

TP N°10 BLOC N°3



**Cryptographie et création d'une
architecture à clé publique (PKI) à partir
l'outil OpenSSL sous Linux Debian**

SOMMAIRE

Supervision du serveur « SRVWEB » hébergeant le site de la ville avec « SRVNAGIOS »

Etape 1 : Mettre à jour les paquets et installer Openssl	Page 1
Etape 2 : Openssl et le chiffrement symétrique.....	Page 1
Etape 3 : Openssl et le chiffrement asymétrique.....	Page 2
Etape 4 : OpenSSL et le hachage (fonction de hachage).....	Page 3
Etape 5 : Réalisation d'une infrastructure à clé publique (PKI) personnalisée à partir de OpenSSL.....	Page 4 à 6
Etape 6 : sécurisation du site "Les jardinsSaint-Eloi", basé sous le service web Apache2 , avec les éléments et votre PKI et des protocoles SSL/TLS.....	Page 7
Etape 7 : Test du dispositif de sécurisation du site Web, "jardin de Saint Éloi", gérer par OpenSSL et basé sur le service apache deux.....	Page 8

Etape 1 : Mettre a jour les paquets et installer Openssl

Pour commencer ce to sur la Cryptographie et l'architecture à clé publique, nous allons commencer par installer et mettre à jour les paquets OpenSSL.

Pour cela si ce n'est pas encore fait saisissez la commande "sudo apt update puis sudo apt upgrade" pour mettre à jour les paquets

Après quoi installons openssl en saisissant la commande "apt install openssl". Pour verifiez la version de open ssl saisissez la commande "openssl version"

```
oot@tepson:/home/tepson# openssl version
penSSL 3.0.19 27 Jan 2026 (Library: OpenSSL 3.0.19 27 Jan 2026)
oot@tepson:/home/tepson#
```

***Version de Openssl utilisé**

A noté que le fichier de configuration OpenSSL ce trouve dans le répertoire " /etc/ssl/openssl.cnf. Pour l'éditer saisissez la commande "nano /etc/ssl/openssl.cnf

Il permet de configurer : les autorités de certification, les paramètres TLS, les politiques de certificats, les extensions X509.

Etape 2 : Openssl et le chiffrement symétrique

***Pour rappelle le chiffrement symétrique utilise la même clé pour chiffrer et déchiffrer.**

Pour lister les algorithmes qui sont disponible saisissez la commande "openssl enc -list"

Maintenant nous allons créer un fichier texte que nous allons appelé "chiffre_sym" dans le quel il contiendra la phrase que vous voulez, pour ma part ce sera "HelloWorld ! Ceci est un test de chiffrement symétrique". Pour cela saisissez la commande "echo **"Texte_que_vous_voulez_saisir"** > **nom_du_fichier_text.txt**". Dans on cas ce sera " echo "HelloWorld ! Ceci est un test de chiffrement symétrique" > chiffre_sym.txt".

Après avoir créée le fichier texte "chiffre_sym.txt", nous allons chiffrer ce fichier texte, pour cela saisissez la commande "openssl enc -aes-256-cbc -salt -in chiffre_sym.txt -out chiffre_sym.enc.

1. **-aes-256-cbc** : C'est l'algorithme de chiffrement utilisé, l'algorithme AES avec clé 256 bits en mode CBC

2. **-salt** : cela ajoute un seul (valeur aléatoire) pour renforcer la sécurité contre les attques par dictionnaire
3. **-in** : c'est le fichier d'entré
4. **-ou** : c'est le fichier de sortie chiffré

Après avoir chiffrer le fichier texte "chiffre_sym.txt" nous allons supprimer le fichier. Pour cela saisissez la commande "rm chiffre_sym.txt".

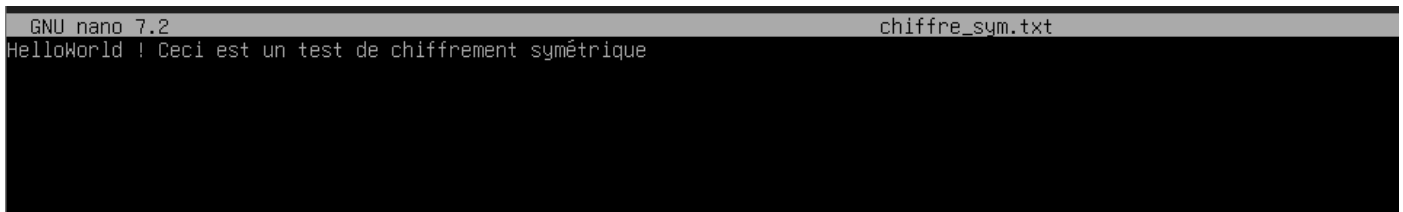
Maintenant déchiffrons le fichier chiffre_sym.enc pour cela saisissez la commande "openssl enc -d -aes-256-cbc -in chiffre_sym.enc -out chiffre_sym.txt"

Ont pourrais pensez la que cette de ligne de commande est pareil que la 1er, nous pourrions pensez qu'ils sont pareil mais la seul chose qui diffère c'est l'ajout de "-d"

1. **-d** : pour le déchiffrement

Maintenant si ont édite le contenue du fichier chiffre_sym.txt en saisissant la commande "nano chiffre_sym.txt", ont peut voire le contenue du fichier chiffre_sym.txt.

Contenue du fichiers chiffre_sym.txt



```
GNU nano 7.2                               chiffre_sym.txt
HelloWorld ! Ceci est un test de chiffrement symétrique
```

***contenue du fichier chiffre_sym.txt**

Sans ouvlier que l'option "-salt" ajoute un sel aléatoire avant chiffremen, ce qui garantit que deux fichiers identiques produisent des chiffrés différents, empechant les attaques pré-calcul.

Etape 3 : OpenSSL et le chiffrement asymétrique

Mais mettons en place le chiffrement asymétrique avec Openssl pour cela créons un fichier texte appeler "chiffre_asym.txt" et pour contenu "HelloWorld ! Ceci est un test de chiffrement asymétrique" pour cela saisissez la commande " echo "HelloWorld ! Ceci est un test de chiffrement asymétrique" > chiffre_asym.txt

A noter : le chiffrement asymétrique avec OpenSSL utilise deux clés différentes :

1. clé publique → pour chiffrer
2. clé privée → pour déchiffrer

Maintenant nous allons générer une clé privée en saisissant la commande "openssl genpkey -algorithm RSA -out clepriv.pem -pkeyopt rsa_keygen_bits:2048 ":

1. genpkey : genere une clé privée
2. -algorithm RSA : c'est l'algorithme utilisé (RSA)
3. -out : présente le fichier de sortie
4. pkeyopt rsa_keygen_bits:2048: c'est la taille de clé qui est (2048 bits)

maintenant listons le répertoire tepson, pour cela saisissez la commande "ls -l"

```
root@tepson:/home/tepson# ls -l
total 20
-rw-r--r-- 1 root root 58 10 avril 21:05 chiffre_asym.txt
-rw-r--r-- 1 root root 80 10 avril 17:30 chiffre_sym.enc
-rw-r--r-- 1 root root 57 10 avril 17:31 chiffre_sym.txt
-rw----- 1 root root 1704 10 avril 21:30 clepriv.pem
-rw-r--r-- 1 root root 451 10 avril 21:55 clepub.pem
```

ont peut constater les fichiers dans le répertoire que nous avons crée précédemment. Ont constate également que la clé privée possède des permissions restrictives (lecture et écriture uniquement pour le propriétaire) afin de garantir sa confidentialité. En revanche, la clé publique est accessible en lecture pour tous les utilisateurs car elle peut être partagée sans risque

Voici ce qu'on constate si on édite les fichiers "clepub.pem" et "clepriv.pem".

