

TP2 RS40 2021



PROBLEMATIQUE

On doit remplacer la connexion http par une connexion https par rapport à la création de 5 certificats afin que les adhérents dans un club privé puissent obtenir un mot de passe.

Professeur : Abdeljalil ABBAS-TURKI

Florent PERRONNET

Réalisé par : Yuan Cao

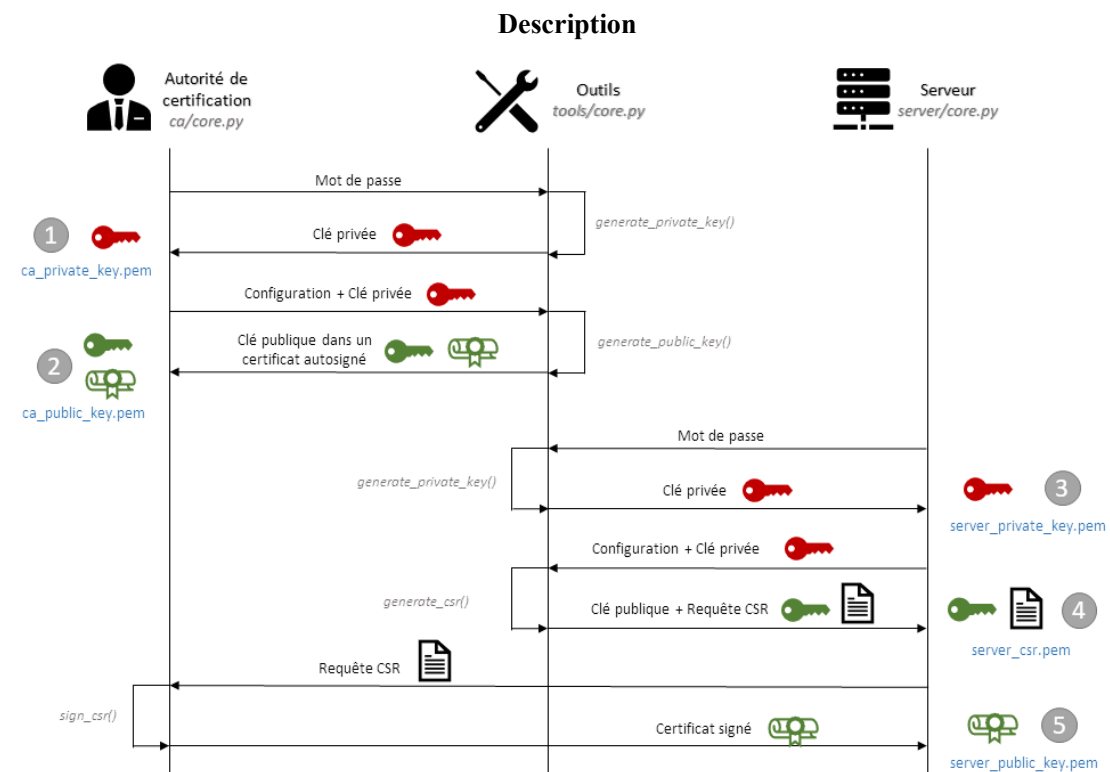
Sommaire

Introduction.....	3
1 Présentation du projet.....	4
1.1 Description.....	4
2 Conception.....	5
3 Conclusion.....	11

INTRODUCTION

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.

