



# **Université de technologie de Belfort Montbéliard**

---

## *TP2 : HTTPS*

---

Année 2022-2023

UV : RS40

**Présenté par : Saad SBAT**

## Contents

Introduction.....	3
Objectif .....	4
Vérification du serveur http .....	4
Génération du certificat de l'autorité de certification .....	5
Qu'est-ce que le <i>protocole HTTPS</i> ?.....	5
Génération de la clé privée et la clé publique de l'autorité de certificat .....	6
Génération de la clé privée du serveur et la requête CSR.....	8
L'envoi de la certificat au CA par le Serveur .....	10
Connexion https .....	13
<b>NET::ERR_CERT_AUTHORITY_INVALID</b> .....	14
Améliorations.....	15
Page login: .....	15
<b>Hachage du mot de passe</b> .....	16

# HTTP/HTTPS

## Introduction

HTTP signifie « **HyperText Transfer Protocol** ». Ce protocole a été développé par Tim Berners-Lee au CERN (Suisse) avec d'autres concepts qui ont servi de base à la création du World Wide Web : le HTML et l'URI. Alors que le HTML définit comment un site Internet est construit, le HTTP détermine comment la page est transmise du serveur au client. Le troisième concept, l'URL (Uniform Resource Locator), fixe la façon dont une ressource (par exemple un site Internet) doit être adressée sur le Web. Lorsque vous saisissez une adresse Internet dans votre navigateur Web et qu'un site vous est affiché quelques secondes plus tard, cela signifie qu'une communication a été établie entre votre navigateur et le serveur Web via HTTP. On peut donc dire que le HTTP est la langue dans laquelle votre navigateur Web parle au serveur Web afin de lui communiquer ce qui est demandé.

Le fonctionnement du HTTP peut être expliqué très simplement à travers la consultation d'un site Internet :

1. L'utilisateur saisit dans la barre d'adresse de son navigateur Internet *http://example.com/*.
2. Le navigateur envoie une requête correspondante, appelée **requête HTTP**, au serveur Web qui administre le domaine *example.com*. Normalement, cette requête est de type : « Merci de m'envoyer le fichier ». Mais le client peut également se contenter de demander : « As-tu ce fichier ? ».
3. Le serveur Web reçoit la requête HTTP, cherche le fichier désiré (dans l'exemple : la page d'accueil de *example.com*, c'est-à-dire le fichier *index.html*) et envoie dans un premier temps l'**en-tête** qui informe le client à l'origine de la requête du résultat de sa recherche à l'aide d'un code de statut. Vous trouverez des détails concernant les codes de statut dans notre article détaillé.

4. Si le fichier a été trouvé et si le client demande à l'obtenir (c'est-à-dire si le client ne souhaite pas uniquement savoir s'il existe), après l'en-tête, le serveur envoie le **corps du message**, à savoir le contenu à proprement parler. Dans notre exemple, il s'agit du fichier *index.html*.
5. Le navigateur reçoit le fichier et l'affiche sous forme de site Internet.

## Objectif

Nous supposons un club privé qui distribue un mot de passe d'entrée aux adhérents une fois par mois. Ils utilisent ce mot de passe pour accéder au club. Les adhérents obtiennent le mot de passe en se connectant sur un lien URL secret. Le lien leur est communiqué lors de leur dernière rencontre physique. Actuellement, il s'agit simplement d'une connexion http, ce qui permet à toute personne observant le trafic de lire le mot de passe du club. L'objectif du projet est de remplacer la connexion http par une connexion https.

## Vérification du serveur http

L'inconvénient principal de ce type de connexion réside dans le fait qu'elle n'est pas sécurisée : aucun chiffrement des données des utilisateurs n'est opéré.

Dans cet exemple j'ai la requête http qui affiche le message secret « RS40 IS THE BEST UV » qu'on peut le voir facilement en filtrant les messages `tcp.port==8081` en utilisant WireShark

The screenshot shows a network traffic capture on the left and a web browser on the right. The browser displays a message: "The Secret Message: RS40 IS THE BEST UV".

**Network Traffic Capture (Wireshark):**

No.	Time	Source	Destination	Protocol	Length	Info
12	0.609745	127.0.0.1	127.0.0.1	TCP	84	53856 → 8081 [ACK] Seq=871
13	0.610526	127.0.0.1	127.0.0.1	TCP	84	8081 → 53856 [FIN, ACK] Seq=842
14	0.610554	127.0.0.1	127.0.0.1	TCP	84	53856 → 8081 [ACK] Seq=871
15	0.623853	127.0.0.1	127.0.0.1	TCP	84	53856 → 8081 [FIN, ACK] Seq=842
16	0.623932	127.0.0.1	127.0.0.1	TCP	84	8081 → 53856 [ACK] Seq=842
17	45.008985	127.0.0.1	127.0.0.1	TCP	86	[TCP Keep-Alive] 53857 → 8081
18	45.009016	127.0.0.1	127.0.0.1	TCP	108	[TCP Window Update] 8081 → 53856
25	90.012735	127.0.0.1	127.0.0.1	TCP	86	[TCP Keep-Alive] 53857 → 8081
26	90.012762	127.0.0.1	127.0.0.1	TCP	108	[TCP Keep-Alive ACK] 8081 → 53856
31	135.022140	127.0.0.1	127.0.0.1	TCP	86	[TCP Keep-Alive] 53857 → 8081
32	135.022172	127.0.0.1	127.0.0.1	TCP	108	[TCP Keep-Alive ACK] 8081 → 53856

