Chiffrement

> Don't be lasagna.
>
> Le douzième docteur

Les méthodes relatives à la sécurité vues précédemment ne permettent pas de s'affranchir de tous les problèmes de sécurité. En effet, si des réponses partielles ont été apportées sur les problèmes d'authentification par exemple, les données échangées entre le serveur et les clients transitent en clair sur le réseau, ce qui n'est pas souhaitable lorsque ces informations sont trop sensibles (c'est typiquement le cas pour le commerce électronique par exemple) ou lorsqu'on souhaite préserver sa vie privée. D'autre part, les clients n'ont jamais la garantie qu'ils s'adressent au serveur auquel ils pensent s'adresser. Il y a donc un problème d'authentification, non plus des utilisateurs, mais des serveurs eux-mêmes.

Pour répondre à ces deux problèmes, le protocole SSL/TLS [1], reposant sur des certificats d'authenticité, est supporté par Apache, Nginx et la plupart des serveurs web. Sous Apache 2, ce protocole est géré par le module `mod_ssl` [2] et `ngx_http_ssl_module` [3] pour Nginx.

ATTENTION On ne dit pas « crypter », « encrypter », « désencrypter » ou « cryptage ». On dit « chiffrer », « déchiffrer » et « chiffrement ». Le « chiffrage », c'est évaluer le coût de quelque chose. Par contre, le « décryptage » existe : c'est retrouver le message de départ d'un message chiffré sans posséder la clé de déchiffrement. Allez lire https://chiffrer.info/ pour plus de détails.

1 Quelques notions de cryptologie

Au niveau cryptologique, on distingue deux grandes familles de méthodes de chiffrement : les symétriques et les asymétriques. Les méthodes symétriques reposent sur l'utilisation d'une seule clé, utilisée aussi bien pour chiffrer que pour déchiffrer une information. La sécurité est alors basée sur la non-divulgation de la clé (qui doit rester secrète) et sur la difficulté de la deviner. Ces méthodes sont très rapides à l'exécution mais posent des problèmes de confidentialité, étant donné que l'expéditeur et le destinataire doivent tous les deux connaître une même clé, qui doit forcément être communiquée avant d'initier les communications chiffrées. Ce type de technique est très difficile à mettre en œuvre à large échelle.

À la différence de ces premières méthodes, les méthodes asymétriques reposent sur l'utilisation de deux clés, une publique et une privée, liées par une relation mathématique. L'une d'entre elles est utilisée pour le chiffrement, tandis que l'autre est utilisée pour le déchiffrement. Les méthodes à base de clés asymétriques sont nettement plus lentes, et donc plus coûteuses en temps de traitement, que les méthodes à base de clés symétriques mais elles procurent un niveau de sécurité très élevé.

SSL (*Secure Sockets Layer*) et TLS (*Transport Layer Security*) reposent sur des algorithmes de chiffrement aussi bien symétriques qu'asymétriques, selon les principes énoncés ci-dessus. Les clés utilisées sont transmises au moyen de certificats, émis par des organismes appelés *autorités de certification*. Les navigateurs Web intègrent par défaut une liste de telles autorités [4]. Les certificats utilisés sont au format X.509 [5]. Lorsqu'un site Web désire proposer des connexions sécurisées au moyen de SSL/TLS, une demande de certification doit être formulée auprès d'une autorité de certification. Ces autorités peuvent être celles intégrées aux navigateurs [6], ce qui est l'usage pour les sites s'adressant au public le plus large, ou être

1. http://fr.wikipedia.org/wiki/Transport_Layer_Security N'ergotons pas, je vais parler de SSL tout du long, mais j'entends par là SSL/TLS. SSL est désormais à proscrire, il ne faut plus utiliser que TLS, mais les habitudes (et les noms des directives) ont la vie dure.
2. https://httpd.apache.org/docs/2.4/mod/mod_ssl.html
3. http://nginx.org/en/docs/http/ngx_http_ssl_module.html
4. sous Firefox, cette liste est consultable dans les préférences, rubrique « Avancé », bouton « Gérer les certificats », onglet « Autorités »
5. https://fr.wikipedia.org/wiki/X.509
6. elles représentent alors des *tiers de confiance*

confondues directement avec le site proposant le chiffrement, ce qui est le cas lorsque les utilisateurs visés sont restreints. Dans ce dernier cas, ce sont alors le plus souvent des certificats auto-signés [7].