1.Présentation du projet

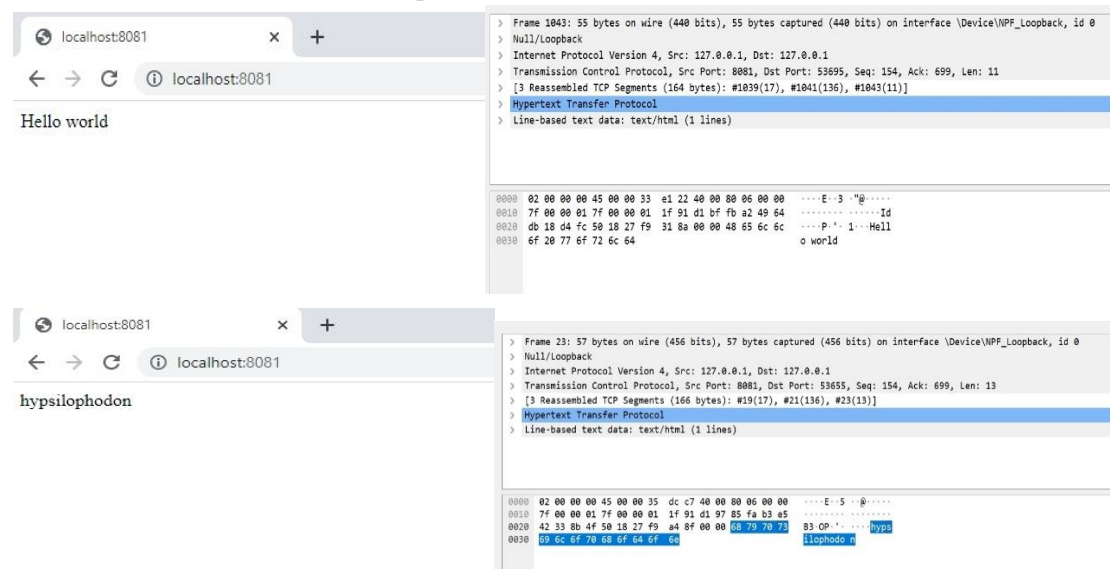


Principalement, on suivra les étapes ci-dessous :

1. Créer une autorité de certification (ca) détenant une paire de clés (publique, privée) et un certificat (le tiers de confiance)
2. Générer un certificat du serveur signé par la ca. Ceci se déroule en deux étapes :
 - a. Générer le certificat de la requête de certification (CSR : Certificate Signing Request)
 - b. Faire signer le certificat par la ca.
3. Remplacer la connexion http par la connexion https
4. Faire l'amélioration par rapport à la création d'un page web pour afficher le mot de passe du club privé après que l'un utilisateur tape son nom et son mot de passe corrects.

2. Conception

Partie 1 : Vérification du serveur http



En utilisant wireshark, on se convainc que tous le monde peut observer le mot de passe, ce qui est dangereux.

Partie 2 : Génération du certificat de l'autorité de certification

On génère le certificat autosigné de la ca ici :

```
class CertificateAuthority:
    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

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

Le mot de passe de la clé privée de la ca est :

```
CA_PASSWORD = 'ca_ycao' # A compléter
```

Les données de la ca est :

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

On visualise la clé privée de la ca et le certificat autosigné :

```
ca_private_key_filename = pem.parse_file(CA_PRIVATE_KEY_FILENAME)
ca_public_key_filename = pem.parse_file(CA_PUBLIC_KEY_FILENAME)
```

```
print(ca_private_key_filename)|
print(ca_public_key_filename)
```

```
In [1]: runfile('C:/Users/18019/Desktop/rs40/TP2/src/print_pems.py', wdir='C:/Users/18019/
Desktop/rs40/TP2/src')
[<RSAPrivateKey(PEM string with SHA-1 digest '52a33f5b73bba1b272506658c8eb2ee3816aa6c7')>]
[<Certificate(PEM string with SHA-1 digest '0ffb5721fc4c79f9b66bb5d675439812059e829d')>]
```

Partie 3 : Génération du certificat du serveur

On génère la clé privée du serveur (server-private-key.pem) ainsi que le certificat csr (server-csr.pem), qui contient la clé publique du serveur dans cette partie.