**HTTP Response (HTML):**

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">\n
<meta name="viewport" content="width=device-width, initial-scale=1.0">\n
<title>Document</title>\n
</head>\n
<body style="display: flex; flex-direction: column; background: linear-gradient(#e66465, #9198e5); height: 40px; margin: auto; border-radius: 8px; color: white; height: 40px; text-align: center; font-family: sans-serif;">\n
  <div style="border: 1px solid white; margin: auto; border-radius: 8px; color: white; height: 40px; text-align: center; font-family: sans-serif;">\n
    <span style="font-size: 2em; font-weight: bold;">The Secret Message:</span> \n
    <span style="font-size: 2em; color: rgb(255, 0, 0);"> RS40 IS THE BEST UV</span>\n
  </div>\n
</body>\n
</html>
```

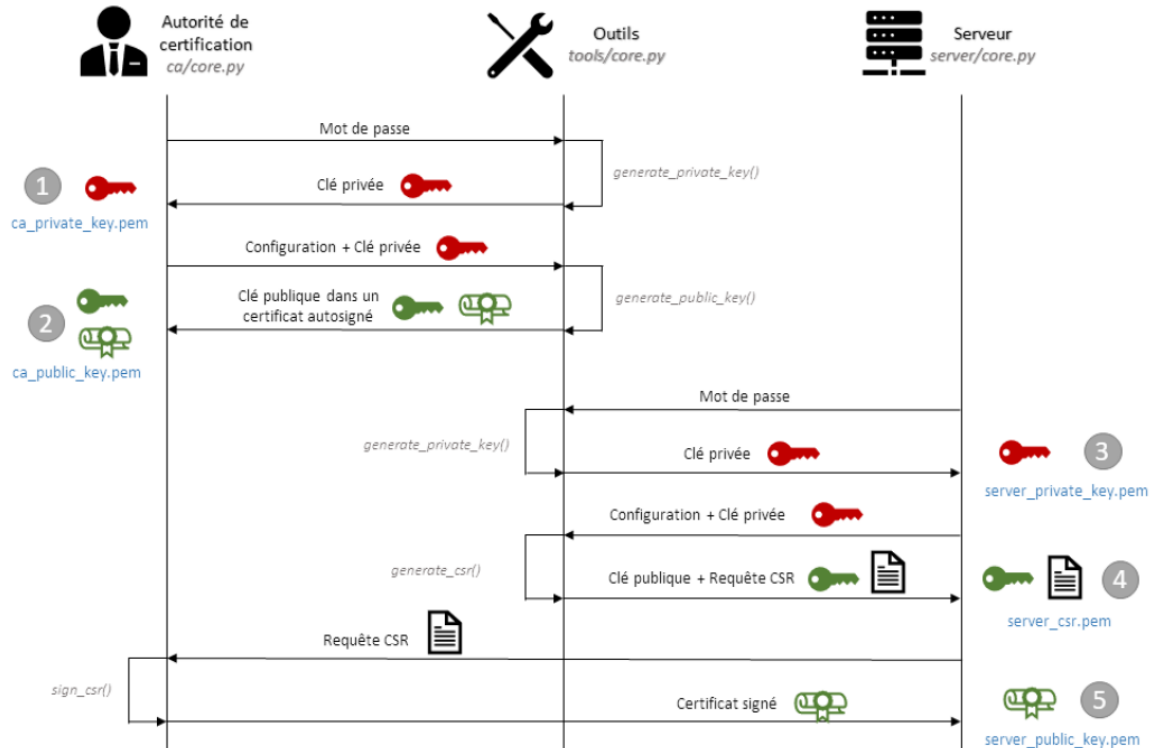
Après le filtrage `tcp.port==8081` on constate que la réponse http peut être observé par tout le monde et donc le compte peut être facilement pirater.

## Génération du certificat de l'autorité de certification

Qu'est-ce que le *protocole HTTPS* ?

Le **protocole HTTPS** est une extension sécurisée du protocole HTTP, le « S » pour « Secured » signifie que les données échangées entre le navigateur de l'internaute et le site web sont chiffrées et ne peuvent en aucun cas être espionnées (**confidentialité**) ou modifiées (**intégrité**). Obtenir le sacro-saint « S » passe par l'acquisition et l'installation d'un **certificat SSL/TLS** auprès d'une Autorité de Certification reconnue. Cela affichera ainsi le **HTTPS**, le cadenas vert et le mot « Sécurisé » dans la barre d'adresse du navigateur.

Dans ce TP on va créer un certificat autosigné en procédant aux étapes suivantes :



## Génération de la clé privée et la clé publique de l'autorité de certificat

Pour générer la clé privée du certificat on va importer le module `rsa` et appeler la méthode `generate_private_key` qui prend en paramètre l'exposant public `e=65537` et on le stocke dans un fichier puis on chiffre avec le « password » qu'on va le passer en paramètre