Contenu du fichiers clepub.pem

```
GNU nano 7.2 clepub.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoDh0kLedzUpNLq++hB1X
/hjMqvoAmvogv61YRc6cxPPTeQ81TVK5gwtG3Gy4EDTbcGPeHiBXtq6cvva+drGu
+5fycQwY03buEV2L5sdUd0bYQnThdFIjPXDV3GL544G9U1QoGNYS3t8Qd8XxpN0
2aqySs14iey1wESLdPTjoI2Ynz9MyAigcfxeYzbEY7WA6goQtG5mJRHfV2MDuuU8
mEtoJAas7x4sfqpfVfM06afk0354rW1ETn3KVp+f22y1G8ULFEv2FfmR88ZKrARC
UcoHQrzkzLdrv4kG+jy9DFxf0TmJe+40eYq1uJXu8hvX080tIdhndvQeRBIJSw0
7QIDAQAB
-----END PUBLIC KEY-----
```

Contenu du fichiers clepriv.pem

```

GNU nano 7.2 clepriv.pem
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBCcwggSJAEEAAQIBAQCKOHSQt53NSk0u
r76EGVf+GMyyq+gCa+1C/qJhFzpE89MRDyJNUrmOC0cbclgQ5NtwY96GIFe2rpy+
9r52UC771/IJDBg7du4RVkvmx1R3RthCd0G118IM9cNXcYvnJgb1SVcGy1Jne3xB
3xfGmc7ZqrJKyXIJ7LXARit0900gJ2lfP0zICKBx/F5JNsTLtYDqChC0bmaNECw9
kuQ65TyYS2gk8qzvHix+C19UWYPP+Q7fnitbUR0fcpWn5/bbKUbxQsUS/YV+2Hz
xkqsBEJRygdCv0Tmt2u+0IQb6PL0MXF/ROY177g55iqW6Ne7yG9fTw62J2Gd29B5
EEgmxaJtAgMBAAECggEASV4dmdAm9KI29F45CCyIdvHecp6p6ZNJUUD87KfWRL
4om34gLM9cJygKUj4QxKC5LaxmKpvHqh9iHBFVKTEY4sJkSNHLwbp25p50Wquw
j5RH10XkaDGeEbE1d51o/CoqltV70U0Gq1IpoZt9yo/aJE1RK2YD90v2GtWTRJt
KH6crqDlWv2DQJAwLW141pi4H/vJg+vT046oGdnUtq0UieAwvCKVjBE/EnevL1AW
Vu1G0n/M1UTJj132z86y4FRd07menCy6v11Tm8y20u7mE1Dvr3+24G1Ks1gb6Eym
cV8pS022SDExH1BiaK4uYRPGRC+qXxpIYuPHuq3tkuKBgQDi52CZv+Uq7IOLREgz
VW+zvz6ti5Cxs2XFT3p4rz5G0SoA2Jnq8begMAv jTaplMKSHIiM/qzDrIVX+U2qK
KXieUR0ueJiUnG+Vkt9f817Xw8odgsNF1t+0pcwJC/80+8h7Mc15U7Qxgz100HJ2
lK12cpICKKJW0YzYgt+cJg0xEuKBGQC5SNbwXT0aksPkzft7t23Ct1NJhKCGa6kT
KwIvYobCx2eDvcxwDqS82cIBeVv2JjyHxQmP+uECR8KkgvZ5H0+2f2C1fMIxgz+k
znANuKhHTUfW87RyB2rmvYk8b0uh1vr2MFHailQX1k1YheFIH3f18Rz4CqtuCGAG
oTUT5b9/uKBG8BQC4hrn3CQAuqb6nX33zmSh22PL7VurFHL/jW6V2nQvrKrPvJK
yP5zRq108DSR6FUAUvawSE+rLNMCGJS32mp/teGBGaYynV8FgFivcWPTx6ggeJ1t
s3nwIuL4N6V4BJR0uKS2tJeKNDY7GbcS1mbT2UyJmm0It969VQUfF8HAoGBAJHh
19nI1CjCBYDQ7YCrG7CDxv/LfNHzS/Xb1wzX9mLFvarsdVqFLQ/EJAu1EVLtKCb7
edvzNQgmmAoPUyBouBROlh+16vAvrxKDtSYHhfRVP4E/YIOnc3P2J4R7/18qzG2o
XcpDv5JFGQDp6E2xAGX7QWPKgtDixzhxPFAFekzhAoGAYe3NGb296vpJjdX2cukX
SSyIeNvGnfys1Fzm0b/brMaQd7mRuYY95DSe8q93IeYQj6k/fAYXqMLJQ9N66Kk
eV5w3qIy61UqLxXqk7vwMrzB8rqtMLauu1Jn1LwFR12HInu1hsg6pTzUR/tDsbC8
d3rAz8J+3hKSEu+L///5TfM=
-----END PRIVATE KEY-----

```

Ici on constate que les fichiers “clepriv.pem” et clepub.pem contiennent des données non lisible directement. on distingue une clé privé et une clé publique grâce aux indications présentes dans les fichiers.

Maintenant nous allons chiffrer le fichier “chiffre_asym.txt” avec la clé publique pour cela saisissez la commande “openssl pkeyutl -encrypt -pubin -inkey clepub.pm -in chiffre_asym.txt -out chiffre_asym.enc”.

1. -pkeyutl : Permet de réaliser des opérations sur des clés asymétrique
2. -encrypt : Spécifie le mode chiffrement. Le fichier entrée sera chiffré avec la clé publique.
3. -pubin : Indique que la clé fournie après -inkey est une clé publique (bien que par défaut, pkeyutl attend une clé privée)
4. -inkey clepub.pm : Cela indique le fichier contenant la clé (ici la clé publique clepub.pem) utilisé pour l’opération
5. -in : indique le fichier d’entrée à chiffrer
6. -out : indique le fichier de sortie qui contiendra le résultat du chiffrement (données chiffrées)

Enfaite cette commande permet de chiffrer le fichier chiffre_asym.txt avec la clé publique clepub.pem et produit le fichier chiffré chiffre_asym.enc. **Seul le détenteur de la clé privé correspondante pourra déchiffrer ce fichier.**

Supprimons le fichier chiffre_asym.txt, pour cela saisissez la commande “rm chiffre_asym.txt”

Maintenant nous allons déchiffrer le fichier “chiffre_asym.enc car , comme je l’ai dit **“le détenteur de la clé privé correspondante pourra déchiffrer ce fichier”**. Pour cela saisissez la commande “openssl pkeyutl -decrypt -inkey clepriv.pm -in chiffre_asym.enc -out chiffre_asym.txt”.

1. **-pkeyutil** : Permet de réaliser des opérations sur des clés asymétrique
2. **-decrypt** : indique que l'opération à réaliser est un déchiffrement (restauration du clair à partir du chiffré)
3. **-inkey clepriv.pem** : chemin du fichier contenant la clé privé. Contrairement au chiffrement, le déchiffrement asymétrique nécessite la clé privé
4. **-in** : indique le fichier d'entrée à chiffrer
5. **-out** : indique le fichier de sortie qui contiendra le résultat du chiffrement (données chiffrées)

Maintenant éditons le fichier "nano chiffre_asym.txt"

Contenue du fichiers chiffre_asym.txt



```
GNU nano 7.2      chiffre_asym.txt
HelloWorld ! Ceci est un test de chiffrement asymétrique
```

Etape 4 : OpenSSL et le hachage (fonction de hachage)

Pour commencer cette étape 4, faut ce rappeler que les fonctions de SHA servent à calculer une empreinte cryptographique/numérique d'un fichier ou d'un texte.

Pour mettre en place les fonctions de SHA nous allons tout d'abord créer un fichier texte que nous allons appeler "hachage.txt" et qui aura pour contenu « HelloWorld ! Ceci est un hash. », (n'oubliez pas le point à la fin de la phrase, nous en aurons besoin ultérieurement). Saisissez la commande "echo "HelloWorld ! Ceci est un hash. " > hachage.txt.

Après avoir créé le fichier texte nous allons créer un condensat du fichier "hachage.txt" et ensuite redirigez le résultat dans un fichier que nous allons appelé "hash.txt" pour cela saisissez la commande "openssl dgst -sha256 hachage.txt > hash.txt".

1. **dgst** : sous-commande pour les fonctions de hachage (digest = condensat/empreinte). Elle calcule une empreinte numérique d'un fichier
2. **-sha256** : Spécifie l'algorithme de hachage à utiliser (SHA-256)
3. **hachage.txt** : représente le fichier d'entrée, le fichier dont on veut calculer l'empreinte numérique. la commande va lire son contenu
4. **>** : l'Opérateur de redirection (un seul >). Il crée un nouveau fichier (ou écrase le fichier existant) et y écrit le résultat de la commande au lieu de l'afficher à l'écran.
5. **hash.txt** : fichier de sortie, c'est le fichier dans lequel le résultat (l'empreinte) est enregistré

Pour vérifiez le contenu du fichier hash.txt saisissez la commande "cat hash.txt".

Contenu du fichiers hash.txt

```
root@tepson:/home/tepson# cat hash.txt
SHA2-256(hachage.txt)= ab2aaeadb463b9ffec21eb5bf381dacdab92951c3361c1b8db7b961da94425f6
```

*Contenu du fichier hash.txt (l'empreinte originale)

Maintenant nous allons modifier le fichier "hachage.txt" afin de supprimer le point à la fin de la phrase du contenu

```
GNU nano 7.2 hachage.txt
HelloWorld ! Ceci est un bash
```

Après avoir enlevé le point, nous allons créer à nouveau un condensat du fichier hachage.txt et ainsi le redirigez, en complément. pour cela saisissez la commande "openssl dgst -sha256 hachage.txt >> hash.txt

1. **dgst** : sous-commande pour les fonctions de hachage (digest = condensat/empreinte). Elle calcule une empreinte numérique d'un fichier
2. **-sha256** : Spécifie l'algorithme de hachage à utiliser (SHA-256)

3. **hachage.txt** : représente le fichier d'entrée, le fichier dont on veut calculer l'empreinte numérique. la commande va lire son contenu
4. **>>** : Contrairement à la 1er commande avec (un seul >), avec deux >> sa ajoute (ne remplace pas) le résultat à la fin du fichier existant. Et si le fichier n'existe pas, il le crée
5. **hash.txt** : fichier de sortie, c'est le fichier dans lequel le résultat (l'empreinte) est enregistré

```
root@tepson:/home/tepson# cat hash.txt
SHA2-256(hachage.txt)= ab2aaeadb463b9ffec21eb5bf381dacdab92951c3361c1b8db7b961da94425f6
SHA2-256(hachage.txt)= 2706147d6c31a318829f73a2690af3147f7264e95a2573772447cec9ef3e6066
root@tepson:/home/tepson#
```

On constate que le fichier hash.txt contient des empreintes différentes après modification de hachage.txt. Cela démontre que toute altération, même minime (suppression d'un point), produit un hash totalement différent. Les fonctions de hachage sont donc fiables pour vérifier l'intégrité des données.