```
class 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)# A compléter
        self.csr = generate_csr(self.private_key, csr_filename, config)# A compléter
        self.private_key_filename = private_key_filename
        self.csr_filename = csr_filename
        self.password = password

    def get_csr(self):
        return self.csr
```

```
#Création du server
server = Server(
    SERVER_CONFIGURATION
    ,SERVER_PASSWORD
    ,SERVER_PRIVATE_KEY_FILENAME
    ,SERVER_CSR_FILENAME
    # A compléter
)
```

Le mot de passe de la clé privée du serveur est :

```
SERVER_PASSWORD = 'server_ycao'
```

Les données du server sont :

```
SERVER_CONFIGURATION = Configuration('CH', 'Shanghai', 'Shanghai', 'SHU', 'www.server.com', ['www.server.alt.com'])
```

On visualise la clé privée du serveur et le certificat csr qui contient la clé publique du serveur :

```
server_private_key_filename = pem.parse_file(SERVER_PRIVATE_KEY_FILENAME)
server_csr_key_filename = pem.parse_file(SERVER_CSR_FILENAME)
```

```
print(server_private_key_filename)
print(server_csr_key_filename)
```

```
[<RSAPrivateKey(PEM string with SHA-1 digest '9e7787321e2b1924ef5c8887916766864c900d67')>]
[<CertificateRequest(PEM string with SHA-1 digest '42896bf0430563c71f7487095adc028b4a469d5f')>]
```

Ensuite, on obtient le certificat définitif du serveur :

```
def sign(self, csr, certificate_filename):
    sign_csr(csr, self.public_key, self.private_key, certificate_filename)
    # A compléter

# Signature du certificat par l'autorité de certification
#print(server.get_csr())
signed_certificate = certificate_authority.sign(server.get_csr(), SERVER_PUBLIC_KEY_FILENAME)#
```

La visualisation du certificat du serveur est :

```
server_public_key_filename = pem.parse_file(SERVER_PUBLIC_KEY_FILENAME)
print(server_public_key_filename)
[<Certificate(PEM string with SHA-1 digest '990a4c2edd2635be56e344b0534b8f74e555d93d')>]
```

Partie 4 : Connexion https / Partie 5 : Améliorations

On modifie le fichier `run_serveur.py` afin de permettre aux membres du club de se connecter en https :

```
from flask import Flask, render_template

# définir le message secret
SECRET_MESSAGE = "Hello world" # A modifier

RESOURCES_DIR = "resources/"
SERVER_PRIVATE_KEY_FILENAME = RESOURCES_DIR + "server-private-key.pem"
SERVER_PUBLIC_KEY_FILENAME = RESOURCES_DIR + "server-public-key.pem"

app = Flask(__name__)

@app.route("/")
def get_secret_message():
    return render_template('index.html', secret_message=SECRET_MESSAGE)

