

PROJET CRYPTO

RAPPORT

Sarhiri Nabill

Relève Jean-Baptiste

2024

SOMMAIRE

01

INTRODUCTION

02

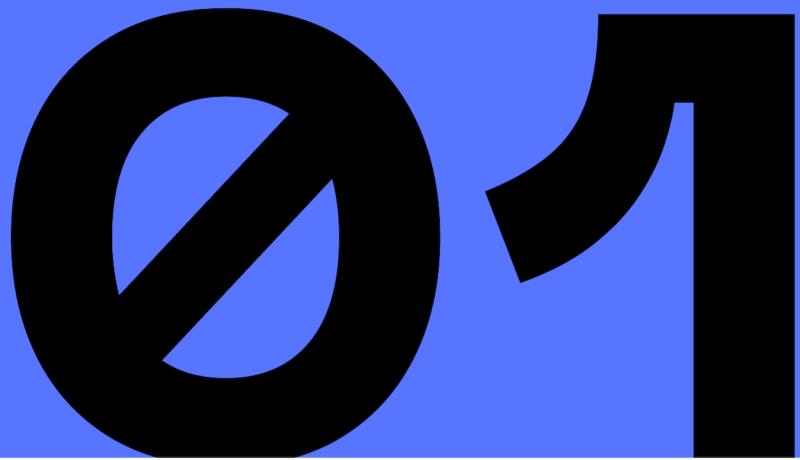
NOTRE PROJET

03

ARCHITECTURE

04

CONCLUSION



INTRODUCTION

Introduction

Le contexte

Dans le cadre de notre cours de cryptographie nous avons étudié les signatures numériques à l'aide de différents algorithmes de cryptographie mais aussi le fonctionnement d'une chaîne de certificats x509 utilisé dans le web.

Le projet qui nous est demandé est de réaliser un programme permettant la vérification/validation d'une chaîne de certificat x509 en vérifiant les signatures ainsi que tous les paramètres allant de la date de validation à la potentielle révocation d'un certificat. Ce programme pourrait constituer un module d'une pile TLS.

Ce rapport explique comment nous avons pensé le projet et comment nous l'avons réalisé.





NOTRE PROJET

Notre projet

Le contexte

Notre idée principale est de réaliser un site web permettant de vérifier une chaîne de certificats x509. Cela permettra à des utilisateurs de vérifier une chaîne à l'aide d'un outil graphique et simple d'utilisation.

Nos choix

Nous avons du faire des choix de technologies et de langage de programmation à utiliser, en voici les explications:

- Programme principal: Python

Nous avons choisis le python pour sa simplicité d'utilisation mais aussi pour ses nombreuses librairies publiques.

- WebFront: Html, Css, Js

Pas besoin de framework nous avons développé une interface simple mais efficace.

- WebBack: Python Flask

Nous avons décidé de baser notre projet sur python il était donc cohérent de faire un serveur web en python, nous avons donc utilisé la librairie Flask qui est largement la plus utilisé aujourd'hui.

Notre projet

Réalisation

A l'heure actuel notre projet comprends différentes fonctionnalités que nous expliquons ci dessous:

- Validation d'un certificat x509 auto signé. Au format PEM et DER
- Validation d'une chaîne de certificats x509 au format PEM et DER
- Validation de la signature d'un certificat pour les algorithmes RSA et ECDSA.
- Vérification de la révocation d'un certificat
- Vérification des dates de validité d'un certificat
- Vérification de l'autorisation de génération de certificat
- Vérification des usages des clés d'un certificat

Nous avons en fait réussi à réaliser une sorte de module TLS qui vérifie toute une chaîne de certificat.

Le seul point qui nous manque est une liste de RCA auquel notre programme peut faire confiance, étant donné que ça n'était pas demandé nous avons décidé de pas nous en occuper.

Réalisation

Nous avons aussi un serveur web qui gère la communication de données entre le client (utilisateur) et notre programme.

Ce serveur web est intuitif et permet à l'utilisateur d'uploader des fichiers de certificats facilement.

The screenshot displays three separate certificate validation results, each with a red header and a white body. Each result includes a 'Load a certificate' button and an 'Envoyer' button at the bottom right.

