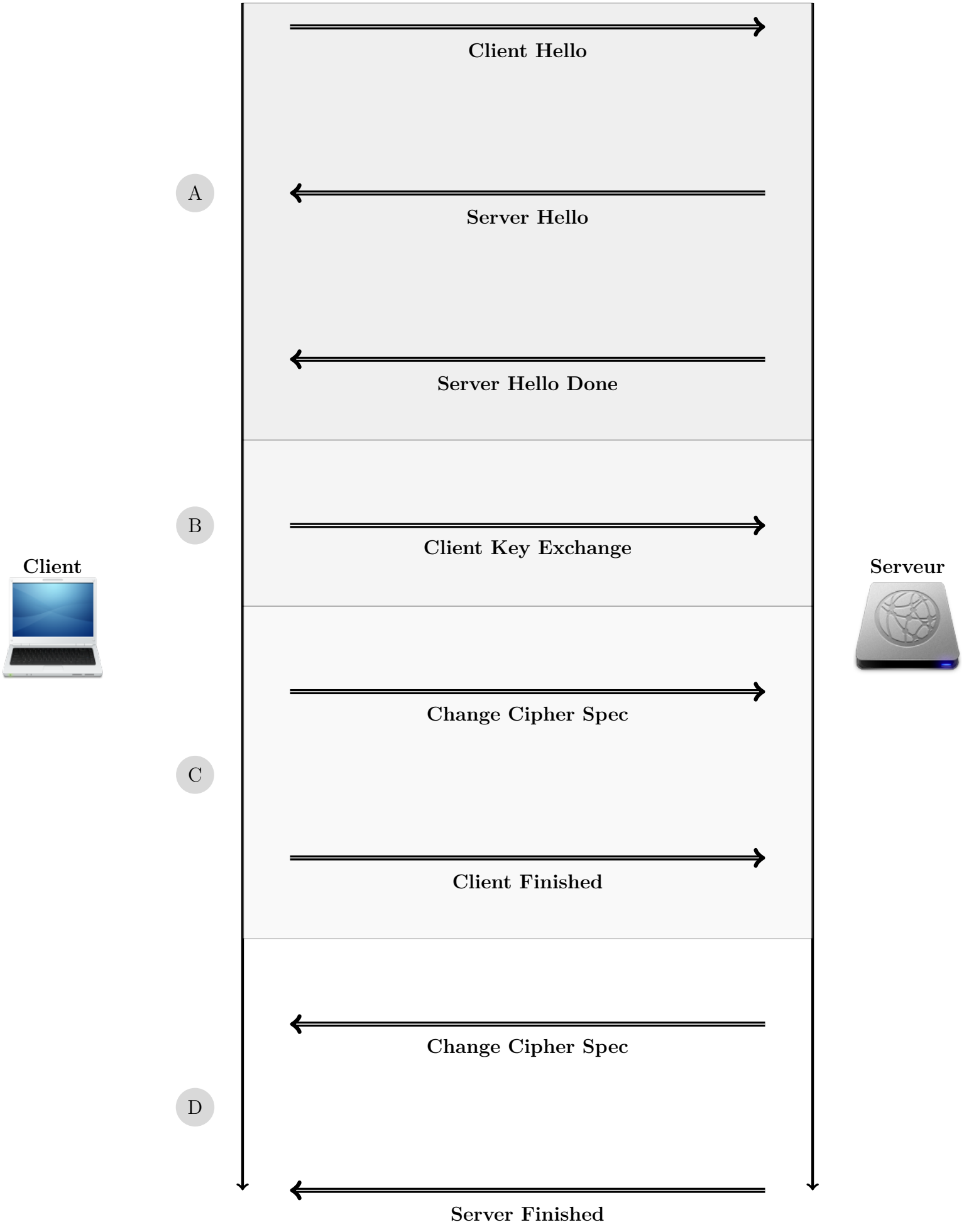


FIGURE 1 – Déroulement d'une connexion SSL/TLS « classique »

Le client envoie le message **CLIENT_HELLO** contenant la version de SSL (ou TLS) (3.x pour TLS, x pour SSL) qu'il supporte, la liste des ciphersuites autorisées [a], les algorithmes de compression supportés, ainsi que des informations de session et des valeurs aléatoires. Ces dernières sont composées de 32 octets, dont 4 constituent la date du client :

```
ClientVersion    3.3
Random[32]
SessionID
CipherSuite
CompressionMethod
```

Le serveur répond alors avec le message **SERVER_HELLO**, en envoyant la version SSL (ou TLS) choisie en fonction de celle du client [b] : la version la plus récente supportée par le client est choisie.
Le serveur envoie également la ciphersuite sélectionnée (la première supportée dans la liste envoyée par le client si acceptée par le serveur), l'algorithme de compression choisi ainsi que des valeurs aléatoires sur 32 octets, dont 4 composent la date du serveur :

A

```
ServerVersion    3.3
Random[32]
SessionID
CipherSuite
CompressionMethod
```

Ensuite, le serveur envoie son certificat [c] contenant sa clé publique. Une fois ces informations envoyées, le serveur informe le client avec **HELLO_SERVER_DONE**.
Le client vérifie l'identité du serveur en utilisant les informations du certificat : date de validité, sujet, autorité de certification.
Le client peut également vérifier si le certificat a été révoqué ou non. En effet, si la clé privée du serveur a été compromise, il est possible de signaler à l'autorité de certification de révoquer le certificat de la clé publique associée : ce faisant, le certificat sera indiqué comme invalide et la connexion ne pourra se faire [d].

a. Voir page 18.
b. Voir page 36 pour plus de détails pratiques sur ce point.
c. Voir page 15
d. Cette vérification de révocation utilise le protocole OCSP ou le mécanisme CRL. Cependant, tous les logiciels clients ne sont pas configurés pour effectuer cette vérification par défaut.

Le client génère alors la pre-master key grâce aux valeurs aléatoires échangées précedemment. Sa longueur dépend de l'algorithme utilisé : dans le cas de RSA, la pre-masterkey fait 48 octets.
Le client chiffre alors la pre-masterkey accompagnée de la version du protocole utilisé (pour éviter les "rollbacks" vers une ancienne version de TLS ou SSL) avec la clé publique du serveur et lui envoie au sein du message **CLIENT_KEY_EXCHANGE**.

B

```
ClientVersion    3.3
Random[46]
```

Une fois envoyé, le client informe le serveur que les prochains messages de sa part seront chiffrés avec le message **CHANGE_CIPHER_SPEC**, selon les informations précedemment échangées.
Enfin, le client envoie le message **CLIENT_FINISHED**. Ce message permet de s'assurer que l'échange de clé et que l'authentification se sont bien déroulés.
Pour cela, le client utilise la PRF [a] comme suit :

$$PRF(secret, label, seed) = P\_ < hash > (secret, label + seed)$$

Où :

C

- *secret* est la masterkey ;
- *label* est *client_finished* côté client, et *server_finished* côté serveur ;
- *seed* est le hash [b] de la concaténation des messages échangés lors des échanges précédents.

```
Verify_data[verify_data_lenght]
```

a. Pseudo-Random Function, fonction générant un résultat d'une longueur donnée à partir d'une entrée. Le résultat doit s'apparenter comme vraiment aléatoire : la distinction entre un résultat issu de la PRF ne doit pas être distinguable d'un résultat réellement aléatoire.

b. Dépend de la ciphersuite utilisée.

Le serveur déchiffre alors la pre-masterkey chiffrée par le client dans le message **CLIENT_KEY_EXCHANGE**. Il génère la masterkey de la même façon que le client :

$$\begin{aligned} master\_secret &= PRF(pre\_master\_secret, "mastersecret" \\ &, \quad ClientHello.random + ServerHello.random) \end{aligned}$$

Où

- *pre_master_secret* est la *pre_master_key* transmise par le client ;
- *ClientHello.random* est la valeur aléatoire sur 32 octets transmise lors du message **CLIENT_HELLO** ;
- *ServerHello.random* est la valeur aléatoire sur 32 octets transmise lors du message **SERVER_HELLO**.

D

Il envoie alors au client le message **CHANGE_CIPHER_SPEC** informant que tous les messages suivants de sa part seront chiffrés. Enfin, le serveur effectue les même calculs que le client sur tous les messages échangés précédemment.
Il compare le résultat avec celle envoyée par le client, et envoit son résultat au client dans le message **SERVER_FINISHED**.
Si chacune des valeurs reçues et calculées localement correspondent, alors la connexion peut continuer. Sinon, il est probable que l'échange ait été modifié par une personne tierce.

1.3 Certificats

Afin d'authentifier le serveur, le protocole TLS utilise le mécanisme de certificats. La norme la plus présente aujourd'hui est *X.509*[26] et plus précisément la RFC 5280[27]. Elle permet de s'assurer que la clé publique envoyée par le serveur appartient bien au serveur et non à une personne tierce se faisant passer pour le serveur.

Pour cela, un certificat est délivré par une autorité de certification[28] une fois qu'une démarche ait été respectée pour s'assurer de l'identité du serveur. Ces autorités sont soumises à des audits réguliers qui permettent de s'assurer ou non de leur viabilité.

Les principaux navigateurs web et systèmes d'exploitation incluent directement les clés publiques de ces autorités, ce qui permet d'identifier le certificat d'un serveur généré par une autorité de certification valide.

Un certificat est composé de plusieurs champs, et notamment, pour le certificat pris en exemple :

- **Version** : Version 3
- **Numéro de série** : 02 :2A :50
- **Algorithme de signature du certificat** : PKCS #1 SHA-512 With RSA Encryption
- **Délivreur** :

  CN = CAcert Class 3 Root

  OU = http ://www.CAcert.org

  O = CAcert Inc.
- **Validité** :

  Pas avant : 08/04/2014 23 :50 :02 (08/04/2014 21 :50 :02 GMT)

  Pas après : 07/04/2016 23 :50 :02 (07/04/2016 21 :50 :02 GMT)
- **Sujet** : CN = *.fr33tux.org
- **Informations sur la clé publique** :

  Algorithme de la clé publique : PKCS #1 RSA Encryption

  Clé publique
- **Extensions**
- **Algorithme de signature du certificat** : PKCS #1 SHA-512 With RSA Encryption
- **Signature du certificat**

26. Créée en 1988 par l'Union Internationale des Télécommunications.
27. RFC 5280 : *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* : https://tools.ietf.org/html/rfc5280
28. Verisign, Gandi, Comodo... en sont les exemples les plus connus.

