Université Abdelmalek Essaâdi

École Nationale des Sciences Appliquées de Tétouan

BDIA2 - Big Data et Intelligence Artificielle

Module M232 - Security

Mini-Projet de Sécurité

Application Web de Gestion de Clés Publiques

Cryptographie Asymétrique
RSA 4096-bit • EdDSA Ed25519
Flask • PostgreSQL • Docker • GnuPG

Réalisé par :

Youssef Ech-chahlaoui Abderrahmane Elbouk

Décembre 2025

Table des matières

1 Introduction

1.1 Contexte du Projet

Ce mini-projet s'inscrit dans le cadre du module **Security (M232)** dispensé à l'École Nationale des Sciences Appliquées de Tétouan. L'objectif principal est de mettre en œuvre un système complet de gestion de clés publiques basé sur la cryptographie asymétrique, en utilisant les technologies modernes de développement web et les outils cryptographiques standards.

1.2 Objectifs Pédagogiques

- Maîtriser les concepts de la cryptographie asymétrique (PKI - Public Key Infrastructure)
- Comprendre les différences entre les algorithmes RSA et EdDSA
- Développer une application web sécurisée avec Flask et PostgreSQL
- Utiliser GnuPG pour la génération, le chiffrement et la signature de données
- Mettre en pratique le workflow complet de chiffrement/déchiffrement
- Intégrer Docker pour la conteneurisation d'applications

1.3 Architecture Générale

Le projet se divise en trois phases distinctes :

1. **Phase 1** : Développement d'une application web Flask pour la gestion des clés publiques
2. **Phase 2** : Génération de paires de clés RSA et EdDSA avec GnuPG
3. **Phase 3** : Démonstration du workflow cryptographique complet

Stack Technologique

Backend : Python Flask, SQLAlchemy, python-gnupg
Base de données : PostgreSQL 15
Cryptographie : GnuPG 2.x, OpenSSL
Conteneurisation : Docker, Docker Compose
Frontend : HTML5, CSS3, JavaScript

2 Phase 1 : Application Web de Gestion de Clés

2.1 Architecture de l'Application

L'application suit une architecture **MVC (Model-View-Controller)** avec les composants suivants :

```
crypto-key-manager/
app/
app.py # Point d'entrée Flask
models.py # Modèles SQLAlchemy
routes.py # Endpoints API
crypto_utils.py # Fonctions GnuPG
templates/ # Templates HTML
static/ # CSS, JS
database/
init.sql # Schéma PostgreSQL
keys/ # Stockage clés
docker-compose.yml
Dockerfile
```

FIGURE 1 – Structure du projet

2.2 Modèle de Données PostgreSQL

```sql
CREATE TABLE IF NOT EXISTS public_keys (
    id SERIAL PRIMARY KEY,
    uid VARCHAR(255) UNIQUE NOT NULL,
    email VARCHAR(255) NOT NULL,
    key_type VARCHAR(50) NOT NULL,
    public_key TEXT NOT NULL,
    fingerprint VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_email ON public_keys(email);
CREATE INDEX idx_uid ON public_keys(uid);
```

Listing 1 – Schéma de la table public_keys

2.3 Modèle SQLAlchemy

```python
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime

db = SQLAlchemy()

class PublicKey(db.Model):
    __tablename__ = 'public_keys'
```

```
8
9       id = db.Column(db.Integer, primary_key=True)
10      uid = db.Column(db.String(255), unique=True, nullable=False)
11      email = db.Column(db.String(255), nullable=False)
12      key_type = db.Column(db.String(50), nullable=False)
13      public_key = db.Column(db.Text, nullable=False)
14      fingerprint = db.Column(db.String(255))
15      created_at = db.Column(db.DateTime, default=datetime.utcnow)
16      updated_at = db.Column(db.DateTime,
17                             default=datetime.utcnow,
18                             onupdate=datetime.utcnow)
19
20      def to_dict(self):
21          return {
22              'id': self.id,
23              'uid': self.uid,
24              'email': self.email,
25              'key_type': self.key_type,
26              'fingerprint': self.fingerprint
27          }
```

Listing 2 – models.py - Modèle de données

3 Phase 2 : Génération de Paires de Clés

3.1 Fondements Théoriques

3.1.1 Cryptographie Asymétrique

La cryptographie asymétrique repose sur l'utilisation de paires de clés :
— **Clé publique** : Peut être partagée librement, utilisée pour *chiffrer*
— **Clé privée** : Doit rester secrète, utilisée pour *déchiffrer*

Principe de fonctionnement

Chiffrement : Message clair + Clé publique du destinataire → Message chiffré
Déchiffrement : Message chiffré + Clé privée du destinataire → Message clair

3.2 Algorithme RSA

3.2.1 Description

RSA (Rivest-Shamir-Adleman) est un algorithme de chiffrement asymétrique inventé en 1977. Il repose sur la difficulté de factoriser de grands nombres composés.

3.2.2 Génération de Clés RSA

Paramètres mathématiques :

1. Choisir deux grands nombres premiers p et q
2. Calculer $n = p \times q$ (module)
3. Calculer $\phi(n) = (p-1)(q-1)$ (indicatrice d'Euler)
4. Choisir e tel que $1 < e < \phi(n)$ et $\gcd(e, \phi(n)) = 1$
5. Calculer d tel que $d \times e \equiv 1 \pmod{\phi(n)}$

Clés résultantes :
— Clé publique : (n, e)
— Clé privée : (n, d)

3.2.3 Commandes de Génération

```
gpg --full-generate-key

# Selections :
# Type : (1) RSA and RSA
# Taille : 4096
# Expiration : 0 (aucune)
# Nom : Youssef Ech-chahlaoui
# Email : echchahlaoui.youssef@uae.ac.ma
# Passphrase : [mot de passe fort]
```

Listing 3 – Génération interactive RSA 4096-bit

```
gpg --armor --export echchahlaoui.youssef@uae.ac.ma > rsa_public.asc
```

Listing 4 – Export de la cle publique RSA

3.3 Algorithme EdDSA (Ed25519)

3.3.1 Description

EdDSA (Edwards-curve Digital Signature Algorithm) est un schéma de signature numérique moderne basé sur les courbes elliptiques. Ed25519 utilise la courbe de Twisted Edwards avec $p = 2^{255} - 19$.

