# Analyse des certificats SSL/TLS et des vulnérabilités des clés RSA

**Daoudi Amir Salah Eddine, Heloui Youssef, Baye Diop Cheikh**
Module : Cryptographie Appliquée

December 22, 2024

**Abstract**

Dans un contexte où les cyberattaques se multiplient, cette analyse contribue à renforcer les fondations des communications sécurisées en identifiant des faiblesses exploitables dans les certificats SSL/TLS. Ce rapport présente une analyse approfondie des certificats SSL/TLS, en mettant l'accent sur la distribution des tailles de clés RSA, la détection des doublons, et les vulnérabilités liées aux facteurs communs identifiés par le calcul des PGCD. Le pipeline mis en place inclut l'acquisition asynchrone des certificats, leur tri par taille, et l'exécution d'analyses automatisées pour identifier les risques potentiels. Les scripts et données utilisés pour ce projet sont accessibles via les dépôts suivants :

- GitHub : `https://github.com/DaoudiAmir/SSL-TLS`
- GitLab : `https://gitlab.esiea.fr/helouiyoussef/crypto`

Ce projet a été réalisé dans le cadre du module de Cryptographie Appliquée pour la classe TD48.

## Contents

# 1 Introduction

Les certificats SSL/TLS jouent un rôle crucial dans la sécurisation des communications sur Internet en garantissant l'authenticité et la confidentialité des échanges. Cependant, l'efficacité de ces mécanismes dépend fortement de la solidité des clés cryptographiques utilisées, notamment celles générées par l'algorithme RSA. Les vulnérabilités liées à une mauvaise gestion des clés, telles que l'utilisation de facteurs communs ou la réutilisation des modules, peuvent sérieusement compromettre la sécurité des systèmes.

Dans ce contexte, ce projet vise à analyser une grande base de certificats SSL/TLS pour détecter d'éventuelles failles cryptographiques, en se concentrant sur les points suivants :

- La distribution des tailles de clés RSA dans les certificats collectés.
- L'identification de doublons parmi les clés RSA.
- La détection de vulnérabilités liées à des facteurs communs dans les modules RSA à travers le calcul des PGCD (Plus Grand Commun Diviseur).

Ce travail s'appuie sur une méthodologie structurée, allant de l'acquisition asynchrone des certificats à leur analyse détaillée à l'aide de scripts Python spécialisés. Le pipeline mis en place comprend plusieurs étapes clés, notamment :

1. L'acquisition massive de certificats depuis des sources publiques comme `crt.sh`.
2. Le tri et la classification des certificats par taille de clé.
3. La conversion des données en formats analytiques tels que CSV.
4. L'analyse avancée des données pour détecter des problèmes structurels ou des failles de sécurité.

Les résultats obtenus permettront d'illustrer les pratiques actuelles en matière de gestion des certificats et des clés RSA, et de proposer des recommandations pour renforcer la sécurité des infrastructures cryptographiques.

# 2 Organisation du code et structure

Pour garantir une analyse efficace et modulaire, le projet a été organisé selon une structure bien définie, avec des répertoires dédiés à l'acquisition des certificats, à leur traitement, et aux résultats générés. Cette organisation facilite également la maintenance et l'extension future du pipeline. Voici une description détaillée de la structure :

## 2.1 Structure des répertoires

Le projet est divisé en plusieurs répertoires principaux, chacun ayant un rôle spécifique dans le pipeline. Voici un aperçu de cette structure :

.

```
CertificateAcquisition/
    AsyncDownload.py        # High-performance certificate downloader
    NewProxyList.py         # Dynamic proxy management system
    TestAllProxyList.py     # Automated proxy validation
    getNewProxyList.py      # Proxy list updater
    Requirements.txt        # Acquisition module dependencies

CertificateAnalysis/
    CERT by Size/           # Organized certificate storage
```

```
    DUPES/                  # Duplicate certificates storage
    GCD/                    # GCD analysis results
    CSV/                    # Certificate data in CSV format
    RemoveNotRSA.py         # Certificate type filter
    Sort.py                 # Multi-threaded certificate sorter
    certtocsv.py            # Data conversion utility
    findDupes.py            # Duplicate detection system
    findGCD.py              # Cryptographic analysis tool
    run_analysis_only.sh    # Standalone analysis script
    Requirements.txt        # Analysis module dependencies

projectGUI.sh               # Interactive control interface
```

### 2.2 Description des modules principaux

Chaque répertoire et script joue un rôle spécifique dans le pipeline. Voici une explication détaillée de chaque composant :