Maintenant nous allons de nouveau modifier le contenu du fichier "hachage.txt" en rajoutant le point et ainsi ressaisir la commande "openssl dgst -sha256 hachage.txt >> hash.txt"

Vérifions le contenu du fichier hash.txt en saisissant la commande "cat hash.txt"

```
root@tepson:/home/tepson# openssl dgst -sha256 hachage.txt >> hash.txt
root@tepson:/home/tepson# cat hash.txt
SHA2-256(hachage.txt)= ab2aaeadb463b9ffec21eb5bf381dacdab92951c3361c1b8db7b961da94425f6
SHA2-256(hachage.txt)= 2706147d6c31a318829f73a2690af3147f7264e95a2573772447cec9ef3e6066
SHA2-256(hachage.txt)= ab2aaeadb463b9ffec21eb5bf381dacdab92951c3361c1b8db7b961da94425f6
```

On conclut que les fonctions de hachage comme SHA-256 produisent une empreinte unique pour chaque fichier. Toute modification, même infime, change complètement cette empreinte. Si le fichier revient à son état original, l'empreinte redevient identique. Le hachage permet donc de garantir l'intégrité des données.

Etape 5 : Réalisation d'une infrastructure à clé publique (PKI) personnalisée à partir de OpenSSL

A. Création de l'arborescence de répertoires et du fichier "openssl.cnf" pour la configuration d'une PKI personnalisée

Pour commencer cette étape qui consiste à la réalisation d'une infrastructure à clé publique personnalisée à partir de Openssl, nous allons mettre en place une arborescence.

Pour cela saisissez ses commande:

- `sudo su`
- `cd`
- `mkdir -p ~/pki/{certs,crl,newcerts,private}`
- `chmod 700 ~/pki/private`
- `touch ~/pki/index.txt`
- `echo 1000 > ~/pki/serial`

Résultat attendu :

```
[root@tepson:~/pki# ls
certs  crl  index.txt  newcerts  openssl.cnf  private  serial
```

*liste des fichiers qui se trouve dans le répertoire "/root/pki"

1. **certs:** Va stocker les certificats, comme le certificat de l'autorité racine, certificat du serveur signé par la root CA
2. **crl :** Va Contenir les listes de révocation de certificats, il va lister les certificats qui ont été révoqués avant leur date d'expiration
3. **index.txt:** Va contenir une base de données textuelle qui enregistre l'état de chaque certificat émis par la CA
4. **newcerts:** Va stocker une copie automatique de chaque nouveau certificat émis par la CA
5. **openssl.cnf :** Va contenir le fichier de configuration personnalisé pour OpenSSL. Il définit les politiques, extensions, emplacements et comportement de la CA
6. **private :** Héberge les clés privées (CA racine, serveur, etc.)
7. **serial :** Va contenir le prochain numéros de série à attribuer à un certificat. OpenSSL lit ce fichier, l'incrémente et l'écrit après chaque émission.

Après avoir expliquer le role et le contenu de chaque répertoire bien que personnellement je l'ai déjà fait vous devez créer le fichier "openssl.cnf" qui sera dans l'emplacement "/root/pki".

Dans ce fichier saisissez ses informations:

```
anleygeffry — tepson@tepson: ~ — ssh tepson@192.168.1.26 — 102x54
GNU nano 7.2                                openssl.cnf
[ ca ]
default_ca = CA_default

[ CA_default ]
dir = /root/pki
certs = $dir/certs
crl_dir = $dir/crl
database = $dir/index.txt
new_certs_dir = $dir/newcerts
certificate = $dir/certs/cajse.crt
serial = $dir/serial
private_key = $dir/private/cajse.key
default_days = 3650
default_md = sha256
policy = policy_loose
x509_extensions = v3_ca

[ policy_loose ]
commonName = supplied

[ req ]
default_bits = 4096
default_md = sha256
prompt = no
distinguished_name = dn
x509_extensions = v3_ca

[ dn ]
CN = RootCAJSE

[ v3_ca ]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ v3_server ]
basicConstraints = CA:false
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = jardinssteloi.com
DNS.2 = www.jardinssteloi.com
```

*Capture d'écran du contenu du fichier Openssl.cnf

Petite explication sur chaque section :

1. **[ca]** : Indique quelle section contient la configuration de la CA
2. **[Ca_default]** : Ce sont les paramètres généraux de la CA (chemins, politique, durée...).
3. **[policy_loose]** : Politique de validation des CSR (seul le CN est requis).
4. **[req]** : Ce sont les paramètres pour générer des CSR ou certificats auto-signés.
5. **[dn]** : Distinguished Name (sujet) pour le certificat CA.
6. **[v3_ca]** : Extensions pour un certificat CA (autorité de certification).
7. **[v3_ca]** : Extensions pour un certificat serveur (utilisation TLS).
8. **alt_names]** : Liste des noms alternatifs (SAN) pour le serveur.

B. Création de la clé privée et du certificat de l'autorité racine (RootCA)

Après avoir mis en place l'arborescence de répertoire et u fichier openssl, nous allons créé la clé privée et du certificat de l'autorité racine commençons par saisir la commande "openssl genrsa -out ~/pki/private/cajse.key 4096", qui va permettre de généré une paire de clés RSA et va enregistrer la clé privée dans un fichier (cajse.key).

Rôle de chaque partie :

1. **genrsa** : cette sous- commande permet de générer une clé privée RSA
2. **-out ~/pki/private/cajse.key** : représente le fichier de sortie ou sera stockée la clé privé
3. **4096** : représente la taille de la clé en bits. (plus la taille est grande, plus le chiffrement est sécurisé.

Ensuite saisissez la commande "chmod 600 ~/pki/private/cajse.key", qui va modifier les permissions du fichier cajse.key qui contient la clé privée de l'autorité racine

Role de chaque partie :

1. **6 (propriétaire) : lecteur (4) + écriture (2) = 6** > le propriétaire peut lire et modifier le fichier
2. **0 (groupe) : aucun droite (lecteur, écriture, exécution)**
3. **0 (Autre) : aucun droite (lecteur, écriture, exécution)**

Après avoir crée la clé privée de l'autorité racine, nous allons créez le certificat de l'autorité grace a la commande "openssl req -x509 -new -nodes -key ~/pki/private/cajse.key -sha256 -days 3650 -out ~/pki/certs/cajse.crt -config ~/pki/openssl.cnf".

Role de chaque partie :

1. **req** : sous-commande de gestion des demandes de certificat (CSR)
2. **-x509** : Indique que l'on peut produire un certficiat auto-signé directement
3. **-new** : Génère une nouvelle demande de certificat (obligatoire même pour un certificat auto-signé).
4. **-nodes** : ne chiffre pas la clé privée (la clé privée d'entrée n'est pas protégée par une pass phrase). Permet d'éviter une demande de mot de passe.
5. **-key ~/pki/private/cajse.key** : Spécifie le chemin vers le fichier de la clé privée à associer au certificat (celle générée par genrsa)
6. **-sha256** : Algorithme de hachage utilisé pour signer le certificat
7. **-days 3650** : Durée de validité du certificat en jours. Ici 3650 jours = 10 ans.
8. **-out ~/pki/certs/cajse.crt** : Chemin du fichier de sortie où sera enregistré le certificat généré
9. **-config ~/pki/openssl.cnf** : Utilise le fichier de configuration personnalisée situé dans /root/pki. Ce fichier contient les sections [req] (avec prompt = no), [dn] (définit CN =

RootCAJSE), et [v3_ca] (extensions CA)