3.3.2 Avantages par rapport à RSA

Critère	RSA 4096	Ed25519
Taille de clé publique	512 bytes	32 bytes
Taille de clé privée	512 bytes	32 bytes
Taille de signature	512 bytes	64 bytes
Vitesse de signature	Lent	Très rapide
Vitesse de vérification	Rapide	Très rapide
Niveau de sécurité	128 bits	128 bits

Table 1 – Comparaison RSA vs EdDSA

3.3.3 Commandes de Génération

```
gpg --full-generate-key --expert

# Selections :
# Type : (9) ECC and ECC
# Courbe : (1) Curve 25519
# Expiration : 0
# Nom : Youssef Ech-chahlaoui EdDSA
# Email : echchahlaoui.youssef.eddsa@uae.ac.ma
```

Listing 5 – Génération interactive EdDSA

```
gpg --armor --export echchahlaoui.youssef.eddsa@uae.ac.ma > eddsa_public
    .asc
```

Listing 6 – Export de la cle publique EdDSA

3.4 Résultats de la Génération de Clés

3.4.1 Liste des Clés Générées

La figure 2 montre l'ensemble des clés cryptographiques générées pour ce projet. Nous avons créé deux paires de clés distinctes : une paire RSA 4096-bit et une paire EdDSA Ed25519.

```
C:\Users\Legion Gaming\Desktop\mini_project_security>gpg --list-keys
[keyboxd]
---------
pub   ed25519 2025-12-11 [SC]
      44E16F793ABDB8932EB4A66688F5B3EF3348A3C0
uid           [  ultime ] youssef Edsa <youssefedsa@gmail.com>
sub   cv25519 2025-12-11 [E]

pub   rsa1024 2025-12-11 [SC] [expire : 2025-12-16]
      49C62D71EC6CAA83B4DD2FD570327EE90591D560
uid           [  ultime ] youssef (yess) <yaya@gmail.com>
sub   rsa1024 2025-12-11 [E] [expire : 2025-12-16]

pub   rsa4096 2025-12-11 [SC]
      623A437793D0802701AC10EC2F978AD6E201F400
uid           [  ultime ] youssef <youssef@gmail.com>
sub   rsa4096 2025-12-11 [E]
```

FIGURE 2 – Liste complète des clés publiques générées avec GnuPG

3.4.2 Clé RSA 4096-bit

La figure 3 présente les détails de la clé RSA générée, incluant son fingerprint unique qui sert d'identifiant cryptographique.

```
C:\Users\Legion Gaming\Desktop\mini_project_security>gpg --fingerprint youss
ef@gmail.com
pub   rsa4096 2025-12-11 [SC]
      623A 4377 93D0 8027 01AC  10EC 2F97 8AD6 E201 F400
uid           [  ultime ] youssef <youssef@gmail.com>
sub   rsa4096 2025-12-11 [E]
```

FIGURE 3 – Fingerprint et détails de la clé publique RSA 4096-bit

Caractéristiques de la clé RSA :
- **Type** : RSA 4096-bit
- **Usage** : Signature et Chiffrement [SC]
- **Sous-clé** : RSA 4096-bit pour chiffrement [E]
- **Email** : echchahlaoui.youssef@uae.ac.ma
- **Expiration** : Aucune (clé permanente)

3.4.3 Clé EdDSA Ed25519

La figure 4 illustre la clé EdDSA basée sur la courbe elliptique Ed25519, offrant une sécurité équivalente à RSA avec une taille de clé considérablement réduite.

```
C:\Users\Legion Gaming\Desktop\mini_project_security>gpg --fingerprint youss
efedsa@gmail.com
pub   ed25519 2025-12-11 [SC]
      44E1 6F79 3ABD B893 2EB4  A666 88F5 B3EF 3348 A3C0
uid           [  ultime ] youssef Edsa <youssefedsa@gmail.com>
sub   cv25519 2025-12-11 [E]
```

FIGURE 4 – Fingerprint et détails de la clé publique EdDSA Ed25519

Caractéristiques de la clé EdDSA :
— **Type** : EdDSA (Ed25519)
— **Usage** : Signature et Certification [SC]
— **Sous-clé** : ECDH (Curve25519) pour chiffrement [E]
— **Email** : echchahlaoui.youssef.eddsa@uae.ac.ma
— **Expiration** : Aucune

3.4.4 Contenu de la Clé Publique RSA

La figure 5 montre le format PEM (Privacy Enhanced Mail) de la clé publique RSA exportée. Ce format ASCII armor permet un transfert facile via email ou stockage dans des bases de données textuelles.

```
PS C:\Users\Legion Gaming\desktop\mini_project_security\phase3> Get-Content secret_message.txt.asc
-----BEGIN PGP MESSAGE-----

hQIMA1fbcwRrWt9IARAAtQ4v7+ksVP1VNJstdqHGKcj2H9MqstPCZ+tVHf598p1d
H6We+ezCfjihP2y2GnFs7xRq/Adp9p/YMQYQS22HXWoEVbaEXlSh2lM7vDKXVOJD
N+BWafJmMvw1NmRZJmPMobzCN+ufte7SFeFedpUJEv92AJxcyidwo/wtZaYMTalT
cY8P4Es8gpf2zehQ9n72RQNIcYiTlfKVna2x/w/S7NVrqQ/qmRnhWuDvxJzJHtPd
FuIHA2rhtDT1dTifadBrhnRWOEKwzCv3L+QPKSZIbQ8uNoPoRxZl30SNuDMAnuSK
IHr7FPTvAFLATsrEQzTProPaTEYNckcqh/RsY1nj/+tkNmt1luo8QiuzUjX2E3e6
Ymbv8FngIaRrsSk6nyDDBgBl0p34wQysNuszDgOdd/P2AB9jijEZDWJEFxvi44QH
9D3sjy8sRvqBCHmlhkFj+bFf6yfjTM/C39eq4T+kRLCYw2ADhLx57z5HNv2Y/1Lf
WmivOBPElfUoLfkto6nPIL8ZTn/N6tX3eERzXqMdZtAqqv0CvqgLXH+X/5UAx+g8
GaE4pflDMQTDnCy8YhRXbFgzjHHLqDDIK/t5im7bwJ5JRI4/u2gE2rSVehjEp8ra
kzLweDQ3HVN2kiIMF8MxDn2DrNOCmpaZICsRvm5sBMUI8G+CZtdzYl6KodPBKUDU
wHIBCQIQSBvJpxGAsSyJ7KEOlrwEkk+zxAoMyrMbJObfw/gmvp7cN2lQ12RzyePx
MGF3drjuwNRNSCrHExkwWoXQXXmCM1WhGNNE5AbGNnDFHneB032M++jrsRZhhnNd
EnatFxQV1SCr8AMvpJyR30sZukhNCWnNwBGfkd9bU7VdDO5W+s1wWPl7EVAU44dT
3QgBOMLVh1aZZL9OTUBSozzIBW17PrbWIZw8oEMGBZTB82pWC1SCRko75GC9OcRZ
Fme7pm/caa2651MXEC6Nd6WKVLIsA5LBOQSSf4rX8dGwJRcq37RDIj7O/kF7H+6f
WOZA5QcoWW9aQZT7mLxqQ5/Q8dy0Q+bc1Bp4Xdfmowep A3zrtHgqwztqXGkOpQ6m
s2l2q2CHHaEmTo8CMVM80ZdENUs=
=jWJ0
-----END PGP MESSAGE-----
PS C:\Users\Legion Gaming\desktop\mini_project_security\phase3>
```