if __name__ == "__main__":
    # HTTP version
    #app.run(debug=True, host="127.0.0.1", port=8081)
    # HTTPS version; Il faut que l'on utilise le terminal ici(tapez: py run_server.py)
    context = (SERVER_PUBLIC_KEY_FILENAME, SERVER_PRIVATE_KEY_FILENAME)
    app.run(debug=True, host="127.0.0.1", port=8081, ssl_context=context)
```

Aussi, je fais l'amélioration par créer un web simple pour lire le message après que l'on tape le nom et le mot de passe corrects. Les codes d'amélioration sont dans le fichier « `index.html` » qui se divise également en trois parties :html, css, javascript. Ce fichier est dans le répertoire « `templates` » pour que la fonction `render_template()` puisse l'utiliser.

Ici, le message secret est : Hello world

Maintenant, je teste le serveur https sur mon navigateur :

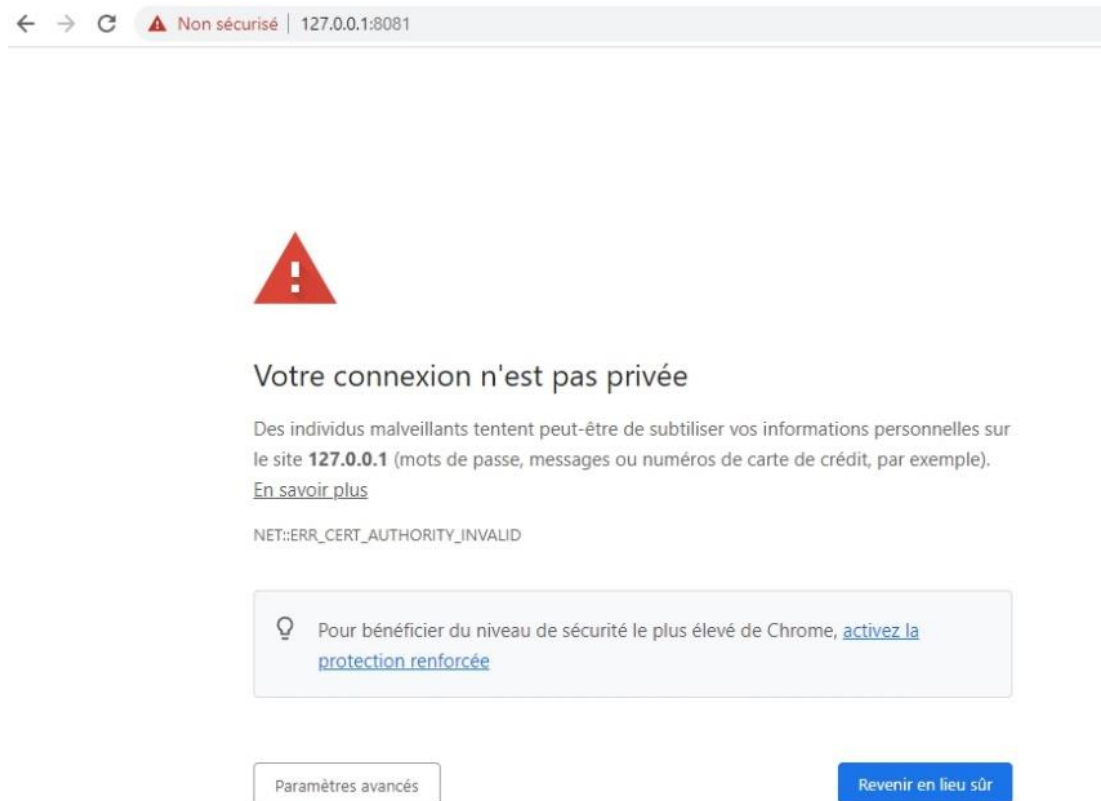
Au lancement du programme `serveur_run.py` on tape la ligne de commande sur le terminal :

`py serveur_run.py`

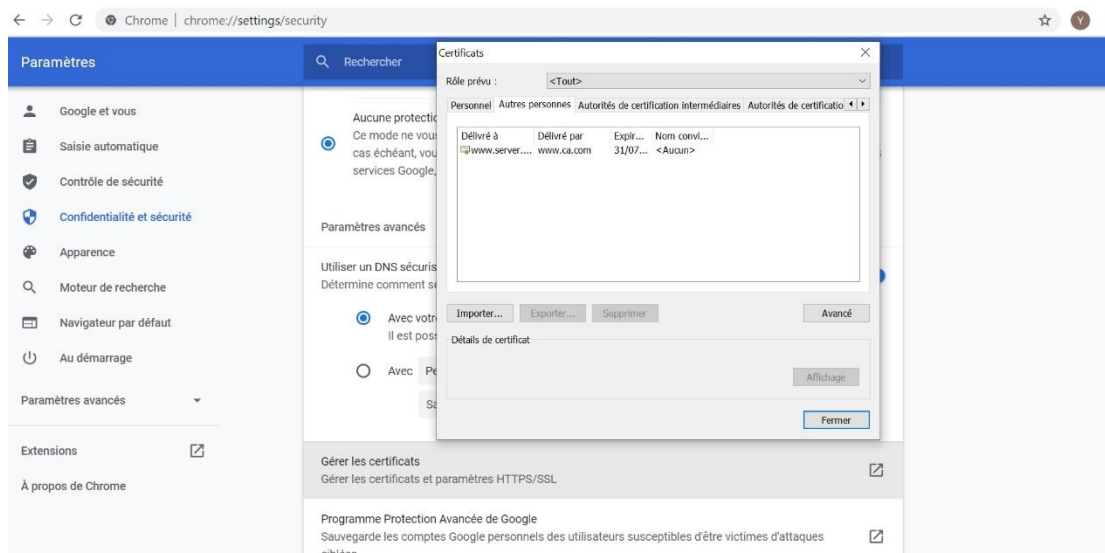
Et après on saisit le mot de passe du serveur :

`server_ycao`

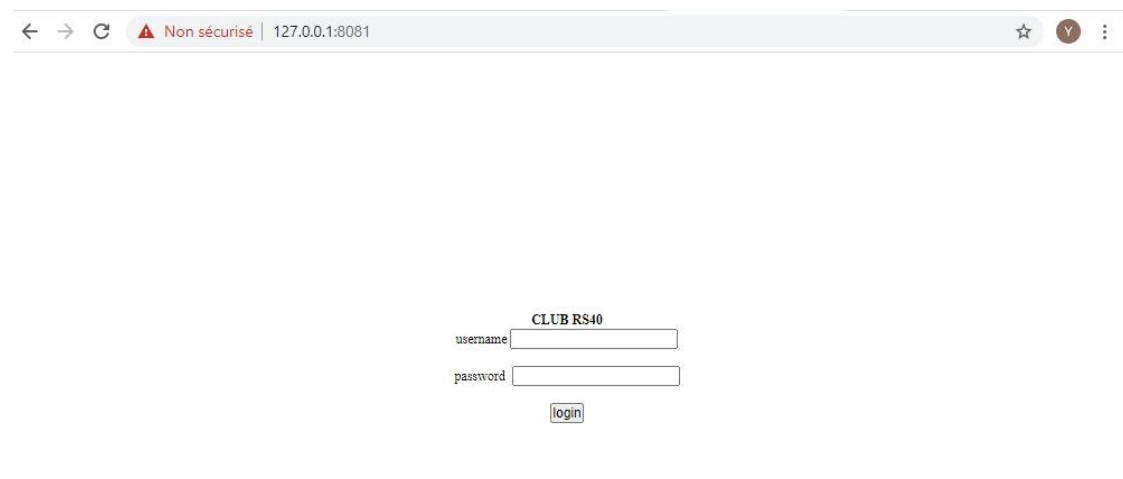
Une fois ce mot de passe saisi, on peut ouvrir notre navigateur en tapant : `https://127.0.0.1:8081` et constater qu'il refuse à priori de nous donner accès à la page. Le message d'erreur obtenu est le suivant :



