

ESIEA

Rapport de cryptographie

Sujet n°1 - Certificats, web et calculs de clés privées

KESSLER Corentin, RUYNAT Eddy
05/01/2025

INTRODUCTION

Ce projet vise à analyser les clés RSA de certificat numériques et plus précisément leurs **clés RSA**. Pour ce faire, nous avons récupéré des certificats via **certstream** que nous avons ensuite décodé et trier selon leurs tailles.

Nous avons ensuite utiliser la méthode **Batch GCD** pour trouver des p et q commun aux clés qui pourrait servir à les casser.

Nous avons choisis de récupérer les certificats dans un fichier **csv** dans un premier temps.

Par la suite nous avons choisi d'utiliser une Base de données **MySQL** afin de faciliter le traitement des données et leurs stockage.

Nous avons également fait le choix d'utiliser **python** afin d'interagir avec la Base de données.

Le repository Github avec l'ensemble de nos codes se trouve à ce lien :

« https://github.com/Gruk92320/Projet_Crypto »

Table des matières

INTRODUCTION.....	1
Table des matières.....	2
Récupération des certificats	3
Insertion aux format PEM dans la table MySQL.....	5
Décodage des certificat et récupération des modulus	7
Tri par taille de clés et suppression des doublons	9
Conclusion.....	10

Récupération des certificats

Pour cette partie nous nous sommes aidés de plusieurs forums ainsi que de chatGPT pour la révision du code.

La première étape du projet consiste à récupérer 1 million de clé via certstream.

```
import logging
import datetime
import certstream
import csv
import os

# Tableau pour stocker les certificats
certificates = []
max_certificates = 1000000 # Limiter à 1 million de certificats
save_interval = 1000 # Sauvegarder toutes les 1000 itérations
```

Nous avons déclaré 2 variables qui nous permettent de limiter le nombre de certification ainsi qu'un tableau qui va stocker les certifications récupérées.

```
# Fonction pour sauvegarder les certificats dans un fichier CSV
def save_certificates_to_csv(certificates, filename='certificates.csv'):
    # Si le fichier existe déjà, on l'ouvre en mode ajout, sinon en mode écriture
    file_exists = os.path.exists(filename)
    with open(filename, mode='a', newline='', encoding='utf-8') as file:
        writer = csv.writer(file)
        # Écrire l'en-tête si c'est un nouveau fichier
        if not file_exists:
            writer.writerow(['Domain', 'As DER'])
        # Écrire les certificats dans le CSV
        for cert in certificates:
            writer.writerow([cert['domain'], cert['as_der']])

    print(f"Certificats sauvegardés dans '{filename}'")
```

Cette fonction permet de créer un fichier CSV de mettre un en-tête dans sa première ligne si aucun fichier CSV existe. De plus ce bout de code va écrire les certificats dans ce fichier.

```
def print_callback(message, context):
    global certificates

    logging.debug("Message -> {}".format(message))

    if message['message_type'] == "heartbeat":
        return

    if message['message_type'] == "certificate_update":
        # Si nous avons atteint le nombre limite de certificats à stocker
        if len(certificates) >= max_certificates:
            return

        # Extraire les données du certificat
        cert_data = message['data']['leaf_cert']

        # Extraire les domaines associés au certificat
```

```

all_domains = cert_data['all_domains']

# Si aucun domaine n'est trouvé, définir à "NULL"
if len(all_domains) == 0:
    domain = "NULL"
else:
    domain = all_domains[0]

# Créer un dictionnaire avec les informations essentielles du certificat
cert_info = {
    "domain": domain,
    "as_der": cert_data['as_der']
}
print(len(certificates))
# Ajouter le certificat au tableau
certificates.append(cert_info)

# Sauvegarder toutes les 1000 itérations
if len(certificates) % save_interval == 0:
    save_certificates_to_csv(certificates)
    certificates.clear() # Réinitialiser la liste des certificats

# Optionnel : Afficher un message toutes les 1000 itérations pour vérifier
if len(certificates) % 1000 == 0:
    print(f"{len(certificates)} certificats sauvegardés jusqu'à présent.")

```

Cette fonction vérifie si le nombre de certification a atteint le million, si ce n'est pas le cas elle va récupérer les informations importantes des certifications. Vérifier que ces certifications sont valides et si c'est le cas enregistre les certifications quand le tableau crée préalablement atteint 1000 unités.

```

# Configurer le logging
logging.basicConfig(format='[%(levelname)s:%(name)s] %(asctime)s - %(message)s', level=logging.INFO)

```

Cette partie initialise la confection a CertyStream a l'aide de la bibliothèque "logging".

```

# Connexion au flux CertStream
certstream.listen_for_events(print_callback,url='wss://certstream.calidog.io/full-stream')

```

La fonction Listern_gor_event provenant de la bibliothèque "certstream" va faire appel à notre fonction print_callback décrit juste au dessus à chaque fois que CertStream nous retourne une certification.