```
# Génération clé privée
def generate_private_key(filename: str, password: str):
    # 65537 est l'exposant public magique
    private_key = rsa.generate_private_key(
        public_exponent=65537, key_size=2048, backend=default_backend()
    )
```

```

# Paramètres d'encodage pour le chiffrement de la clé privée
utf8_pass = password.encode("utf-8")
algorithm = serialization.BestAvailableEncryption(utf8_pass)

# Création du fichier "filename" contenant le clés privés (p,q,n) chiffrées avec le "password"
with open(filename, "wb") as keyfile:
    keyfile.write(
        private_key.private_bytes(
            # FIXME Expected type 'Encoding', got 'str' instead
            encoding=serialization.Encoding.PEM,
            # FIXME Expected type 'PrivateFormat', got 'str' instead
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=algorithm,
        )
    )
return private_key

```

Cette methode est appele a la creation d'une instance de la classe *CertificateAuthority*

```

# Création de l'autorité de certification
certificate_authority = CertificateAuthority(
    CA_CONFIGURATION
    ,CA_PASSWORD
    ,CA_PRIVATE_KEY_FILENAME
    ,CA_PUBLIC_KEY_FILENAME
)

```

```

# Création de l'autorité de certification
certificate_authority = CertificateAuthority(
    CA_CONFIGURATION
    ,CA_PASSWORD
    ,CA_PRIVATE_KEY_FILENAME
    ,CA_PUBLIC_KEY_FILENAME
)

```

*CA\_CONFIGURATION* représente les informations de l'autorité ce qui sont dans notre cas :

```

CA_CONFIGURATION = Configuration('FR', 'Franche-Comté', 'Belfort', 'UTBM', 'www.ca.com',
['www.ca_2.com'])

```

`CA_PASSWORD` C'est le mot de pass pour chiffrer la cle privée

`CA_PASSWORD = '321'`

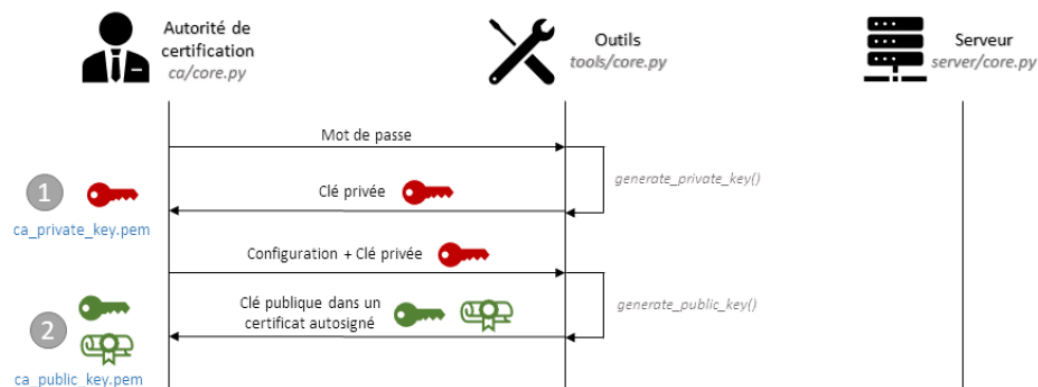
et Finalement `CA_PRIVATE_KEY_FILENAME` et `CA_PUBLIC_KEY_FILENAME` les noms des fichier qu'ils vont contenir les clés.