Result 1 (Top):

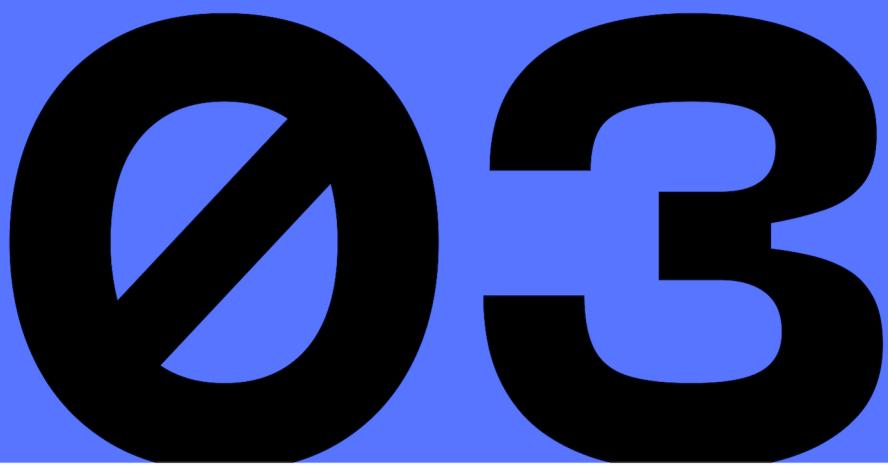
Attribut	Value
file_name	DigiCert_Global_Root_02.cer
cert_id	42917404946573785148798632253841
signAlgo	sha256
cryptoAlgo	RSA
sign	-----BEGIN PUBLIC KEY----- MIGfMA0GCSqDQSB2J... END PUBLIC KEY-----
dateBefore	2013-01-01 12:00:00
dateAfter	2018-01-15 12:00:00
expired	False
revoked	False
subject	CN=DigiCert Global Root G2,O=www.digicert.com,O=DigiCert Inc,C=US
issuer	CN=DigiCert Global Root G2,O=www.digicert.com,O=DigiCert Inc,C=US
pubk	-----BEGIN PUBLIC KEY----- MIGfMA0GCSqDQSB2J... END PUBLIC KEY-----
keyUsage	-----
keyUsageValid	True
autoSign	True
isCA	True
valid	True

Result 2 (Middle):

Attribut	Value
file_name	www.amazon.fr.cer
cert_id	111718114324461870928746297151500029
signAlgo	sha256
cryptoAlgo	RSA
sign	-----BEGIN PUBLIC KEY----- MIGfMA0GCSqDQSB2J... END PUBLIC KEY-----
dateBefore	2013-01-01 12:00:00
dateAfter	2018-01-10 23:59:59
expired	False
revoked	False
subject	CN=www.amazon.fr
issuer	CN=DigiCert Global CA G2,O=DigiCert Inc,C=US
pubk	-----BEGIN PUBLIC KEY----- MIGfMA0GCSqDQSB2J... END PUBLIC KEY-----
keyUsage	-----
keyUsageValid	True
autoSign	False
isCA	False
valid	Nabil & Jb, 2024

Result 3 (Bottom):