Pour vérifiez le contenu du certificat du CA saisissez la commande “openssl x509 -in /root/pki/certs/cajse.crt -text -notout| more”

```
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number:
    36:d3:9a:af:00:3b:e7:28:d5:68:1a:d9:e0:93:23:74:92:bc:e8:cc
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: CN = RootCAJSE
  Validity
    Not Before: Apr 14 18:01:39 2026 GMT
    Not After : Apr 11 18:01:39 2036 GMT
  Subject: CN = RootCAJSE
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (4096 bit)
    Modulus:
      00:a1:46:6e:98:3a:2f:38:41:69:0d:43:da:67:fd:
      61:43:08:2c:e8:dc:10:8b:e4:be:52:c4:58:b1:43:
      b3:df:cc:70:79:3a:37:08:4b:d0:14:94:b0:1c:bc:
      a0:8e:91:95:73:d5:0b:41:cd:31:14:4a:5a:c8:b2:
      02:55:6b:5e:5a:a9:65:75:be:63:01:57:fc:45:73:
      eb:57:f0:63:de:fb:56:c1:53:4e:ad:28:af:f4:47:
      d4:6a:e7:0a:0a:5d:92:3a:9d:76:f5:31:5f:4d:54:
      f5:8a:af:c4:23:b3:bd:1b:a5:b8:29:83:03:5e:2c:
      10:bb:c4:aa:f8:c7:2b:3c:28:3d:2b:cb:0a:a5:51:
      bc:23:b4:2f:58:15:c3:a4:26:db:aa:6e:46:18:10:
      37:7c:cc:4a:35:85:98:12:b8:c5:73:3f:86:cb:9a:
      84:cd:49:07:5b:66:a4:07:19:2a:04:7d:0f:e2:ff:
      11:c7:5a:65:8e:3f:6f:cb:05:ee:8a:b4:0b:a4:7e:
      51:9d:81:1b:22:e3:9b:42:b1:73:ed:5a:db:c8:a7:
      13:3c:79:8a:75:b8:f0:3b:a6:61:0b:64:9d:51:0d:
      91:e5:de:b6:87:3a:c3:09:9d:60:80:b5:27:77:6e:
      0c:1e:e4:d7:fd:83:c7:11:f0:82:b6:6c:7b:0b:72:
      67:98:3e:bd:10:a4:ad:ef:ae:88:08:ed:00:5e:46:
      40:b9:c1:d7:7a:e6:71:1f:63:90:c1:0f:3f:73:88:
      3d:c3:8c:aa:e5:1b:97:48:4f:ec:2a:d8:b9:4c:23:
      66:aa:92:83:e4:e0:13:51:e2:ca:51:86:d8:90:6d:
      99:1e:01:5f:ba:6b:bf:df:83:19:4f:81:ad:77:81:
      f1:d4:55:17:3f:9d:ab:e5:a2:c1:10:13:45:e5:2e:
      c0:1d:a4:55:89:fa:03:07:38:80:ad:85:75:09:ba:
      8d:bc:0d:b4:3f:c4:e1:20:17:d8:0b:a9:91:b3:55:
      7e:25:89:ed:03:d6:1b:a3:1f:d4:5e:b7:d5:f8:67:
      70:2b:80:82:cb:32:ae:07:56:1c:08:d3:44:86:e9:
      73:a9:28:d0:c4:67:71:38:bd:b7:59:75:2b:90:47:
      20:58:f6:e9:5a:8d:e7:0f:49:93:ef:fa:ef:62:ef:
      1a:24:8e:79:f0:39:ce:4f:54:83:08:e3:26:2a:88:
      8a:78:52:6f:1d:52:f7:01:79:03:c8:47:54:5d:57:
      9a:a2:bcb:a2:de:48:5c:5e:dc:28:26:0f:f6:0b:
      92:21:49:da:07:6b:c6:3d:6b:b0:a1:eb:6b:fa:ba:
      0d:6d:96:af:45:d3:c8:87:ce:4c:91:c5:e0:c0:d3:
      b7:e4:d7
    Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Subject Key Identifier:
      E8:B9:55:64:76:DA:7A:F7:63:59:97:F4:5C:16:57:42:69:D6:DE:F6
    X509v3 Authority Key Identifier:
      E8:B9:55:64:76:DA:7A:F7:63:59:97:F4:5C:16:57:42:69:D6:DE:F6
    X509v3 Basic Constraints: critical
      CA:TRUE
    X509v3 Key Usage: critical
      Digital Signature, Certificate Sign, CRL Sign
  Signature Algorithm: sha256WithRSAEncryption
  Signature Value:
    18:4e:87:79:e3:57:b2:71:37:c8:ec:c1:d8:91:58:93:26:
```

Signature Value:

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

-----BEGIN CERTIFICATE-----

MIIFGTCCAwGgAwIBAgIUntOarWA5yJVaBrZ4JMjdJK86MmwDQYJKoZIhvcNAQEL
BQAwFDESMBAGA1UEAwJUm9vdENBS1NFMB4XDTI2MDQxNDE4MDEzOV0XDTM2MDQx
MTE4MDEzOV0wFDESMBAGA1UEAwJUm9vdENBS1NFMIICIjANBgkqhkiG9w0BAQEF
AAOCAg8AMIICCgKCAgEAoUzmDovOEFPdUPaZ/1hQwgs6NwQ1+S+UsRYsUOz38xw
eTo3CEvQFJSwHLygpGvC9ULQc8xFEpayLICVwteWq1ldV5jAVf8RXPzV/Bj2vtW
wVN0rShP9FUaqcKC12S0p129TffTvt11q/E17096W4KYMDXiWYu8Sq+McRPCg9
K8sKpVG8I7QvWBXOpCbbqm5GGBA3R8xKNYWErJFcZ+Gy5qEbUmHW6akhxkuBH0P
4v8Rx1p1j9v68XuhrQLpH5RnYEBIuObQzFz7VrbyKcTPHEdbjw00Zhi2SduQ2R
5d62hzrDC21ggLund24MHUTX/YPHEfCCtmx7C3JnmD69EKST766IY08AXKZAuchX
auZkH20Qw08/c449wlyq5RuXSE/sLti5TcNmapK050ATUeLKUbyYK62ZhgFfumu/
34MZ74Gtd4Hx4FUXP52zsa1BEBNFS7AhaRv1foD9zi1ArV40qbNvA20P8TBTfY
O6CRs1V+JYntA9Ybox/UXrfv+6dWk4COyzKuB1YcONNEhu1zqSjDqGdxOL23WkUr
KEcgWPbpNo3nD0mT7/cvYu8aJi558DnOT13SDV0mkoikF3vHVL3AXkdyEdUXVea
oxy64t5IXF7cKCaP9puSIUnaB2y9Wuwoetr+r4NbZavRdP1h85MkcXgmN035Nc
lWEAAaANjMGEWHQYDVR0BBYEF0i5VWR22nz3Y1mX9FwW03p1t72MA8GAIUdEwEB/wQFMAMBAf8wDgYDVR0P
lMBaAF0i5VWR22nz3Y1mX9FwW03p1t72MA8GAIUdEwEB/wQFMAMBAf8wDgYDVR0P
AQH/BAQDAgGGMA8GCScGS1b3DQEBOWIAA4ICAQ8ZG6HeeNXsnE3yOzB2JFYkyau
xZjkyB3Uew4sKP3Wc5IVmvIoPdbwSdlc0WELTFzBT/Y1VMtr2eDcJ+w8k3wWAvO
lW+MjrkU6X+/f+qHydgapBAEKWiIdH9rsmPyEsYvd15UACkmdZTgWFe15K8n234
5Q/p0ktJHxck4foazaBdKoghLpDjFc0q5Yk3nH5MOSHsXaYFWf1sCbOpELL5TxE
9IkVXn8VYu2Hn20nnFAWjThd4+vcxXW38xbmQUVaKFWitY09EAYvFgSbRC3A115
idBY6zCkm1aK611xTE2DQtfCtdeFm5EjcfdnigjSaHHG1kuvqbgZ1ED276HOCH+3
2rv242NHTJSrTSIK7hFemFf/cK090dtLrIqkEWaZoTRLsbq4WuYDJhhGMPJx+jnB
FtPhGfH4zBv8wmnvKSGxp4DU7cbR5waqGY/MQqkyfR1bh59pIOSwL4u5vN2kBi8
g0NNB+r+7wIpp7wkyMhCcgFdpWv+RJK4KTvUOwoasTONbArHfxspZfvc+eqECYqx
byP40oeg0Ymg0P0p4ghaF/BjsUyOf5Pt dx8DD01fnxRzUQLGSYg6Jo1wIRfZ1nEE
+iwofAvLTtFpRIQDMnSBnnW6Nf288kmMIXbdoKCPwRdR2ZZZxMjvqJDCx9Suyt3
rvAqWevHHg8FWef33w==

-----END CERTIFICATE-----

C. création d'une clé privée du CA Racine, et d'un certificat serveur pour "srvlinux" signé cette même CA Racine (Root CA)

Pour commencer la partie C de l'étape 5, Nous allons tout d'abord commencer par générer une clé privée pour le serveur est Linux pour cela saisissez la commande "openssl genrsa -out ~/pki/private/srvlinux.key 2048.

Après avoir généré la clé privée, nous allons créer une demande de signature de certificat en format CSR pour cela saisissez la commande, "openssl req -new -key ~/pki/private/srvlinux.key -out ~/pki/certs/srvlinux.csr" et remplissez les informations suivante selon vos besoins:

Mon certificat :

1. **Pays :** FR
2. **Etat/province :** Guadeloupe
3. **Localité :** Les Abymes
4. **Organisation/entreprise :** Jardins de StEloi
5. **Section :** Entreprise
6. **Nom commun :** www.jardinssteloi.com
7. **Email :** m.bourgeois@jardinssteloi.com

Après quoi nous allons signer le certificat serveur avec la clé privé du CA pour cela saisissez la commande "openssl ca -config ~/pki/openssl.cnf -extensions v3_server -in ~/pki/certs/srvlinux.csr -out ~/pki/certs/srvlinux.crt"

Si la commande précédente ne fonctionne pas, ajoutez à la fin de la commande "2>&1", faites y deux fois.

