Projet de Fin de Module : Crypto-Java

Université Cheikh Anta Diop de Dakar
Faculté des Sciences et Techniques
Département Mathématiques et Informatique
Master 2 Transmission des Données et Sécurité de l'Information
Année Scolaire 2023 / 2024
Module : Cryptographie Appliquée
Professeur : Pr Demba SOW

Djibril FALL

October 11, 2024

Abstract

Ce mémoire présente le développement d'une application de cryptographie. Ce projet est une application de cryptographie développée en Java avec JavaFX pour l'interface graphique. L'objectif est de fournir une interface utilisateur intuitive permettant de générer des clés, d'effectuer des opérations cryptographiques (symétriques et asymétriques). L'application repose sur les algorithmes de chiffrement couramment utilisés et propose une gestion flexible des clés selon le type de chiffrement sélectionné.

Le code de l'application est disponible sur mon GitHub : `https://github.com/Papus-Jubus/ProjetCrypto`.

Contents

Chapter 1

Introduction

Ce mémoire aborde le thème de la cryptographie, une discipline essentielle pour assurer la sécurité des informations. Il traite également des technologies utilisées pour le développement de l'application.

Chapter 2

Première Partie : La Cryptographie

La cryptographie est l'art de protéger les informations en les transformant en un format sécurisé. Cette partie aborde les concepts clés de la cryptographie, y compris :

2.1 Concepts de Base

Définition et importance de la cryptographie, types de cryptographie (symétrique, asymétrique, hachage).

2.2 Algorithmes de Cryptographie

Cette section présente les principaux algorithmes de cryptographie utilisés dans l'application, ainsi que leur principe de fonctionnement.

2.2.1 AES (Advanced Encryption Standard)

AES est un algorithme de chiffrement symétrique qui remplace le DES (Data Encryption Standard). Il utilise des blocs de 128 bits et des tailles de clé de 128, 192 ou 256 bits. Le chiffrement se fait en plusieurs tours (10, 12 ou 14, selon la taille de la clé) comprenant des substitutions, des permutations et des opérations de mélange pour transformer le texte en clair en texte chiffré.

2.2.2 RSA (Rivest-Shamir-Adleman)

RSA est un algorithme de chiffrement asymétrique qui repose sur la difficulté de factoriser de grands nombres premiers. Il utilise deux clés : une clé publique pour le chiffrement et une clé privée pour le déchiffrement. RSA est souvent utilisé pour échanger des clés symétriques en toute sécurité.

2.2.3 SHA-256 (Secure Hash Algorithm 256 bits)

SHA-256 est une fonction de hachage cryptographique qui produit un hachage de 256 bits. Il est utilisé pour garantir l'intégrité des données. SHA-256 transforme une entrée de taille variable en une sortie fixe et est conçu pour être résistant aux collisions, ce qui signifie qu'il est difficile de trouver deux entrées différentes ayant le même hachage.

2.2.4 HMAC (Hash-based Message Authentication Code)

HMAC est un mécanisme qui combine une fonction de hachage cryptographique avec une clé secrète pour créer un code d'authentification. Il garantit l'intégrité et l'authenticité des messages, en s'assurant que seul le détenteur de la clé secrète peut produire un HMAC valide pour un message donné.

2.2.5 Diffie-Hellman

L'algorithme de Diffie-Hellman permet à deux parties de générer une clé secrète partagée sur un canal de communication non sécurisé. Chaque partie choisit un nombre secret et les échange sous forme de puissances d'un nombre de base, permettant ainsi de calculer une clé secrète sans l'envoyer directement.

Chapter 3

Deuxième Partie : Rappels quelques classe Java Security

Cette section décrit les technologies et classes Java utilisées pour le développement de l'application.

3.1 Java et Cryptographie

Java offre une API de cryptographie qui comprend plusieurs classes essentielles :

3.1.1 KeyPairGenerator

La classe 'KeyPairGenerator' est utilisée pour générer des paires de clés (une clé publique et une clé privée) pour les algorithmes de cryptographie asymétrique. Pour générer une paire de clés, vous devez d'abord instancier un objet 'KeyPairGenerator' avec l'algorithme souhaité (par exemple, RSA). Ensuite, en appelant la méthode 'generateKeyPair()', la classe génère et renvoie une paire de clés, qui peut être utilisée pour le chiffrement et le déchiffrement.