```
def __init__(self, config: Configuration, password: str, private_key_filename: str, public_key_filename: str):
    self._config = config
    self._private_key = generate_private_key(private_key_filename, password) # A compléter
    self._public_key = generate_public_key(self._private_key, public_key_filename, config) # A compléter
    self._private_key_filename = private_key_filename
    self._public_key_filename = public_key_filename
    self._password = password
```

à la création de l'objet «*certificate\_authority*» l'autorité de certificat aura ses propre clés publique et privées.

on a réaliser les 2 premières étape...

## Génération de la clé privée du serveur et la requête CSR



Une demande de certificat SSL impose donc de suivre un certain nombre d'étapes. La première consiste alors à prendre contact avec une Autorité de Certification (CA). La deuxième étape est autrement plus délicate : il s'agit en effet de générer une demande de signature de certificat (ou CSR certificat pour *Certificate Signing Request*) et de l'adresser à l'CA dans le but d'obtenir un certificat numérique.

En substance, une **CSR** est donc un message vers une Autorité de Certification par un demandeur, dans le but d'obtenir un certificat d'identité numérique.

La demande de signature de certificat est générée par le demandeur. Celui-ci doit créer une **clé publique** (qui sera incluse dans la CSR) et une **clé privée** (qu'il utilisera pour signer numériquement la demande et qu'il gardera secrète)



Comme la CA, le serveur va générer sa clé privée à la création de l'objet « server »

```
def __init__(self, config: Configuration, password: str, private_key_filename: str, csr_filename: str):
    self._config = config
    self._private_key = generate_private_key(private_key_filename, password)
```

En plus le serveur génère la requête CSR qui contient la clé publique ensuite la CSR va être signée par sa clé privée afin de réaliser l'authentification de la part du serveur

```
self._csr = generate_csr(self._private_key, csr_filename, config)
```

```
# Génération du fichier de requête de certification
def generate_csr(private_key, filename: str, config: Configuration):
    # Construction des informations qui font l'objet de la certification
    subject = read_configuration(config)

    # Génération des alternatives de serveurs DNS valides pour le certificat
    alt_names = []
    for name in config.alt_names:
        alt_names.append(x509.DNSName(name))
    san = x509.SubjectAlternativeName(alt_names)

    # Génération des différents constructeurs d'objet des attributs du CSR
    builder = (
        x509.CertificateSigningRequestBuilder()
        .subject_name(subject)
        .add_extension(san, critical=False)
    )

    # Signature du CSR avec la clé privée
    csr = builder.sign(private_key, hashes.SHA256(), default_backend())

    # Écriture de la requête de signature du certificat dans le fichier PEM
    with open(filename, "wb") as csrfile:
        # FIXME Expected type 'Encoding', got 'str' instead
        csrfile.write(csr.public_bytes(serialization.Encoding.PEM))

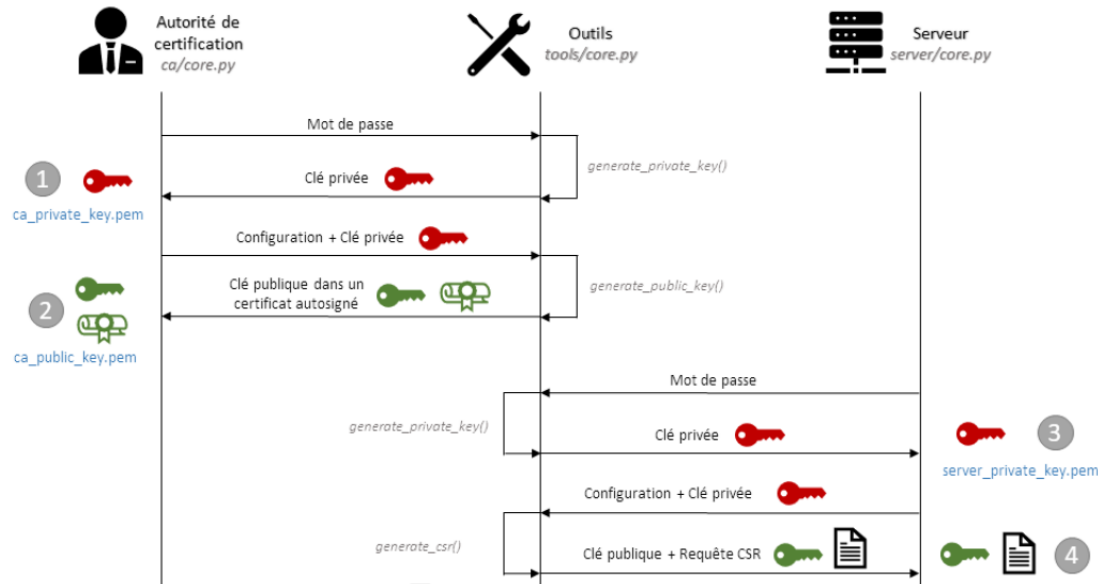
    return csr
```

Maintenant on a les 4 fichiers « .pem »

1. Clé privée de CA
2. Clé publique de CA
3. La requête CSR du serveur
4. Clé privée du serveur