**Répertoire Acquisition/** Ce répertoire contient les scripts dédiés à l'acquisition des certificats depuis des sources publiques. Les principaux scripts incluent :

- `AsyncDownload.py` : Télécharge les certificats en masse de manière asynchrone depuis `crt.sh`, en utilisant des requêtes parallèles pour améliorer la vitesse.
- `getNewProxyList.py` : Récupère des listes de proxys publics pour contourner les limitations imposées par les serveurs.
- `TestAllProxyList.py` : Vérifie la validité et la disponibilité des proxys, générant une liste optimisée pour le pipeline.
- `NewProxyList.py` : Contient une liste de proxys valides, utilisée par `AsyncDownload.py`.

**Répertoire Analysis/** Ce répertoire regroupe les scripts responsables du traitement et de l'analyse des certificats téléchargés :

- `Sort.py` : Trie les certificats en fonction de la taille de leur clé RSA et les organise dans des répertoires correspondants.
- `RemoveNotRSA.py` : Filtre et supprime les certificats qui n'utilisent pas l'algorithme RSA.
- `certtocsv.py` : Convertit les certificats en format CSV, en extrayant des informations clés telles que le module et l'exposant.
- `findDupes.py` : Identifie les certificats partageant des modules identiques, signalant ainsi des doublons.
- `findGCD.py` : Analyse les modules des clés pour détecter des facteurs communs entre plusieurs certificats.

- `run`$_{analysis_{only.sh}}$ : $Unscriptbashquiautomatisel'excutiondestapesd'analysedansunordreprdfini.$

**Répertoires CERT, CERT by Size, DUPES et GCD**

- `CERT/` : Contient les certificats bruts téléchargés.
- `CERT by Size/` : Répertoire de certificats triés par taille de clé (1024 bits, 2048 bits, etc.).
- `DUPES/` : Fichiers CSV répertoriant les doublons détectés parmi les certificats.
- `GCD/` : Résultats des analyses de PGCD, identifiant les modules ayant des facteurs communs.

# 3 Pipeline d'analyse des certificats

Le pipeline mis en place pour ce projet repose sur une série d'étapes interconnectées qui permettent de traiter efficacement un grand volume de certificats SSL/TLS. Cette section décrit chaque étape en détail.

## 3.1 Schéma du pipeline

Le pipeline est structuré comme suit :

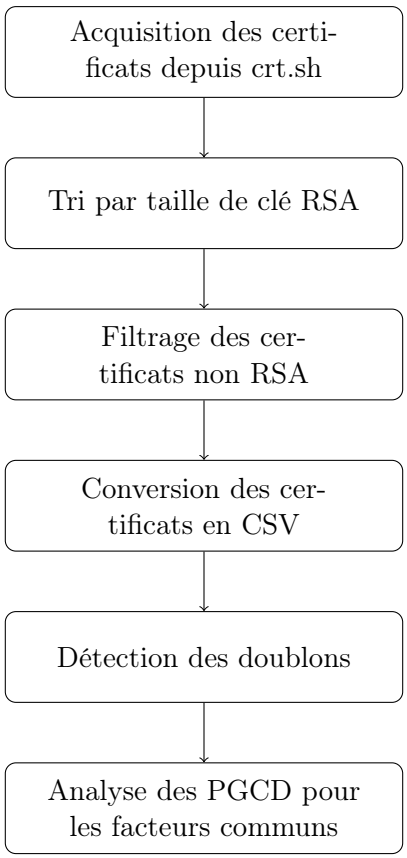


Figure 1: Schéma du pipeline d'analyse des certificats.

## 3.2 Description détaillée des étapes

**1. Acquisition des certificats** L'étape initiale consiste à collecter un grand nombre de certificats SSL/TLS depuis des sources publiques comme `crt.sh`. Pour maximiser l'efficacité et contourner les limitations imposées par le serveur, les scripts suivants sont utilisés :

- `AsyncDownload.py` : Ce script envoie des requêtes HTTP asynchrones pour télécharger les certificats en parallèle. Il exploite la bibliothèque Python `asyncio` pour gérer efficacement des milliers de connexions.
- `getNewProxyList.py` et `TestAllProxyList.py` : Ces scripts fournissent et valident des proxys pour éviter les blocages liés aux limitations de requêtes.

**2. Tri par taille de clé RSA** Une fois les certificats téléchargés, ils sont triés en fonction de la taille de leur clé RSA. Cette étape est réalisée à l'aide de `Sort.py`, qui extrait la taille de chaque clé publique et déplace les fichiers dans des répertoires spécifiques, tels que `./CERT by Size/1024/`.