FIGURE 2 – Exemple d'un certificat à la norme X.509 tel qu'affiché dans le navigateur Mozilla Firefox

Il est également possible d'afficher le certificat en format brut via la commande suivante :

```
openssl s_client -connect git.fr33tux.org:443
```

On obtient alors les différents champs comme ci-dessus :

```
depth=2 O = Root CA, OU = http://www.cacert.org,
CN = CA Cert Signing Authority, emailAddress = support@cacert.org
verify error:num=19:self signed certificate in certificate chain
verify return:0
---
Certificate chain
 0 s:/CN=fr33tux.org
   i:/O=CAcert Inc./OU=http://www.CAcert.org/CN=CAcert Class 3 Root
 1 s:/O=CAcert Inc./OU=http://www.CAcert.org/CN=CAcert Class 3 Root
   i:/O=Root CA/OU=http://www.cacert.org/CN=CA Cert Signing Authority/
   emailAddress=support@cacert.org
 2 s:/O=Root CA/OU=http://www.cacert.org/CN=CA Cert Signing Authority/
 emailAddress=support@cacert.org
   i:/O=Root CA/OU=http://www.cacert.org/CN=CA Cert Signing Authority/
   emailAddress=support@cacert.org
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIF4TCCA8mgAwIBAgIDAipSMA0GCSqGSIb3DQEBDQUAMFQxFDASBgNVBAoTC0NB
Y2VydCBJbmMuMR4wHAYDVQQLExVodHRwOi8vd3d3LkNBY2VydC5vcmcxHDAaBgNV
c3MzLXJldm9rZS5jcmwwMQYDVR0RBCowKIILZnIzM3R1eC5vcmegGQYIKwYBBQUH
CAWgDQwLZnIzM3R1eC5vcmcwDQYJKoZIhvcNAQENBQADggIBAHRAEdj8hKgARflE
XA6K4i9ieCIx1qixsusAGPdNRuvkCUyOvI4cyKtjETtkdOGGAQMhMeJzibFDIGV6
BrSPo+RUWs5zxiMG0LAda4IIc7Fx0DvOwHZ2ZCbA44Hk1T8tGLSCG1KxEGnDkq/n
lFkA50ZJ7odTqn9g6b0wbN6T3kfUN2W9XYGepbYf8J8KTHggqirVLO51LxcFBccv
JkTd2z4C42NxG0r9Y+MUZvpY+f8zHoYzh/Kc8gorZJekcJUgWkw/sgkrxFRDHKhK
oblFCke9CtxEbCszk3uY9/cz1RzsZX1aJ6RXcGJTM9jM5D2XK2k5urm2DSSf2ty4
qTX8EqVYe8Bh7qk/SWjkWi8y/uX9GCyulny1RuWfggxCEGM6+72MeXG9xeIF4qFj
OJcBOZjE5AaE8nt+iFAjw5jyXMH3iSL9ThEMpNa+XzowIna+mfrxJJWh4WVvcFpm
sNvHzMsS+CJEBc4m+rNz9bDbze/stfCaSn4lLbsQXljSNa63q4DhtMK634iRuBXv
UksJwFfmo2On8zUcGfHo19YhKjEYJv5wUWUeEnW4YO28jyeK4xGQrakk0P3cX7Q5
o574un9YncMaZ5Y6BfaDZkV9LoLAM7AK0QQUEMahpTbsnmFkc7tdqOcUnfRFW25J
+tWZ3qG13SF9JXPIAVbejgSGPW+t
-----END CERTIFICATE-----
subject=/CN=fr33tux.org
issuer=/O=CAcert Inc./OU=http://www.CAcert.org/CN=CAcert Class 3 Root
```

FIGURE 3 – Exemple du même certificat (tronqué) en format brut

Ce certificat est donc signé par l'autorité de certification (représenté aux lignes précédées par **i :**) **CA Cert** pour le domaine fr33tux.org.
Il est valide durant deux ans, à partir du 27/01/2014 et jusqu'au 27/01/2016.
Le certificat ne peut être utilisé que pour un seveur SSL (il est donc invalide pour signer des binaires par exemple).

En dehors de ces dates ou si le domaine qui utilise ce certificat change, la négociation [29] échoue, l'authentification du serveur ne pouvant pas se faire de manière sûre.

1.4 Ciphers suites

Lors des messages **CLIENT_HELLO** et **SERVER_HELLO** [30], les deux parties échangent une liste de ciphers.

Une CipherSuite est une combinaison d'algorithmes permettant l'authentification, le chiffrement, l'échange de clés et fournissant une PRF. Il en existe de multiples, fournissant chacune un certain niveau de sécurité et éventuellement le forward secrecy et pour certaines d'entre elles, le perfect forward secrecy [31].

Prenons par exemple la ciphersuite utilisée sur le site accessible sur https://fr33tux.org :

TLS_DHE_RSA_WITH_AES_256_CBC_SHA

Cette ciphersuite est donc composée de quatre algorithmes :

Échange de clé DHE : *Diffie-Hellman-Ephemeral* ;
Authentification RSA : *Rivest Shamir Adleman* ;
Chiffrement AES : *Advanced Encryption Standard* ;
Intégrité et authentification SHA : *Secure Hash Algorithm*.

L'échange de clé Diffie-Hellman est vu dans la partie 2.1 page 22.

L'authentification s'effectue via la clé asymétrique envoyée par le serveur dans le certificat, comme expliqué dans la partie 1.3 page 15

Le chiffrement utilise ici AES [32]. C'est un algorithme de chiffrement par bloc de 128 bits à clé symétrique : la même clé est utilisée pour le chiffrement et pour le déchiffrement. Il est donc important que la clé soit partagée de façon sûre entre les

29. Voir partie 1.2.
30. Voir partie 1.2 page 9.
31. Voir partie 3 page 27.
32. Aussi appelé *Rijndael*, cet algorithme est publié en 1997 par le NIST. Il est aujourd'hui recommandé par la NSA pour protéger les informations classées *Top Secret*.

deux parties de la communication.

L'intérêt réside dans le fait que le chiffrement symétrique est plus performant que le chiffrement asymétrique : des clés plus courtes sont nécessaires pour offrir une sécurité similaire.

Cet algorithme utilisé une clé de 128, 192 ou 256 bits. Cette clé définit le nombre de répétitions de l'algorithme :

— 10 cycles pour une clé de 128 bits ;
— 12 cycles pour une clé de 192 bits ;
— 14 cycles pour une clé de 256 bits.

Il utilise des suites de permutations et substitutions. Chaque cycle ayant besoin d'une clé de 128 bits, il est nécessaire de générer le nombre de clés adéquat à partir de la clé originale au sein de l'étape *AddRoundKey*[33]

Une fois le nombre de clés requis généré, l'algorithme effectue une série d'étapes après avoir divisé le message d'entrée en blocs de 128 bits[34] :

1. *SubBytes* : chaque élément de la matrice 4×4 est substitué selon la table de substitution de Rijndael : les lignes représentent le bit de poids fort du nibble[35] et les colonnes celui de poids fort (voir figure 1.4 page 20 et l'Annexe A.1 page 44) ;
2. *ShiftRows* : un décalage de chaque octet est effectué (voir Annexe A.2 page 47) ;
3. *MixColumns* : chaque colonne de la matrice d'octets est alors multipliée par un polynôme fixe $a(x) = [03]x3 + [01]x2 + [01]x + [02]$, le tout modulo $x^4 + 1$ (Voir Annexe A.3 page 48) ;
4. *AddRoundKey* : la clé correspondant au cycle actuel (générée précédemment) est alors ajoutée à la matrice via un *xor* (Voir Annexe A.4 page 49).

Le cycle ci-dessus est répété dix, douze ou quatorze fois selon la taille de la clé. Ensuite, un dernier cycle légèrement différent est effectué :

— *SubBytes* ;
— *ShiftRows* ;
— *AddRoundKey*.

Nous obtenons alors le message chiffré, découpé en blocs de 128 bits.

33. Pour générer ce nombre de clés, AES utilise le Rijndael key schedule.
34. Ce qui correspond à 4 octets.
35. Un demi-octet, soit quatre bits.

Grâce à la nature reversible des opérations effectuées (voir Annexe A page 44, le déchiffrement d'un message fait appel aux étapes inverses :

1. *Inverse ShiftRows* ;
2. *Inverse SubBytes* ;
3. *AddRoundKey* ;
4. *Inverse MixColumns*.

Le dernier cycle est alors légèrement différent, de la même façon que pour le dernier cycle du chiffrement :

1. *Inverse ShiftRows* ;
2. *Inverse SubBytes* ;
3. *AddRoundKey*.

	0	1	2	3	4	5	6	**7**	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	**94**	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

FIGURE 4 – Table de substitution de Rijndael - Par exemple, la valeur en hexadécimal *0xe7* est substituée par la valeur *0x94*.

Pour l'intégrité et l'authenticité des messages, le *Message Authentification Code* se base sur la fonction de hash *SHA* [36]. Une fonction de hash génère un nombre fixe

36. Acronyme de *Secure Hash Algorithm*.

de caractères à partir d'un message en entrée de longueur variable.
Il ne nécessite pas de clés et sa particularité est également d'être à sens unique : il n'est pas possible de retrouver le message original à partir du hash généré. Une fonction de hashage doit donc respecter les propriétés suivantes :

- connaissant y, il est impossible de trouver x tel que $y = h(x)$
- résistance aux collisions "faibles" : Connaissance x, il est impossible de trouver $x' \neq x$ tel que $h(x) = h(x')$
- résistance forte aux collisions "fortes" : il est impossible de trouver deux messages distincts x et x' tels que $h(x) = h(x')$

Le MAC est envoyé avec le message sous forme d'un *tag* que chaque partie vérifie ensuite de cette façon :

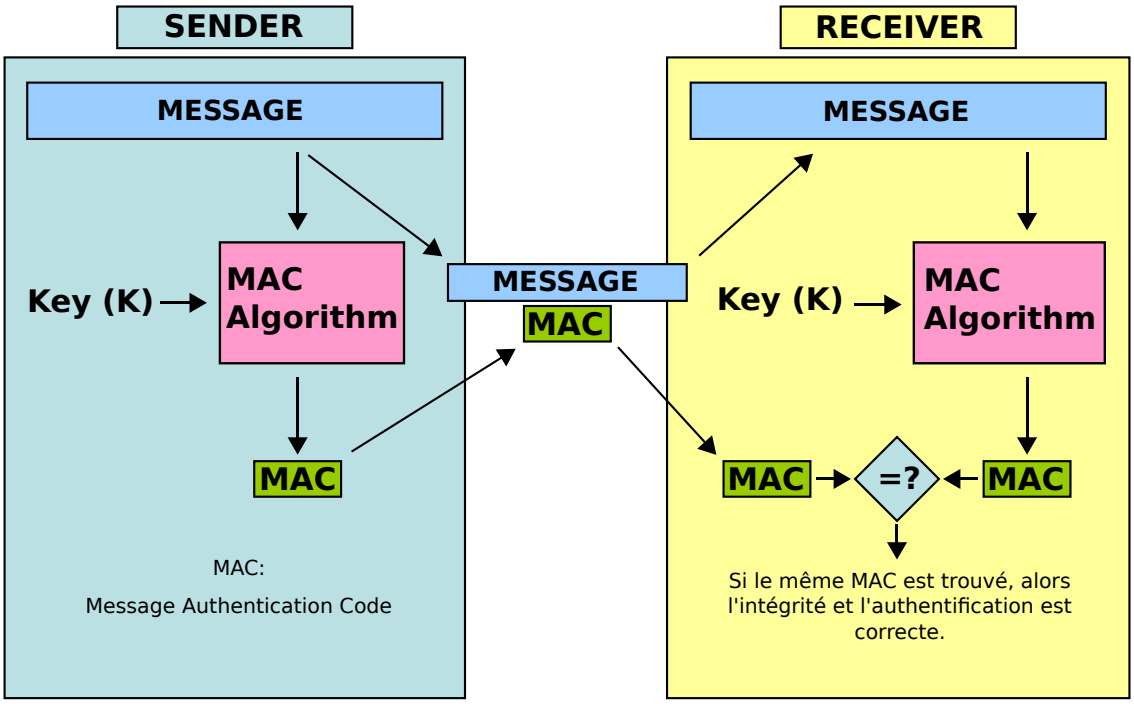


FIGURE 5 – Message Authentification Code

Le serveur génère alors un hash du message et de la clé partagée entre les deux parties.

Grâce au respect de chaque étape de ce protocole, le client et le serveur communiquent de façon chiffrée, authentifiée, et vérifiant l'intégrité de leurs messages.

2 Algorithmie : Diffie-Hellman

2.1 L'échange avec Diffie-Hellman

Comme vu précédemment, l'établissement d'une connexion TLS nécessite un échange de clés préalable. Cet échange doit pouvoir se faire sans qu'un tiers ne puisse en obtenir le résultat ni au moment de la connexion, ni plusieurs années plus tard. Pourtant, l'établissement d'une connexion « simple » [37] ne permet pas cette sécurité : la compromission de la clé privée compromet l'intégralité des connexions précédentes puisque la clé symétrique générée lors de chaque connexion est envoyée chiffrée par la paire de clé asymétrique du serveur.