Insertion aux format PEM dans la table MySQL

Le but de cette étape était d'avoir l'ensemble de nos certificats stocker aux format PEM dans une table MySQL. Le code réalisant cette étape se trouve dans le fichier « csv_to_bdd.py » dans le repository.

Cependant, en regardant de plus près les certificats récupérer à l'étape précédente, il manquait « -----BEGIN CERTIFICATE----- » et « -----END CERTIFICATE----- » pour qu'il soit au format PEM.

Nous avons donc passé le code décrit après à ChatGPT en lui demandant d'ajouter ces balises sur chacune des lignes avant de les insérer dans la BDD. Chat GPT a également ajouté une suppression des lignes avec des champs manquants et la vérification des colonnes du documents CSV :

```
# Vérifier les colonnes du DataFrame
print("Colonnes disponibles :", df.columns)

# Vérifier que les colonnes attendues sont présentes
if 'name' not in df.columns or 'cle' not in df.columns:
    raise ValueError("Les colonnes 'name' ou 'cle' sont manquantes dans le fichier CSV.")

# Nettoyer les données : Supprimer les lignes avec des valeurs manquantes dans 'name' ou 'cle'
df = df.dropna(subset=['name', 'cle'])

# Ajouter les balises BEGIN CERTIFICATE et END CERTIFICATE autour des clés dans 'cle'
df['cle'] = df['cle'].apply(lambda x: f"-----BEGIN CERTIFICATE-----{x}-----END CERTIFICATE-----")
```

Pour ce faire, nous avons ouvert le fichier CSV contenant les certificats à l'aide de Pandas en python

Nous avons ensuite initié la connexion à la BDD et créer un curseur à l'aide de :

```
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="1234",
    database="mydb"
)

cursor = conn.cursor()
```

Une fois cela fait, nous utilisons une boucle qui itère sur chaque ligne du Dataframe et l'insère dans la table :

```
for _, row in df.iterrows():
    cursor.execute("INSERT INTO crypto (name, cle) VALUES (%s, %s)", (row['name'], row['cle']))
    counter += 1
```

```
# commit tout les 1000
if counter == 1000:
    counter = 0
    conn.commit()
    print(f"{counter} entrees")
```

A l'aide du print, nous pouvons suivre l'avancement du programme et le commit toutes les 1000 entrées permettent de ne pas perdre l'avancement si le programme plante.

Enfin nous ajoutons les entrées résiduelles et nous fermons la connexion à l'aide de :

```
# commit avec les dernières entrées
conn.commit()
print(f"+ {counter} entrees")

# Fermer la connexion
cursor.close()
conn.close()
```

Ci-dessous un échantillon du résultat obtenus pour les 10 premiers certificats :

```
1
2  select * from crypto where idcrypto <10;
3
4
```

Result Grid			
Filter Rows:			
Edit:			
Export/Import:			
Wrap Cell Content:			
	idcrypto	name	de
1	1	test.wjtuzd.info	-----BEGIN CERTIFICATE-----MIIETCCA9mgAwIBAgISA0khjo1ZdBjNW...
2	2	test.www132betorspin.com	-----BEGIN CERTIFICATE-----MIIFHZCCBAegAwIBAgISBFoHnzQsTiQdbc...
3	3	*.3767565500af4670964b5cc2073c3a87.int.cu...	-----BEGIN CERTIFICATE-----MIIH8zCCBdugAwIBAgITMwEVgUu9u3i6h8...
4	4	*.st666top1.com	-----BEGIN CERTIFICATE-----MIIDrDCCA1GgAwIBAgIRAKyS88h9gsL4Ec...
5	5	www.wwwwwwwwwwwwwwwdemo.givetogo.org	-----BEGIN CERTIFICATE-----MIIFGDCCBACgAwIBAgISBEbtT9lvVntfL29...
6	6	*.go88-sunwin-iwin-b52-stockroms.pages.dev	-----BEGIN CERTIFICATE-----MIIDCDCCAq+gAwIBAgIRAIbPqJ0lxV9JDI...
7	7	cpanel.freepalestine369.com	-----BEGIN CERTIFICATE-----MIIFzDCCBLsgAwIBAgISBLPmfUJ14oaEBz...
8	8	spencer-weaver.com	-----BEGIN CERTIFICATE-----MIIDhzCCAww2gAwIBAgISA32GxfMlzsipm2...
9	9	bs-mat-alertmetrics-timeout-ezw-dev.securionp...	-----BEGIN CERTIFICATE-----MIID5DCCA4ugAwIBAgIRAOIPntJ7PFJ2EY...
*	NULL	NULL	NULL

Décodage des certificats et récupération des modulus

Dans cette partie, nous avons cherché à décoder les certificats et à stocker leur modulus et la taille de leurs clés dans la BDD. Le code réalisant cette étape se trouve dans le fichier « decode.py » dans le repository.

