Créer un certificat autosigné

Présentation

Un certificat autosigné peut servir :

- à un utilisateur pour lui permettre de s'authentifier auprès d'un serveur par exemple à distance avec SSH ;
- à un serveur pour configurer des communication chiffrées comme htpps pour un serveur Web.

Ce document utilise le logiciel OpenSSH soit sous Linux ou Windows.

- Pour en savoir plus sur l'utilisation d'OpenSSL en CLI :
  - https://wiki.openssl.org/index.php/Command_Line_Utilities
  - https://www.madboa.com/geek/openssl/

Démarche

La création d'un certificat auto-signé se réalise en trois étapes

- Création de la **clé privée** ;
- Création de la **requête de certification de la clé publique** ;
- Création du **certificat** qui va contenir la clé publique et les informations d'identification de l'utilisateur ou du serveur

Dans ce document est présenté la création d'un certificat autosigné pour un utilisateur

> Avec OpenSSL, la clé privée générée contient également les informations de la clé publique. Il n'est donc pas nécessaire de générer la clé publique séparément.

Création de la clé privée

Dans l'invite de commandes lancez la commande suivante :

```
$ openssl genrsa -aes256 -out user-private.pem 2048
```

Indiquez une pass-phrase pour votre nouvelle clé, pass-phrase qui sera demandé lors de la création du certificat.

Explication :

- **genrsa** : Génération de la clé avec l'algorithme RSA
- **-aes256** : Utilisation d'une passe phrase (facultatif) en précisant l'algorithme de chiffrement de la clé privée AES256
- **-out** : Indique le fichier de sortie
- **2048** : Taille de la clé

Pour visualiser le contenu de cette clé privée :

```
openssl rsa -in user-private.pem -text
```

> La commande suivant permet de connaître la liste des chiffrements disponibles :
>
> ```
> $ openssl list -cipher-algorithms
> ```

Générer la clé publique RSA (contenu dans le fichier user.pem)

```
$ openssl rsa -in user-private.pem -out user-public.pem -pubout
Enter pass phrase for user-private.pem
```

```
Writing RSA key
```

Pour visualiser le contenu de cette clé privée :

```
$ openssl rsa -in user-public.pem -pubin -text
```

Vous disposez maintenant :

- d'une clé privée **user-private.pem** ;
- d'une clé publique **user-public.pem**.

Il est possible de supprimer la pass-phrase associée à la clé privée en générant une nouvelle clé privée à partir de cela déjà créer. Pour cela exécutez la commande suivante :

```
$ openssl rsa -in user-private.pem -out user-private_nopass.pem
```

Création de la requête de demande de signature du certificat

Pour générer la requête de demande de signature, il faut utiliser le fichier contenant la clé privée.

Lors de la création de la requête vous avez à saisir vos informations d'identification et il est important d'indiquer un **Common Name** unique.

- Saisissez la commande suivante :

```
$ openssl req -new -key user-private.pem  -out user-private.csr

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:Haute-Vienne
Locality Name (eg, city) []:Limoges
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Lycée Syzanne Valadon
Organizational Unit Name (eg, section) []:BTS SIO
Common Name (e.g. server FQDN or YOUR name) []:user
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Explication :

- **req** : Gestionnaire de requête
- **new** : Nouvelle requête
- **-x509** : indique le type de certificat Nouvelle requête
- **-key** : Indique la clé à utiliser
- **-out** : Indique le fichier de sortie

Génération du certificat au format x509 avec une validité de 365 jours auto-signé

```
$ openssl x509 -req -days 365 -in user.csr -signkey user-private.key -out user.crt
```

Le certificat généré est le fichier **.crt** et la clé privée associée, le fichier **.key**.

Notes

Création clé privée du CA :

```
openssl genrsa 2048 -out ca-key.pem
```

Une clé maintenant :

- la clé privée de votre autorité de certification (CA) ca-key.pem.

===== Création certificat du CA en renseignant les informations d'identité. Il est important que le Common Name soit unique pour chaque certificat:

```
openssl req -new -x509 -nodes -days 365000 -key ca-key.pem -out ca-cert.pem
```

Deux clés maintenant :

- la clé privée de l'autorité de certification (CA) ca-key.pem.
- le fichier du certificat de l'Autorité (CA) ca-pert.pem ;

Création de la clé privée pour un serveur

Création de la clé privée du serveur en utilisant les deux fichiers créés auparavant et en indiquant un Common Name unique et en définissant un mot de passe comme chalenge password pour protéger la clé privée.