FIGURE 6 – Hellman, Diffie, Merkle

En 1976, Whitfield Diffie, Martin Hellman et Ralph Merkle rédigent *New Directions in Cryptography*[38]. Cet article introduit la méthode d'échange de clés appelée *Diffie-Hellman*.

Cette méthode se base sur le problème du logarithme discret : si l'on possède g^a, il n'existe pas de méthodes efficaces pour obtenir a.

Lors de la mise en place du serveur, ce dernier a généré p, un nombre premier, et g une racine primitive modulo p[39] :

37. Comme décrite dans la partie 1.2 page 9.
38. Disponible sur le site de l'Université de Stanford.
39. Voir partie 4 page 36 pour plus d'informations sur la mise en place pratique.

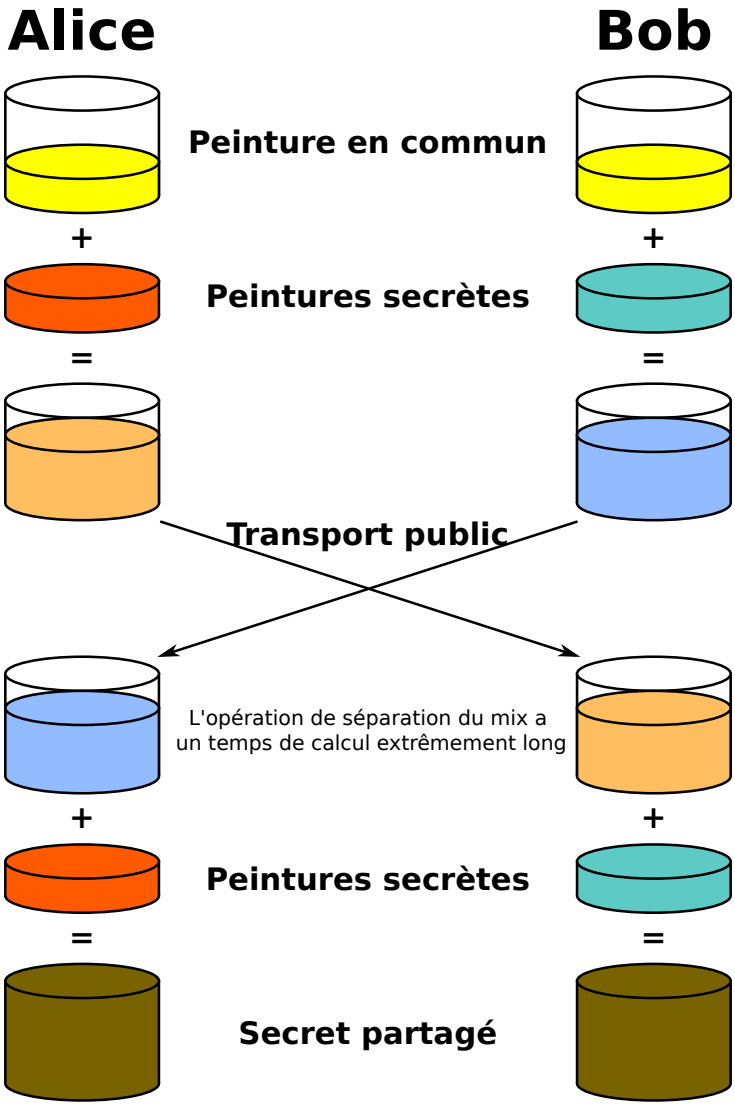


FIGURE 7 – Principe d'un échange Diffie-Hellman

Racine primitive modulo n Soit n un entier strictement positif. Un groupe pour la multiplication est alors formé par l'ensemble des nombres premiers avec n.
Une racine primitive modulo n est un entier qui génère tous les autres entiers du groupe modulo n :

Par exemple, pour $n = 14$, les éléments de $\mathbb{Z}_{14}^{x}$ sont $1, 3, 5, 9, 11$ et 13.
On obtient ainsi :

1	1
3	3, 9, 13, 11, 5, 1
5	5, 11, 13, 9, 3, 1
9	9, 11, 1
11	11, 9, 1
13	13, 1

FIGURE 8 – Exemple des puissances des éléments du groupe $\mathbb{Z}_{14}^x$ modulo 14

On observe donc bien que seulement 3 et 5 sont des racines primitives modulo 14 : leurs puissances génèrent l'ensemble des éléments du groupe.

Afin de calculer la puissance d'un nombre, une méthode possible est celle de l'**exponentiation rapide** : cette méthode décompose x en puissances de 2 et réduit ainsi le nombre d'opérations nécessaires :
Soit :

$$n = \sum_i e_i 2^i$$

Avec $e_i = 0$ ou 1, ce qui donne donc :

$$g^n = \prod_{e_i=1} g^{2^i}$$

Et par exemple :

$$\begin{aligned} g^{78} =& g^{64} \times g^8 \times g^4 \times g^2 \\ =& (((g^2 \times g^2 \times g^2 \times g^2 \times g)^2 \times g)^2 \times g)^2 \end{aligned}$$

On peut donc aisément remarquer que la puissance souhaitée peut s'obtenir à l'aide de carrés et de multiplications.

Données : x, g
$r \leftarrow 1$
Tant que $x > 0$ **faire**
  Si $x\%2 = 1$ **alors**
    $r \leftarrow r \times g$
  FinSi
  $g \leftarrow g \times g$
  $x \leftarrow \frac{x}{2}$
FinTantque
retourner r

Algorithme 1 : Exponentiation rapide

g^n avec $n = 2^k$. On effectue au plus k divisions par deux, donc la complexité est de $k = 2 \log n$

Les paramètres p et g sont alors utilisés pour chaque échange Diffie-Hellman :

1. le serveur envoie au client le nombre premier p et la racine primitive g, signés avec la clé publique du certificat ;
2. le client génère un nombre entier aléatoire a ;
3. le serveur fait de même, et génère b ;
4. le client envoie au serveur $g^a \bmod p$;
5. le serveur envoie au client $g^b \bmod p$;
6. le client calcule alors $(g^b)^a \bmod p$ et le serveur $(g^a)^b \bmod p$;
7. le client et le serveur partagent dès lors la même clé.

Cette méthode nécessite de grands nombres pour fonctionner de manière à rendre une recherche exhaustive impossible en un temps raisonnable : a, b et p doivent être composés de plus d'une centaines de chiffres chacun.

Le problème du logarithme discret Un tiers écoutant l'échange entre le client et le serveur est en possession des informations suivantes :

— g : la racine, élément d'un groupe ;
— p : le nombre premier ;
— $g^a \bmod p$: envoyé par le serveur ;
— $g^b \bmod p$: envoyé par le client.

L'intérêt pour lui est donc ici de trouver x (élément de $\mathbb{Z}$) tel que $g^x = g^a (\bmod p)$: si x existe, il est appelé le logarithme discret de g^a dans la base g.

Pour résoudre ce problème, une première solution est de tester chaque x :

Données : g^a, g
$x \leftarrow 0$
Tant que $g^x \neq g^a$ **faire**
| x←x+1
FinTantque
retourner *Vrai*

Algorithme 2 : Naïf

La complexité de cet algorithme est donc de $n = 2^k$, c'est à dire exponentielle en k, la longueur en bits de n.

	Exponentiation rapide	**Naïf**
Principe	Calcul g^n	trouve a tel que $g^n = g^a$
Complexité en n	$\log n$	n
Complexité en k	O(k) - polynômiale	$O(2^k)$ - exponentielle

Il existe d'autres algorithmes mais qui ne sont guère plus efficaces que l'algorithme dit « naïf ». Ainsi, la génération de la clé d'échange est de complexité polynômiale alors que pour retrouver la clé, la complexité est de nature exponentielle.

La technique du *Forward Secrecy* introduit la nature éphémère des primitives du serveur (g et p) : ces dernières ne sont pas générées uniquement lors de la mise en place du serveur et réutilisées à chaque fois, mais le sont pour chaque client se connectant et leur usage est unique.

Le calcul de telles primitives étant consommateur de ressources serveur, répéter l'opération pour chaque client l'est encore plus et peut rapidement s'avérer pénalisant lorsque la fréquentation du serveur est élevée. Il est possible d'utiliser les courbes elliptiques afin de réduire le coût de calcul qui peut poser problème à grande ampleur : des chiffres nettement plus petits sont alors nécessaires pour obtenir un résultat équivalent à l'usage de Diffie-Hellman « classique » [40].

40. Voir partie 3.2 page 27

3 Le Forward Secrecy

3.1 Principe

Un échange Diffie-Hellman « classique » utilise des paramètres p, g et a générés une unique fois et utilisés par la suite pour l'ensemble des connexions.

Un échange Diffie-Hellman éphémère utilise ainsi les mêmes paramètres de façon éphémère :

Au lieu de générer les paramètres Diffie-Hellman p, g et a, ces derniers le sont pour chaque connexion et supprimés ensuite. Ce faisant, chaque connexion est liée avec les paramètres éphémères qui ne peuvent plus être compromis puisqu'effacés après utilisation.
Par conséquent, les connexions précédemment enregistrées ne peuvent plus être déchiffrées, même si le serveur est saisi ou victime d'une intrusion et que la clé privée a été compromise.

Comme l'explique l'EFF : *the beauty of forward secrecy is that the privacy of today's sessions doesn't depend on keeping information secret tomorrow.* [41]

Cependant, pour générer ces paramètres « à la volée », il est nécessaire d'avoir une machine possédant les ressources suffisantes, car le calcul n'est pas anodin, en particulier lorsqu'il implique de grands nombres avec plusieurs centaines de chiffres chacun.

Par conséquent, il devient dans certains cas nécessaire de trouver un moyen moins coûteux offrant le même niveau de sécurité, et c'est ici que les courbes elliptiques interviennent.

3.2 Fonctionnement avec les courbes elliptiques

3.2.1 Définitions

Une courbe elliptique est une courbe de degré trois, et possédant plusieurs caractéristiques lui conférant la nature de groupe commutatif [42].

La courbe elliptique « vit » dans le plan projectif $\mathbb{P}^2$, au sein duquel les points de la courbe sont représentés par un tuple :

$$\mathbb{P}^2 = \mathbb{R}^3 / \sim$$

Tels que, avec $\forall \lambda \neq 0$:

$$(X, Y, Z) \sim \lambda(X, Y, Z)$$

et :

$$(X : Y : Z) \neq (0 : 0 : 0)$$

41. *Why the web needs Perfect Forward Secrecy More Than Ever*, 8 Avril 2014.
42. Voir annexe C page 57.

Donc les points de $\mathbb{R}^2$ sont des triplets notés (X :Y :Z) tels que $(X : Y : Z) \neq (0 : 0 : 0)$ et $(X : Y : Z) \equiv (\lambda X : \lambda Y : \lambda Z)$.

Une courbe elliptique est une courbe lisse qui possède la structure d'un groupe, c'est à dire qu'elle comporte :

— une loi de composition interne, commutative et associative

— un élément neutre

Une courbe projective de degré trois possède un point d'inflexion, c'est à dire un point tel que la tangente en ce point n'a pas d'autre intersection avec la courbe. On notera ce point spécifique O_E.

Il est ainsi possible d'additionner deux points de la courbe C, dans $\mathbb{P}^2$. Pour cela, il suffit de définir P,Q tels que :

$$\Delta_{(PQ)} \cap C = (PQ)$$

$$\Delta_{(PQ)O_E} \cap C = P + Q$$

Dans $\mathbb{R}^2$, cette construction équivaut à :

1. On trace la droite passant par les points P et Q, tous deux sur la courbe ;
2. Celle-ci coupe la courbe en un troisième point, PQ ;
3. On trace alors la droite passant par le point neutre O_E et PQ. Elle intersecte la courbe en un point, $P + Q$. Si le point neutre est placé à l'infini, on trace la droite parallèle à la direction du point à l'infini et passant par PQ.