```
resources
ca-private-key.pem
ca-public-key.pem
server-csr.pem
server-private-key.pem
```

Les étapes réalisées à l'instant :



## L'envoi de la certificat au CA par le Serveur

Après avoir générer la requête CSR signée par le serveur, la CA construit la certificat avec les données dans la requête CSR :

- le nom de domaine qui a reçu le certificat ;
- la personne, l'entreprise ou l'appareil qui a reçu le certificat ;
- l'Autorité de certification qui a émis le certificat ;
- la signature électronique de l'Autorité de certification ;
- les sous-domaines associés ;
- la date d'émission du certificat ;
- la date d'expiration du certificat ;
- **la clé publique du serveur (la clé privée n'est pas divulguée).**

```

# Création de la clé publique signée par le ca
def sign_csr(csr, ca_public_key, ca_private_key, filename: str):
    # Definition de la validité du certificat qui sera généré à 60 jours
    valid_from = datetime.utcnow()
    valid_until = valid_from + timedelta(days=60)

    # Attributs du certificat
    builder = (
        x509.CertificateBuilder()
        .subject_name(csr.subject) # l'objet est bien celui du CSR
        .issuer_name(ca_public_key.subject) # issuer est le ca
        .public_key(csr.public_key()) # obtient la clé publique du CSR.
        .serial_number(x509.random_serial_number())
        .not_valid_before(valid_from)
        .not_valid_after(valid_until)
    )

    # Ajoute les extensions existantes dans le certificat csr
    for extension in csr.extensions:
        builder = builder.add_extension(extension.value, extension.critical)

```

et La CA signe le certificat par sa clé privée

```

# Signature de la clé publique avec la clé privée du ca
public_key = builder.sign(
    private_key=ca_private_key,
    algorithm=hashes.SHA256(),
    backend=default_backend(),
)

# Génération du certificat signée par le ca
with open(filename, "wb") as keyfile:
    # FIXME Expected type 'Encoding', got 'str' instead
    keyfile.write(public_key.public_bytes(serialization.Encoding.PEM))

```

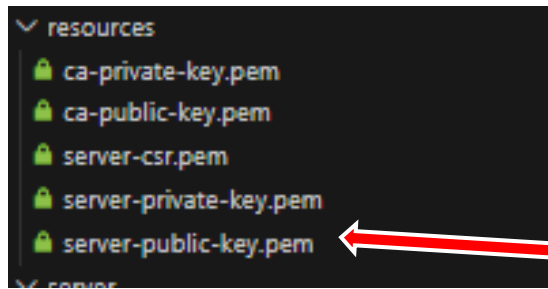
l'appel de la fonction est fait en appelant la fonction :

```

signed_certificate = certificate_authority.sign(server.get_csr(), SERVER_PUBLIC_KEY_FILENAME)# A
compléter

```

maintenant on a bien les 5 fichiers .pem



Le Certificat

Affichage du certificat signé :

```
def print_perms(filename: str):
    file=pem.parse_file(filename)
    for cert in file:
        print(cert.as_text())

-----BEGIN CERTIFICATE-----
MIIDaDCCAICgAwIBAgIUUq9G/GDXjYef1Nr7TMjkOZIld+4wDQYJKoZIhvcNAQEL
BQAwXDELMAkGA1UEBhMCRIxLc3J0MQ0wCwYDVQQKDARVVEJNMwEQYDQDDAp3d3cuY2EuY29t
VQQHDAZCZWxmb3J0MQ0wCwYDVQQKDAZVVEJNMwEQYDQDDAp3d3cuY2EuY29t
MB4XDTEzMDYyMDIxNTg0NlloXDTIzMDgwOTIxNTg0NlowXDELMAkGA1UEBhMCRIxLc3J0MQ0wCwYDVQQKDAZVVEJNMwEQYDQDDAp3d3cuY2EuY29t
DzANBgNVBAgMBkJsXj1dDEQMA4GA1UEBwwHVHJpcG9saTENMAsgA1UECgwEU0FB
RDEbMBkGA1UEAwwSc3d3LnNlcnZlcNhYWQuY29tMIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBCgKCAQEAswjrTNvq/GOP/P8N9SqnCRsCaIa0qbh8wPZ1cfgqsxLk
W2iyFm55wVCIP6K9QwF/wz2UtSfxYwP8ZUDK2EBmpBR7dWZNpNL8gRLcMS/9f8+k
0R66TOmKEMEowkja4c8k1LB3w6U3km6BFkhVIPdZWQnnojSgIOELrK/6/aG253Y
BOieo5kxTR6c6el+ltpdwFcPi/1RRTyw17W5gH4o2rVyW7RIICvcUBIGQxMoNg7T
/cBlI8Nhp9HIQBGt8ZcWQIjtLq4Cs91V6jxw4sRHbzHdg/GImXd9MqUDj8WITvW6
EAxyddBDvbdLCLf0hfsFh7gWXSzgjhiYZCrPys4HywIDAQABoyIwIDAeBgNVHREE
FzAVghN3d3cuc2VydmdyU2FhZDIuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQAAtQvva
MP7Ja363H9xdhR8/asHO1gI4Hy58ww2+70sxK4pYj00icsnFp+VRBOs+5xA/zklo
L5iwNidWYkO8urlWOcIVn2ToDx1ee6eF10Xo/mD9VnxRsuxMiNAHq3mIP/Ub1nP/
16MM2k+z5ruaHlh/Ntkfwd93ZRWzT10LxS1UB5G7x2l6HI0W6bOm7FeBBJkEprWK
RA9cg895IaY7ZgyEUApBur6zZXwjexuPPhxhYCOdW/O4fhqc3uElkFz4TxQtQh80
n+bA2NORq5bVIRCHmT+EcWE0qqaKNy81T26DrXRGTI9aOQTji0xeXZjiMSXJ1wBl
MqXW8OM6FxP92iBR
-----END CERTIFICATE-----
```

## Connexion https

Pour assurer une connexion HTTPS il faut qu'on lance le serveur avec le certificat déjà généré aussi :