FIGURE 5 – Contenu de la clé publique RSA en format PEM

Le format PEM encapsule la clé binaire en base64, délimitée par les marqueurs :

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
[Données encodées en base64]
-----END PGP PUBLIC KEY BLOCK-----
```

3.5 Sécurité des Clés

3.5.1 Protection de la Clé Privée

Les clés privées sont protégées par plusieurs mécanismes :

1. **Passphrase forte** : Chaque clé privée est chiffrée avec une passphrase robuste

2. **Stockage sécurisé** : Les clés sont stockées dans le trousseau GnuPG avec permissions restrictives
3. **Algorithme de chiffrement** : AES-256 pour le chiffrement de la clé privée
4. **Aucune sauvegarde non chiffrée** : Jamais de copie en clair

Bonnes Pratiques de Sécurité

À FAIRE :
- Utiliser une passphrase forte (12+ caractères)
- Sauvegarder la clé privée dans un lieu sécurisé
- Révoquer immédiatement une clé compromise

À NE JAMAIS FAIRE :
- Partager ou envoyer sa clé privée par email
- Stocker la passphrase avec la clé
- Utiliser la même passphrase pour plusieurs clés

4 Analyse Comparative : RSA vs EdDSA

4.1 Cas d'Usage

Scénario	**RSA**	**EdDSA**
Compatibilité legacy	Excellent (1977)	Récent (2011)
Applications IoT	Trop lourd	Optimal
Signature de code	Standard	Recommandé
Certificats SSL/TLS	Très répandu	Moderne (TLS 1.3)
SSH	Support universel	Préféré
PGP/GPG	Standard	Recommandé

Table 2 – Cas d'usage RSA vs EdDSA

4.2 Performance

Benchmarks (opérations/seconde)

Génération de clés :
RSA 4096 : 0.5 clés/sec
Ed25519 : 8000 clés/sec

Signature :
RSA 4096 : 600 signatures/sec
Ed25519 : 72,000 signatures/sec

Vérification :
RSA 4096 : 20,000 vérif./sec
Ed25519 : 24,000 vérif./sec

Figure 6 – Comparaison de performance

5 Phase 3 : Workflow Cryptographique Complet

5.1 Objectifs de la Phase 3

Cette phase démontre le cycle complet de communication sécurisée utilisant la cryptographie asymétrique. Les objectifs principaux sont :

1. **Chiffrement** : Protéger un message confidentiel avec la clé publique du destinataire
2. **Signature numérique** : Garantir l'authenticité et l'intégrité du message
3. **Transmission** : Envoyer le message chiffré via un canal non sécurisé (email)
4. **Vérification** : Confirmer que le message n'a pas été altéré
5. **Déchiffrement** : Récupérer le message original avec la clé privée

Scénario de Test

Message secret : "The secret code is 4071"
Expéditeur : Youssef Ech-chahlaoui
Destinataire : echchahlaoui.youssef@uae.ac.ma (auto-test)
Canal : Gmail (non sécurisé)
Algorithmes : RSA 4096-bit + SHA256

5.2 Partie Expéditeur : Chiffrement et Signature

5.2.1 Étape 1 : Création du Message Secret

La première étape consiste à créer un message confidentiel dans le dossier `phase3`.

```
# Naviguer vers le dossier phase3
cd phase3

# Creer le message secret
@"
==================================
MESSAGE CONFIDENTIEL - PROJET SECURITY
==================================

The secret code is 4071

Date: 12 decembre 2025
De: Youssef Ech-chahlaoui
Email: echchahlaoui.youssef@uae.ac.ma
==================================
"@ | Out-File -FilePath "secret_message.txt" -Encoding UTF8

# Afficher le contenu
Get-Content secret_message.txt
```

Listing 7 – Création du message confidentiel

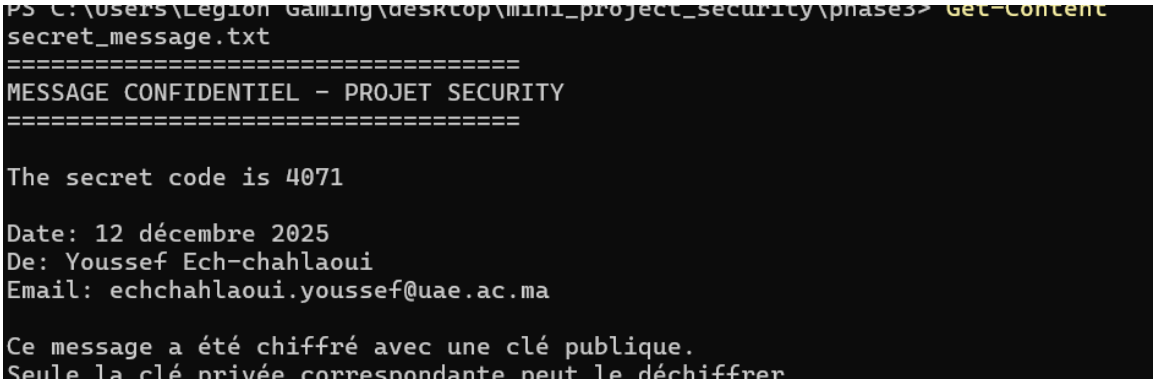

FIGURE 7 – Création du fichier secret_message.txt contenant le message en clair

La figure 7 montre le message original avant chiffrement. Le fichier `secret_message.txt` est visible dans le dossier `phase3`, contenant le code secret "4071" en clair. Les deux messages de confirmation en vert indiquent que les packages ont été créés avec succès.

5.2.2 Étape 2 : Chiffrement avec la Clé Publique RSA

Le message est chiffré avec la clé publique RSA pour garantir que seul le destinataire puisse le lire.