Il suffit alors de cliquer sur Advanced puis Accept the Risk and Continue pour accepter le certificat mais on peut simplement éviter cet avertissement en important mon certificat dans le navigateur.

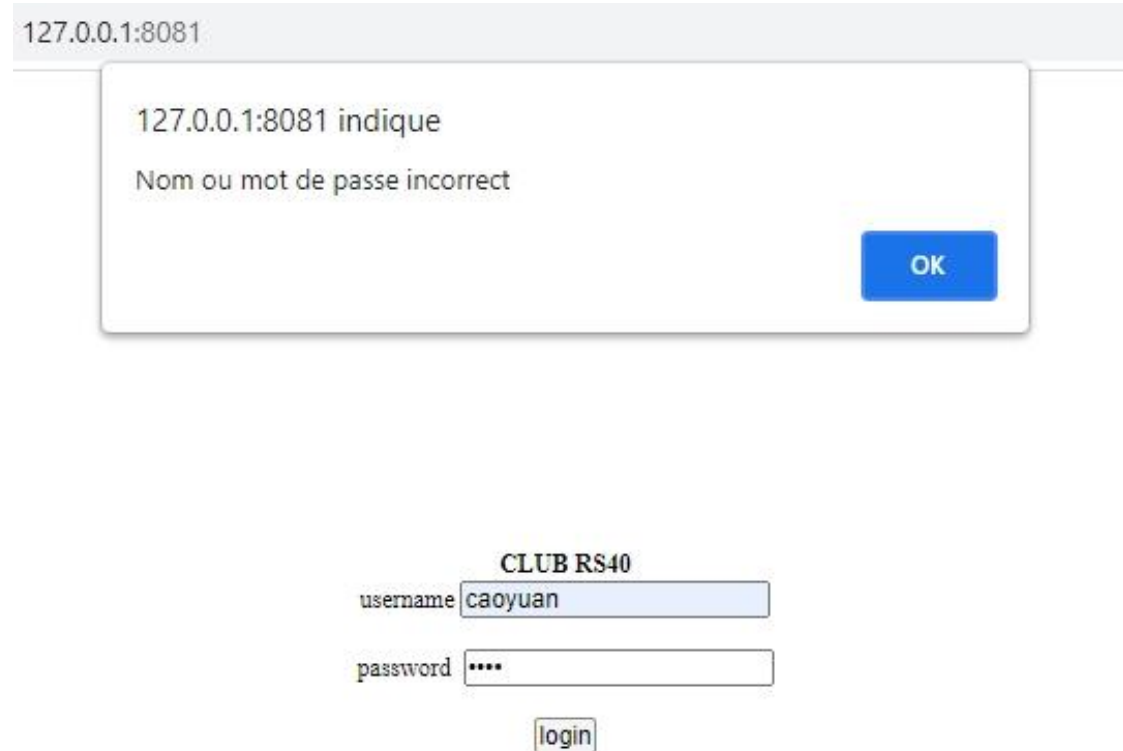


Maintenant il affiche la page finale :



A screenshot of a web browser window. The address bar shows a warning icon and the text "Non sécurisé | 127.0.0.1:8081". The page content is a login form with the title "CLUB RS40". It contains two input fields: "username" and "password". Below the "password" field is a "login" button.

Si on tape le nom et le mot de passe incorrects :



A screenshot of a web browser window showing an error message. The address bar displays "127.0.0.1:8081". A modal dialog box is centered on the screen with the text "127.0.0.1:8081 indique" and "Nom ou mot de passe incorrect". A blue "OK" button is in the bottom right corner of the dialog. Below the dialog, the login form is visible again, but the "username" field now contains the text "caoyuan" and the "password" field contains four dots "....". The "login" button remains below the fields.

Si on tape le nom et le mot de passe corrects comme

username : ycao

password : rs40

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8081'. A notification box in the center of the page states: '127.0.0.1:8081 indique Login avec succès, votre mot de passe est : Hello world'. Below this, there is a login form titled 'CLUB RS40'. The form contains two input fields: 'username' with the value 'ycao' and 'password' with masked characters '****'. A 'login' button is positioned below the password field.

Le terminal s'affiche :

```
Microsoft Windows [version 10.0.19042.985]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\18019>cd C:\Users\18019\Desktop\rs40\TP2\src

C:\Users\18019\Desktop\rs40\TP2\src>py run_server.py
* Serving Flask app 'run_server' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 137-624-975
Enter PEM pass phrase:
* Running on https://127.0.0.1:8081/ (Press CTRL+C to quit)
127.0.0.1 - - [08/Jun/2021 22:03:03] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [08/Jun/2021 22:03:03] "GET /favicon.ico HTTP/1.1" 404 -
```

Note : Ce TP a été fait dans un environnement de développement python : Spyder (Anaconda3)

3. Conclusion

Ce projet me permet de mettre en pratique les connaissances acquises en cours de l'UV RS40.

Grâce à celui, je me familiarise avec la notion de certificat et aussi l'implémentation d'une connexion sécurisée HTTPS, qui est indispensable dans le domaine qui nécessite une connexion internet en toute sécurité tels que le site de banque.

Ce TP me permet aussi de comprendre l'utilisation de package flask, pem, cryptography et aussi le logiciel Wireshark. Ils sont tous importants dans le domaine de réseau, serveur web et cybersécurité.