1. **ca :** Sous-commande qui agit comme une autorité de certification (elle vérifie et la demande et signe le certificat)
2. **-config ~/pki/openssl.cnf :** Spécifie le fichier de configuration personnalisé de notre PKI (Elle contient les paramètres de la CA)
3. **-extensions v3_server :** ici, on utilise la section nommée du (v3_server) fichier de configuration pour ajouter les extensions X.509 approprié à un certificat serveur.
4. **-in ~/pki/certs/srvlinux.csr :** Représente le fichier d'entrée contenant la demande de signature de certificat (CSR)
5. **-out ~/pki/certs/srvlinux.crt :** Représente le fichier de sortie où sera enregistré le certificat signé

Ici cette commande va permettre de signer une demande de certificat (CSR) avec l'autorité de certification (CA) que vous que nous avons créé, afin de produire un certificat serveur valide pour le pour le site de jardin sainteloi (www.jardinssteloi.com)

```
[root@tepson:~/pki# cat /root/pki/serial
1001
[root@tepson:~/pki# cat /root/pki/index.txt
V          360412214012Z          1000      unknown /CN=www.jardinssteloi.com
root@tepson:~/pki# █
```

***Contenue des fichier serial et index.txt**

Ici le fichier **SERIAL** contient 1001. Ce fichier indique le prochain numéro de série qui sera attribué au prochain certificat émis par la CA. Dans ce cas, le dernier certificat émis (celui du serveur) a reçu le numéro 1000 après cette émission on peut Open automatiquement incrémenter le compteur à 1001, pour le certificat suivant.

Dans le fichier index .txt, il s'agit de la base de données textuel des certificats émis de la CA. Chaque ligne correspond à un certificat. Ici une seule ligne est présente:

1. **V** : Certificat valide = V (R = Revoked // E = Expired)
2. **360412214012Z** : Date d'expiration (36 = 2036 // 04 = avril // 12 /// Heure 21:40:12) Soit 10 ans après la signature car, dans les précédente étape nous avosn noté (-days 3650)
3. **1000** : Le numéro de série. Le certificat porte le numéro de série 1000
4. **unknow** : le certificat n'est pas révoqué
5. **CN=www.jardinssteloi.com** : Ceci est le sujet du certificat (Le Common Name, correspondant au serveur web)

D. Installation du certificat de l'autorité de certification (CA) sur le serveur « srvlinux » basé sous Linux Debian

Maintenant nous allons installer le certificat de l'autorité de Certification sur le serveur linux, ainsi pour que le système d'exploitation fasse confiance à notre autorité de certification, nous allons saisir la commande "sudo cp ~/pki/certs/cajse.crt /usr/local/share/ca-certificates/", afin d'installer le certificat RootCAJSE sur le serveur Linux.

observons le fichiers "cajse.pem" qui ce trouve dans l'emplacement "/etc/ssk/certs", pour clea saissisez la commande "ls -l /etc/ssl/certs/ | grep cajse"

```
[root@tepson:~/pki# ls -l /etc/ssl/certs/ | grep cajse
root@tepson:~/pki# █
```

Ici ont constate que le fichier cajse.Pem n'y pas. Pour y remédier saissisez la commande "sudo update-ca-certificates" afin de mettre à jour la liste des autorités de certifications présents sur le serveur linux

```
[root@tepson:~/pki# ls -l /etc/ssl/certs/ | grep cajse
lrwxrwxrwx 1 root root      9 Apr 16 01:57 7fc50a5a.0 -> cajse.pem
lrwxrwxrwx 1 root root    42 Apr 16 01:57 cajse.pem -> /usr/local/share/ca-certificates/cajse.crt
root@tepson:~/pki# █
```

Ont peut voir qu'il est présent, après avoir saissi la commande pour y remédier.

La génération du même certificat à été généré au format PEM car c'est un format texte standard qui permet au système de le reconnaître comme autorité de confiance

Etape 6 : sécurisation du site “Les jardinsSaint-Eloi”, basé sous le service web Apache2 , avec les éléments et votre PI et des protocoles SSL/TLS

Maintenant nous allons sécuriser notre site web, mais avant il faudra créer une chaine de confiance entre les certificat pour ce faire tapez la commande "cat ~/pki/certs/srvlinux.crt ~/pki/certs/cajse.crt > ~/pki/certs/fullchain.crt"

Pour consulter, appréhender et comprendre le contenu du fichier saisissez la commande `cat -n /root/pki/certs/fullchain.crt | more`. On constate que le fichier `fullchain.crt` contient deux certificats (deux blocs PEM) :

1. **Premier bloc : certificat du serveur (srvlinux.crt)**
2. **Second bloc : certificat de l'autorité racine (cajse.crt)**

```
Issuer: CN=RootCAJSE
Validity
    Not Before: Apr 15 21:40:12 2026 GMT
    Not After : Apr 12 21:40:12 2036 GMT
Subject: CN=www.jardinssteloi.com
```

```

1-----BEGIN CERTIFICATE-----
2MIIEVjCCAg16gAwIBAgIACEAAwDQYJKoZIhvcNAQELBQwFDESMBAGA1UEAwu3Um9v
3dENBS1NFBM4XDTI2MDQxNTIiXDAxMj0xMTI1IjBjAGBgkqhG9wBAQFAQCAQ8A
4AwBwD3d3LmPhcmRpbmNzdGVsb2kuY29tMTI1IjBjAGBgkqhG9wBAQFAQCAQ8A
5MIIBGKCAQEA2qdy4YXpNG35kHEfIDB1KMS2tqbyaocPUBAg/RJnvcVfY1SEF3+E
6BkJdKj5A5kadwtRlrlr1HIq0smC21j/UfVLzscFX2LkQKKB6C3F7pS9gzG7Bwj1JKFw
7M3IdKjY1Iaw3Q80jk3Z/RftYOnmPQZ2YbajZfFLbVt+oY2aZkK+bPPVbWjJqO
8gQN+alPn5qaYeI3gn79r/XsbvJ82w4JBuKf7+BiZLh/5EfCe3wVc2im1Aw0A4ZLc
992npC34rtPgk4o0TbEJcA6sGIhf0mZkc2u7FVLbVt+oY2aZkK+bPPVbWjJqO
10XSGSf6vBie2PX61p1lrNrvPLBYQw5jZFQ1DAQAB04IMGI6MAkGA1UEQmQCAAAw
11CWYDR9PBPAQdAgwMBMGA1UDJQMMQAAGCCGAgEwBwMDMGA1UEQ0Q3MzCwPeph
12cmRpbmNzdGVsb2kuY29tY3d3cuamFvZGluZD02W2xvvaas5j2b0WQYDR00BBYE
13FP88ab2Aeig1yOQ+U0uovYjHUBMBGA1UDJ0WYBMAAF0I5VWR22nr3Y1mx9FwW
14V03p1t72MA0GCSqGSIb3DQEBChUA41CAQAQD1kIhXGwKq08e1Af056j2w1UZE
1561nwFwP1WH9SSXSjYVWY48C31ccp03rhzbd1UjXwnkwoLLd1XCF96V/PlmT1e1r
16fZU08x2mfD0WU5a59oP3UjzV15jikyEKF2K3s41jyB815rzm0APNZLxp1RH
171179sbvdf0wJD2j+M0p1Zx3BFLQY5zme0k0eKvPPUAg/RJmOLuY0bKul/38FA
18aDx0XHaU2lb9y1E5QL/MIPq20SQFMl00e1onF37+9yNf081kffe70C2zeVJvr4P
19uCBV148wKH0e10AB4t1pU9yNxy4ykj7fAo7B3J28WGM5wJ+1Jdnf5SursrUH5MQZ2
20LQWPH20BKUkHwRiH9RFuiT3yBqIh+2IGe/y8y3KcZ/FOAcUWmPjPRFFXQCp
21k+he8BMk0a9Xo+S0mK13n5p1Q1t4BtdNryKzw15p1L21EFCR0F33cun+FA5nBw
22n0e5AD91ilzEtNpaM39aeEZP27f7cdwpzr7iG0VTFwMDnvobvz7D4XGROCPwbk
231SKTjP0i0ib+u/xMyVsVh+6A/YXDncPQK8j6wDECk1xqr0N9GS0I46174VIKUdwv
24567v7p1FDVT/6oS0ror9WqHeskebl0rIkt+FL4EYsf984vmvVad5U5iGQ4Yt9K
25KGeEnXyPUJucg==
26-----END CERTIFICATE-----
27-----BEGIN CERTIFICATE-----
28MIIFGTCCAgwGAwIBAgIUNT0arwA75yJvVaBrZ4JmJdJK86MwDQYJKoZIhvcNAQEL
29BQwFDESMBAGA1UEAwu3Um9vY29tMTI1IjBjAGBgkqhG9wBAQFAQCAQ8A
30ME4MD2eZ0VFDSEMBAGA1UEAwu3Um9vY29tMTI1IjBjAGBgkqhG9wBAQFAQ
31AAOCAGAMIIICQKCAQEA6UzUmDovOEFpDUPaZ7l4Wqps6wQi+S4gY9YU0238Xw
32O3CEvQFJ3SwHlygjpGv9UL0ce8XfEApLjCVWteWqllldv5E1vAF8RPv/Bj4Vjw
33vN0rShp9FfUaqqK12S0p129TFtVt11j/ET7096q64KYMDm1yVf8Sq+McrPCg9
34K85gPvG8I7QvWBXdpCbbqm5GGBA3R8XKNYVeyfJcz+Gy5eBulmHw8zhXkuUBH0
354vBRk1plj9jv68XuhrPLH5RnIYU0b0rfZ7vbyKcTPHMEdbjW0aHk21SduQDQ1
36562hzrDCZ1ggLUnd24MHuTX/YPHFEc0ctmx7C33nmd6EKSt5k671Y00AXkZuXCHX
37euzZ0wQ08/c49w4yq5RuxSE/slt512CNmapK50A0tLU6yVkg62GZFfumu/
3834MZT4Gtd4Hx1FUXP62r5aLBEBNFd5HArVf0d8Z1ArY1VcbqNua2P8BTh1BfY
39G6C8rs1+YJntA9Ybox/UXrFv+GdwK4CYzKwB1rCCNNEuhdlzSG5jGdxGL23WXRu
40kEcqGPbpw03nD0mT7/rvYU8aJ1558nd01tSDY0Wm0aKofJEFJ3W1L3AXkdyUdXvea
41ory64t51XF7cKCaP9xjUlnaB2vGPWUuoetr+Rn4GzvdrRtInH5mK6Q9u35NCn
42M5EAaAnjMGewHqYDR00BBYEFOI5VWR22nr3Y1mx9FwW03p1t72MB88A1UDJ0WY
43BMAAF0I5VWR22nr3Y1mx9FwW03p1t72MB88A1UDJ0WYBMAAF0I5VWR22nr3Y1mx9
44AQH/BAQDAQdAgwK0GCSqGSIb3DQEBChUA41CAQAQD1kIhXGwKq08e1Af056j2w1UZE
45Xqj3V8UeW4GKpA59oP3UjzV15jikyEKF2K3s41jyB815rzm0APNZLxp1RH
46W+MjkrU6X+/f+qHydgdpBAEkiid49rsmPyZsYvd15U4CKmcdZ7GfWf1E5K8n234
475/p0k7JHzxc4foazbDKoghLPdjCfQo5Yk3N5H05XSH5Yaf15CbofSBL15tXE
489IKV8nBVYU2Hn20nnFAWjThd4+vcxXW38xbmQ0uKvHf1y09EAYVf8cmSBR3A115
49IdB8y6Ckm1aK611XTE2DQ2fCtDeFm5EjcfDn3jSAHhG1kuvqgqZ1ED276HOHC+3
502rv24AD1NJTJSrTSIK7hFemFj/Ck090DtlrIdgKEwAZoTLR35WuYDjJhGMpJz+jNB
51FtPHn4Z4v8B0wmvVZKSGsp4DU7cbr5wqG/YMQqkfYrlb1sh59P10SwL4f5w2Kb18
52qPNB+r+7W1pp7wkyMHcGfFdpWv+RJK4TtUv0oas+T0NABrH3ScZfVc+eqCYqX
53byP40eeg0Ymg0P0p4ghaF/8jsUyOf5Ptd3BD01fnxRZQLG5G6JoJIDrfZlNEe
54+iwoFAVLTtFpRIQDMnSbnW6Nf288kmX1MBX5d0sCPwR22ZzXmkwYJDR39SUYt3
55rvAQWvWHq8FWEf3W==
56-----END CERTIFICATE-----