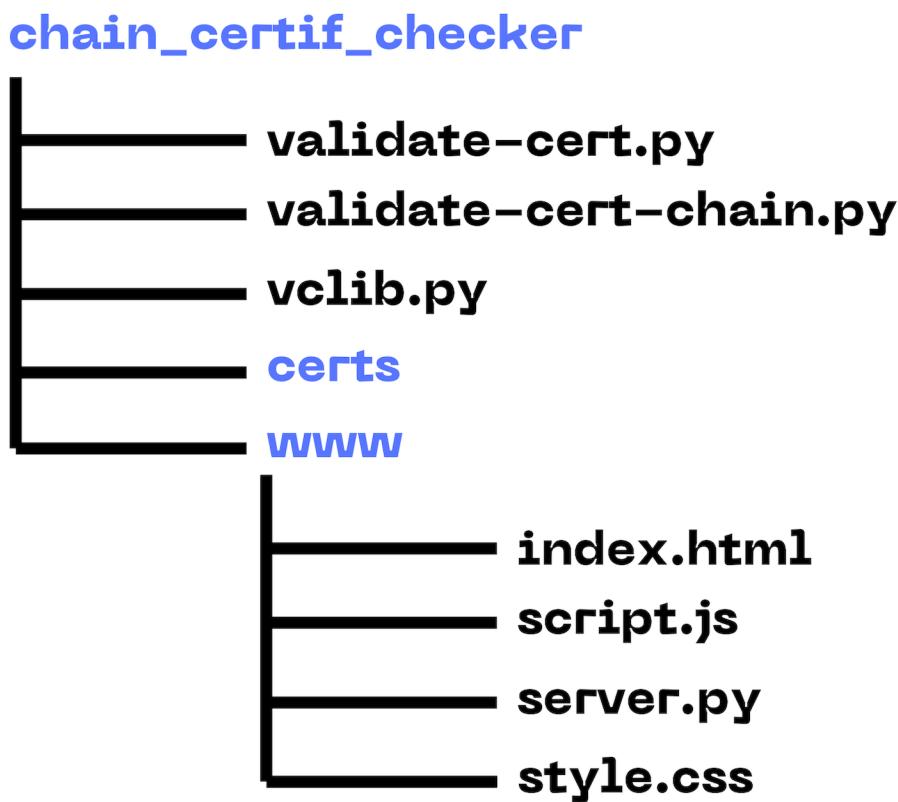
Attribut	Value
file_name	DigiCert_Global_Root_02.cer
cert_id	164426479309447963511630271144267
signAlgo	sha256
cryptoAlgo	RSA
sign	-----BEGIN PUBLIC KEY----- MIGfMA0GCSqDQSB2J... END PUBLIC KEY-----
dateBefore	2013-01-01 12:00:00
dateAfter	2018-01-15 12:00:00
expired	False
revoked	False
subject	CN=DigiCert Global Root G2,O=www.digicert.com,O=DigiCert Inc,C=US
issuer	CN=DigiCert Global Root G2,O=www.digicert.com,O=DigiCert Inc,C=US
pubk	-----BEGIN PUBLIC KEY----- MIGfMA0GCSqDQSB2J... END PUBLIC KEY-----
keyUsage	-----
keyUsageValid	True
autoSign	True
isCA	True
valid	Nabil & Jb, 2024



ARCHITECTURE

Architecture

Notre arborescence



L'ensemble de cette arborescence est détaillé ci-après dans le document mis à part le répertoire **certs** qui est destiné simplement à stocker les certificats téléchargé depuis l'interface web de notre application.

Architecture

Notre librairie

vclib.py

La VClib est une sorte de librairie python que nous avons développé dans le but de faire fonctionner nos scripts de validation de chaîne de certificats.

Librairies annexes utilisées

Afin de faciliter notre implémentation nous avons utilisé des librairies annexes qu'il faut installer sur sa machine à l'aide du gestionnaire de paquets **pip**:

- **cryptography**
- **ecdsa**
- **pyasn1**
- **requests**
- **json**
- **binascii**
- **sys**

class chain

Cette classe permet de stocker une chaîne et de vérifier sa validité.

Elle permet aussi au serveur web de récupérer les informations suite à cette vérification.

Architecture

Notre librairie

class certificat

Cette classe contient tous les attributs nécessaires à la vérification de la validité d'un certificat.

Elle contient aussi différentes méthodes permettant de valider:

- La signature (RSA / ECDSA)
- La révocation
- La validité dans le temps
- S'il est en droit de générer des certificats
- Les usages de la clé

Cette classe permet aussi au serveur web de récupérer toutes les informations des certificats afin de les renvoyer au client web.

Passons en revu certaines méthodes de cette classe notamment celles permettant de vérifier les signatures et les paramètres de validation.

Architecture

Notre librairie

Méthode checkSignEC

Cette méthode permet de vérifier la signature d'un certificat signé en ECDSA sans utiliser de librairie de vérification de signature. Pour se faire, cette méthode s'appuie sur des calcul mathématiques modulaires. Afin de parvenir à réaliser les calculs correctement, nous récupérons le nom de la courbe utilisée pour la signature pour ensuite retrouver ses paramètre et plus particulièrement l'ordre et le générateur.

Nous récupérons également la signature du certificat à l'intérieur de ce dernier (les composantes R et S), puis nous générerons un hash du contenu du certificat en utilisant le même algorithme de hash utilisé pour signer le certificat.