Pour ce faire, nous avons ajouté les colonnes « modulus » et « key_size » à notre table crypto à l'aide de la commande dans MySQL workbench :

```
ALTER TABLE crypto
ADD COLUMN modulus TEXT;

ALTER TABLE crypto
ADD COLUMN key_size INT;
```

Pour commencer, nous avons à nouveau créé une connexion à notre base de données à l'aide de :

```
# Connexion à la base de données
db_connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="1234",
    database="mydb"
)
cursor = db_connection.cursor()
```

Par la suite nous avons réalisé la fonction permettant de décoder les certificats et de renvoyer leurs modulus ainsi que la taille de leurs clés :

```
def extract_rsa(cert_pem):
    cert = x509.load_pem_x509_certificate(cert_pem, default_backend())
    public_key = cert.public_key()

    if isinstance(public_key, rsa.RSAPublicKey):
        public_numbers = public_key.public_numbers()
        modulus = public_numbers.n
        exponent = public_numbers.e
        key_size = public_key.key_size
        return modulus, key_size
    else:
        return None, None
```

On récupère les lignes de la table avec leurs id afin de savoir à quelle ligne insérer le modulus et la taille de la clé, on récupère également le certificat au format PEM à décoder :

```
cursor.execute("SELECT idcrypto, c1e FROM crypto")
```

On réalise ensuite une boucle qui va itérer sur chaque ligne de la BDD en passant le certificat PEM à la fonction extract_rsa et en récupérer le modulus et la taille de la clé.

Ensuite si l'on a bien récupéré ces deux éléments, alors on insert leurs valeurs dans la table SQL :

```
i=0
# Parcourir l'ensemble des certificats récolté précédemment
for row in cursor.fetchall():
    cert_id = row[0]
    cert_pem = row[1]

    # Extraire le certificat decoder
    modulus, key_size = extract_rsa_key_details(cert_pem.encode())

    if modulus and key_size:
        #modifier la valeurs de modulus et de key_size pour la ligne traiter
        cursor.execute("""
            UPDATE crypto
            SET modulus = %s, key_size = %s
            WHERE idcrypto = %s
            """, (str(modulus), key_size, cert_id))
        i=i+1
print(i)
```

Nous avons également ajouté un compteur avec i nous permettant de suivre l'avancement.

On termine par un commit de l'ensemble des lignes de la table et la fermeture de la connexion à la BDD :

```
db_connection.commit()
cursor.close()
db_connection.close()
```

Ainsi, on dispose des modulus et des tailles de clés pour les certificats RSA lisible

On obtient la table ci-dessous (il s'agit d'un échantillon) :

	idcrypto	name	de	modulus	key_size
1		test.wjtuzd.info	-----BEGIN CERTIFICATE-----MIE8TCCA9mgAwIBAgISA0kHjo1ZdBjNW...	22212926549002050284514280201842508005...	2048
2		test.www132betorspin.com	-----BEGIN CERTIFICATE-----MIIFHzCCBAegAwIBAgISBFoHNzQsTiQdbc...	26341795199831585535755497284823918730...	2048
3		*.3767565500af4670964b5cc2073c3a87.int.cu...	-----BEGIN CERTIFICATE-----MIIH8zCCBdugAwIBAgITMwEVgUu9u3i6h8...	21235559568598902434197330192657118881...	2048
4		*.st666top1.com	-----BEGIN CERTIFICATE-----MIIDrDCCA1GgAwIBAgIRAKyS88h9gsl4Ec...	NULL	NULL
5		www.wwwwwwwwwwwwdemo.givetogo.org	-----BEGIN CERTIFICATE-----MIIFGCCBACgAwIBAgISBebtT9lvVntfL29...	27310374577416796252415100178019911822...	2048
6		*.go88-sunwin-iwin-b52-stockroms.pages.dev	-----BEGIN CERTIFICATE-----MIIDCCCAq+gAwIBAgIRAIbPqJ0lxV9JDI...	NULL	NULL
7		cpanel.freepalestine369.com	-----BEGIN CERTIFICATE-----MIIFzCCBLSgAwIBAgISBLPmfUV14oaEBz...	19206969234466202265463619435597859714...	2048
8		spencer-weaver.com	-----BEGIN CERTIFICATE-----MIIDhzCCAww2gAwIBAgISA32Gxflmzsjpm2...	NULL	NULL
9		bs-mat-alertmetrics-timeout-ezw-dev.securionp...	-----BEGIN CERTIFICATE-----MIID5DCCA4ugAwIBAgIRAOIPntI7PFJ2EY...	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL

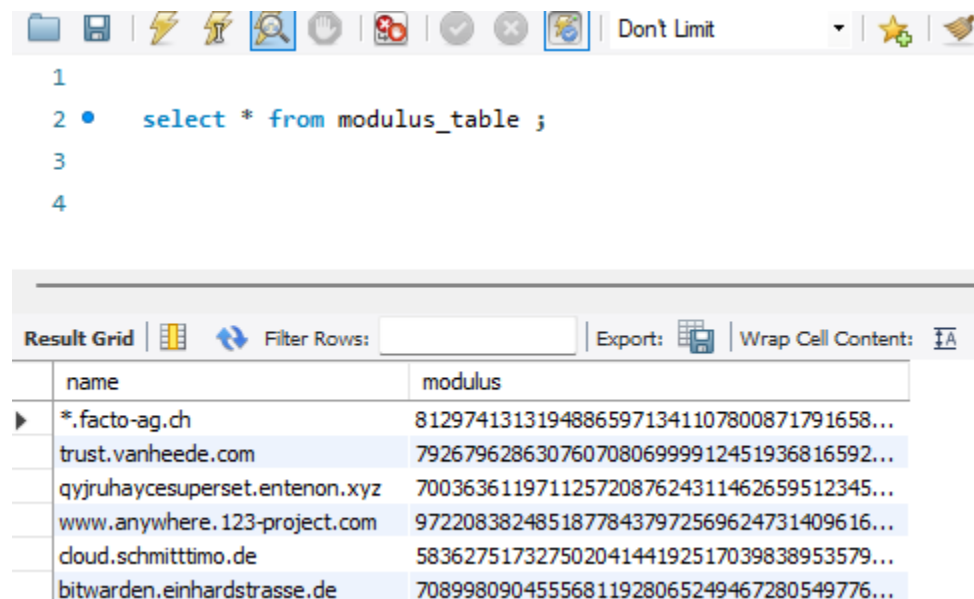
Tri par taille de clés et suppression des doublons

Pour cette partie, nous n'avons utiliser que le SQL ci-dessous :

```
INSERT INTO modulus_table (name, modulus)
SELECT
    MAX(name),
    modulus
FROM crypto
WHERE modulus IS NOT NULL -- Exclure les lignes où modulus est NULL
GROUP BY modulus
ORDER BY key_size DESC
```

Nous avons créer la table modulus_table dans laquelle nous avons insérer uniquement les modulus sur lesquelles nous avons réaliser un group by afin de supprimer les doublons et nous avons trier sur les key_size.

Le résultat obtenu est le suivant :



The image shows a SQL query editor interface. The top toolbar includes icons for file operations, execution, and settings. The query entered is 'select * from modulus_table ;'. Below the query, a 'Result Grid' is displayed with columns 'name' and 'modulus'. The grid contains six rows of data, with the first row selected.

	name	modulus
▶	*.facto-ag.ch	81297413131948865971341107800871791658...
	trust.vanheede.com	79267962863076070806999912451936816592...
	qyjruihaycesuperset.entenon.xyz	70036361197112572087624311462659512345...
	www.anywhere.123-project.com	97220838248518778437972569624731409616...
	cloud.schmittimo.de	58362751732750204144192517039838953579...
	bitwarden.einhardstrasse.de	70899809045556811928065249467280549776...

Conclusion

certstream

Ce projet nous a permis de mettre en pratique des notions que nous avons vues pendant notre cours.

Nous avons réussi à recouper 1 million de certifications et les traiter dans une base de données afin de les décoder.

Cependant nous n'avons pas réussi à mener le projet jusqu'au bout car nous ne trouvons pas de solution fonctionnelle pour la partie certstream.

Nous avons pu également décoder les certificats et les trier selon leur taille ainsi que récupérer leurs modules et supprimer les doublons.

Cependant nous n'avons pas réussi à arriver au bout du projet en appliquant le batch GCD sur les modules obtenus.