# Un coffre-fort numérique simplifié

## GS15 - A24 - Projet Informatique

Thomas DEVINCENZI & Hamza LAKHAL

A24

utt
UNIVERSITÉ DE TECHNOLOGIE
TROYES

# SOMMAIRE

# Introduction

Ce rapport présente le développement d'un coffre-fort numérique permettant de stocker et gérer des fichiers sensibles de manière sécurisée. Ce projet vise à implémenter des techniques cryptographiques, notamment RSA, ZKP (Zero-Knowledge Proof), Diffie-Hellman, Cobra, ainsi qu'un algorithme de hachage personnalisé. Toutes les étapes sont étudiées et mises en œuvre dans un environnement Python, respectant les restrictions imposées sur l'utilisation de bibliothèques externes de cryptographie.

# 1. Fonctionnalités principales

## 1.1 Création de compte utilisateur

- **Objectif :** Permettre à un utilisateur de créer un compte protégé par un couple de clés RSA, garantissant la sécurité de ses fichiers.
- **Processus :**
  - L'utilisateur choisit un mot de passe, qui est utilisé pour dériver une clé privée RSA via une fonction de dérivation (KDF).
  - Une clé publique correspondante est générée.
  - La clé publique est stockée dans le coffre-fort, tandis que la clé privée est conservée dans un espace utilisateur sécurisé.
  - Un certificat utilisateur est généré pour assurer l'authenticité du coffre fort.
- **Stockage des données :**
  - Clé publique : /coffre_fort/<nom_utilisateur>/public_key.key
  - Clé privée : /users/<nom_utilisateur>/private_key.key
  - Certificat : /users/<nom_utilisateur>/certificat.txt
- **Implémentation :**
  - La génération des clés est gérée par generation_cle.py, utilisant des nombres premiers générés avec Rabin-Miller.
  - Le certificat est créé dans certificat_coffre.py.

## 1.2 Connexion au coffre

- **Objectif :** Authentifier un utilisateur pour accéder à son espace sécurisé dans le coffre-fort.
- **Processus :**
  1. L'utilisateur entre son identifiant, permettant de localiser ses fichiers et certificats.
  2. Le certificat utilisateur est vérifié pour s'assurer qu'il est valide et non expiré afin de vérifier l'authenticité du coffre fort.

3. Une preuve de possession de la clé privée est réalisée via le protocole Guillou-Quisquater.
4. Une clé de session est négociée avec le coffre-fort via le protocole Diffie-Hellman.

- **Implémentation :**
  - La gestion de la vérification des certificats et ZKP est définie dans Guillou_Quisquater.py et certificat_coffre.py.

## 1.3 Gestion des fichiers

- **Objectif :** Permettre un dépôt et une récupération sécurisés des fichiers par l'utilisateur.
- **Processus détaillé :**
  1. **Dépôt de fichier :**
     - L'utilisateur place un fichier en clair dans son espace personnel (/users/<nom_utilisateur>).
     - Le fichier est chiffré avec l'algorithme COBRA, puis déplacé vers le coffre-fort (/coffre_fort/<nom_utilisateur>).
     - Un hash du fichier est calculé et stocké pour assurer son intégrité avant et après le transport.
  2. **Récupération de fichier :**
     - Le fichier est extrait du coffre-fort et déchiffré à l'aide de la clé privée de l'utilisateur.
     - L'intégrité est vérifiée en comparant le hash initial et final.
- **Implémentation :**
  - Les opérations de chiffrement et de gestion des fichiers sont réalisées dans cobra.py et RSA.py.

## 1.4 Journalisation

- **Objectif :** Assurer une traçabilité complète des actions pour chaque utilisateur.
- **Processus :**
  - Chaque événement, comme la création de compte, les connexions et les transferts de fichiers, est enregistré dans un fichier de log.
- **Implémentation :**
  - La journalisation est gérée par le module log.py.
  - Les logs sont stockés dans le fichier log.txt dans le coffre fort

# 2. Détails techniques

## 2.1 Algorithme de chiffrement COBRA

- **Caractéristiques :**

    - Basé sur l'algorithme Serpent, modifié pour inclure des S-Boxes personnalisées et des étapes supplémentaires, comme une transformation Feistel.
    - Fonctionne sur des blocs de 128 bits avec des clés jusqu'à 256 bits.
- **Étapes principales :**
    - Substitution via des S-Boxes.
    - Transformation Feistel utilisant des permutations et des XOR complexes.
    - Ajout de clés de tour générées dynamiquement.
- **Implémentation :**
    - Définie dans cobra.py, elle inclut également des fonctions pour le chiffrement et le déchiffrement des messages et fichiers.

## 2.2 Chiffrement asymétrique RSA

- **Utilisation :**
    - Chiffrement des fichiers avant stockage dans le coffre.
    - Déchiffrement lors de la récupération.
    - Implémentation de padding PKCS#1 v1.5 pour sécuriser les blocs.
- **Détails :**
    - La génération des clés RSA est assurée par generation_cle.py, utilisant des modules mathématiques pour garantir la robustesse des clés.
    - Les fichiers sont chiffrés par blocs, comme décrit dans RSA.py.

## 2.3 Fonction de hachage et HMAC

- **Objectif :** Garantir l'intégrité des fichiers tout au long des opérations.
- **Détails techniques :**
    - Basée sur une construction Merkle-Damgård, la fonction de hachage inclut des permutations et des XOR complexes.
    - Implémentée dans Merkle.py, elle est utilisée pour calculer des hash et des HMAC lors des dépôts et récupérations.

## 2.4 Preuve à divulgation nulle (ZKP)

- **Protocole utilisé :**
    - Guillou-Quisquater, qui prouve la possession de la clé privée sans la révéler.
- **Détails :**
    - Le protocole inclut la génération de challenges et de réponses vérifiables.
    - Implémentation dans Guillou_Quisquater.py.

# 3. Résultats et démonstration

## 3.1 Exemple de scénario utilisateur

1. **Création de compte :**
    - L'utilisateur "Alice" choisit un mot de passe. Une clé publique est stockée dans /coffre_fort/Alice, et une clé privée dans /users/Alice.
    - Un certificat est généré et validé.
2. **Connexion :**
    - Alice entre son identifiant, vérifie son certificat, et s'authentifie via ZKP.
    - Une clé de session est établie avec le coffre-fort pour sécuriser les échanges.
3. **Dépôt de fichier :**
    - Alice place un fichier "document.txt" en clair dans /users/Alice.
    - Le fichier est chiffré avec COBRA, déplacé dans /coffre_fort/Alice, et son hash est calculé.
4. **Vérification d'intégrité :**
    - Lors de la récupération, le hash du fichier est recalculé et comparé pour garantir l'absence de modification.

# 4. Conclusion

- **Objectifs atteints :**
    - Création d'un coffre-fort numérique fonctionnel et sécurisé.
    - Mise en œuvre réussie de techniques cryptographiques avancées.
    - Gestion complète des fichiers avec traçabilité assurée.
- **Perspectives d'amélioration :**
    - Développement d'une interface utilisateur graphique pour améliorer l'expérience utilisateur.
    - Intégration d'un système de partage sécurisé entre utilisateurs.
    - Optimisation de l'algorithme COBRA pour des performances accrues.