```
context = (SERVER_PUBLIC_KEY_FILENAME, SERVER_PRIVATE_KEY_FILENAME)
app.run(debug=True, host="127.0.0.1", port=8081, ssl_context=context)
```

Le PEM code c'est le password du serveur

```
SERVER_PASSWORD = '123'
```

### Infos du certificat :

Certificate Viewer: www.serverSaad.com

GeneralDetails

Issued To

Common Name (CN)www.serverSaad.com  
Organization (O)SAAD  
Organizational Unit (OU)<Not Part Of Certificate>

Issued By

Common Name (CN)www.ca.com  
Organization (O)UTBM  
Organizational Unit (OU)<Not Part Of Certificate>

Validity Period

Issued On

Saturday, June 10, 2023 at 11:58:46 PM

Expires On

Wednesday, August 9, 2023 at 11:58:46 PM

Fingerprints

SHA-256 Fingerprint

E3 E2 EA 07 74 32 B4 0E 2E E0 F3 C6 AC 97 4B 7B 78 31 0E 83 F2 57 4B F7 78 AB 75 26 7A 0E 9E E8 B0 84 A0 B4 10 78 6C 7B D4 AE 8D D1 2C 11 42 FE 9C DD 66 62

SHA-1 Fingerprint

B0 84 A0 B4 10 78 6C 7B D4 AE 8D D1 2C 11 42 FE 9C DD 66 62

## NET::ERR\_CERT\_AUTHORITY\_INVALID

L'erreur "**NET::ERR\_CERT\_AUTHORITY\_INVALID**" est une erreur courante qui se produit dans les navigateurs Web, tels que Google Chrome, lorsqu'un certificat SSL/TLS d'un site Web n'est pas valide. Cette erreur indique généralement que le certificat présenté par le site Web a été signé par une autorité de certification non reconnue car les navigateurs Web sont configurés pour faire confiance aux autorités de certification bien connues, telles que Let's Encrypt, Comodo, GoDaddy, etc. Lorsque on a utilisé un certificat auto-signé, le navigateur ne reconnaît pas l'autorité de certification et affiche donc l'erreur.



### Your connection is not private

Attackers might be trying to steal your information from **127.0.0.1** (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR\_CERT\_AUTHORITY\_INVALID

**Subject:** www.serverSaad.com

**Issuer:** www.ca.com

**Expires on:** Aug 9, 2023

**Current date:** Jun 11, 2023

**PEM encoded chain:**