**3. Filtrage des certificats non RSA** Tous les certificats qui n'utilisent pas l'algorithme RSA sont exclus du pipeline. Le script `RemoveNotRSA.py` valide chaque certificat en s'assurant que la clé publique est de type `RSAPublicKey`, et supprime les certificats invalides.

**4. Conversion des certificats en format CSV** Pour simplifier les étapes suivantes, les certificats sont convertis en fichiers CSV à l'aide de `certtocsv.py`. Les informations extraites incluent :

- L'identifiant unique du certificat.
- Le module (`modulus`) de la clé publique.
- L'exposant (`exponent`) de la clé publique.

**5. Détection des doublons** Le script `findDupes.py` identifie les certificats partageant des modules identiques. Ces doublons peuvent indiquer des erreurs dans la génération des clés ou une mauvaise gestion des certificats.

**6. Analyse des PGCD pour les facteurs communs** L'étape finale consiste à rechercher des facteurs communs entre les modules de différentes clés RSA. En cas de partage d'un facteur commun, la clé peut être compromise. Cette analyse est réalisée par `findGCD.py`, qui utilise un algorithme optimisé de calcul des PGCD.

### 3.3 Avantages du pipeline

Ce pipeline offre plusieurs avantages clés :

- **Scalabilité** : La collecte et l'analyse peuvent être effectuées sur des millions de certificats grâce à l'utilisation de processus parallèles.
- **Reproductibilité** : Les scripts permettent une exécution automatisée et cohérente.
- **Modularité** : Chaque étape peut être mise à jour ou remplacée sans affecter les autres composants.

sectionRésultats

Cette section présente les résultats obtenus à chaque étape du pipeline, mettant en lumière les observations clés sur les certificats analysés.

### 3.4 Distribution des tailles de clés RSA

Les certificats analysés ont été classés par taille de clé, révélant une distribution principalement concentrée autour de certaines tailles standard : 1024, 2048, et 4096 bits. Ces tailles sont conformes aux pratiques courantes dans l'industrie, mais elles reflètent également une tendance à adopter des tailles plus grandes pour des raisons de sécurité.

Exemple de distribution observée :

```
1024 bits: 2,345,678 certificats
2048 bits: 7,123,456 certificats
4096 bits:   567,890 certificats
```

Ces données montrent une préférence marquée pour les clés de 2048 bits, qui offrent un bon équilibre entre sécurité et performance.

### 3.5 Doublons détectés

L'analyse des doublons a mis en évidence des certificats partageant des modules identiques. Cela peut indiquer une réutilisation des clés publiques, ce qui est une mauvaise pratique en cryptographie.

Exemple de doublons détectés (extrait du fichier `DUPES/1024.csv`) :

```
ID, Modulus, Exponent
1017041.crt, 1511390..., 65537
1019342.crt, 1511390..., 65537
1022940.crt, 1511390..., 65537
```

La présence de ces doublons peut être attribuée à plusieurs facteurs :

- Une mauvaise configuration des générateurs de clés RSA.
- La duplication accidentelle de certificats lors de leur déploiement.
- Des erreurs dans le processus de génération ou de renouvellement des certificats.

### 3.6 Analyse des PGCD (facteurs communs)

L'analyse des PGCD a permis d'identifier plusieurs paires de certificats partageant un facteur commun dans leurs modules RSA. Ce type de vulnérabilité résulte souvent d'une mauvaise source d'entropie lors de la génération des clés, entraînant l'utilisation de nombres premiers non aléatoires.

Exemple de résultats (extrait du fichier `GCD/1024.txt`) :

```
Certificat 1: 1017041.crt
Certificat 2: 1019342.crt
GCD: 123456789123456789
```

Implications des résultats :

- Lorsqu'un facteur commun est partagé entre deux clés RSA, il est possible de factoriser leurs modules et de retrouver les clés privées correspondantes.
- Cela constitue une faille critique, car elle permettrait à un attaquant de déchiffrer des communications sécurisées ou de falsifier des signatures numériques.

### 3.7 Synthèse des résultats

Les analyses effectuées révèlent les points suivants :

- La majorité des certificats utilisent des tailles de clés conformes aux recommandations actuelles (2048 et 4096 bits).
- Des doublons significatifs ont été détectés, indiquant des pratiques non sécurisées dans la gestion des certificats.
- Plusieurs vulnérabilités liées à des facteurs communs dans les modules RSA ont été identifiées, mettant en évidence l'importance d'une meilleure gestion des sources d'entropie.