La figure 10 montre la construction dans $\mathbb{R}^2$.

Si la droite passant par P et Q rejoint O_E. Alors la somme de P et Q est considérée comme nulle : $P + Q = O_E$ [43].

Cette construction par sécante et tangente d'une courbe de degré 3 est bien définie car d'après le théorème de Bezout, une courbe de degré 3 dans le plan projectif intersecte une droite en 3 points (avec multiplicité).

La loi de composition du groupe est donc définie sur la courbe elliptique. On peut la noter +.
Cette loi est commutative : $P + Q = Q + P$ (autrement dit, la droite passant par P et Q passe également par Q et P).

43. O_E est considéré comme le point à l'infini.

La loi est également associative :

$$(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$$

On peut montrer l'associativité de cette loi grâce au théorème de Max Noether [44]. Ce théorème concernant les courbes algébriques a notamment une conséquence sur l'intersection d'une courbe elliptique et de cubiques : si une courbe elliptique intersecte deux cubiques en 9 points dont 8 sont communs aux deux cubiques, alors le $9^{ème}$ est aussi commun.

De cette manière, on peut retrouver le résultat d'associativité : $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$.

La figure 9 illustre cette propriété.

Enfin, il reste à définir l'élément neutre pour compléter le groupe commutatif. Comme vu dans les cas particuliers ci-dessus, il existe un *point d'inflexion* O_E tel que :

$$P + O_E = P$$

Ce point est donc l'élément neutre de notre groupe.

De la même façon qu'il est possible de définir les points formant un groupe sur l'ensemble $\mathbb{Z}_{14}$, nous pouvons effectuer la même manipulation sur les courbes elliptiques.

Prenons pour cela la courbe $y^2 = x^3 + 3x + 4 \bmod 6$:

1^{er} **cas** : $x = 0 \longrightarrow y^2 \equiv 4 \bmod 6 \longrightarrow y \in \{2, 4\}$
$2^{ème}$ **cas** : $x = 1 \longrightarrow y^2 \equiv 2 \bmod 6$. y n'est donc pas entier.
$3^{ème}$ **cas** : $x = 2 \longrightarrow y^2 \equiv 0 \bmod 6 \longrightarrow y = 0$.
$4^{ème}$ **cas** : $x = 3 \longrightarrow y^2 \equiv 4 \bmod 6 \longrightarrow y \in \{2, 4\}$.
$5^{ème}$ **cas** : $x = 4 \longrightarrow y^2 \equiv 2 \bmod 6$. y n'est donc pas entier.
$6^{ème}$ **cas** : $x = 5 \longrightarrow y^2 \equiv 0 \bmod 6 \longrightarrow y = 0$.

Ainsi, l'ensemble des points (auquel il faut rajouter les points à l'infini) composant le groupe est :

$$\{(0, 0); (0, 2); (0, 4); (2, 0); (3, 2); (3, 4); (5, 0)\}$$

44. Mathématicien allemand de la fin du $XIX^{ème}$ siècle et spécialiste de la géométrie algébrique.

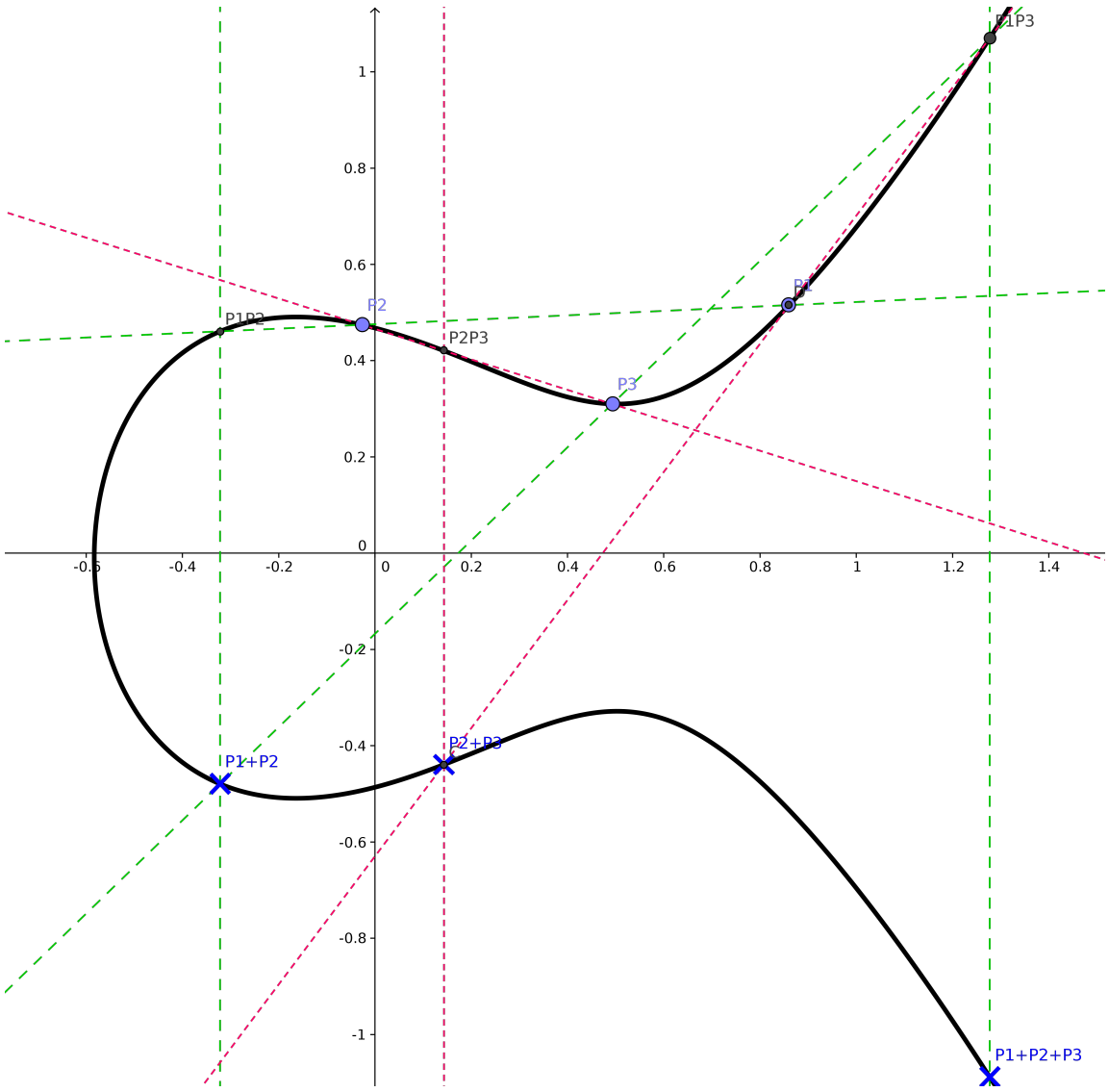


FIGURE 9 – Observation de l'associativité de la loi de façon géométrique.

3.2.2 Exemple d'addition

Soit la courbe elliptique C définie par $F(X,Y,Z)$:

$$ZY^2 + YZ^2 - X^3 + X^2Z = 0$$

Soit l'élément neutre $O_E = (0 : 1 : 0)$. Dans la carte $Y = 1$:

$$0 \times 1^2 + 1 \times 0^2 - 0^3 + 0^2 \times 0 = 0$$

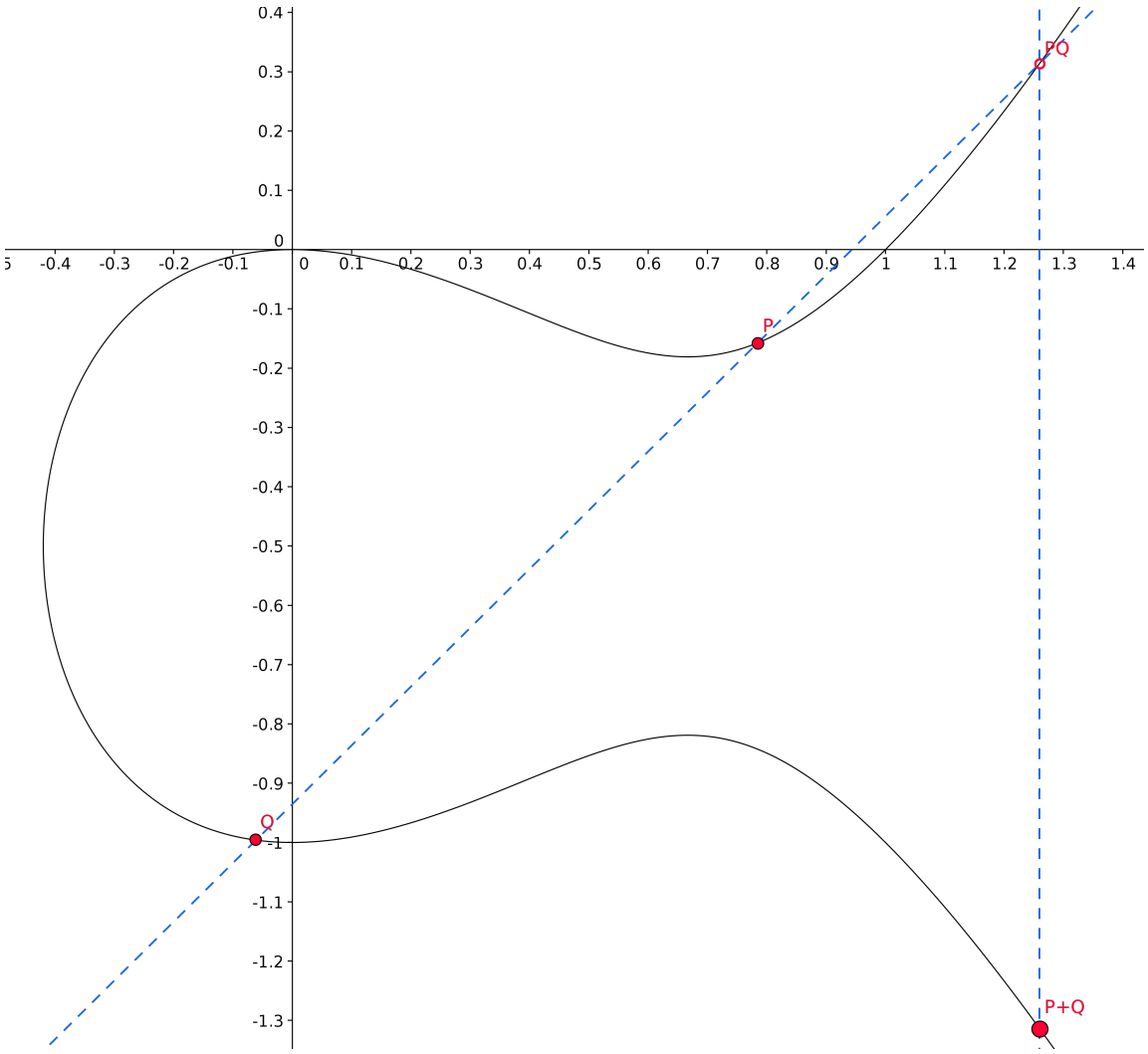


FIGURE 10 – Exemple d'addition de deux points P et Q de la courbe d'équation $-x^3 + x^2 + y^2 + y = 0$

Donc $O_E = (0 : 1 : 0) \in C$
Soit le point P :

$$P = (1 : 0 : 1)$$

Calculons *2P*.