Les certificats auto-signés, ainsi que les certificats signés par des autorités atypiques comme CAcert, sont sans aucun doute voué à quasiment disparaître depuis la création de Let's encrypt [8].
Let's encrypt est une initiative de différents acteurs du Net pour fournir une autorité de certification gratuite et promouvoir l'utilisation du chiffrement.
Gratuit, simple à mettre en place et très simplement automatisable, Let's encrypt a réussi, en quatre ans, à générer plus d'un milliard de certificats [9] et plus de 221 millions de ses certificats sont actifs à ce jour [10].

Pour résumer, il convient donc de bien différencier les *clés*, utilisées pour chiffrer les communications entre deux acteurs, et les *certificats*, dont le but est d'établir l'authenticité d'un acteur (un serveur Web dans notre cas, mais cela peut aussi être un serveur mail, un serveur XMPP…).

2 Préparation du certificat

Le logiciel `openssl` est généralement associé à la bibliothèque `libssl` ; il permet en particulier de créer des clés à partir de plusieurs méthodes de chiffrement. Une des méthodes les plus populaires est la méthode RSA, autrefois protégée par un brevet qui a expiré en septembre 2000.

2.1 Méthode classique, sans Let's encrypt

1. Créer une clé de serveur.
   Cette clé doit être unique pour chaque site Web à protéger et constituera une partie du certificat qui, une fois signé par une autorité de certification, permettra l'authentification du site auprès des clients (typiquement les navigateurs Web). Le nom de la clé est arbitraire, mais est généralement construit à partir du nom du site associé à la clé, et possède généralement l'extension `.key`.
   Lors de la création de la clé, une phrase de passe peut être demandée. Cette phrase fonctionne comme un mot de passe, mais peut être plus longue, ce qui renforce la sécurité qui lui est associée. Il faut donc, comme pour les mots de passe, la retenir pour pouvoir utiliser par la suite la clé qui sera générée (lors de la création du certificat en particulier).
   On se passe généralement de phrase de passe (mettre une phrase de passe vide) pour un certificat serveur : la phrase de passe n'est généralement utilisée que pour les certificats *signants*, c'est à dire les certificats utilisés pour signer votre certificat. En effet, en cas de redémarrage du serveur, il faudrait redonner la phrase de passe et si le serveur crashe à 3h du matin, il faudrait attendre que vous soyez levé pour pouvoir démarrer le serveur Web.

   Un exemple typique d'utilisation d'`openssl` pour générer une clé de serveur est présentée ci-dessous.

   ```
   openssl genrsa [-des3] -out example.org.key 4096
   ```

   L'option `des3`, si elle est présente, chiffre la clé, ce qui nécessitera une phrase de passe pour l'utiliser (à éviter, donc, pour un certificat serveur). Il existe d'autres options de chiffrement au résultat similaire.
2. Créer une demande de signature numérique de certificat X.509 (fichier `.csr` par convention, pour *Certificate Signing Request*, mais on peut choisir l'extension qu'on veut).
   La phrase clé demandée au cours de cette étape est la même que celle fournie lors de la création de la clé de serveur, si vous en aviez une.

   ```
   openssl req -new -key example.org.key -out example.org.csr
   ```

7. CAcert (http://cacert.org) est une autorité de confiance non présente dans les navigateurs. Elle n'est pas considéré comme un tiers de confiance, mais les certificats signés par cette autorité ne sont pas des certificats auto-signés.
8. https://letsencrypt.org/
9. https://letsencrypt.org/2020/02/27/one-billion-certs.html
10. https://letsencrypt.org/stats/

3. Pour créer une clé et une demande de signature d'un seul coup, on peut utiliser :

```
openssl req -new -newkey rsa:4096 -sha512 -nodes -out example.org.csr \
   -keyout example.org.key -subj "/C=FR/ST=/L=Example/O=Example/CN=example.org"
```

4. Obtenir le certificat.
   À partir du fichier `.csr` obtenu, il est alors possible de s'adresser à une autorité de certification pour faire établir le certificat correspondant. Dans l'optique d'un site Web destiné à être accessible par un grand nombre d'utilisateurs, il est alors judicieux de choisir une autorité reconnue par le plus grand nombre possible de navigateurs. Cette opération se fait auprès de sociétés commerciales (par exemple Verisign, Thawte…), est payante et est limitée dans le temps (les certificats délivrés ont une date d'expiration).
   Pour d'autres types d'utilisations, si l'intérêt principal d'un certificat réside plus dans le chiffrement des données que dans l'authentification du site hébergeur, il est également possible de gérer sa propre autorité de certification et ainsi de produire des certificats *autosignés*. Les certificats produits devront alors être acceptés explicitement par tous les navigateurs utilisés. Cette solution est généralement adoptée pour les réseaux d'entreprise, ne souhaitant pas assurer des communications sécurisées pour le grand public, mais pour un nombre restreint d'utilisateurs, typiquement les employés de cette entreprise. Cette solution est décrite dans le paragraphe suivant.
   Dans tous les cas, le certificat établi correspond *in fine* à un fichier à extension `.crt` ou `.pem`.