```

***Contenu du fichier "fullcahin.crt" ont remarque les deux certificats**

Maintenant nous allons configurer le “VirtualHost” pour que le service Apache2 prend en compte la clé privé et du certificat x509 du serveur linux. Pour cela saisissez la commande “nano /etc/apache2/sites-available/sainteloi.conf”

```
<VirtualHost 192.168.4.2:443>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName www.example.com

    ServerName jardinsainteloi.com
    ServerAlias www.jardinsainteloi.com
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/sainteloi

    <Directory /var/www/sainteloi>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/sainteloi/error.log
    CustomLog ${APACHE_LOG_DIR}/sainteloi/access.log combined

    SSLEngine on
    SSLProtocol TLSv1 TLSv1.1 TLSv1.2 TLSv1.3
    SSLCertificateFile /root/pki/certs/fullchain.crt
    SSLCertificateKeyFile /root/pki/private/srvlinux.key
    SSLCipherSuite HIGH:!MD5:!NULL
    SSLHonorCipherOrder on

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf
</VirtualHost>
```

*Contenu du fichier sainteloi.conf

Les chose a modifier à ajouter :

1. Remplacer le port 80 part port 443
2. Ensuite ajouter cette partie :

SSLEngine on

SSLProtocol TLSv1 TLSv1.1 TLSv1.2 TLSv1.3

SSLCertificateFile /root/pki/certs/fullchain.crt

SSLCertificateKeyFile /root/pki/private/srvlinux.key

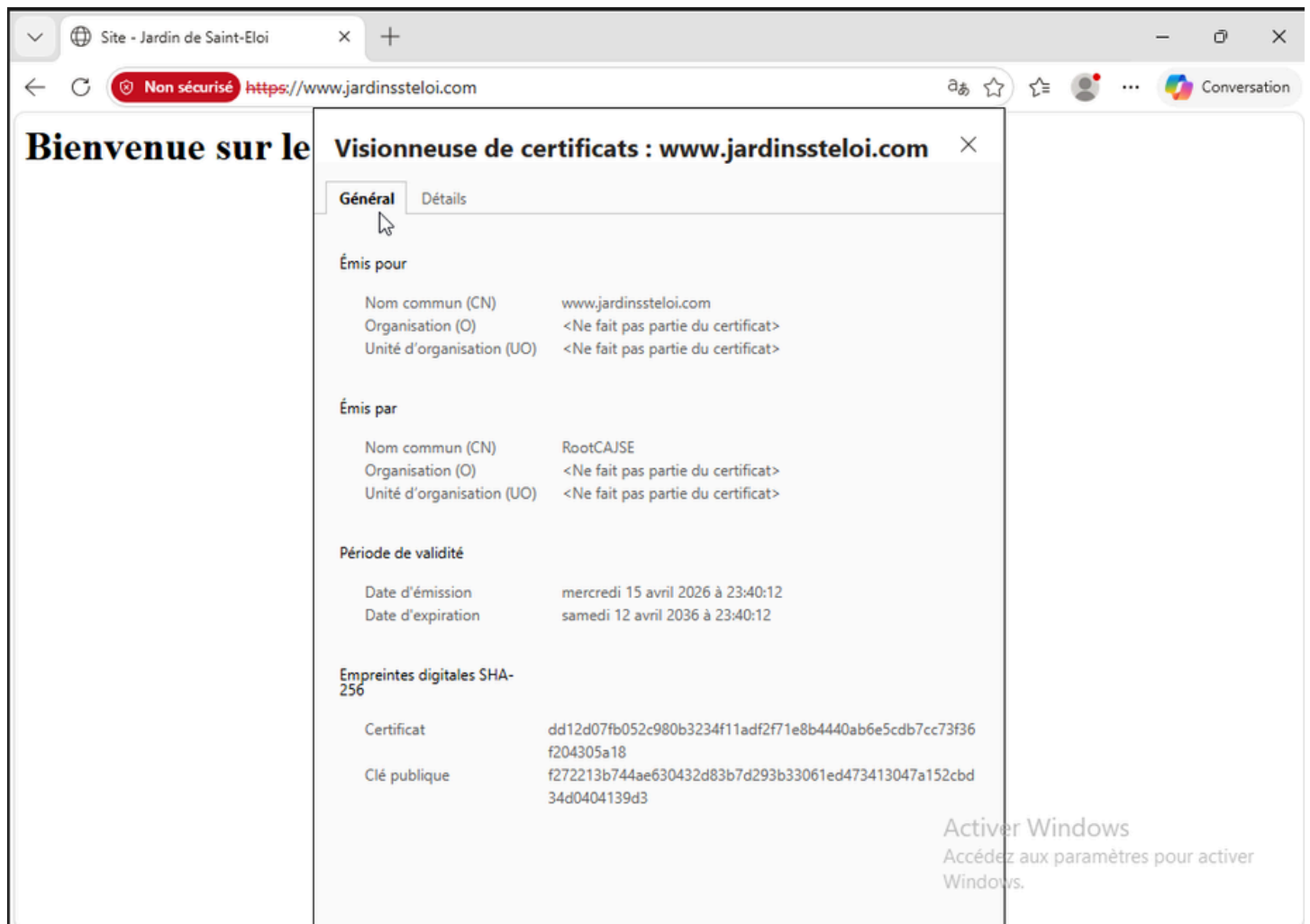
SSLCipherSuite HIGH:!MD5:!NULL

SSLHonorCipherOrder on

role des paramètres de la valeurs “SSLCipherSuite”

1. **HiGH** : va lister les algorithmes de chiffrement fort
2. **!MD5** : va exclure tous les algorithmes MD5 (Fonction de hachage vulnérable)
3. **!NULL** : va exclure les algorithmes suites sans chiffrement

Maintenant activons le module SSL en saisisant la commande “**sudo a2enmod ssl**” puis relancez le service apache2 en saisisant la commande “**systemctl restart apache2**”. n’oubliez pas de désactiver le site par défaut et activez le site jardinsaint, pour cela saisissez la commande “ **sudo a2dissite 000-default.conf** puis **sudo a2ensite sainteloi.conf**” puis relancez le service apache2



Etape 7 : Test du dispositif de sécurisation du site Web, “jardin de Saint Éloi”, gérer par OpenSSL et basé sur le service apache deux

Pour réaliser cette dernière étape et ainsi récupérer le certificat de l'autorité de certification racine “RootCAJSE” et qui représente le certificat numérique du CA “cajse.crt” et ainsi l'intégrer dans le navigateur Web de notre poste client, il existe une procédure toute simple.