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
keyGen.initialize(2048);
KeyPair keyPair = keyGen.generateKeyPair();
```

3.1.2 KeyFactory

La classe 'KeyFactory' est utilisée pour convertir des clés encodées en objets clés (et vice versa). Lorsqu'une clé est stockée ou transmise, elle est souvent sous forme encodée (par exemple, sous forme de tableau d'octets). La classe 'KeyFactory' permet de reconstruire une clé à partir de cette représentation encodée à l'aide d'un spécificateur de clé, tel que 'PKCS8EncodedKeySpec' pour les clés privées ou 'X509EncodedKeySpec' pour les clés publiques.

```
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
PrivateKey privateKey = keyFactory.generatePrivate(new PKCS8EncodedKeySpec(encodedK
```

3.1.3 PKCS8EncodedKeySpec

Cette classe représente un spécificateur de clé pour les clés privées encodées au format PKCS#8. 'PKCS8EncodedKeySpec' est utilisé avec 'KeyFactory' pour créer une instance de clé privée à partir d'une clé encodée en PKCS#8. Vous passez un tableau d'octets (la clé encodée) lors de l'instanciation.

```
PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(encodedKey);
PrivateKey privateKey = keyFactory.generatePrivate(spec);
```

3.1.4 Cipher

La classe 'Cipher' fournit des méthodes pour chiffrer et déchiffrer des données à l'aide d'algorithmes de cryptographie symétrique et asymétrique. Pour chiffrer des données, vous devez d'abord créer une instance de 'Cipher' avec l'algorithme souhaité (par exemple, AES). Ensuite, vous initialisez l'objet avec la clé appropriée et utilisez les méthodes 'doFinal()' pour chiffrer ou déchiffrer les données.

```
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.ENCRYPT_MODE, secretKey);
byte[] encryptedData = cipher.doFinal(dataToEncrypt);
```

3.1.5 MessageDigest

La classe 'MessageDigest' est utilisée pour générer des hachages de données (comme SHA-256). Vous créez une instance de 'MessageDigest' avec l'algorithme de hachage souhaité. Ensuite, vous pouvez mettre à jour l'objet avec les données à hacher et obtenir le hachage final sous forme de tableau d'octets.

```
MessageDigest digest = MessageDigest.getInstance("SHA-256");
byte[] hash = digest.digest(data);
```

3.1.6 Signature

La classe 'Signature' est utilisée pour créer et vérifier des signatures numériques. Vous pouvez instancier 'Signature' avec un algorithme de signature (par exemple, "SHA256withRSA"). Ensuite, vous initialisez l'objet avec la clé privée pour signer ou la clé publique pour vérifier, puis utilisez les méthodes appropriées pour générer ou vérifier la signature.

```
Signature signature = Signature.getInstance("SHA256withRSA");
signature.initSign(privateKey);
signature.update(data);
byte[] digitalSignature = signature.sign();
```

3.1.7 KeyAgreement

La classe 'KeyAgreement' est utilisée pour implémenter des protocoles d'échange de clés, comme Diffie-Hellman. Vous devez créer une instance de 'KeyAgreement', initialiser l'objet avec votre clé privée et ensuite l'utiliser pour générer une clé secrète partagée à partir de la clé publique de l'autre partie.