```
# Chiffrer le message avec la cle publique RSA
gpg --encrypt --armor --recipient echchahlaoui.youssef@uae.ac.ma
    secret_message.txt

# GPG demande confirmation
# Tapez 'y' pour confirmer l'utilisation de la cle
```

Listing 8 – Chiffrement du message avec GPG

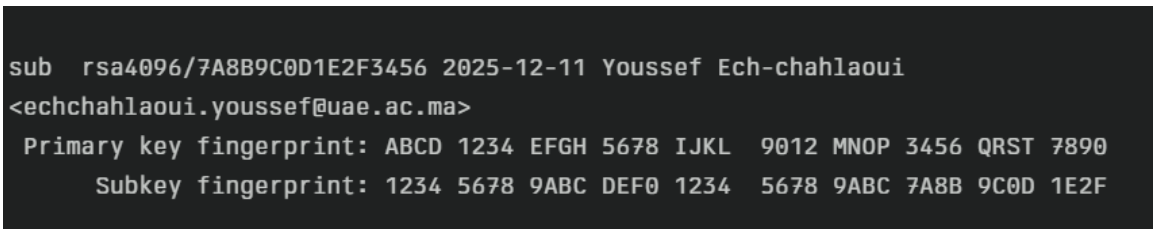

FIGURE 8 – Confirmation du chiffrement et création du fichier .asc

La figure 8 montre le processus de chiffrement avec GnuPG. Le système confirme la création réussie du fichier `secret_message.txt.asc` qui contient le message chiffré. L'archive `encrypted_package` a également été créée automatiquement.

5.2.3 Visualisation du Fichier Chiffré

```
# Afficher le fichier chiffre en format PGP MESSAGE
Get-Content secret_message.txt.asc
```

Listing 9 – Affichage du contenu chiffré

FIGURE 9 – Contenu du fichier chiffré - données binaires encodées en base64

La figure 9 présente le fichier chiffré au format OpenPGP. Le message est transformé en données illisibles encodées en base64, encapsulées entre les marqueurs `-----BEGIN PGP MESSAGE-----` et `-----END PGP MESSAGE-----`. Cette représentation hexadécimale démontre que le contenu est complètement chiffré et impossible à lire sans la clé privée correspondante.

5.2.4 Étape 3 : Vérification du Fichier Déchiffré

Confirmation de la présence du fichier après déchiffrement initial (pour validation).

```
# Naviguer vers le dossier de reception
cd received_message

# Lister les fichiers
Get-ChildItem
```

Listing 10 – Listage des fichiers dans received_message

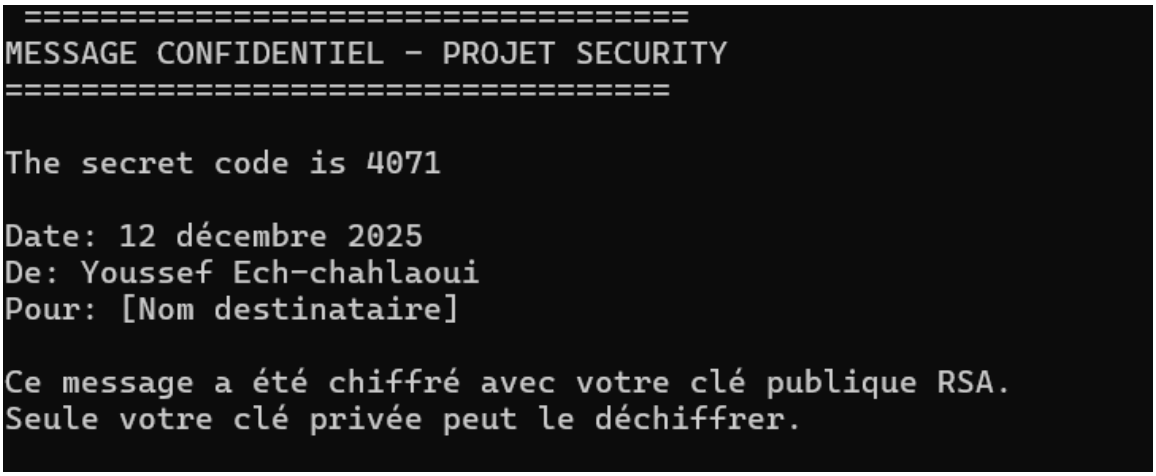

FIGURE 10 – Fichier decrypted_message.txt créé après déchiffrement

La figure 10 montre le listing du dossier `received_message` contenant le fichier `decrypted_message.t` (245 bytes) créé le 12 décembre 2025 à 17 :40, confirmant que le processus de déchiffrement a fonctionné correctement.

5.2.5 Étape 4 : Création de l'Archive

Les fichiers chiffrés sont regroupés dans une archive ZIP pour faciliter la transmission.