Nous avons tout d'abord besoin de la tangente à la courbe C passant par le point P. La dérivée de F est donc :

$$\partial F = \begin{pmatrix} -3X^2 + 2X \\ \\ 2YZ + Z^2 \\ \\ 2ZY + Y^2 + X^2 \end{pmatrix} = \begin{pmatrix} \frac{\partial F}{\partial X} \\ \\ \frac{\partial F}{\partial Y} \\ \\ \frac{\partial F}{\partial Z} \end{pmatrix}$$

En l'évaluant au point P :

$$\partial F(P) = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}$$

Ainsi :

$$\partial F(P) \times \begin{pmatrix} X - 1 \\ Y + 0 \\ Z - 1 \end{pmatrix} = 0$$

On obtient dès lors le système suivant :

$$\left.\begin{aligned} -(X-1) + Y + Z - 1 = 0 \\ ZY^2 + YZ^2 - X^3 + X^2 Z = 0 \end{aligned}\right\} \tag{1}$$

Posons $Z \neq 0$ (par exemple $Z = 1$). Alors le système devient :

$$\left.\begin{aligned} -(x-1) + y = 0 \\ y^2 + y - x^3 + x^2 = 0 \end{aligned}\right\} \tag{2}$$

$$\left.\begin{aligned} x - 1 = y \\ (x-1)^2 + (x-1) - x^3 + x^2 = 0 \end{aligned}\right\} \tag{3}$$

$$\left.\begin{aligned} x-1 &= y \\ x(x-1)^2 &= 0 \end{aligned}\right\} \tag{4}$$

Ainsi :

$$\begin{cases} x = 0 \Rightarrow y = -1 \longrightarrow (0:-1:1) \\ x = 1 \Rightarrow y = 0 \longrightarrow (1:0:1) = P \end{cases}$$

Enfin, si $Z = 0$, on n'obtient pas de solutions :

$$\begin{cases} y = 1 + x - 1 \\ x^3 = 0 \end{cases}$$

Ainsi, on a donc :

$$\Delta P = (0:-1:1)$$

On « recommence » alors pour trouver léquation de la droite passant par ΔP et O_E et enfin trouver le point d'intersection entre cette droite et la courbe.
On observe cependant que la droite $(\Delta P O_E)$ est la droite d'équation $X = 0$

$$\left.\begin{aligned} X &= 0 \\ ZY^2 + YZ^2 - X^3 + X^2Z &= 0 \end{aligned}\right\} \tag{5}$$

$$\left.\begin{aligned} X &= 0 \\ ZY^2 + YZ^2 &= 0 \end{aligned}\right\} \tag{6}$$

On se place dans la carte $Z = 1$:

$$\left.\begin{aligned} x &= 0 \\ y + y^2 &= 0 \end{aligned}\right\} \tag{7}$$

Ainsi, on obtient :

$$y = \begin{cases} -1 \\ 0 \end{cases}$$

D'où les points d'intersection ont pour coordonnées $(0 : -1 : 1)$ et $(0 : 0 : 1)$. Le premier point est l'intersection ΔP. **Par conséquent, le résultat de l'addition est le point** $2P = (0 : 0 : 1)$**.**

3.2.3 Comparaison avec l'échange Diffie-Hellman « classique »

Le déroulement d'un échange Diffie-Hellman utilisant les courbes elliptiques se déroule donc de cette façon :

1. Le serveur et le client choisissent publiquement une courbe qu'ils vont utiliser. Ils choisissent également un point P de cette courbe.
2. Le client génère un nombre entier E_a
3. Le serveur fait de même et génère E_b
4. Le client envoie au serveur $E_a \times P$
5. Le serveur envoie au client $E_b \times P$
6. Le serveur calcule alors $(E_b \times P) \times E_a$ et le serveur $(E_a \times P) \times E_b$
7. Le client et le serveur partagent dès lors la même clé.

La propriété du Forward Secrecy est donc obtenue ici grâce à l'unicité des points et entiers choisis par le serveur pour chaque connexion. Une fois la connexion achevée, ces données ne sont pas gardées ni réutilisées, ce qui garanti la non-compromission des données a posteriori, et même si quelqu'un accède à la machine, cette personne ne pourra pas déchiffrer les connexions passées.

Pour autant, il convient de rappeler que la génération de ces paramètres a un coût non négligeable en calcul, et donc en ressources derrière.

Cependant, la taille des clés fournissant une sécurité à niveau équivalent est nettement inférieure lorsque le système est basé sur les courbes elliptiques. Une des raisons pour cela est que toutes les attaques sur les corps finis ne s'appliquent pas sur les courbes elliptiques. Il est donc nettement plus difficile de résoudre le problème de

Diffie-Hellman sur les courbes elliptiques avec des clés « petites » que de le résoudre sur les corps finis.

Le NIST [45] publie régulièrement des recommandations [46] concernant la longueur des clés selon les différents systèmes. Et l'on constate effectivement une grande différence entre les systèmes basés sur les courbes elliptiques et ceux sur les corps finis :

Date	**Corps finis**	**Courbes elliptiques**	**Sécurité**
2010	1024	**160**	court terme - obsolète
2030	2048	**224**	moyen terme
>2030	3072	**256**	long terme
> >2030	7680	**384**	long terme
> > >2030	15360	**512**	long terme

Les tailles des clés sont exprimées en bit.

45. National Institude Of Standards and Technology, responsable de la publication des standards et des normes aux États-Unis. Site officiel : `http://nist.gov/`.

46. Chiffres obtenus sur le site `http://www.keylength.com`.

4 Mise en place sur des serveurs web

4.1 Apache

Apache est un logiciel libre [47] servant de serveur web très utilisé dans le monde, Son usage décroît progressivement à l'avantage de solutions moins gourmandes telles qu'*NginX*.

La version utilisée ici est la **2.2.22** telle que présente dans les dépôts Debian Stable à la date de rédaction du présent rapport.
Afin de mettre en place le SSL/TLS, il est nécessaire de configurer certaines options.

Ci-dessous un exemple de configuration type :

```
SSLEngine on

SSLCompression off

SSLProtocol +TLSv1.2 +TLSv1.1 +TLSv1

SSLHonorCipherOrder on
SSLCipherSuite ALL:!aNULL:!eNULL:!LOW:!EXP:!RC4:!3DES:+HIGH

Header set Strict-Transport-Security "max-age=31536000"


SSLCertificateFile /emplacement/du/certificat/ssl.crt
SSLCertificateKeyFile /emplacement/de/la/clef.private
SSLCertificateChainFile /emplacement/de/la/chaine/CAcert.pem
```

- *SSLEngine* Cette option précise si le SSL/TLS est activé ou non ;
- *SSLCompression* Cette option active ou désactive la prise en charge de la compression ;
- *SSLProtocol* Précise les versions de SSL ou TLS supportées ;
- *SSLHonorCipher* Impose ou non au client l'ordre de préférence des ciphers du serveur ;

47. Sous licence du même nom, Apache, et soutenu par la fondation *Apache*.

— *SSLCipherSuite* Liste de préférence des ciphers ;
— *Header set* Strict-Transport-Security Ajoute le champ *Strict-Transport-Security* aux requêtes, avec comme valeur *max-age=31536000* ;
— *SSLCertificateFile* Emplacement du certificat SSL du serveur ;
— *SSLCertificateKeyFile* Emplacement de la clé privée du serveur ;
— *SSLCertificateChainFile* Emplacement de la chaîne de confiance.

La ligne correspondant à la liste de préférence des ciphers n'affiche aucun Cipher reconnaissable, excepté *RC4* et *DES*.
En effet, cette liste de préférences peut également être triée par « caractéristiques » plutôt que par nom de ciphers : ici, l'usage de ciphers utilisant des clés de longueur inférieure ou égale à 64 bits sont proscrits (via la chaîne *!LOW*) alors que les ciphers autorisés doivent utiliser une clé de longueur strictement supérieure à 128 bits (chaîne *+HIGH*). De la même façon, les ciphers utilisant *RC4* et *3DES* sont proscrits. Enfin, tous les ciphers n'offrant pas de chiffrement (*eNULL*) ni d'authentification (*aNULL*) sont interdits.

Cette ligne peut dont également s'écrire comme la liste des ciphers respectants ces caractéristiques :

```
SSLCipherSuite DHE-RSA-AES256-GCM-SHA384:
DHE-RSA-AES256-SHA256:DHE-RSA-AES128-GCM-SHA256:
DHE-RSA-AES128-SHA256
```

Cependant, cette liste est généralement plus longue à écrire, il est donc avantageux de préférer la méthode présente dans le fichier de configuration.

Un point « intéressant » est l'absence du support des courbes elliptiques. En effet, la branche stable d'Apache n'offre pas le support des ciphers basés sur les courbes elliptiques.
Afin de mettre en place le perfect forward secrecy, il convient donc d'utiliser *Diffie-Hellman Ephemeral*, qui coûte plus cher en ressources serveur ce qui peut poser problème selon la fréquentation de celui-ci.

Pour résumer les actions définies par le fichier de configuration :

1. lorsqu'un client se connecte au serveur, il ne pourra négocier qu'une connexion basée sur le protocole TLS, version 1 à 1.2 comprises. Si le client ne supporte aucune des trois ou essaie de se connecter en SSL, le serveur refusera et la connexion échouera ;
2. la négociation du cipher respectera **impérativement** l'ordre établi par le serveur à la ligne *SSLCipherSuite* ;

3. le serveur informe le client qu'il utilisera automatiquement une connexion sécurisée lors de sa prochaine visite, si celle-ci a lieu moins d'un an après la précédente. Cette information est indiquée via l'instruction *Strict-Transport-Security* ;
4. le serveur utilisera alors le fichier contenant le certificat public, celui contenant la clé privée, et enfin celui contenant la chaîne de signature du certificat listés à la fin du fichier de configuration.

Un serveur configuré de cette façon offre donc le support du perfect forward secrecy en utilisant un échange Diffie-Hellman éphémère.

4.2 NginX

NginX est également un logiciel libre de plus en plus utilisé dans le monde du fait de son plus faible impact en ressources serveur et de sa rapidité.

Il existe en plusieurs versions, dont une gratuite et une payante offrant plus de services. Les explications ci-dessous sont effectuées avec la version 1.6 telle que présente dans la branche stable de NginX.

Afin de mettre en place une connexion TLS, il est nécessaire de configurer certaines options, de la même façon que pour Apache (voir ci-dessus). Cependant, la configuration diffère en certains points détaillés ci-après.

Ci-dessous un exemple de configuration type :

```
listen 443 ssl deferred;

ssl_session_cache shared:SSL:10m;
ssl_session_timeout 5m;

ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

ssl_prefer_ciphers on;
ssl_ciphers ALL:!aNULL:!eNULL:!LOW:!EXP:!RC4:!3DES:+HIGH;

add_header Strict-Transport-Security "max-age=31536000;
            includeSubDomains";
```

```
ssl_certificate /emplacement/du/certificat_et_chaine.crt;
ssl_certificate_key /emplacement/de/la/clef.private;
```

- *listen* Cette option précise sur quel port et protocole écoute NginX ;
- *ssl_session_cache* Cette option précise la durée du cache de session ;
- *ssl_session_timeout* Cette option précise la durée après laquelle la session s'achève ;
- *ssl_prefer_ciphers* Impose ou non au client l'ordre de préférence des ciphers du serveur ;
- *ssl_ciphers* Liste de préférence des ciphers ;
- *add_header* Strict-Transport-Security Ajoute le champ *Strict-Transport-Security* aux requêtes, avec comme valeur *max-age=31536000 ; includeSubdomains* ;
- *ssl_certificate* Emplacement de l chaîne entière de certificats (chaîne de confiance et certificat du serveur) ;
- *ssl_certificate_key* Emplacement de la clé privée du serveur.

De la même façon que pour Apache, la liste de préférence des ciphers peut se faire selon des caractéristiques [48].
Cependant, le support des ciphers est différente : NginX supporte plus de ciphers, y compris les courbes elliptiques.