Enfin nous calculons les deux coefficients multiplicateur des points représentés par le générateur et la clé publique que nous appelons U1 et U2 :

$$U1 = ((S^{-1}) \bmod N \times \text{Hash}(M)) \bmod N$$

$$U2 = ((S^{-1}) \bmod N \times R) \bmod N$$

Avec :

S et **R** les composantes de la signature calculée à partir de la clé privée.

S⁻¹ mod N l'inverse modulaire de **S**.

N l'ordre du sous-groupe généré par le générateur.

M le contenu du certificat qui est hashé.

Architecture

Notre librairie

Pour finir nous calculons P qui sera notre point dont la coordonnée en X devra être égal à R.

$$P = U_1 \times G + U_2 \times Q$$

Avec :

Q la clé publique du certificat de l'autorité de certification qui a émis le certificat en cours de vérification.

G le générateur de la courbe elliptique.

Il ne nous reste plus qu'à vérifier la relation suivante :

$$Px \equiv R \pmod{N}$$

Avec :

Px la coordonnée en x du point P.

Architecture

Notre librairie

Méthode checkSignRSA

Cette méthode permet de vérifier la signature d'un certificat signé en RSA sans utiliser de librairie de vérification de signature. Pour se faire, cette méthode s'appuie sur des calcul mathématiques modulaires. Afin de parvenir à réaliser les calculs correctement, nous récupérons l'exposant public ainsi que le modulo depuis le contenu du certificat de l'autorité de certification qui a émis le certificat en cours de vérification.

Nous récupérons également la signature du certificat à l'intérieur de ce dernier, puis nous générons un hash du contenu du certificat en utilisant le même algorithme de hash utilisé pour signer le certificat.

Enfin nous calculons D équivalent à l'exponentiation modulaire de la signature à l'exposant publique:

$$D = (S^E) \bmod N$$

À ce niveau, D n'est pas le message hashé avant signature mais le hash ainsi que d'autres informations encodées selon le standard ASN.1. Il a donc fallut décoder cette donnée avant de pouvoir en extraire le hash.

Pour finir nous comparons le hash calculé à partir du contenu du certificat avec le hash obtenu après vérification de la signature et décodage de la sortie de cette même vérification.

Architecture

Notre librairie

Methode checkSign

Cette méthode permet de vérifier la signature d'un certificat signé en RSA ou en ECDSA en utilisant la fonction `verify()` de la librairie `cryptography`. cette fonction s'adapte selon les paramètre qui lui sont fournis pour vérifier la signature avec le bon algorithme.

Methode checkParam

Cette méthode permet de vérifier les paramètres d'un certificat. Autrement dit elle permet de vérifier la date de validité, le fait que le certificat soit autosigné, le fait que le certificat ait été émis pour un autorité de certification et les usages de la clé privée qui a servis à signer le certificat. En effet une clé qui sert à signer un certificat ne pas servir à chiffrer des données dans l'idéal.

Passons maintenant en revue les autres fonctions comprises dans le fichier vclib.py.

Fonction checkArgs

Cette fonction permet de vérifier les paramètres passé à aux scripts validate-cert.py et validate-cert-chain.py. Elle a été pensée pour fonctionner de manière optimisée avec les deux scripts sans disfonctionnement. Cette fonction vérifie que le bon nom d'arguments a été passé en paramètre et que les paramètres sont valides

Architecture

Notre librairie

Fonction initCertif

Cette fonction permet de récupérer un certificat et d'en extraire ces données grâce à la librairie cryptography. Elle va générer un objet de type certificat.

Fonction is_not_revoked

Cette fonction permet de télécharger toutes les liste de certificats révoqués mentionnés par leur url dans un certificat d'une autorité de certification afin de vérifier si les certificat émis par cette autorité de certification sont révoqué ou non. Après avoir téléchargé les liste de certificats révoqué cette fonction va vérifier dans chaque fichier s'il contient le numéro de série du certificat qui a été émis par l'autorité de certification et qu'il faut vérifier. Pour se faire la fonction **is_certificate_not_revoked()** est appelée pour chaque fichier.