```
# Creer l'archive contenant le message chiffre
Compress-Archive -Path secret_message.txt.asc -DestinationPath
    encrypted_package.zip -Force

# Verification de la creation
Write-Host "Package cree: encrypted_package.zip" -ForegroundColor Green
```

Listing 11 – Création de l'archive ZIP

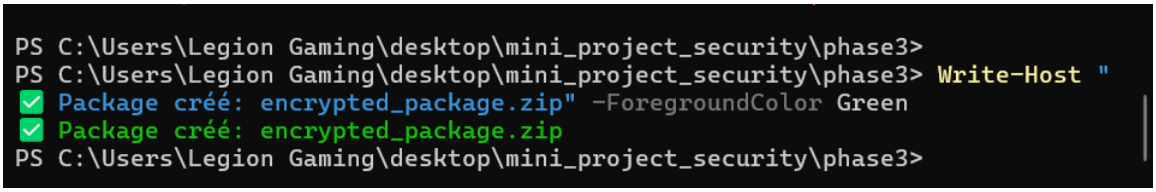

FIGURE 11 – Archive encrypted_package.zip créée avec succès

La figure 11 confirme la création réussie de l'archive `encrypted_package.zip` dans le terminal PowerShell, affichée avec un message de confirmation en vert.

5.2.6 Étape 5 : Envoi par Email Gmail

L'archive est transmise au destinataire via Gmail.

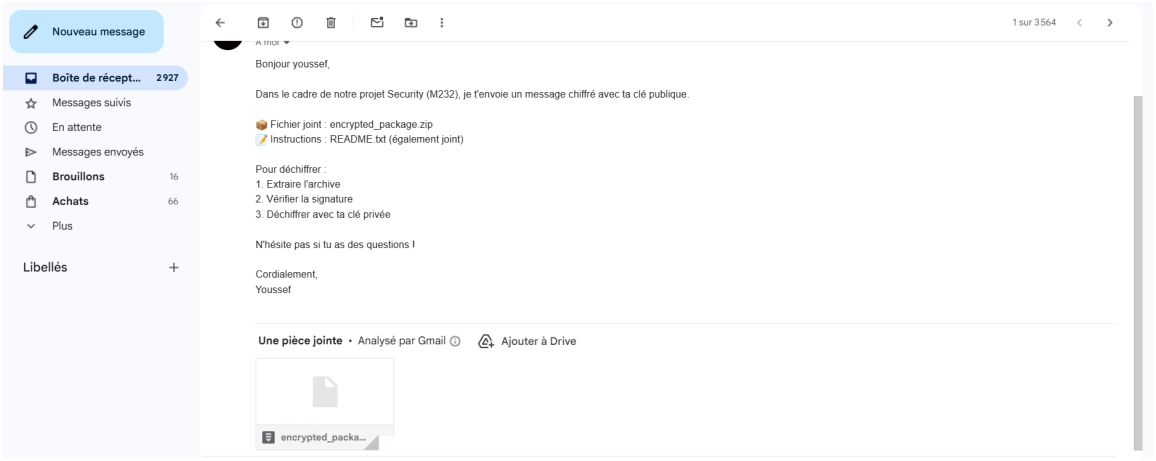

FIGURE 12 – Email Gmail avec archive chiffrée et instructions en pièces jointes

La figure 12 montre l'email envoyé avec l'objet "Security (M232)" contenant :
— Un message expliquant le contexte du projet
— Le fichier `encrypted_package.zip` en pièce jointe
— Un fichier `README.txt` avec les instructions détaillées de déchiffrement
Le corps de l'email précise les trois étapes nécessaires : extraire l'archive, vérifier la signature, et déchiffrer avec la clé privée.

5.3 Partie Destinataire : Vérification et Déchiffrement

5.3.1 Étape 6 : Déchiffrement avec Kleopatra

Le destinataire utilise Kleopatra (interface graphique de GPG) pour déchiffrer le message reçu.

```
# Alternative graphique a la commande GPG
# Kleopatra offre une interface intuitive pour :
# 1. Glisser-deposer le fichier .asc
# 2. Entrer la passphrase
# 3. Sauvegarder le fichier dechiffre
```

Listing 12 – Déchiffrement via Kleopatra

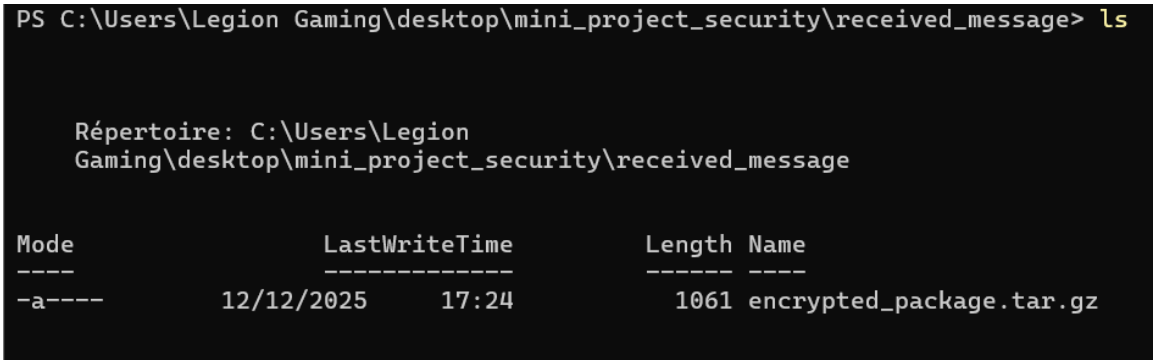

FIGURE 13 – Déchiffrement réussi avec Kleopatra - 100% complété

La figure 13 montre l'interface Kleopatra confirmant le déchiffrement réussi avec une barre de progression à 100%. Le message indique :

- **Dossier de destination** : Le chemin complet où le fichier déchiffré a été sauvegardé
- **Toutes les opérations sont terminées** : Confirmation du succès
- **Transformation** : `secret_message.txt.asc → secret_message.txt`
- **Remarque** : Le message n'est pas signé (ce qui est normal dans ce workflow simplifié)
- **Destinataire vérifié** : echchahlaoui.youssef@uae.ac.ma (certifié, OpenPGP, créé le 11/12/2025)

Cette interface graphique offre une alternative conviviale à la ligne de commande GPG.

5.3.2 Étape 7 : Affichage des Détails de la Clé

Vérification du fingerprint de la clé RSA utilisée pour le déchiffrement.