```
KeyAgreement keyAgreement = KeyAgreement.getInstance("DH");
keyAgreement.init(privateKey);
keyAgreement.doPhase(otherPartyPublicKey, true);
byte[] sharedSecret = keyAgreement.generateSecret();
```

Chapter 4

Troisième Partie : REALISATION

Cette section présente les réalisations de l'application avec des captures d'écran.

4.1 Interface de Génération des Clés

L'interface de génération des clés est conçue pour permettre aux utilisateurs de créer des clés pour les algorithmes de cryptographie. Elle comprend plusieurs éléments clés :

1. **Titre de la Page** : Affiche clairement "Génération de Clés", indiquant la fonctionnalité principale de cette section de l'application.
2. **Sélection du type de cryptographie** :
   - `ComboBox "Crytographie"` : Permet à l'utilisateur de choisir cryptographie Symetrique ou Asymetrique .
3. **Sélection de l'Algorithme de Génération** :
   - `ComboBox "Algorithme"` : Permet à l'utilisateur de choisir l'algorithme de génération de clés approprié, tel que "RSA", "DSA" ...
4. **Champ de Spécification de la Taille de la Clé** :
   - `TextField "Taille de la Clé"` : Permet à l'utilisateur de spécifier la taille de la clé (par exemple, 2048 bits pour RSA).
5. **Bouton de Génération de Clés** :
   - `Bouton "Générer Clé"` : Déclenche le processus de génération et sauvegarde de la paire de clés (publique et privée), affichant un message de succès ou d'erreur.

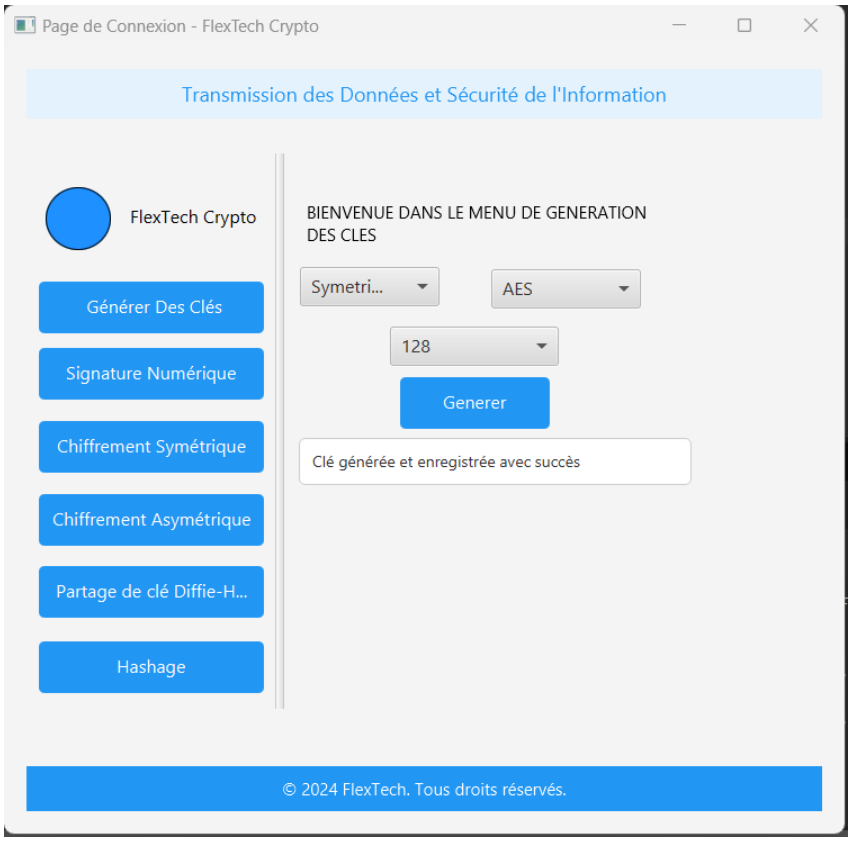

Figure 4.1: Menu de génération d'une cle AES.

Figure 4.2: Menu de génération d'une cle RSA.

4.2 Interface de Chiffrement Symétrique

L'interface de chiffrement symétrique est conçue pour permettre aux utilisateurs de chiffrer des données en utilisant des clés symétriques. Elle comprend plusieurs éléments clés :