```
$ sudo openssl req -newkey rsa:2048 -nodes -keyout server-key.pem -out server-req.pem
Generating a RSA private key
writing new private key to 'server-key.pem'
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:Haute-Vienne
Locality Name (eg, city) []:Limoges
Organization Name (eg, company) [Internet Widgits Pty Ltd]:BTS SIO CNED
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:MariaDB server
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Traitement de la clé privée du serveur MariaDB pour obtenir un type de clé RSA :

```
$ openssl rsa -in server-key.pem -out server-key.pem
writing RSA key
```

Signature du certificat du serveur par votre autorité de certification (CA) :

```
$ openssl x509 -req -in server-req.pem -days 365000 -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 -
out server-cert.pem
Signature ok
subject=C = FR, ST = Haute-Vienne, L = Limoges, O = Internet Widgits Pty Ltd, OU = BTS SIO, CN = server
Getting CA Private Key
```

Vous disposez maintenant de deux fichiers supplémentaires qui permettront de sécuriser la communication le serveur :

- Le fichier de certificat du serveur : server-cert.pem ;
- Le fichier de clé privée du serveur : server-key.pem

Vérification que le certificat du serveur est correct avec la commande suivante :

```
$ openssl verify -CAfile ca-cert.pem server-cert.pem
server-cert.pem: OK
```

Création du certificat SSL pour un utilisateur

Création de la clé privée de l'utilisateur en définissant un mot de passe comme chalenge password :

```
$ sudo openssl req -newkey rsa:2048 -nodes -keyout user-key.pem -out user-req.pem
Generating a RSA private key
writing new private key to 'user-key.pem'
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:Haute-Vienne
Locality Name (eg, city) []:Limoges
Organization Name (eg, company) [Internet Widgits Pty Ltd]:BTS SIO
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:user
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Traitement de la clé privée du client pour obtenir un type de clé RSA :

```
$ sudo openssl rsa -in user-key.pem -out user-key.pem
writing RSA key
```

Signature du certificat du client MariaDB par votre autorité de certification (CA) :

```
$ sudo openssl x509 -req -in user-req.pem -days 365000 -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01
-out user-cert.pem
Signature ok
subject=C = FR, ST = Haute-Vienne, L = Limoges, O = Internet Widgits Pty Ltd, OU = BTS SIO, CN = User
Getting CA Private Key
```

Vérification du certificat du client MariaDB :

```
$ openssl verify -CAfile ca-cert.pem user-cert.pem
user-cert.pem: OK
```

Format de fichier

Les identités numériques sont enregistrées dans des conteneurs protégés et standardisés au format PKCS#12. Un fichier de ce type contient :

- la clé privée ;
- le certificat de l'utilisateur(contenant sa clé publique) ;
- le certificat de la CA signataire

Ce fichier qui contient l'identité numérique d'un utilisateur a en général l'extension .p12 et est protégé par mot de passe.

Convertir le fichier PFX/P12 au format PEM

```
# openssl pkcs12 -in moncertificat.p12 -out moncertificat.pem -nodes
```

Extraire la clé privée de votre certificat au format PFX/P12

```
# openssl pkcs12 -in moncertificat.p12 -out maclessh.pem -nodes -nocerts
```

Extraire la clé publique du certificat au format PEM et ajout de celle-ci dans le fichier contenant la clé privée

```
# openssl x509 -in moncertificat.pem -pubkey -noout >> maclessh.pem
```

Utilisation de ssh-keygen pour créer l'empreinte a copier sur le serveur d'accès dans le fichier ~/.ssh/authorizedkeys

```
# ssh-keygen -i -m PKCS8 -f maclessh.pem
```

La ligne obtenu doit être copié sur le serveur, dans le fichier ~/.ssh/authorizedkeys

Test de la connexion Maintenant, vous pouvez tester la connexion au serveur avec de l'authentification forte.

ssh -i maclessh.pem utilisateur@NOM_DU_SERVEUR

Optionnel : ajout du fichier maclessh.pem dans la configuration du client Si vous le souhaitez, vous pouvez ajouter les lignes suivantes dans le fichier de configuration de votre client SSH. Celui se trouve généralement à l'endroit ~/.ssh/config Host NOM*DU*SERVEUR IdentityFile chemin/vers/maclessh.pem

Une fois que cela est fait, vous n'avez plus qu'à taper ceci pour accéder au serveur :

ssh utilisateur@NOM_DU_SERVEUR

Pour extraire Ortu extraire Ressources : https://tbs-certificats.com/FAQ/fr/auth_forte_openssh.html

CSR (Certificate Signing Request) : demande externe de certificat à la CA en fournissant la clé publique et des informations d'identité et obtention après vérification d'un certificat signé

Format de fichier

Lien : https://qastack.fr/server/9708/what-is-a-pem-file-and-how-does-it-differ-from-other-openssl-generated-key-file

- **.csr** : fichier de demande de signature d'un certificat contenant la clé publique et les information d'identité. Le format actuel est PKCS10, défini dans la RFC 2986. Après signature par l'autorité de certification et un certificat est renvoyé au demandeur.
- **.pem** (Privacy Enhanced Mail) : format de conteneur pouvant inclure :
  - uniquement le certificat public ,
  - une chaîne de certificats complète comprenant une clé publique, une clé privée et le certificat racine. De manière confuse ;
  - un CSR car le format PKCS10 peut être traduit en PEM. Le nom vient de Privacy Enhanced Mail (PEM).
- **.key** : fichier au format PEM contenant uniquement la clé privée d'un certificat spécifique. Il s'agit d'un nom conventionnel et non normalisé.
- **.pkcs12 .pfx .p12** : format de conteneur entièrement chiffré avec mot de passe contenant les paires de certificats publics et privés. Openssl peut transformer ce conteneur en un fichier .pem avec des clés publiques et privées (openssl pkcs12 -in file-to-convert.p12 -out converted-file.pem -nodes)
- **.der** : fichier dont les données sont encodées en binaire à la différence du format **.pem** codé en Base64. OpenSSL peut convertir un fichier .der en .pem (openssl x509 -inform der -in to-convert.der -out converted.pem). Windows considère les fichiers .der comme des fichiers de certificat et exportera par défaut les certificats sous forme de fichiers au format .der avec une extension différente.
- **.cert .cer .crt** : fichier au format .pem (ou rarement .der) portant une extension différente, reconnu par l'explorateur Windows comme un certificat, contrairement à **.pem**.
- **.p7b .keystore** - Défini dans le RFC 2315 en tant que PKCS numéro 7, il s'agit d'un format utilisé par Windows pour l'échange de certificats. Java les comprend de manière native et les utilise souvent .keystorecomme une extension. Contrairement aux certificats de style .pem, ce format comporte une méthode définie pour inclure les certificats de chemin de certification.
- **.crl** : liste de révocation de certificats utilisée par les autorités de certification comme moyen de désautoriser les certificats avant

leur expiration.

En résumé, il existe quatre manières différentes de présenter les certificats et leurs composants:

- PEM : Régi par les RFC et utilisé préférentiellement par les logiciels open source. Il peut avoir une variété d'extensions (.pem, .key, .cer, .cert, etc.)
- **PKCS7** : Norme ouverte utilisée par Java et prise en charge par Windows. Ne contient pas de matériel de clé privée.
- **PKCS12** : offret une sécurité renforcée par rapport au format PEM en texte brut. Cela peut contenir du matériel de clé privée. Il est utilisé préférentiellement par les systèmes Windows et peut être librement converti au format PEM via OpenSL.
- **DER** : format parent de PEM et est à considéré comme une version binaire du fichier PEM codé en base64. Pas couramment utilisé en dehors de Windows.