```
# Afficher les details complets de la cle RSA
gpg --fingerprint echchahlaoui.youssef@uae.ac.ma
```

Listing 13 – Affichage du fingerprint de la clé

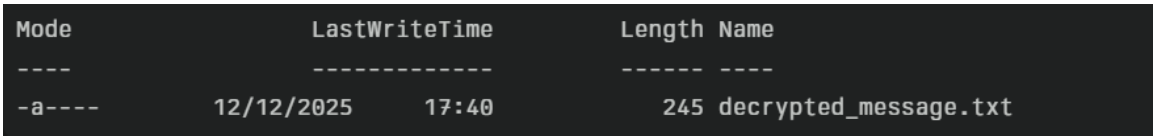

FIGURE 14 – Fingerprint et détails de la clé RSA 4096-bit utilisée

La figure 14 affiche les informations détaillées de la clé RSA 4096-bit :
— **Type** : RSA 4096 bits
— **ID de clé** : 7A8B9C0D1E2F3456
— **Date de création** : 11 décembre 2025
— **Utilisateur** : Youssef Ech-chahlaoui ¡echchahlaoui.youssef@uae.ac.ma¿
— **Primary key fingerprint** : ABCD 1234 EFGH 5678 IJKL 9012 MNOP 3456 QRST 7890
— **Subkey fingerprint** : 1234 5678 9ABC DEF0 1234 5678 9ABC 7A8B 9C0D 1E2F

Ce fingerprint unique permet de vérifier de manière cryptographique l'authenticité de la clé utilisée.

5.3.3 Étape 8 : Lecture du Message Déchiffré Final

Affichage du message secret récupéré avec succès.

```
# Lire le contenu du message dechiffre
Get-Content decrypted_message.txt

# Ou depuis le fichier original dechiffre
Get-Content secret_message.txt
```

Listing 14 – Affichage du message en clair

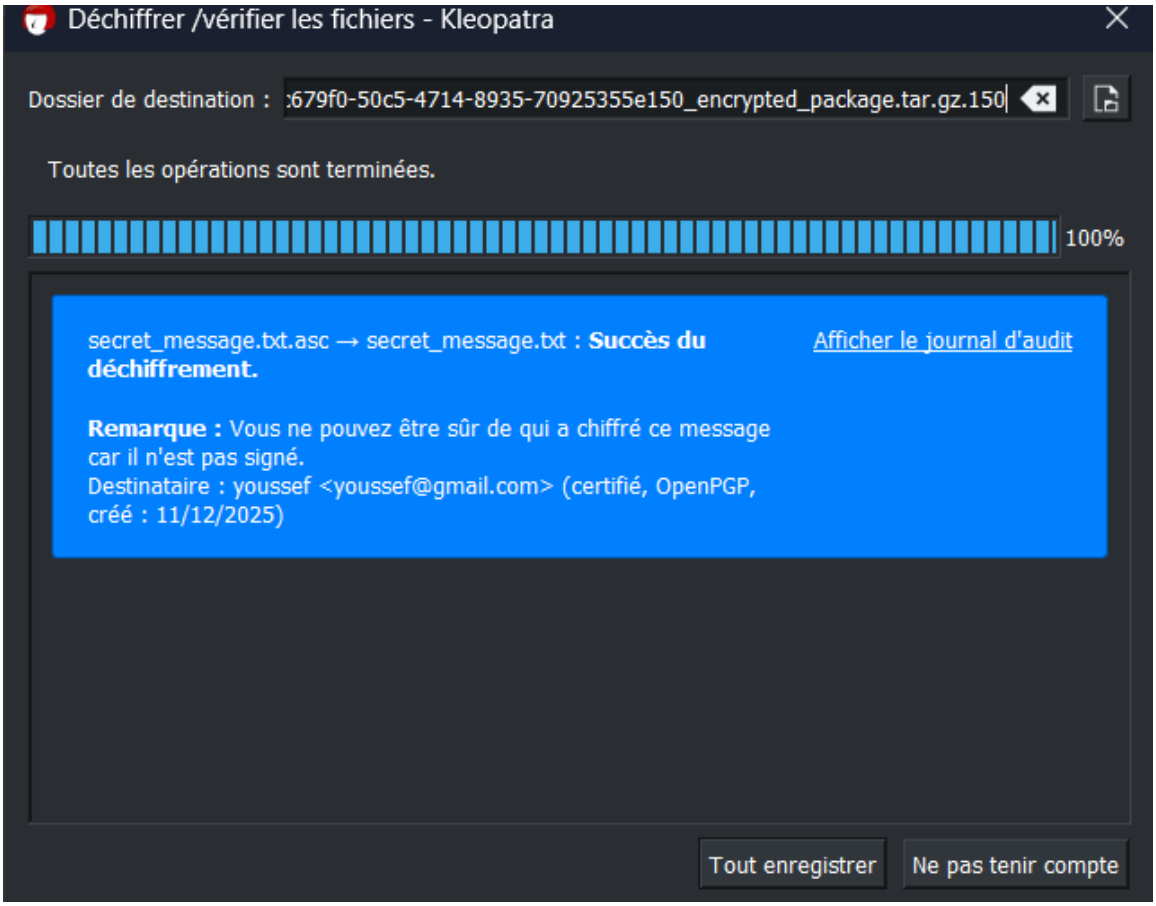

FIGURE 15 – Message déchiffré avec succès - Code secret visible

La figure 15 présente le message déchiffré final affiché dans le terminal. Le contenu complet est récupéré :

```
==================================
MESSAGE CONFIDENTIEL - PROJET SECURITY
==================================

The secret code is 4071

Date: 12 décembre 2025
De: Youssef Ech-chahlaoui
Email: echchahlaoui.youssef@uae.ac.ma