```
-----BEGIN CERTIFICATE-----
MIIDaDCCA1CgAwIBAgIUUq9G/GDXjYef1Nr7TMjkOZI1d+4wDQYJKoZIhvcNAQEL
BQAwXDElMAkGA1UEBhMCRR1IxFAVBGnVBAGMDkZyYW5jaGUTQ29tdMOpMRAwDgYD
VQOHDADCBxmb3J0MQ0wCwYDVQOQDARVVEJNMRRwEQYDVQOQDAP3d3cuY2EuY29t
MB4XDTIzMDYxMDIxNTg0N1oXDTIzMDgwOTIxNTg0N1owXDElMAkGA1UEBhMCTE1x
DzANBgNVBAgMBk1laXJldDEQMA4GA1UEBhWHVHJpcG9saTENMA5GA1UECgwEU0FB
RDEbMBkGA1UEAww5d3d3LnN1cnZlc1NhYWQuY29tMIIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBIICgKCAQEA5wjrTNvq/G0P/P8N9SqnCRsCaIa0qbh8wPZ1cFgqslLk
W2iyFm55wVICIP6K9QwF/wz2UtSfxYwP8ZUDK2EBmpBR7dWZNPnL8gRLcMS/9f8+k
0R66T0mKEMEowkja4c8k1LB3w6U3km6BFkhVIPdZWQnnojSgIOELrK/6/aG253Y
B0ieo5kxTR6c6e1+1tpdwFcPi/1RRTyw17W5gH4o2rVyw7R1ICvcUBIGQxMoNg7T
/cB118Nhp9HIQBGt8ZcwQIjtLq4Cs91V6jxw4sRHbzHdg/GImXd9MqUDj8WITvW6
EAXydbBDbdLCLf0hfsFh7gWXSzgjhiYZCrPys4HywIDAQABoyIwIDAeBgNVHREE
FzAVghN3d3cuc2VydmVylU2FhZDIuY29tMA0GC5qGSIb3DQEBChUAA4IBAQAQvva
MP7Ja363H9xdhR8/asH01gI4Hy58ww2+70sXK4pYj00icsnFp+VRB0s+5xA/zk1o
L51wN1dWYk08ur1W0cIVn2ToDx1ee6eF10Xo/mD9VnxRsuxM1NAHq3mIP/UbInP/
16MM2k+z5ruaHIh/Ntkfwd93ZRwzT10LxS1UB5G7x216HI0W6bOm7FeBBJkEprWK
RA9cg895IaY7ZgyEUApBur6zZXwjexuPPPhxhYCOdW/O4fhqc3uEIkFz4TxQtQh80
n+bA2NORq5bVrCHmT+EwE0qqaKNy81T26dXRGTI9aOQTj10xeXZj1MSXJ1wB1
MqXW80M6FxP2iBR
-----END CERTIFICATE-----
```



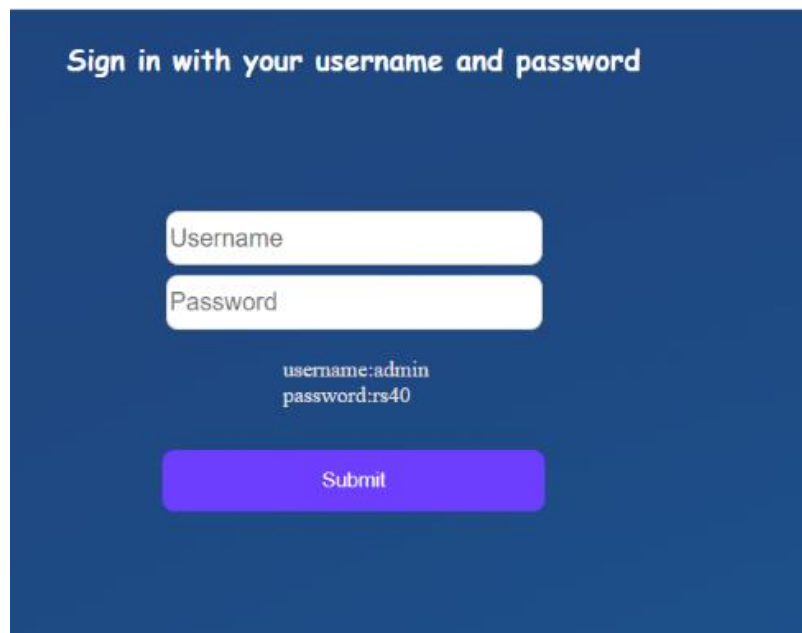
To get Chrome's highest level of security, [turn on enhanced protection](#)

### Solution proposée :

1. Obtenir un certificat auprès d'une autorité de certification de confiance : on peut acheter un certificat auprès d'une autorité de certification réputée. Il existe de nombreuses autorités de certification disponibles, telles que Comodo, GoDaddy, Symantec, etc. Ces autorités de certification sont pré-installées dans les navigateurs et sont généralement reconnues par défaut.

### Améliorations

Page login:



Sign in with your username and password

Username

Password

username:admin  
password:rs40

Submit

Username :admin

Password :rs40

## Hachage du mot de passe