Ainsi, la ligne *ssl_ciphers* est équivalente à :

```
ssl_ciphers ECDHE-RSA-AES256-GCM-SHA384:
ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-GCM-SHA256:
ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA:
DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-SHA256:
DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-SHA256:
DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA:
RC4-SHA:AES256-GCM-SHA384:AES256-SHA256:CAMELLIA256-SHA:
ECDHE-RSA-AES128-SHA:AES128-GCM-SHA256:AES128-SHA256:
AES128-SHA:CAMELLIA128-SHA;
```

L'avantage de la ligne triée par caractérisques est ici clairement plus avantageuse.

Pour résumer les actions définies dans le fichier de configuration (similaire aux directives d'Apache) :

1. lorsqu'un client se connecte au serveur sur le port 443, il ne pourra négocier

48. Se référer à la configuration d'Apache page 36 pour des explications sur cette ligne.

qu'une connexion TLS, versions 1 à 1.2 comprises. Toute autre version sera refusée par le serveur et la connexion échouera ;

2. l'expiration d'une session s'effectue au bout de 5 minutes d'inactivité de la part du client ;
3. la négociation du cipher respectera **impérativement** l'ordre établi par le serveur à la ligne *ssl_ciphers* ;
4. le serveur informe le client qu'il devra utiliser automatiquement une connexion sécurisée lors de sa prochaine visite sur le même domaine ou sur un sous-domaine, si celle-ci a lieu moins d'un an après la précédente. Cette information est indiquée au client via la ligne *Strict-Transport-Security* ;
5. le serveur utilisera alors le fichier contenant le certificat et la chaîne de signatures, ainsi que dans un fichier séparé la clé privée listés à la fin du fichier.

Un serveur configuré de cette façon offre donc le support du perfect forward secrecy en utilisant un échange Diffie-Hellman éphémère basé sur les courbes elliptiques.

Conclusion

Au travers de ce rapport, j'ai pu étudier les différents mécanismes nécessaires à la mise en place de la propriété du *Forward Secrecy* lors de la connexion entre deux machines : poignée de main entre le client et le serveur, suivie d'une négociation pour enfin établir un système d'échange de clé, préliminaire à l'envoi chiffré de données entre les deux machines.

De plus en plus de personnes privilégient le système de Diffie-Hellman basé sur les courbes elliptiques pour cet échange, mécanisme mis en lumière grâce à la prise de conscience de l'importance de la cryptographie pour faire face aux pratiques de surveillance de masse notamment révélées par Edward Snowden en 2013.

Utiliser les courbes elliptiques permet donc de faire des calculs sur des nombres nettement inférieurs par rapport à ceux pris sur des corps finis. Ainsi, l'économie en ressources, et donc en machines est nettement avantageuse, sans compter le gain en vitesse lors de la connexion.
Par conséquent, il est désormais accessible, grâce à ce faible impact sur les ressources, de générer les paramètres temporaires propre à chaque client pour profiter du Forward Secrecy, et ainsi garantir la sécurité des connexions passées même en cas de compromission de la machine.

Cette UV m'a permis d'approfondir des notions que je côtoyais quotidiennement au travers de mon temps libre en partie consacré à l'administration de serveurs : SSL, TLS, chiffrement, Diffie-Hellman, courbes elliptiques... étaient des mots qui ne m'étaient pas inconnus sans pour autant que j'en perçoive tout le fonctionnement et les détails que j'ai étudiés ici.

Je tiens à remercier les différentes personnes qui ont pris le temps de relire ce document, de me soumettre des améliorations... ainsi que Frédéric Holweck sans qui tout ce travail (et notamment tout l'aspect mathématique) n'aurait pas été possible.

Références

[1] Julian Assange, Jacob Appelbaum, Andy Müller-Maghun et Jérémie Zimmermann : *Cypherpunks, Freedom and the future of the Internet*. OR Books, 2012, 186p. ISBN : 978-1-939293-00-8.

[2] Dan Boneh : Stanford : Cryptography 1, 2014. MOOC suivi durant six semaines sur `https://www.coursera.org/course/crypto`.

[3] Whitfield Diffie, Martin Hellman et Ralph Merkle : New directions in cryptography. Novembre 1976.

[4] Nadia Heninger, Tanja Lange et Daniel Bernstein : The year in crypto. Chaos Communication Congress, Décembre 2013. `http://cdn.media.ccc.de/congress/2013/webm/30c3-5339-en-de-The_Year_in_Crypto_webm.webm`.

[5] NIST : Specification for the advanced encryption standard (aes). Rapport technique, Novembre 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

Licences

Serveur page 11
- Auteur : Thrasos Varnava
- Licence : Creative Commons Attribution Non-commercial (by-nc)

Client page 11
- Auteur : Asher
- Licence : Creative Commons Attribution (by)

MAC page 21
- Auteur : Twisp
- Licence : Domaine public

Principe d'un échange Diffie-Hellman page 23
- Image originale : A.J. Han Vinck
- Licence de l'adaptation utilisée : Attribution-Share Alike 1.0 Generic

Hellman, Diffie, Merkle page 22 (montage composé des images ci-dessous)
- **Hellman** - Licence Creative Commons Attribution-Share Alike 3.0 Unported
- **Diffie** - Licence Creative Commons Attribution 2.0 Generic
- **Merkle** - Auteur : David Orban - Licence Creative Commons Attribution 2.0 Generic

Exemple de données récupérables via la faille HeartBleed page 54
- Auteur : James Long

xkcd : HeartBleed Explanation page 56
- Auteur : Randall Munroe
- Licence : Creative Commons Attribution Non-Commercial 2.5

Apache : Logo page 36
- Auteur : Fleshgrinder
- Licence : Apache License 2.0

NginX Logo page 38
- Auteur : Igor Sysoev
- Licence : Domaine public

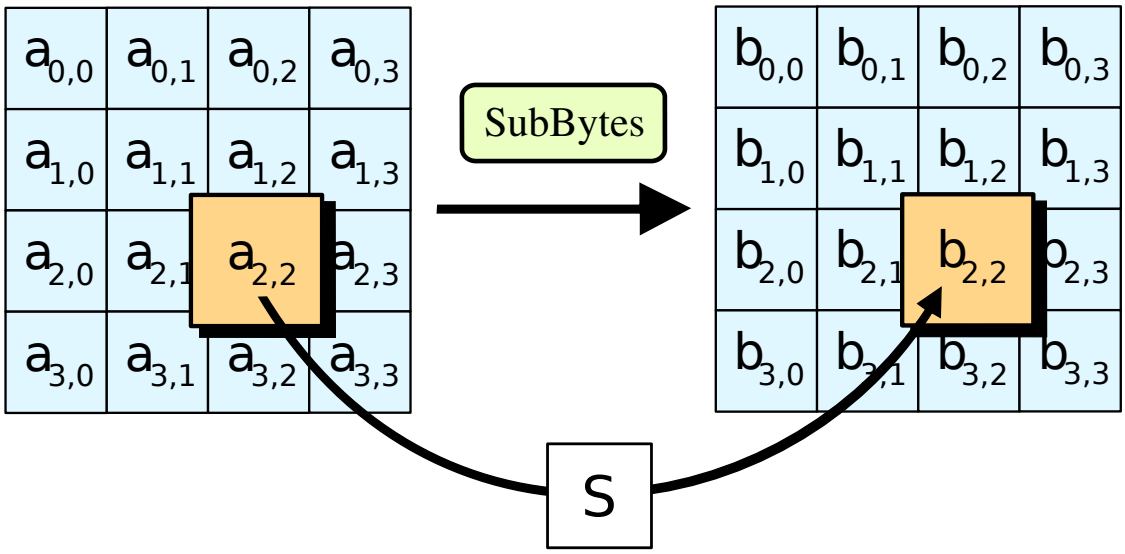
Annexes

A AES

Cette annexe a pour but de détailler les différentes étapes de l'algorithme de chiffrement AES.

A.1 Étape *SubBytes*

Cette étape est destinée à permuter chaque élément de la matrice de base.



Cette première étape d'un cycle AES est ainsi une substitution par la table de Rijndael (disponible page 20)
Chaque entrée de la table de substitution provient de l'inverse d'un élément du corps 2^8.
Ce dernier est alors transformé par l'application affine :

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Où :

$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$ est l'inverse d'un élément du corps 2^8 ;

$\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$ correspond à l'élément nul de la table, 0x63.

Cette dernière matrice est essentielle puisqu'elle permet de s'assurer que chaque substitution est différente de l'entrée. Or l'élément 0 n'a pas d'inverse. Et donc sa subsitution laisserait cette entrée inchangée et affaiblirait le cipher. Ainsi, grâce à cette matrice, on s'assure que même l'élément 0 est modifié lors de la substition sans pour autant toucher à la nature bijective de l'opération pour les autres termes.

Et la première matrice correspond à la somme $A^4 + A^3 + A^2 + A$ où :

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Ainsi, chaque octet à substituer peut donc s'écrire sous forme binaire dans $\mathbb{GF}(2)[X]$ modulo $1 + X + X^3 + X^4 + X^8$, qui est un polynôme irréductible de $\mathbb{GF}(2^8)$.

On obtient dès lors des coefficients $(\alpha_0, \alpha_1, \ldots, \alpha_7)$ qui appartiennent à $\mathbb{GF}(2^8)$.

Ces coefficients sont mis sous forme polynômiale : $\alpha_0 + \alpha_1 \times x + \ldots + \alpha_7 \times x^7$.

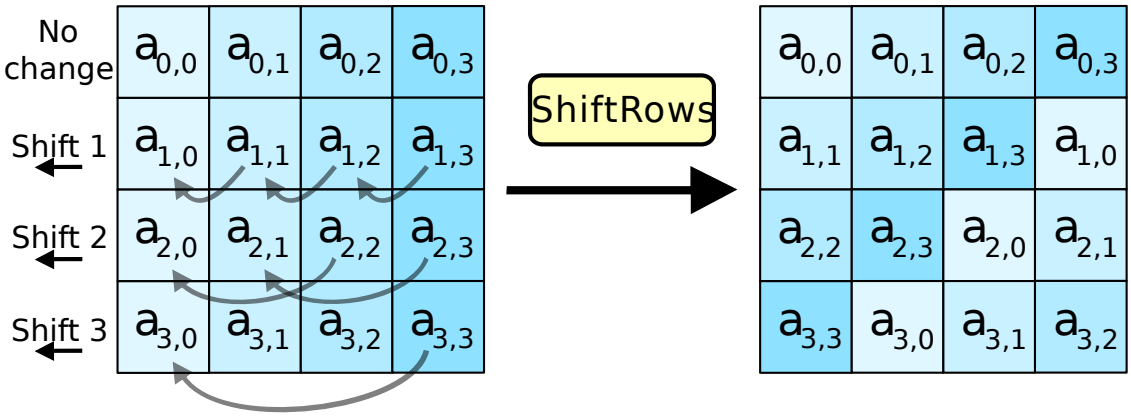
Ce polynôme appartenant à $\mathbb{GF}(2^8)$, il est inversible et son polynôme inverse est : $\beta_0 + \beta_1 \times x + \ldots + \beta_7 \times x^7$.

Enfin, on peut y appliquer la transformation décrite ci-dessus.

Ce faisant, la table de substitution permet ainsi d'éviter tout « point fixe » : chaque entrée a une et une seule sortie et chaque sortie est différente de l'entrée associée.