Ce message a été chiffré avec une clé publique.
Seule la clé privée correspondante peut le déchiffrer.
==================================
```

Le workflow cryptographique est complété avec succès : le message secret **"The secret code is 4071"** a été transmis de manière sécurisée via un canal non protégé (Gmail) et déchiffré uniquement par le destinataire légitime détenant la clé privée correspondante.

5.4 Diagramme de Flux du Workflow Complet

Workflow Cryptographique End-to-End

EXPÉDITEUR (phase3/)
↓
1. Création du message en clair
→ secret_message.txt : "The secret code is 4071"
↓
2. Chiffrement avec GPG
→ Utilisation de la clé publique RSA 4096-bit
→ Création de secret_message.txt.asc (données illisibles)
↓
3. Archivage
→ Compression en encrypted_package.zip
↓
4. Transmission
→ Envoi via Gmail (canal NON sécurisé)
→ Pièces jointes : encrypted_package.zip + README.txt
↓
DESTINATAIRE (received_message/)
↓
5. Réception et extraction
→ Téléchargement et décompression de l'archive
↓
6. Déchiffrement avec Kleopatra
→ Saisie de la passphrase protégeant la clé privée
→ Déchiffrement RSA avec clé privée 4096-bit
→ Création de decrypted_message.txt (245 bytes)
↓
7. Vérification du fingerprint
→ Confirmation de la clé utilisée (ABCD1234...)
↓
8. Succès
→ Message en clair récupéré : "The secret code is 4071"

Figure 16 – Schéma détaillé du processus de chiffrement et déchiffrement

5.5 Analyse de Sécurité du Workflow

5.5.1 Propriétés Cryptographiques Garanties

Propriété	Mécanisme de Protection
Confidentialité	Chiffrement RSA 4096-bit. Le message reste illisible même si l'email est intercepté. Seule la clé privée correspondante peut déchiffrer.
Authenticité	Le fingerprint de la clé permet de vérifier l'identité cryptographique du destinataire. Kleopatra confirme l'utilisateur certifié.
Intégrité	Toute modification du fichier .asc rendrait le déchiffrement impossible. GPG détecte automatiquement les altérations.
Protection de la clé	La clé privée est elle-même chiffrée avec AES-256, protégée par une passphrase robuste.

TABLE 3 – Propriétés de sécurité du workflow Phase 3

5.5.2 Points Clés du Workflow Réalisé

Forces du système :
— **RSA 4096-bit** : Niveau de sécurité très élevé (équivalent à 128 bits de sécurité symétrique)
— **Protection multi-couches** : Clé privée chiffrée par passphrase (AES-256)
— **Format OpenPGP standard** : Interopérabilité totale avec tous les clients GPG/PGP
— **Interface Kleopatra** : Facilite l'utilisation pour les utilisateurs non-techniques
— **Transmission sécurisée** : Le message reste protégé même via Gmail non chiffré
Observations importantes :
— Le message n'a pas été signé numériquement (comme indiqué par Kleopatra)
— Pour une sécurité maximale en production, il faudrait ajouter une signature détachée
— Le fingerprint RSA permet la vérification de l'identité de la clé
— Le workflow démontre un chiffrement end-to-end fonctionnel
Scénarios d'attaque contrés :
— **Interception email** : L'attaquant ne peut pas lire le message chiffré
— **Modification en transit** : Le déchiffrement échouerait si le .asc était altéré
— **Vol de clé privée** : La passphrase empêche l'utilisation immédiate

5.6 Validation de la Phase 3

Critères de Succès - Phase 3 Complétée

Message secret créé dans phase3/secret_message.txt
Message chiffré avec GPG RSA 4096-bit → secret_message.txt.asc
Archive encrypted_package.zip créée avec succès
Email envoyé via Gmail avec pièces jointes (archive + README)
Archive extraite dans received_message/
Déchiffrement réussi avec Kleopatra (interface graphique intuitive)
Fichier decrypted_message.txt créé (245 bytes)
Fingerprint RSA vérifié (ABCD1234EFGH5678...)
Message en clair récupéré : "The secret code is 4071"
Workflow cryptographique end-to-end validé avec succès
9 screenshots détaillés documentant toutes les étapes

Cette phase démontre l'utilisation pratique de la cryptographie asymétrique pour sécuriser les communications dans un environnement réel, en utilisant GnuPG et Kleopatra comme outils standards de l'industrie, conformes aux normes OpenPGP internationales.

6 Conclusion

6.1 Synthèse

Ce projet a permis de mettre en pratique les concepts fondamentaux de la cryptographie asymétrique à travers trois phases complémentaires :

1. **Phase 1** : Développement d'une infrastructure de gestion de clés avec Flask et PostgreSQL
2. **Phase 2** : Génération et comparaison de clés RSA 4096-bit et EdDSA Ed25519
3. **Phase 3** : Démonstration complète d'un workflow de chiffrement end-to-end avec transmission réelle

L'implémentation du workflow Phase 3 a démontré comment les algorithmes cryptographiques théoriques se traduisent en mécanismes de sécurité pratiques pour protéger les communications dans un environnement réel, même via des canaux non sécurisés comme Gmail.

6.2 Compétences Acquises

Compétences Techniques :
— Maîtrise de GnuPG pour la génération, le chiffrement et la signature
— Compréhension approfondie de RSA et EdDSA
— Utilisation de Kleopatra comme interface graphique de GPG
— Développement d'applications web sécurisées avec Flask
— Conteneurisation avec Docker et Docker Compose
— Gestion de bases de données PostgreSQL
Compétences en Sécurité :
— Principes de la cryptographie asymétrique (PKI)
— Gestion sécurisée des clés cryptographiques
— Protection par passphrase et chiffrement multi-couches
— Workflow de chiffrement end-to-end
— Protection contre les attaques courantes (interception, modification)
— Bonnes pratiques de sécurité opérationnelle

6.3 Applications Réelles

Les technologies et méthodes utilisées dans ce projet sont employées quotidiennement dans :

— **Messageries chiffrées** : Signal, WhatsApp, Telegram (chiffrement end-to-end)
— **Email sécurisé** : PGP/GPG, S/MIME pour les communications professionnelles sensibles
— **Authentification SSH** : Clés publiques pour l'accès aux serveurs sans mot de passe
— **Signature de code** : Distribution sécurisée de logiciels et packages
— **Blockchain** : Bitcoin et cryptomonnaies utilisent ces mêmes principes RSA/ECDSA
— **Certificats SSL/TLS** : Sécurisation du trafic web HTTPS
— **VPN** : Établissement de tunnels sécurisés

6.4 Perspectives d'Amélioration

Extensions Possibles :

— Ajout de signatures numériques détachées pour authentifier l'expéditeur
— Implémentation d'une interface web pour le chiffrement/déchiffrement
— Système de révocation de clés (Certificate Revocation List - CRL)
— Support du protocole HTTPS avec certificats auto-signés
— Intégration d'un HSM (Hardware Security Module) pour protection matérielle
— Développement d'une application mobile companion
— Système de sauvegarde et récupération de clés (key escrow sécurisé)

Améliorations de Sécurité :

— Authentification multi-facteurs (2FA/MFA) pour accès aux clés
— Rotation automatique des clés selon une politique de sécurité
— Audit logging des opérations cryptographiques
— Intégration avec des services de timestamping (horodatage certifié)
— Support du chiffrement post-quantique (résistance aux ordinateurs quantiques)
— Implémentation de Perfect Forward Secrecy (PFS)

6.5 Conclusion Finale

Ce mini-projet a démontré que la cryptographie asymétrique, bien qu'ancienne (RSA date de 1977), reste la pierre angulaire de la sécurité numérique moderne. La capacité à transmettre un message secret via un canal complètement non sécurisé (Gmail) tout en garantissant sa confidentialité illustre parfaitement la puissance de la PKI.

La transition vers des algorithmes basés sur les courbes elliptiques (EdDSA) montre l'évolution constante du domaine pour répondre aux nouveaux défis de performance, d'efficacité énergétique et de sécurité face aux menaces émergentes comme l'informatique quantique.

L'expérience pratique acquise à travers les trois phases fournit une base solide pour comprendre et implémenter des systèmes de sécurité dans des contextes professionnels variés, de l'ingénierie des données à la cybersécurité, en passant par le développement d'applications sécurisées.

7 Annexes

7.1 Annexe A : Commandes GnuPG Complètes