Nous allons utiliser le module `http.server` de python pour mettre à disposition des fichiers via le port, 8000.

Pour cela déplacez vous depuis le serveur linux dans le répertoire “/root/pki/certs” en saisisant la commande “`cd /root/pki/certs`”. Puis saisissez la commande “`python3 -m http.server 8000`”.

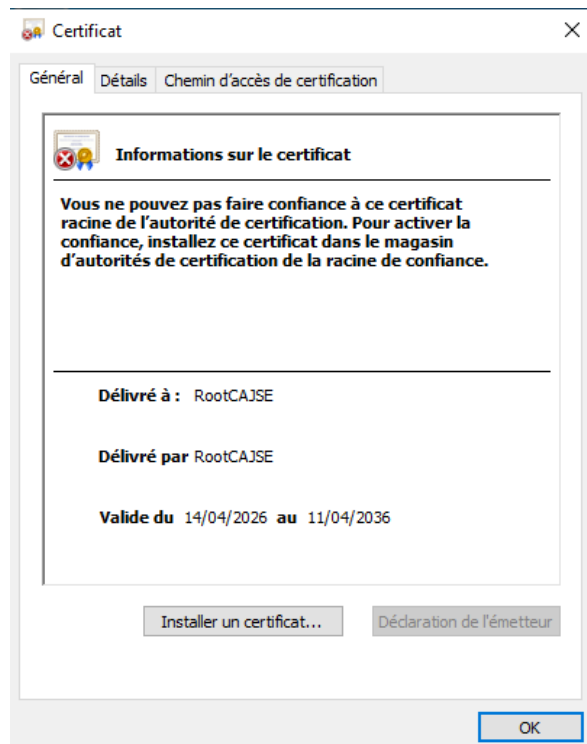
Maintant allons sur le poste client et ouvrez votre navigateur et tapez sesi “<http://192.168.4.2:8000/cajse.crt>”

Ceci télécharge le fichier certificat “cajse.crt”



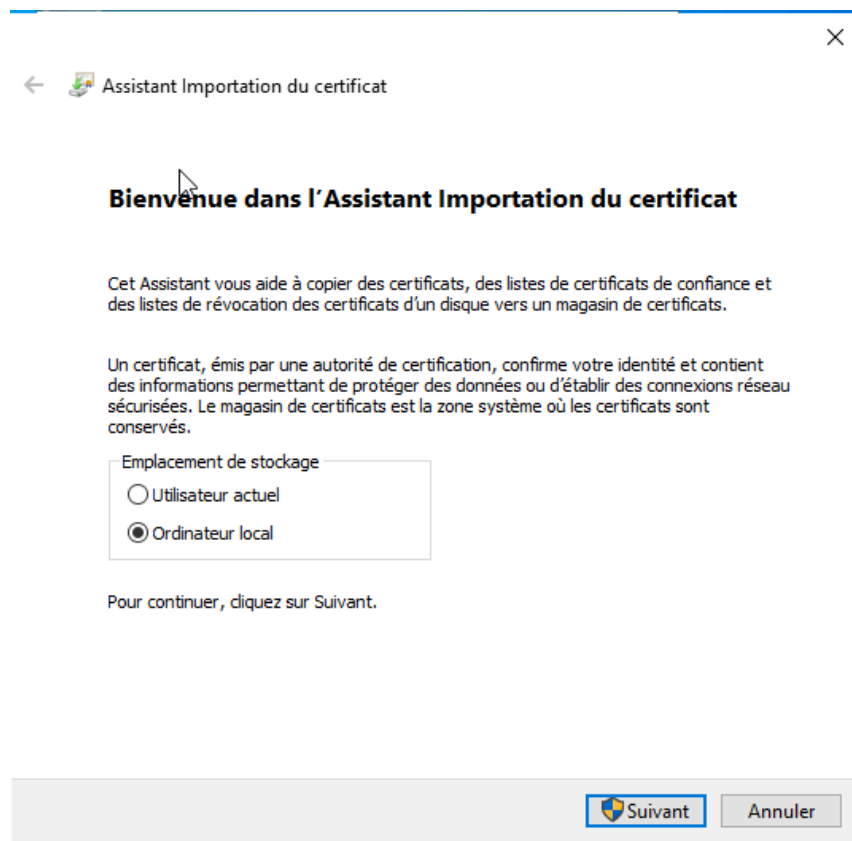
*J'ai déplacé mon certificat sur le bureau

Ouvrez le



Ont constate que le navigateur ne reconnait pas RootCAJSE, elle ne peut faire confiance à ce certificat.

Pour cela cliquer sur “Installer un certificat” puis “ok” puis cocher sur “ordinateur local”



L'installation sur l'ordinateur local permet à tout les utilisateur sur la meme machine d'accéder au site jardin saint eloi avec un certificat valide. Selectionner suivant.



  Assistant Importation du certificat

Magasin de certificats
Les magasins de certificats sont des zones système où les certificats sont conservés.

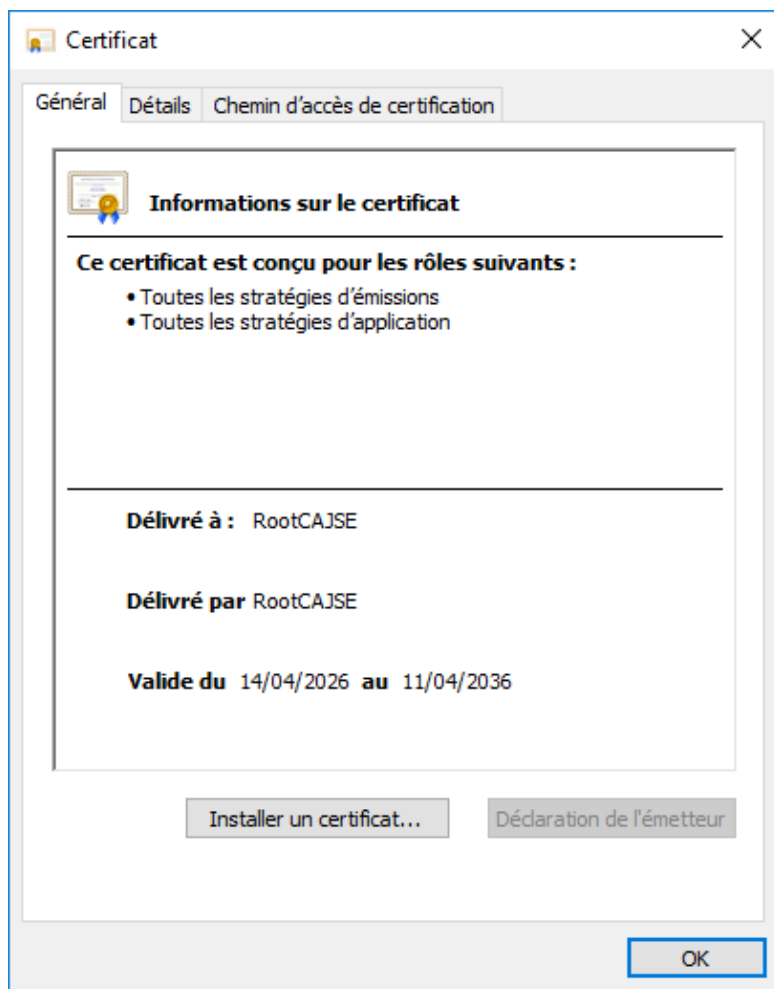
Windows peut sélectionner automatiquement un magasin de certificats, ou vous pouvez spécifier un emplacement pour le certificat.