```
username="admin"  
password="rs40"  
hashed_password=bcrypt.hashpw(password.encode("utf-8"),bcrypt.gensalt())
```

### 1. **bcrypt.hashpw(password.encode("utf-8"), bcrypt.gensalt())**

prend le mot de passe encodé et le sel généré, puis les utilise pour calculer le hachage du mot de passe. Le résultat est un hachage sécurisé du mot de passe.

Il est également important de ne pas stocker les mots de passe en texte clair. En utilisant une fonction de hachage sécurisée comme celle fournie par bcrypt, on stocke uniquement le hachage du mot de passe, ce qui rend le processus de vérification des mots de passe plus sûr et protège les utilisateurs en cas de compromission de la base de données.

Et pour la vérification de login on utilise :

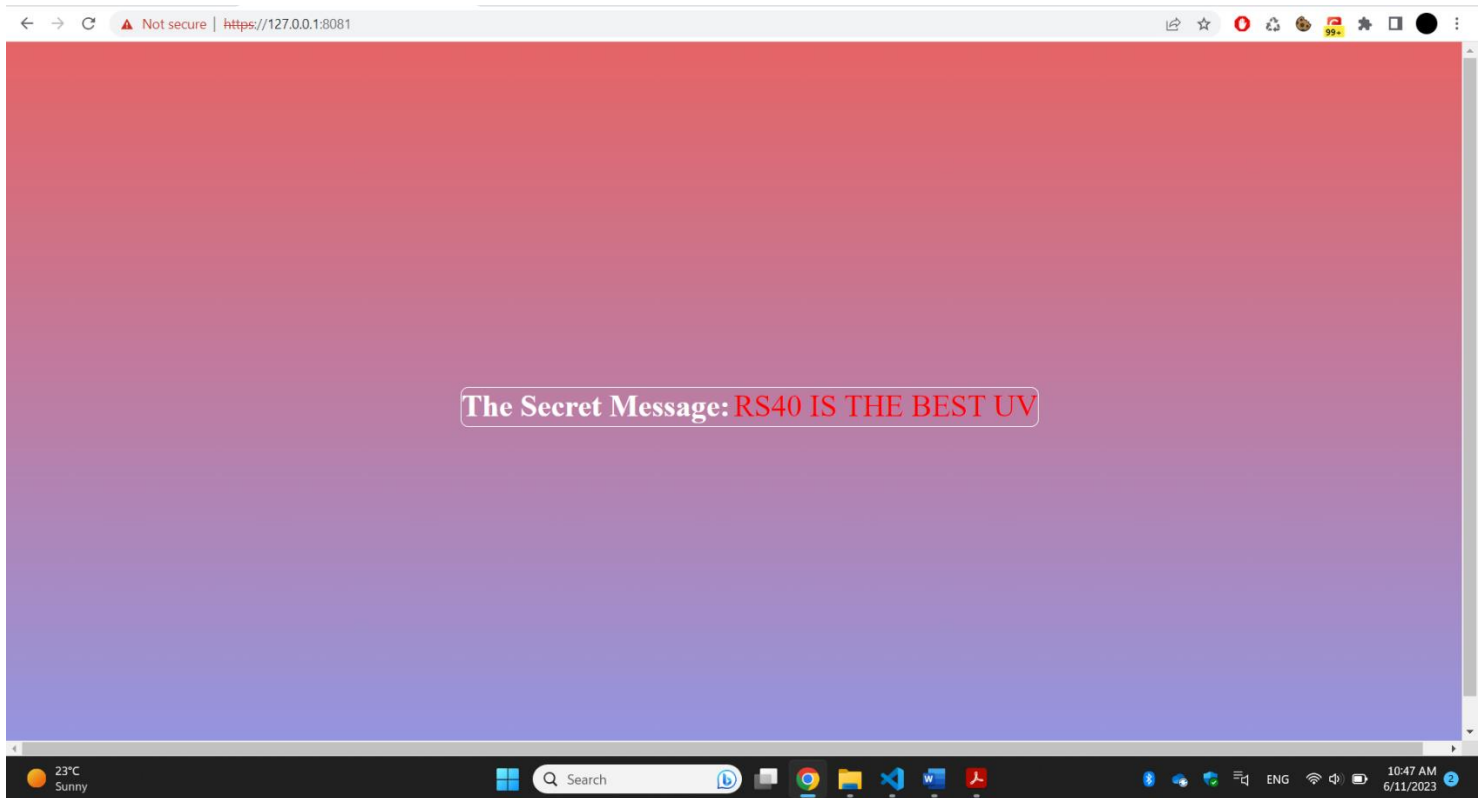
```
bcrypt.checkpw(request.form["password"].encode('utf-8'),hashed_password)
```

pour vérifier si un mot de passe fourni correspond au hachage de mot de passe stocké.

Il est important de noter que `bcrypt.checkpw()` compare les hachages et ne nécessite pas de déchiffrer le mot de passe stocké. Cela garantit une vérification sécurisée du mot de passe sans exposer le mot de passe lui-même.



Si on a réussi de s'authentifier on aura la page du message secret :



```
SECRET_MESSAGE = "RS40 IS THE BEST UV" # A modifier
username="admin"
password="rs40"
hashed_password=bcrypt.hashpw(password.encode("utf-8"),bcrypt.gensalt())
app = Flask(__name__)
@app.route("/",methods=['GET','POST'])
def get_secret_message():
    error=''
    if request.method=="POST":
        if request.form['username']!=username or not
bcrypt.checkpw(request.form["password"].encode('utf-8'),hashed_password):
            error="invalid"
        else:
            return render_template("home.html",secret_message=SECRET_MESSAGE)

    return render_template("index.html",error=error)
if __name__ == "__main__":
    context = (SERVER_PUBLIC_KEY_FILENAME,SERVER_PRIVATE_KEY_FILENAME)
    app.run(debug=True, host="127.0.0.1", port=8081, ssl_context=context)
```

# MERCI !