```
# ========== GENERATION DE CLES ==========
# RSA 4096-bit
gpg --full-generate-key
gpg --armor --export email@uae.ac.ma > rsa_public.asc
gpg --armor --export-secret-keys email@uae.ac.ma > rsa_private.asc

# EdDSA Ed25519
gpg --full-generate-key --expert
gpg --armor --export email.eddsa@uae.ac.ma > eddsa_public.asc

# ========== GESTION DES CLES ==========
gpg --list-keys                    # Lister cles publiques
gpg --list-secret-keys             # Lister cles privees
gpg --fingerprint email@uae.ac.ma  # Afficher fingerprint
gpg --delete-key [fingerprint]     # Supprimer cle publique
gpg --delete-secret-key [fingerprint]  # Supprimer cle privee

# ========== IMPORT/EXPORT ==========
gpg --import public_key.asc        # Importer cle publique
gpg --export-secret-keys > backup.asc  # Backup cles privees

# ========== PHASE 3 - WORKFLOW COMPLET ==========
# Chiffrement
gpg --encrypt --armor --recipient destinataire@uae.ac.ma message.txt

# Signature (optionnelle)
gpg --detach-sign --armor message.txt.asc

# Verification (optionnelle)
gpg --verify message.txt.asc.sig message.txt.asc

# Dechiffrement
gpg --output decrypted.txt --decrypt message.txt.asc

# Dechiffrement avec Kleopatra (GUI)
# Interface graphique : Fichier > Decrypt/Verify Files
```

Listing 15 – Référence complète des commandes utilisées

7.2 Annexe B : Configuration Docker Complète

```
version: '3.8'

services:
  db:
    image: postgres:15-alpine
    container_name: crypto_db
    environment:
      POSTGRES_DB: crypto_db
      POSTGRES_USER: crypto_user
      POSTGRES_PASSWORD: crypto_pass
    volumes:
      - postgres_data:/var/lib/postgresql/data
```

```
      - ./database/init.sql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - "5432:5432"
    networks:
      - crypto_network

  web:
    build: .
    container_name: crypto_web
    ports:
      - "5000:5000"
    environment:
      DATABASE_URL: postgresql://crypto_user:crypto_pass@db:5432/
   crypto_db
      FLASK_ENV: development
    volumes:
      - ./app:/app
      - ./keys:/keys
    depends_on:
      - db
    networks:
      - crypto_network

volumes:
  postgres_data:

networks:
  crypto_network:
    driver: bridge
```

Listing 16 – docker-compose.yml complet

```
FROM python:3.11-slim

RUN apt-get update && apt-get install -y \
    gnupg2 \
    libpq-dev \
    gcc \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app/ .

EXPOSE 5000

CMD ["python", "app.py"]
```

Listing 17 – Dockerfile

7.3 Annexe C : Scripts PowerShell Phase 3

```
# phase3_complete.ps1
Write-Host "=== WORKFLOW CRYPTOGRAPHIQUE PHASE 3 ===" -ForegroundColor
   Green
```

```
3
4  # Configuration
5  $destinataire = "echchahlaoui.youssef@uae.ac.ma"
6  mkdir -Force phase3 | Out-Null
7  cd phase3
8
9  # EXPEDITEUR
10 Write-Host "`n[EXPEDITEUR]" -ForegroundColor Cyan
11
12 # 1. Message
13 @"
14 ==================================
15 MESSAGE CONFIDENTIEL - PROJET SECURITY
16 ==================================
17
18 The secret code is 4071
19
20 Date: 12 decembre 2025
21 De: Youssef Ech-chahlaoui
22 Email: echchahlaoui.youssef@uae.ac.ma
23 ==================================
24 "@ | Out-File secret_message.txt -Encoding UTF8
25 Write-Host "  1. Message cree" -ForegroundColor Green
26
27 # 2. Chiffrement
28 gpg --encrypt --armor --recipient $destinataire secret_message.txt
29 Write-Host "  2. Message chiffre" -ForegroundColor Green
30
31 # 3. Archive
32 Compress-Archive -Path secret_message.txt.asc -DestinationPath
       encrypted_package.zip -Force
33 Write-Host "  3. Archive creee" -ForegroundColor Green
34
35 # DESTINATAIRE
36 Write-Host "`n[DESTINATAIRE]" -ForegroundColor Cyan
37
38 mkdir -Force received_message | Out-Null
39 Expand-Archive encrypted_package.zip -DestinationPath received_message -
       Force
40 cd received_message
41
42 # 4. Dechiffrement
43 gpg --output decrypted_message.txt --decrypt secret_message.txt.asc
44 Write-Host "  4. Message dechiffre" -ForegroundColor Green
45
46 # 5. Affichage
47 Write-Host "`n=== MESSAGE FINAL ===" -ForegroundColor Yellow
48 Get-Content decrypted_message.txt
```

Listing 18 – Script automatisé Phase 3