Fonction is_certificate_not_revoked

Cette fonction permet de lire les fichiers de liste de certificats révoqués pour les parcourir et vérifier si le numéro de série d'un certificat à vérifier se trouve à l'intérieur.

Architecture

Notre premier script

validate-cert.py

Ce programme permet de tester la validité d'un seul certificat cela nous a permis de commencer le développement de notre librairie en se basant uniquement sur les certificat RCA (root certificate authority) étant donné que se sont des certificats auto signés. Ce script vérifie tout d'abord les arguments passés en paramètre avec la fonction **checkArgs()**, puis initialise un certificat grâce à la fonction **initCertif()**. Ensuite, la fonction **checkSign()** est appelée pour vérifier les signatures du certificat. Enfin, les fonctions **checkParam()** et **displayJson()** permettent de vérifier les paramètres et extensions du certificat et de retourner si le certificat est valide ou non.

Architecture

Notre script final

validate-cert-chain.py

Ce programme permet de tester la validité d'une chaîne de certificat. Il fonctionne dans l'ensemble à peu près comme le script **validate-cert.py**. La plus grosse différence réside dans le fait que ce script utilise les fonctions **checkSignRSA()** et **checkSignEC()** pour vérifier la signature des certificats. Nous utilisons les informations contenues dans les certificats pour extraire l'algorithme de signature afin de faire appel à la bonne fonction. De plus, grâce aux fonctions **is_not_revoked()** et **is_certificate_not_revoked()**, ce script permet de vérifier si les certificats sont révoqués. Par ailleurs, ce script permet de vérifier des chaînes de certificat de grandes tailles (pas de limites théoriquement) en gardant le fait que le premier certificat est un certificat autosigné appartenant à une autorité de certification racine et que tous les suivants appartiennent à des autorités de certification intermédiaires. Le dernier certificat (la feuille) doit être émis pour un objet (site web, objet connecté etc...). Pour finir, le script renvoie un JSON pour renseigner de la validité du certificat ou non.

Architecture

Notre serveur web et son interface

server.py

Ce serveur Flask récupère des certificats x509 via une requête post et les stock en local.

Il va par la suite exécuter notre programme de validation de chaîne en lui passant les fichiers locaux puis récupère la sortie standard afin de la renvoyer au client.

Le client reçoit donc les informations sur sa chaîne de certificat et adapte son design pour faire comprendre facilement à l'utilisateur le résultat de notre programme.

script.js

Ce script Javascript permet de récupérer les fichiers chargés depuis la page web et de les envoyer vers le serveur via une requête post **Ajax** sous la forme d'un chaîne JSON contenant pour chaque éléments le nom du fichier ainsi que son contenu. La réponse du serveur est analysée pour modifier en conséquence la page html.

index.html

Ce fichier sert de template pour permettre de charger les certificats et d'afficher le résultat de la vérification avec une grille qui affiche les informations des certificats et une couleur de fond qui permet de se rendre compte rapidement de la validité d'une chaîne à chaque certificat envoyé vers le serveur.

style.css

Ce fichier permet de modifier l'affichage stylistique de notre page web en fonction de l'état d'une vérification de certificat



CONCLUSION

Conclusion

Un projet réussi

Bilan technique

Ce projet a été une réussite dans son ensemble, nous avons pu intégrer toutes les fonctionnalités qui étaient demandées initialement. Par ailleurs, nous avons rajouté une interface pour rendre notre programme accessible et en faire une application web. Différents tests nous ont permis de confirmer le bon fonctionnement de notre application quelque soit la configuration des certificats chargés depuis l'interface web. En outre, nos scripts validate-cert.py ainsi que validate-cert-chain.py fonctionnent également directement en ligne de commande pour les plus téméraires.

Bilan pédagogique

Ce projet nous a permis de réellement toucher du doigt les différents algorithmes et standards utilisés dans les différentes applications cryptographiques. Il nous a permis également de comprendre le fonctionnement des certificats x509 et saisir la nécessité de vérifier et valider ces documents numériques primordiaux dans bon nombres de protocoles visant à sécuriser les échanges de données numériques.

Ressources

<https://cedricvanrompay.gitlab.io/tp-rsa/instructions.html>

<http://milletseb.free.fr/crypto/cryptographie-pratique.html>