A.2 Étape *ShiftRows*

Cette étape a pour but d'effectuer un décalage sur de chaque lignes de la matrice. Chaque octet des trois dernières lignes sont décalés différemment :

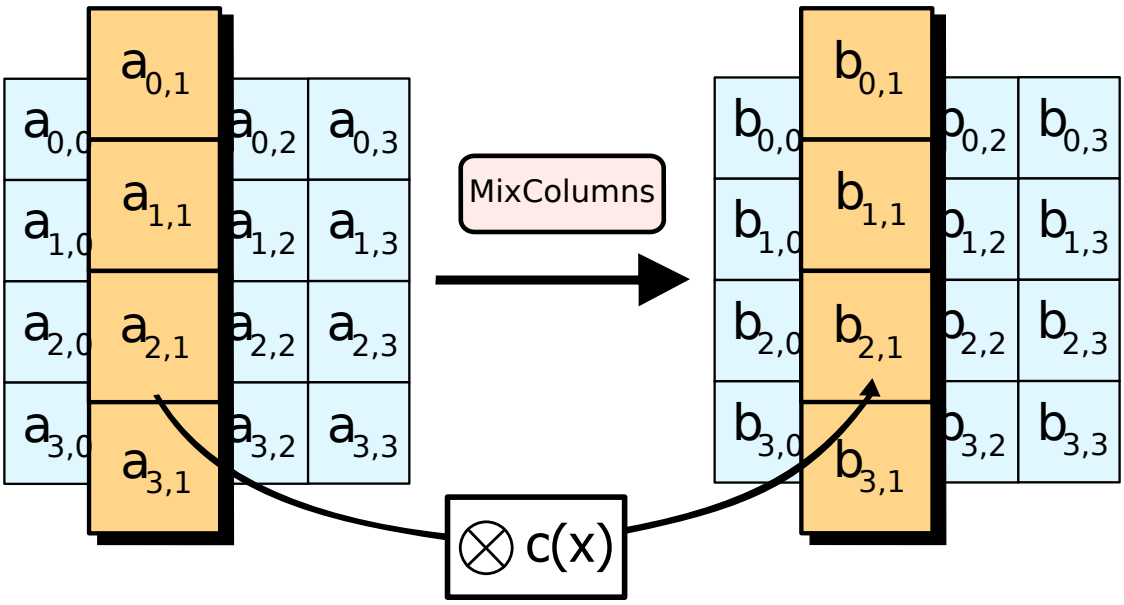


— deuxième ligne : un décalage vers la gauche ;
— troisième ligne : deux décalages vers la gauche ;
— quatrième ligne : trois décalages vers la gauche.

Cela permet donc d'avoir des lignes indépendantes les unes des autres.

A.3 Étape *MixColumns*

Cette étape applique ensuite une transormation linéaire de chaque élément de la matrice.



Chaque colonne est alors multipliée par un polynôme fixe $a(x) = [03]x3+[01]x2+[01]x+[02]$, le tout modulo x^4+1 qui est un polynôme non inversible dans $\mathbb{GF}(2^8)$.

Le polynôme correspond à la matrice :

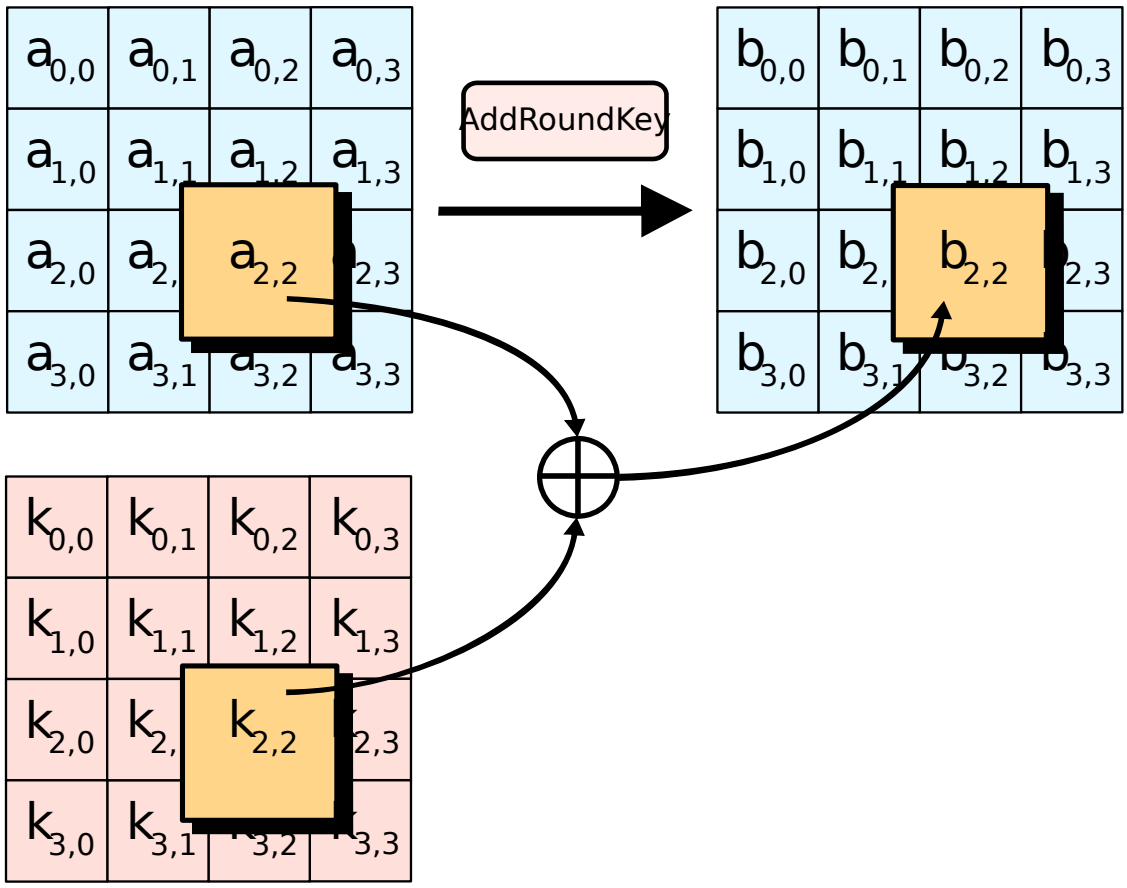
$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

Cette matrice possède une propriété de diffusion [49] élevée et est donc essentielle au sein du processus de chiffrement.

49. Propriété où la relation entre le texte d'entrée et le texte de sortie est très faible.

A.4 Étape *AddRoundKey*

Enfin, la dernière étape utilise la clé du cycle pour faire une opération sur chaqye élément de la matrice.



Chaque octet est alors xoré avec l'octet correspondant de clé du cycle actuel.

B HeartBleed, retour sur la publication du 7 Avril 2014

Comme expliqué dans la partie 4 (page 36), la gestion du SSL/TLS est très souvent assurée par OpenSSL, un outil libre et opensource utilisé par environ deux tiers des serveurs dans le monde [50]

Lundi 7 Avril 2014, deux chercheurs en sécurité ont publié une faille dans une extension d'OpenSSL, utilisée dans le protocole TLS. Cette faille n'est pas un problème de sécurité au sein du protocole TLS, mais simplement une erreur d'implémentation au sein de la bibliothèque OpenSSL. Pour autant, ses conséquences sont extrêmement graves puisqu'elle permet de récupérer potentiellement clés privées, identifiants, mots de passe...

Cette découverte donc l'occasion de revenir sur cette extension d'OpenSSL et son rôle au sein d'un échange TLS.

La faille porte le nom d'*HeartBleed* et vise l'extension *HeartBeat* de TLS.

HeartBeat Connaître l'état de la machine à qui on parle est essentiel, et encore plus lors d'un échange sécurisé comme avec TLS. HeartBeat est un mécanisme prévu pour tester la connexion dans l'optique de savoir si les deux parties de l'échange sont toujours présentes ou non.

L'utilisation de l'extension HeartBeat est négociée lors des messages **CLIENT_HELLO** et **SERVER_HELLO** [51] :

```
enum {
    peer_allowed_to_send(1),
    peer_not_allowed_to_send(2),
    (255)
} HeartbeatMode;

struct {
    HeartbeatMode mode;
} HeartbeatExtension;
```

- une machine peut spécifier qu'elle utilisera l'extension sans elle-même répondre aux requêtes *HeartBeatRequest* en utilisant le mode *peer_not_allowed_to_send* ;
- le mode *peer_allowed_to_send* signale donc que la machine utilisera et répondra elle aussi aux requêtes *HeartBeatRequest*.

50. D'après un sondage réalisé par Netcraft disponible sur http://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html.

51. Voir partie 1.2.2.

Une machine peut donc utiliser ce test de *battements de cœur* à tout moment et à n'importe quelle fréquence lors de la connexion TLS afin de connaître l'état de la machine avec qui elle communique.

Ce test de battement de cœur est constitué d'un message dont la structure est la suivante :

```
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```

- **type** correspond à la nature du message :
  - *heartbeat_request* pour une requête,
  - *heartbeat_response* pour une réponse.
- **payload_length** correspond à la longueur de la charge utile ;
- **payload[HeartbeatMessage.payload_length]** correspond au contenu de la charge utile ;
- **padding[padding_length]** correspond à la longueur du padding.

Fonctionnement Une des parties de l'échange TLS peut envoyer une requête *HeartbeatRequest*, contenant des données quelconques (stockées dans **payload**). Ce champ de données quelconques est de longueur définie par **payload_length**. En réponse, la machine recevant la requête doit répondre avec **exactement** le même contenu du champ **payload**.

HeartBleed La faille découverte concernant cette extension porte particulièrement sur la longueur de la charge utile : la machine devant faire une copie exacte de la requête d'origine, il est essentiel de vérifier la longueur effective du contenu à copier afin d'éviter de répondre avec une charge utile contenant d'autres données en plus de la charge utile d'origine[52].

Cependant, la vérification n'était pas faite.

Ainsi, le commit[53] ayant introduit la faille introduisait le support de *HeartBeat* avec notamment l'extrait de code suivant :

```
1 /* Read type and payload length first */
2 hbtype = *p++;
3 n2s(p, payload);
4 pl = p;
```

52. Voir la figure B 56 pour une illustration simple du fonctionnement
53. Commit 4817504d069b4c5082161b02a22116ad75f822b1 publié le 1^{er} Janvier 2012.