2.2 Avec Let's encrypt

Pour Let's encrypt, la procédure est bien plus simple. On installe un client Let's encrypt, par exemple le client officiel `certbot` [11], et on le lance avec les options idoines, genre :

```
certbot certonly --rsa-key-size 4096 --webroot -w /var/www/html/ -d example.org
```

Et boum ! Pour peu que l'on ait bien choisi ses options et que le serveur web soit bien configuré, on a un certificat qui nous attend dans `/etc/letsencrypt/live/example.org`.

Bien évidemment, les options et les chemins peuvent changer selon le client Let's encrypt choisi ou la méthode (ici `webroot`) choisie pour le *challenge* avec le serveur de Let's encrypt.

Pour créer un certificat Let's encrypt, il faut être en mesure de prouver que l'on maîtrise bien le nom de domaine du certificat demandé, que ce soit par un *challenge* HTTP (le client dépose un fichier sur votre serveur web et l'autorité va le récupérer) ou DNS (vous créez un enregistrement DNS que l'autorité va consulter).

3 Chiffrer les connexions avec Apache

L'installation et la configuration d'Apache se réalise de la manière suivante :

1. Vérifier que `mod_ssl` est bien installé (`ls /etc/apache2/mods-available/ssl.*`) [12].
2. Configurer Apache pour l'utilisation de SSL pour le certificat obtenu. Il faut tout d'abord activer le module SSL sous Apache, avec `a2enmod ssl` et en créant un hôte virtuel permettant de traiter les requêtes relatives à ces transactions sécurisées :

```
<IfModule mod_ssl.c>
    <VirtualHost *:443>
        SSLEngine On
        SSLCertificateFile /etc/apache2/ssl/example.org.pem
        SSLCertificateKeyFile /etc/apache2/ssl/example.org.key
```

11. https://certbot.eff.org/
12. s'il ne l'est pas… faut-il que je vous fasse un dessin ? :p

```
        ...
    </VirtualHost>
</IfModule>
```

La directive `<IfModule>` permet d'éviter des plantages d'Apache lors d'un restart alors que le module ssl a été désactivé par mégarde.

4 Chiffrer les connexions avec Nginx

1. Ajoutez `ssl` à la directive `listen` ;
2. Ajoutez les chemins vers les fichiers du certificat et de la clé à l'aide des directives `ssl_certificate` et `ssl_certificate_key` ;

```
listen 443 ssl;
ssl_certificate /etc/nginx/ssl/example.org.pem;
ssl_certificate_key /etc/nginx/ssl/example.org.key;
```

NB Il existe un certain nombre de directives Apache et Nginx pour spécifier les protocoles utilisés (TLS 1, 1.1, 1.2), les *ciphers*, etc. Je vous conseille **fortement** de lire leurs documentations et de suivre les recommandations de https://ssl-config.mozilla.org/ pour leur utilisation.

Pour vérifier la solidité de votre configuration SSL, vous pouvez tester votre site web sur https://www.ssllabs.com/ssltest/ et sur https://cryptcheck.fr/ (attention, ce dernier site note de façon **très** sévère).

5 Gestion des certificats autosignés

Là encore, suivant la version d'Apache considérée et suivant les configurations déjà effectuées par la distribution utilisée (si Apache n'a pas été installé à partir de ses sources), plusieurs techniques permettent de gérer les certificats autosignés :

— Avec les versions 1.x d'Apache, un script `sign.sh` était fourni en standard. Ce script permettait alors de générer un fichier `.crt` à partir d'un fichier `.csr`[13] :
```
./sign.sh example.org.csr
```
— Avec les versions 2.x d'Apache, un script `apache2-ssl-certificate` permet de générer automatiquement un certificat autosigné sans même avoir à générer au préalable une clé de serveur. C'est donc une procédure simple permettant d'obtenir très rapidement le certificat souhaité. Ce script n'étant pas fourni dans Debian, on peut se tourner vers l'utilitaire `make-ssl-cert` du paquet `ssl-cert`.
— Sinon, il est toujours possible de signer son certificat à la main via l'utilitaire `openssl` :
```
openssl genrsa -out example.org.key 4096
openssl req -new -key example.org.key -out example.org.csr
openssl x509 -in example.org.csr -out example.org.pem -req \
   -signkey example.org.key -days 365
```
On crée une clé (`example.org.key`) avec la commande pour créer une clé de serveur. Cette clé est alors utilisée pour signer un certificat à partir d'une demande de signature de certificat (`example.org.csr`) créé avec cette même clé d'où le terme de *certificat auto-signé*.
En plus court :
```
openssl genrsa -out example.org.key 4096
openssl req -new -x509 -days 365 -key example.org.key \
   -out example.org.pem
```

13. Ceci dit, les configurations d'Apache via cette technique posaient un certain nombre de problèmes. En particulier, Apache devait être exécuté de manière manuelle de façon à ce que la phrase clé puisse être saisie lors du démarrage. Des directives telles que `SSLPassPhraseDialog` — indiquant l'emplacement d'un programme devant retourner la phrase en question — pouvaient alors contourner le problème et permettre une exécution automatique d'Apache, ce qui est généralement le mode de lancement souhaité.

6 Utiliser sa propre autorité de certification

On peut créer sa propre autorité de certification pour signer des certificats. Même si elle ne sera pas reconnue de base par les navigateurs, cela peut avoir du sens en entreprise où les postes de travail sont gérés par son département informatique : rien de plus simple que d'ajouter une autorité de certification à un système d'exploitation ou un navigateur.

Il faut d'abord générer un certifat auto-signé, celui de l'autorité :

```
openssl genrsa -aes256 -out ca.key 4096
openssl req -new -x509 -days 365 -key ca.key -out ca.pem
```

Notez l'option `-aes256` : cette option va chiffrer la clé avec un mot de passe qui vous sera demandé. En effet, si la clé du certificat d'un site web doit être lisible par le serveur web pour pouvoir redémarrer n'importe quand et donc devoir être stockée non chiffrée, la clé d'une autorité de certificat gagne en sécurité à nécessiter un mot de passe pour s'en servir. Une personne malveillante récupérant le fichier de la clé de l'autorité ne pourrait pas s'en servir pour signer des certificats.

On génére ensuite une clé pour le site, ainsi qu'une demande de signature :

```
openssl genrsa -out example.org.key 4096
openssl req -new -key example.org.key -out example.org.csr
```

Enfin, on crée le certificat avec la clé de l'autorité de certification :

```
openssl x509 -req -CA ca.pem -CAkey ca.key -CAcreateserial \
  -days 365 -in example.org.csr -out example.org.pem
```

7 Liens utiles

- https://blog.eleven-labs.com/fr/comprendre-ssl-tls-partie-1/
- https://blog.eleven-labs.com/fr/comprendre-ssl-tls-partie-2-chiffrement/
- https://blog.eleven-labs.com/fr/comprendre-le-ssltls-partie-3-certificats/
- Recommandations sur les *ciphers* et autres éléments de configuration à utiliser : https://ssl-config.mozilla.org/
- Tester la sécurité et les bonnes pratiques de son serveur :
  - https://www.ssllabs.com/ssltest/
  - https://cryptcheck.fr (Attention, celui-ci note **très** sévèrement)

8 Un peu de pratique

Attention : ne pas inclure ces manipulations dans les réponses aux exercices.

Apache et Nginx

1. Mettez en place SSL en utilisant un certificat autosigné. Profitez-en pour étudier attentivement la documentation des composantes que vous aurez à manipuler.
2. Faites cohabiter sur votre serveur un hôte sans chiffrement ainsi qu'un hôte sécurisé.
3. Modifiez votre serveur sécurisé en utilisant cette fois-ci un certificat signé par une autorité de certification que vous aurez créée.
   - Que constatez-vous lors de la connexion à votre serveur ?
   - Installez le certificat de l'autorité de certification dans votre navigateur.

— Que constatez-vous lors de la connexion à votre serveur [14] ?

4. Utilisez une règle de réécriture pour obliger l'utilisation du serveur sécurisé si on arrive sur le serveur non sécurisé.

14. non, je ne suis pas gâteux, la question est répétée pour une bonne raison