1. **Champ de Sélection de Fichier** :
   - `Bouton "Charger Fichier"` : Permet à l'utilisateur de sélectionner le fichier à chiffrer, avec une zone de texte affichant le chemin du fichier sélectionné pour confirmation.
2. **Sélection de l'Algorithme de Chiffrement** :
   - `ComboBox "Algorithme de Chiffrement"` : Permet à l'utilisateur de choisir l'algorithme de chiffrement approprié, tel que "AES", "DES", ou "3DES".
3. **Charger la Clé** :
   - `Bouton "Charger Clé"` : Permet à l'utilisateur de sélectionner un fichier contenant la clé symétrique, avec une zone de texte qui affiche son chemin.
4. **Champ d'Entrée de Clé** :
   - `TextField "Entrer Clé"` : Un champ où l'utilisateur peut saisir manuellement la clé à utiliser pour le chiffrement.
5. **Bouton de Chiffrement** :
   - `Bouton "Chiffrer"` : Déclenche le processus de chiffrement des données en utilisant l'algorithme et la clé sélectionnés, affichant le résultat dans la zone de texte prévue à cet effet.
6. **Affichage des Résultats** :
   - `Zone de Texte` : Présente les données chiffrées ou un message d'erreur en cas de problème lors du chiffrement.

Figure 4.3: chiffrement symétrique d'un fichier Texte.

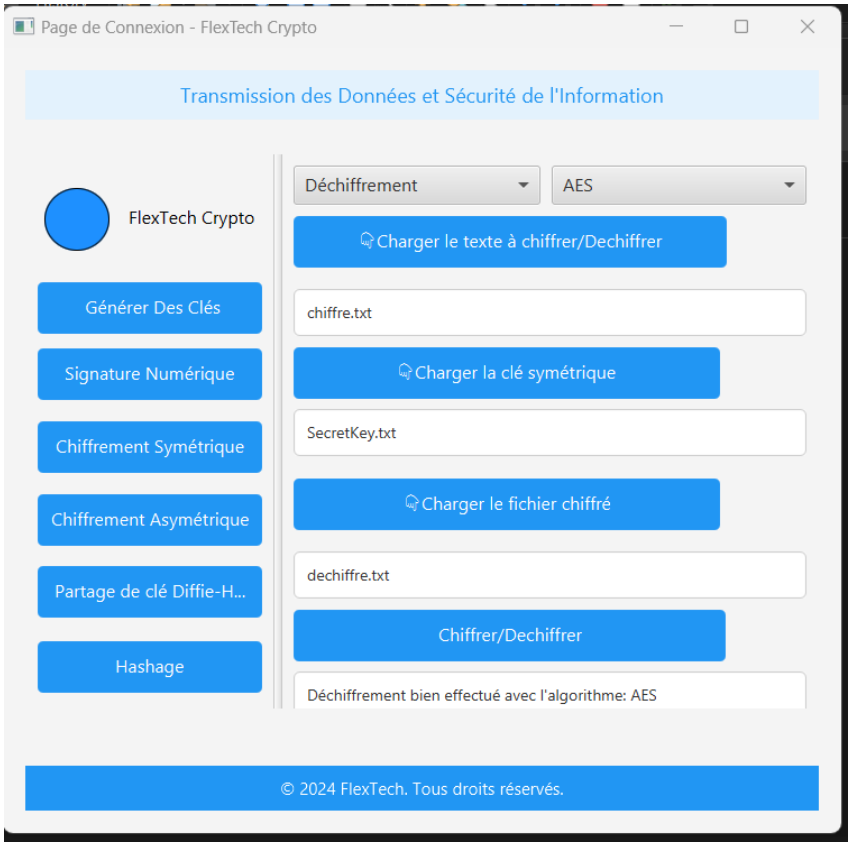

Figure 4.4: Déchiffrement symétrique d'un fichier Texte.

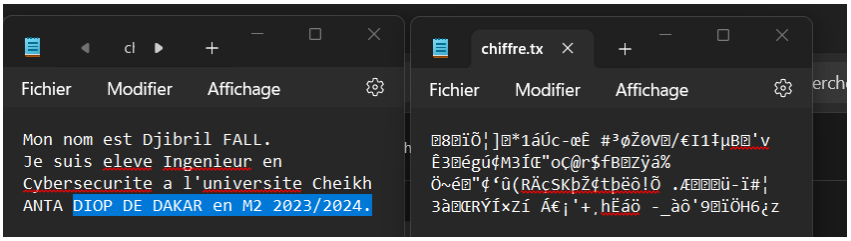

Figure 4.5: Résultat Du chiffrement avec AES.

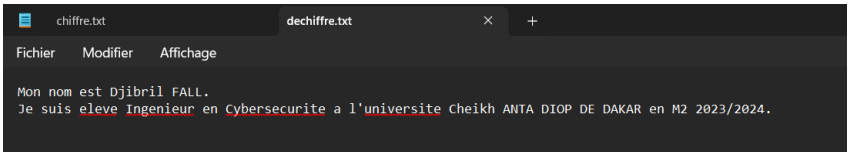

Figure 4.6: Résultat Du Déchiffrement avec AES.

4.3 Interface de Chiffrement Asymétrique

L'interface de chiffrement asymétrique est conçue pour permettre aux utilisateurs de chiffrer des données en utilisant des clés asymétriques. Elle comprend plusieurs éléments clés :

1. **Titre de la Page** : