Certificat x509

→ confidentialité de tls basé sur chiffrement asymétrique

CSR : Certificate Signing Request : Autorité de certification
genre letencrypt ↵

↳ signature :
- entité
- date
- clé publique
- tampon d'un tier de confiance
- condition d'utilisation valide

↝ formats : x509, ssh

x509 ↝ RFC 5280, largement utilisé (SSL/TLS ...)

↳ version, numéro de série, issuer, subject, date de validité (début, fin), clé publique, extensions, signature de l'emetteur du certificat

on send ces données à l'autorité de certification

après vérification l'autorité emet un certificat avec les données + la signature de l'autorité

↳ signature = hash(data) chiffré avec clé privé de l'autorité

→ il suffit de déchiffrer avec clé publique la signature et comparer avec le hash qu'on produit

↳ dec(data, P_k) = hash(data) ???

↳ à vérifier !!!

1 Manipulation de certificats existants

L'outil `openssl x509` permet d'effectuer diverses manipulations sur les certificats.

1. De nombreux certificats se trouvent dans le répertoire `/etc/ssl/certs`. Choisir un certificat et trouver la commande `openssl x509` permettant d'afficher son contenu sous format texte.
2. Relever les différents types d'information contenus dans ce certificat.
3. Dans le répertoire `/etc/ssl/certs`, trouver deux certificats utilisant des mécanismes de signature différents. Pour chacun, relever les caractéristiques cryptographiques embarquées dans le certificat.
4. Le format de certificat le plus courant dans `/etc/ssl/certs` est le format `PEM`, mais le format `DER` reste également très répandu. Choisir un certificat `PEM` dans `/etc/ssl/certs` et trouver la commande `openssl x509` permettant le convertir en un certificat `DER`.
5. De même, convertir un certificat `DER` vers un certificat `PEM`.

TP x509

I
1. openssl x509 -in /etc/ssl/certs/.....pem -text
2. —
3. "Signature Algorithm" | sha256WithRSAEncryption
   sha1WithRSAEncryption
4. openssl x509 -in /etc/ssl/certs/...pem -inform PEM -out cert.der -outform DER
5. openssl x509 -in cert.der -inform DER -out cert.pem -outform PEM

2.1 Certificats auto-signés

1. Le type de certificats le plus simple à générer est les certificats auto-signés. En effet, il ne nécessite la mise en place d'aucune PKI particulière. Ce sont des certificats à usage interne. A l'aide de l'outil `openssl req`, choisir un nom de subject quelconque et générer un certificat auto-signé valable 10 ans, signé avec une clé RSA 4096.

2.2 CSR et CA

L'outil `openssl req` permet également de générer des demandes de signature de certificat (CSR).

1. Commencer par générer une paire de clés RSA 2048 privée/publique à l'aide de `openssl genrsa`.
2. A l'aide de `openssl req`, utiliser la paire de clé générée pour générer une nouvelle CSR, avec SHA256 comme algorithme de hachage, et popeye comme subject.
3. Afficher le contenu de la CSR sous format texte avec `openssl req` et relever les différentes informations qu'elle contient.

Cette CSR devra ensuite être envoyée à une autorité de certification (CA) pour obtenir le certificat correspondant, signé par elle. Dans le cadre du TP, nous allons nous contenter de créer une CA en local.

1. Générer une nouvelle paire de clé, mais pour la CA cette fois-ci.
2. Générer un certificat auto-signé par la CA (CN=TLSSEC). La CA possède alors tout le matériel cryptographique pour signer des CSR.
3. A l'aide de `openssl x509`, signer la CSR obtenue précédemment avec cette CA.
4. A l'aide de `openssl x509`, afficher le contenu du certificat obtenu. En particulier, vérifier que le subject est bien popeye et que l'issuer est bien TLSSEC.
5. Enfin, à l'aide `openssl verify`, vérifier que le certificat est correct.

II
1. self signed : openssl req -x509 -newkey rsa:4096 -keyout key.pem -out req.pem -days 3650
   ↑ 10 ans

CSR
1. openssl genrsa -out key 2048 (privée)
   openssl rsa -in key -pubout -out key.pub (publique)
2. openssl req -new -key -out key.csr -sha256 -subj "/CN=popeye"

③ openssl req -in key.csr -noout -text

CA

① openssl genrsa -out ca.key 2048
openssl rsa -in ca.key -pubout -out ca.pub

② openssl req -new -x509 -key ca.key -out ca.crt -sha256 -days 3650 -subj "/CN=TLSSEC"

③ openssl x509 -req -in key.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out key.crt
-days 365 -sha256

④ openssl x509 -in key.crt -noout -text

⑤
```
→  tp_x509 openssl verify -CAfile ca.crt key.crt
key.crt: OK
```

```
→ tp_x509 openssl x509 -in key.crt -noout -text

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            52:13:40:12:49:9c:c0:1c:2c:1b:67:9e:12:f1:c7:0b:8a:64:f5:86
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=TLSSEC
    Validity
        Not Before: Nov 15 11:01:13 2025 GMT
        Not After : Nov 15 11:01:13 2026 GMT
    Subject: CN=popeye
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
```

3 Utilisation de PKCS#12

Dans certains cas, il est pratique de pouvoir distribuer le certificat et la clé privée dans un même fichier ; par exemple lors de la distribution des identifiants à nouvel utilisateur voulant se connecter à des ressources données. Le format PKCS#12 permet cela.

1. A l'aide de `openssl pkcs12`, créer un fichier PKCS#12 contenant la clé privée et le certificat de l'utilisateur popeye ayant fait la CSR à l'exercice précédent.

① openssl pkcs12 -export -inkey key -in key.crt -out key.p12
↳ demande un mot de passe (car contient la clé privée)

```
→  tp_x509 openssl x509 -in key.crt -noout -text

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            52:13:40:12:49:9c:c0:1c:2c:1b:67:9e:12:f1:c7:0b:8a:64:f5:86
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=TLSSEC
        Validity
            Not Before: Nov 15 11:01:13 2025 GMT
            Not After : Nov 15 11:01:13 2026 GMT
        Subject: CN=popeye
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a0:e0:68:b6:41:30:e3:6f:f3:df:45:2e:6e:1d:
                    32:5d:33:b0:0c:3e:b0:18:24:55:0b:42:11:9b:b0:
                    ef:ad:6e:e4:b7:e8:2f:7a:79:a4:28:21:8d:d4:35:
                    7b:d5:24:92:48:38:30:35:71:27:28:c5:ec:18:16:
                    98:b3:37:9d:12:ab:7f:e4:18:54:6a:cb:8c:bc:12:
                    ad:a9:79:c6:ac:10:60:3c:5a:8f:ed:4f:d0:42:6b:
                    19:ae:f2:d3:da:85:8f:70:3e:22:fa:68:9a:d8:3f:
                    55:c9:75:6d:09:78:87:f1:35:5c:af:18:56:d2:20:
                    a8:c7:8b:6a:b8:d8:47:01:5a:e6:93:2e:a3:19:6b:
                    aa:c2:91:ef:e3:94:17:a4:8a:ab:01:64:aa:eb:5c:
                    dc:ab:07:54:4f:6e:f2:f2:6e:4c:eb:f2:48:02:ae:
                    a1:9c:38:a3:3c:d4:e1:e0:b0:a6:d1:0c:88:f9:b3:
                    ec:30:f9:04:77:ee:15:6f:2b:01:ea:0d:09:85:85:
                    00:ee:ad:72:5b:51:97:c6:fd:26:f9:64:f7:63:0d:
                    39:1f:ce:a2:82:da:98:bf:9b:d1:20:f4:d1:8c:07:
                    21:45:10:2e:91:b9:9b:c5:a4:83:a4:5f:cf:14:a6:
                    ad:18:3e:50:15:ce:c5:97:e0:fd:c5:57:59:99:20:
                    72:db
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                F8:C5:BD:DC:19:49:35:53:12:2A:88:C9:60:D3:EA:07:21:5E:05:B5
            X509v3 Authority Key Identifier:
                05:3B:DE:F8:73:70:E9:34:8C:BA:D1:DC:E3:55:BC:F9:26:85:1D:01
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        7b:7b:ea:b3:6a:85:f7:2e:56:2c:e7:60:43:27:00:7c:c8:d6:
        fb:16:2b:75:f5:b0:54:38:b2:cb:36:ef:1e:55:2e:af:0a:2e:
        b5:ff:13:0a:57:ce:11:cb:1f:13:77:74:51:a8:cd:f7:3c:d5:
        0f:d8:0a:7d:a7:b9:32:e9:06:18:2c:97:36:96:be:51:39:1e:
        52:54:97:dc:12:43:1b:be:b5:c4:37:c6:63:1f:23:9a:79:0b:
        2f:66:c7:28:b3:17:fe:da:87:40:5e:b7:22:19:76:e6:6b:9a:
        0c:ec:0f:0a:ec:5a:2f:85:f6:e3:f0:a0:3b:19:2a:28:4b:90:
        fb:ca:ea:d7:e3:a1:b8:48:5a:fb:23:6a:e7:c5:e6:44:7e:61:
        5f:91:78:1f:22:5f:33:9b:33:e9:b0:38:c3:cd:6e:30:70:f3:
        77:c0:c9:e6:be:78:aa:0d:c9:47:32:bf:9f:f0:cd:c1:16:55:
        ae:01:3f:8f:32:08:e0:8f:ba:a2:f5:d6:c1:21:80:fb:9e:eb:
        71:fa:e1:a8:6a:7e:ce:fd:02:c9:ca:f6:ea:63:8c:fb:96:6b:
        99:8c:bd:5f:21:8a:18:07:0f:1f:ea:15:07:90:e4:a0:b9:78:
        96:1d:39:4c:fa:5b:00:71:51:54:18:10:38:50:1c:5b:24:01:
        72:35:6f:a0:44:b2:16:16:f9:91:2f:17:da:1b:ff:cd:c0:fd:
        82:d3:2a:d4:66:ca:83:7a:13:d9:f1:a2:f3:e5:78:fd:28:08:
        11:a3:d6:56:de:e9:c0:35:c4:33:b1:00:85:96:d3:67:e7:83:
        f6:44:21:c1:43:d1:0b:63:a5:ed:f5:67:c3:5c:17:ff:db:b7:
        8f:d0:b0:31:63:ef:6f:d4:c8:8c:72:2d:0a:b8:71:b9:c6:a0:
        2a:9d:a2:ce:68:4b:9c:bb:d1:af:38:a6:e4:9a:ad:2e:fe:cb:
        7c:7c:ad:0e:7d:b4:3a:03:b6:33:d5:1b:5f:22:e6:94:61:4d:
        f4:1b:c5:09:5f:7b:80:86:64:63:57:76:1e:07:be:fa:31:3b:
        d6:89:76:fa:04:07:56:a8:06:89:8a:1a:a4:78:ff:f5:8d:ee:
        bc:ea:97:07:5b:1a:41:0a:51:77:94:62:ba:14:99:c3:d4:cf:
        d8:76:ad:8e:a4:31:fd:54:41:3e:e6:34:ee:ea:1b:3f:25:9a:
        40:23:a8:fc:ad:8d:be:70:21:39:4f:f2:4d:08:8a:16:d4:bb:
        27:8b:61:0c:c4:14:9a:2e:ef:02:0c:f4:b7:87:b3:7c:d7:47:
        94:0b:f1:1e:6a:2f:6e:c9:1f:d1:6a:07:87:ab:e2:29:0a:59:
        a9:10:cd:9b:9d:20:8e:e3
```