☐ Sélectionner automatiquement le magasin de certificats en fonction du type de certificat

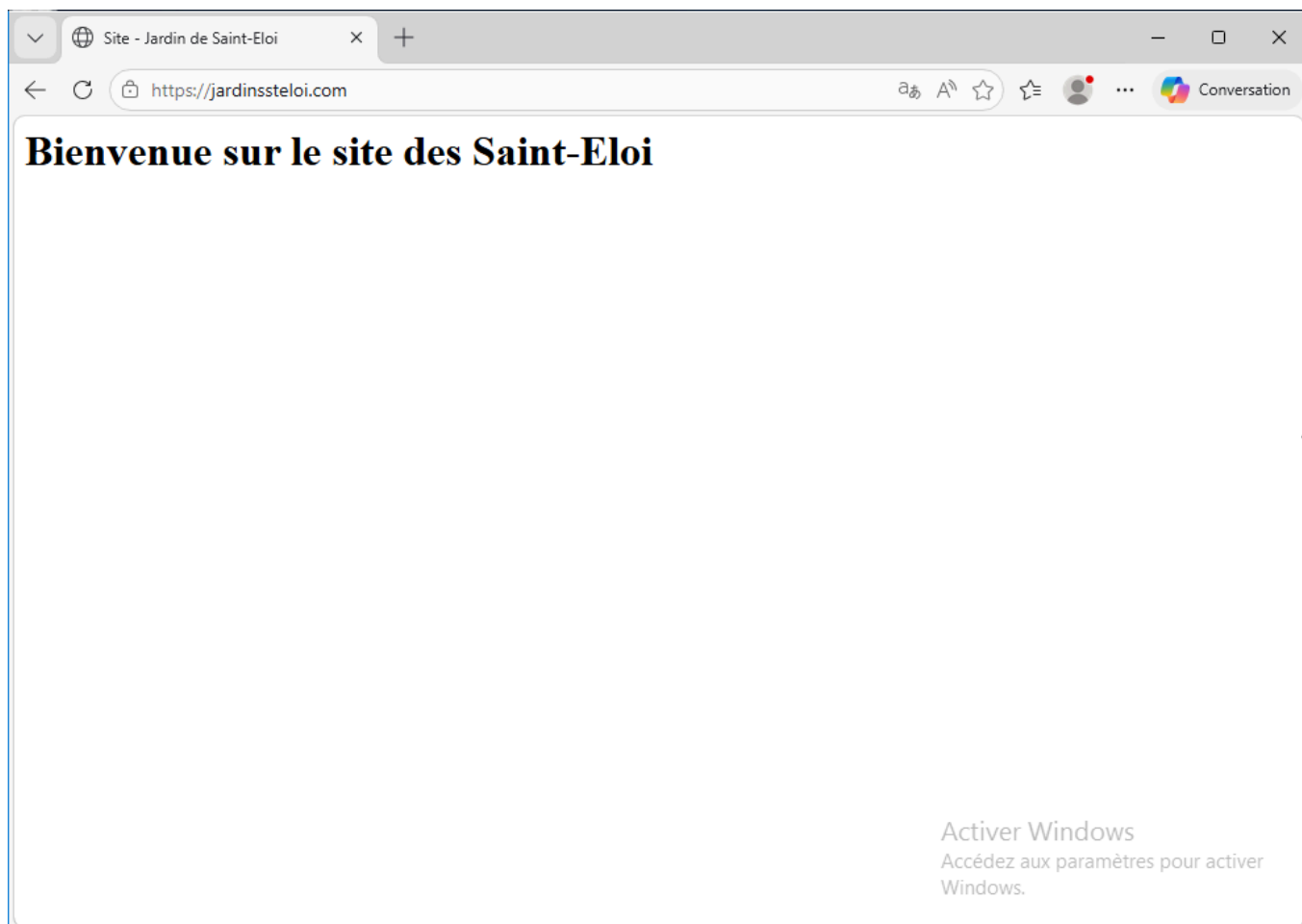
☒ Placer tous les certificats dans le magasin suivant

Magasin de certificats :

Cocher l'option "Placer tous les certificats dnas le magasin suivant" puis faite parcourir, pui sélectionner "Autorités de certification racines de confiance" et faite terminer

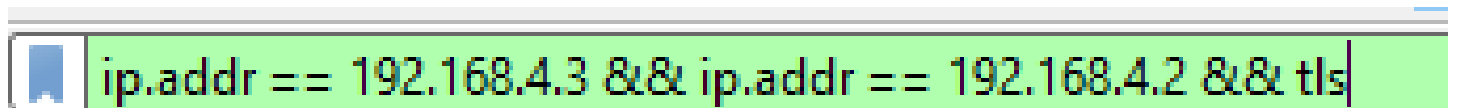


Si on réouvre le certificat on voit que le certificat est valide et fiable et si on redémarre notre navigateur et observez



On constate qu'avant d'importer le certificat "cajse.crt" dans le magasin des autorités racine, le navigateur affichais une erreur de sécurité, car il faisait pas confiance à ce certificat. Après importation, le cadenas ne s'affiche plus, sans avertissement. Le navigateur ne fait confiance qu'aux autorités de certificat présent dans son magasin. en important notre propre certificat, il reconnaît le certificat serveur, signé par cette CA, établi une connexion TLS et valide l'identité du serveur.

Pour réaliser les teste et vérifier que les trames entre le poste client et le seveursoit bien sécurisé, nous allons utilisé wireshark et ainsi analyser si les trames échangé entre le poste client et le serveur et vérifier si ils utilise le protocole TLS, pour cela " ouvrez wireshark et selectionner l'interface sur la quel vous etes connecté à votre réseau ensuite dans la partie "filtre actuel, saisissez le filtre d'affichage qui se trouve dans la capture d'écran



RESULTAT

No.	Time	Source	Destination	Protocol	Length	Info
7	0.009907100	192.168.4.3	192.168.4.2	TLSv1.3	1912	Client Hello (SNI=www.jardinssteloi.com)
9	0.011787500	192.168.4.2	192.168.4.3	TLSv1.3	314	Server Hello, Change Cipher Spec, Application Data, Application ...
10	0.016860400	192.168.4.3	192.168.4.2	TLSv1.3	134	Change Cipher Spec, Application Data
11	0.018146600	192.168.4.2	192.168.4.3	TLSv1.3	133	Application Data
12	0.018303100	192.168.4.3	192.168.4.2	TLSv1.3	911	Application Data
13	0.019358300	192.168.4.2	192.168.4.3	TLSv1.3	622	Application Data
14	0.020669500	192.168.4.3	192.168.4.2	TLSv1.3	1880	Client Hello (SNI=www.jardinssteloi.com)
16	0.022333800	192.168.4.2	192.168.4.3	TLSv1.3	314	Server Hello, Change Cipher Spec, Application Data, Application ...
17	0.027497400	192.168.4.3	192.168.4.2	TLSv1.3	134	Change Cipher Spec, Application Data
19	0.029529200	192.168.4.2	192.168.4.3	TLSv1.3	133	Application Data
21	5.023794300	192.168.4.2	192.168.4.3	TLSv1.3	78	Application Data
35	20.060547600	192.168.4.2	192.168.4.3	TLSv1.3	78	Application Data

***analyse de trames faite sur mon serveur DNS car, les nouvelle version de wireshark ont des problème de compatibilité avec mon poste client**

ont constate que les trames échangé entre le poste ou le serveur DNS et le serveur sont sécurisé car ils utilise le protocole tlsv1.3 et non pas le protocole http.

Lecteur du certificat : www.jardinssteloi.com

Général
Détails

Émis pour

Nom commun (CN)
www.jardinssteloi.com

Organisation (O)
<Ne fait pas partie du certificat>

Unité d'organisation (OU)
<Ne fait pas partie du certificat>

Émis par

Nom commun (CN)
RootCAJSE

Organisation (O)
<Ne fait pas partie du certificat>

Unité d'organisation (OU)
<Ne fait pas partie du certificat>

Durée de validité

Émis le
mercredi 15 avril 2026 à 23:40:12

Expire le
samedi 12 avril 2036 à 23:40:12

Empreintes SHA-256

Certificat
dd12d07fb052c980b3234f11adf2f71e8b4440ab6e5cdb7cc73f36f204305a18

Clé publique
f272213b744ae630432d83b7d293b33061ed473413047a152cbd34d0404139d3

Active
Accède
Windows

Sur le certificat ont peut constater :

Émis pour : Pour qui à tel été émise (notre site www.jardinssteloi.com)

Émeteur : Notre autorité de certification (RootCAJSE)

La date de validité : du 15 avril 2026 au 12 avril 2036 (10 ans)

l'algorithme de signature : SHA-256

L'empreinte SHA-256

La clé publique : RSA 4096 bits

En Conclusion les données échangées sont entièrement sécurisées : la mise en place de la PKI avec OpenSSL, la signature du certificat serveur par notre autorité racine RootCAJSE, la configuration HTTPS d'Apache2 avec des chiffrements forts, et l'import du certificat CA dans le navigateur permettent d'établir une connexion TLS authentifiée, comme en témoigne l'affichage du cadenas vert, garantissant ainsi la confidentialité et l'intégrité des transactions entre les clients et le site web des Jardins de Saint-Eloi.

Au revoir