```
5
6  if (s->msg_callback)
7          s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
8      &s->s3->rrec.data[0], s->s3->rrec.length,
9      s, s->msg_callback_arg);
10
11 if (hbtype == TLS1_HB_REQUEST)
12 {
13         unsigned char *buffer, *bp;
14         int r;
15
16         /* Allocate memory for the response, size is 1 bytes
17         * message type, plus 2 bytes payload length, plus
18         * payload, plus padding
19         */
20         buffer = OPENSSL_malloc(1 + 2 + payload + padding);
21         bp = buffer;
22
23         /* Enter response type, length and copy payload */
24         *bp++ = TLS1_HB_RESPONSE;
25         s2n(payload, bp);
26         memcpy(bp, pl, payload);
27
28         r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT,
29     buffer, 3 + payload + padding);
```

Ligne 2 Le premier octet correspond au type de message (**hbtype**) ;

Ligne 3 On déplace les deux octets suivants (fonction *n2s*) de **p** dans **payload** ;

Ligne 4 Le reste de **pl** correspond donc aux données à renvoyer telles quelles ;

Ligne 20 La mémoire nécessaire à l'élaboration de la réponse est allouée :
- — *un octet* pour le type,
- — *deux octets* pour la longueur des données,
- — *payload* octets de données,
- — *padding* octets pour le padding.

Ligne 24 Le premier octet de la réponse prend le type de message (fonction *s2n*) ;

Ligne 25 On déplace alors les deux octets stockant la taille des données de **payload** à **bp** ;

Ligne 26 On effectue ensuite une copie de **payload** octets de **pl** dans **bp** (fonction *memcpy*).

On peut donc le voir ligne 3, aucune vérification sur la longueur de la charge utile (**payload**) n'est faite.

Pourtant, ligne 20, on alloue de la mémoire pour le message de réponse en tenant compte de la longueur annoncée de la charge utile.

Enfin, ligne 26, une copie de la mémoire est effectuée pour former le message de réponse afin qu'il soit copie conforme de la charge utile d'origine.

Cette absence de vérification de la taille de **payload** peut donc conduire à copier de la mémoire qui ne contient pas les données de la charge utile d'origine qui est censée être renvoyée.

Jusqu'à 64Ko de la mémoire de la machine peuvent ainsi être envoyés au sein de la charge utile du message *heartbeat_response*. Cette limite s'appliquant à chaque message, la répétition de telles requêtes peut conduire à une grande quantité de mémoire fuitée.

Correctif et conséquences La publication de la faille s'est accompagnée d'un correctif[54] ajoutant les vérifications de la taille de la charge utile :

```
1 /* Read type and payload length first */
2 if (1 + 2 + 16 > s->s3->rrec.length)
3     return 0; /* silently discard */
4 hbtype = *p++;
5 n2s(p, payload);
6 if (1 + 2 + payload + 16 > s->s3->rrec.length)
7     return 0; /* silently discard per RFC 6520 sec. 4 */
8 pl = p;
```

On vérifie dès lors que la taille annoncée de la charge utile correspond à la taille effective :

- **1** (octet) correspond à la taille de la variable stockant le type du message (requête ou réponse) ;
- **2** (octets - $2^{16} = 64Ko$) permet de stocker la taille de la charge utile ;
- **payload** (octets) correspond à la charge utile ;
- **16** (octets) correspond au padding.

Ce faisant, si la longueur de la charge utile envoyée est de 64Ko mais que la taille de la charge utile effective est inférieure, la fonction renvoie 0 et ne va pas plus loin (et donc pas de copie de mémoire).

Les données stockées dans la zone mémoire copiée (si la charge utile est de longueur inférieure à celle annoncée) peut contenir de nombreuses informations :

- identifiants utilisateur ;

54. Commit 96db9023b881d7cd9f379b0c154650d6c108e9a3 publié le Lundi 7 Avril 2014.

— mots de passe ;
— contenu de la page ;
— cookies de session ;
— clés utilisées pour le chiffrement.

De fait, il est donc possible de récupérer la clé privée d'un serveur, et par conséquent être apte à déchiffrer l'intégralité du traffic.

Un grand nombre de serveurs ont donc été touchés, par exemple :
— banques ;
— fournisseurs de mails et autres services ;
— sites marchands ;
— réseaux sociaux.

Beaucoup de serveurs n'ont pas été corrigés immédiatement, et beaucoup sont encore vulnérables.

Ainsi, Yahoo ! a été particulièrement long pour corriger le problème et exposait les identifiants (identifiant et mot de passe) de ses utilisateurs durant plusieurs heures. Il en a été de même pour Darty qui a corrigé la faille le 9 Avril, soit deux jours après sa divulgation...

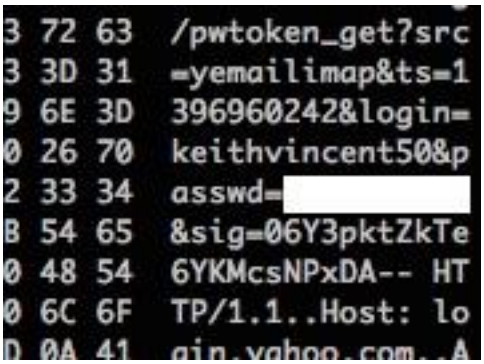


FIGURE 11 – Exemple de données aisément récupérables en exploitant la faille : on aperçoit l'identifiant (*keithvincent50*) et le mot de passe (ici masqué).

Résolution Cette faille, introduite dans OpenSSL 1.0.1 est présente dans OpenSSL 1.0.1 jusqu'à la version 1.0.1f inclue.

Les branches 1.0.0 et 0.9.8 ne sont pas touchées par cette faille.

Le correctif a été intégré dans OpenSSL et rendu disponible au sein de la version **1.0.1g**. Afin de corriger la faille sur une machine, il est donc nécessaire de [55] :

55. Ces opérations ont été effectuées le mardi 8 Avril au matin sur les serveurs présentés dans la

1. mettre à jour OpenSSL vers la version 1.0.1g (ou revenir sur une branche précédente non affectée par cette faille) ;
2. révoquer les certificats SSL ;
3. regénérer les clés ;
4. générer de nouveaux certificats [56] ;
5. demander un changement de mots de passe des utilisateurs une fois la faille corrigée.

Perfect Forward Secrecy Comme expliqué par cet article [57] de l'EFF [58], les serveurs ayant le (Perfect) Forward Secrecy en place ont pu limiter les conséquences de la faille.

En effet, même en cas de compromission de la clé privée du serveur, les connexions passées ne peuvent pas être déchiffrées [59].

Malheureusement, peu de sites l'ont mis en place, mais on peut imaginer que cet évènement fera progresser son adoption.

partie 4.

56. Les certificats de la partie 1.3 ont donc été révoqués et regénérés.
57. *Why the Web Needs Perfect Forward Secrecy More Than Ever* : https://www.eff.org/deeplinks/2014/04/why-web-needs-perfect-forward-secrecy.
58. Association américaine de défense des libertés numériques. Site officiel : https://www.eff.org.
59. Voir partie 3 page 27.

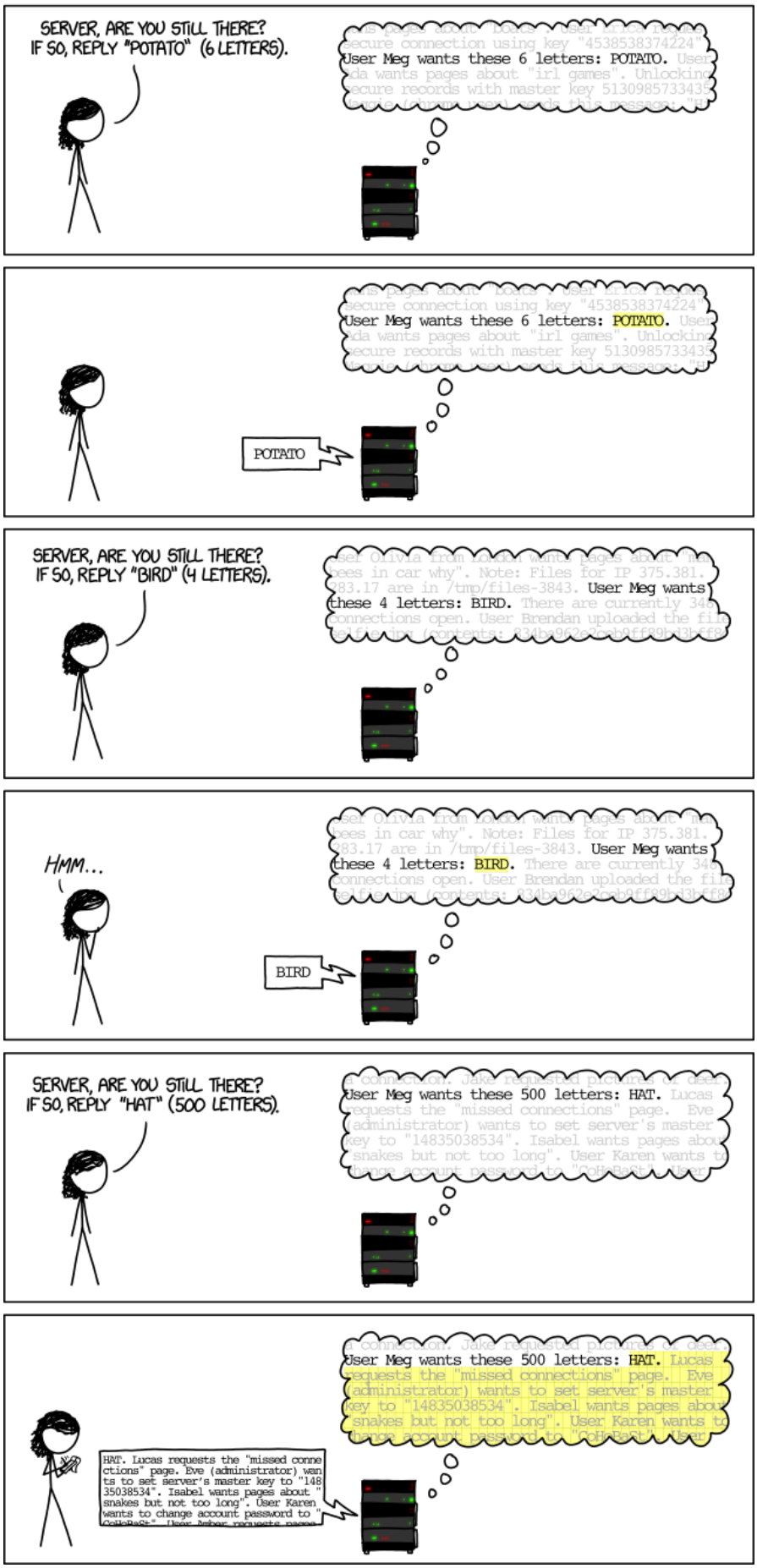


FIGURE 12 – xkcd : HeartBleed Explanation

C Notion de groupe, anneau et corps

Groupe

La notion de groupe est très utilisée par les mécanismes liés à la cryptographie : Diffie-Hellman l'utilise comme principe de base, AES comme un des principes dans l'étape *MixColumns*[60]...

Un groupe est un ensemble muni d'une loi de composition interne $\star$ possédant les propriétés :

— associativité : $\forall x, y, z \in G : x \star (y \star z) = (x \star y) \star z$;
— il existe un unique élément neutre, noté e : $\forall x \in G : x \star e = e \star x = x$;
— tout élément est inversible : $\forall x \in G, \exists y \in G, x \star y = y \star x = e$.

Prenons pour exemple $\mathbb{Z}$, l'ensemble des entiers relatifs :

— la somme de deux entiers est aussi un entier ;
— $\forall a \in \mathbb{Z} : a + 0 = a$;
— $\forall a \in \mathbb{Z}, \exists b : a + b = 0$;
— $\forall a, b, c \in \mathbb{Z} : (a + b) + c = a + (b + c)$.

Ainsi, Z est un groupe pour l'addition, noté $(\mathbb{Z}, \star)$.

Anneau

Un anneau est un ensemble muni de deux lois de composition interne $+$ et $\star$ telles que :

— $(A, +)$ soit un groupe, et contient donc un élément neutre ;
— $\star$ est une loi :
  — associative,
  — distributive comparée à $+$.

Prenons une nouvelle fois pour exemple $\mathbb{Z}$:

— comme vu précédemment, $(\mathbb{Z}, +)$ est un groupe ;
— la multiplication est une loi :
  — associative : $\forall x, y, z \in \mathbb{Z} : (x \times y) \times z = x \times (y \times z)$,
  — distributive par rapport à l'addition : $\forall x, y, z \in \mathbb{Z} : x \times (y + z) = (x \times y) + (x \times z)$.
— l'élément neutre est 1.

Ainsi, $\mathbb{Z}$ est un anneau pour l'addition et la multiplication.

60. Voir Annexe A.3 page 48

Corps

Un corps est un anneau où tous les éléments (excepté les éléments nuls) sont inversibles pour la multiplication.

Par exemple, pour l'ensemble $\mathbb{Z}/p\mathbb{Z}$ correspondant aux congruences modulo p (avec p premier) : $\mathbb{Z}_p = \{0, 1, \ldots, p-1\} \bmod p$:

— groupe pour l'addition ;
— anneau pour l'addition et la multiplication ;
— chaque élément non-nul possède un inverse pour la multiplication.

Ainsi, $\mathbb{Z}_p$ est un corps p premier.