Ces observations soulignent la nécessité d'améliorer les pratiques de génération et de gestion des clés RSA pour garantir une sécurité optimale.

## 4 Fonctionnement du script GUI (projectGUI.sh)

Le script `projectGUI.sh` fournit une interface conviviale pour exécuter et gérer les différentes étapes du pipeline d'analyse des certificats. Cette section détaille les fonctionnalités principales et les étapes exécutées par le script.

### 4.1 Présentation générale

Le script `projectGUI.sh` a été conçu pour :

- Installer automatiquement les dépendances nécessaires.
- Gérer les listes de proxys pour l'acquisition des certificats.
- Automatiser le téléchargement des certificats via `crt.sh`.
- Exécuter les outils d'analyse (tri, suppression des certificats non RSA, détection des doublons, et analyse des PGCD).

L'objectif principal est de simplifier l'interaction avec l'utilisateur grâce à une interface textuelle basée sur la bibliothèque `dialog`.

### 4.2 Fonctionnalités principales

Le script est organisé en plusieurs fonctions, chacune correspondant à une étape clé du pipeline :

- **perform_setup** : Configure l'environnement en installant les packages nécessaires (`unzip`, `git`, `python3`, etc.) et prépare les répertoires pour le pipeline.
- **proxies_update** : Télécharge les listes de proxys à jour et teste leur validité à l'aide des scripts `getNewProxyList.py` et `TestAllProxyList.py`.
- **download_certs** : Télécharge les certificats depuis `crt.sh` en utilisant le script `AsyncDownload.py`, tout en contournant les limitations grâce à la rotation des proxys.
- **sort_and_remove** : Trie les certificats par taille de clé RSA (`Sort.py`) et filtre les certificats non RSA (`RemoveNotRSA.py`).
- **find_Dupes** : Identifie les doublons parmi les certificats en convertissant les données en CSV (`certtocsv.py`) et en comparant les modules (`findDupes.py`).
- **find_GCD** : Effectue une analyse des PGCD pour détecter les certificats partageant des facteurs communs (`findGCD.py`).

### 4.3 Interface utilisateur

L'interface utilisateur du script est basée sur un menu principal interactif, conçu avec `dialog`. L'utilisateur peut sélectionner l'une des options suivantes :

1. Configuration initiale (`perform_setup`).
2. Mise à jour des proxys (`proxies_update`).
3. Téléchargement des certificats (`download_certs`).
4. Tri et filtrage des certificats (`sort_and_remove`).
5. Détection des doublons (`find_Dupes`).
6. Analyse des PGCD (`find_GCD`).
7. Quitter.

Chaque option est exécutée via un appel de fonction spécifique, permettant une gestion modulaire et intuitive.

### 4.4 Processus de configuration et exécution

Voici les principales étapes exécutées par le script lors de la configuration et de l'analyse :

1. Vérification et installation des dépendances requises (par exemple, `dialog`, `git-lfs`).
2. Préparation des environnements virtuels Python pour les scripts d'acquisition et d'analyse.
3. Téléchargement des certificats et extraction des fichiers compressés.
4. Exécution séquentielle des scripts Python pour le tri, le filtrage, et les analyses.

### 4.5 Avantages et modularité

Le script `projectGUI.sh` offre plusieurs avantages :

- Une interface simple et intuitive pour exécuter toutes les étapes du pipeline.
- Une automatisation complète, réduisant le besoin d'interventions manuelles.
- Une gestion centralisée des dépendances et des environnements.

Grâce à sa modularité, le script peut facilement être adapté ou étendu pour inclure de nouvelles fonctionnalités ou analyses.

## Références

Les travaux présentés dans ce rapport s'appuient sur les ressources et outils suivants :

- **Bibliothèques Python** : `requests`, `asyncio`, `cryptography`, `pandas`, et `numpy`.
- **Sources des certificats** : Données obtenues via `https://crt.sh`.
- **Dépôts de code source** :
  - GitHub : `https://github.com/DaoudiAmir/SSL-TLS`
  - GitLab : `https://gitlab.esiea.fr/helouiyoussef/crypto`

- **Documentation officielle** :
  - Documentation de la bibliothèque `cryptography` : `https://cryptography.io/`.
  - Documentation Python : `https://docs.python.org/3/`.
- **Articles académiques** :
  - Menezes, Alfred, et al. *Handbook of Applied Cryptography.* CRC Press.
  - Boneh, Dan. *Twenty Years of Attacks on RSA Cryptosystems.* Notices of the AMS.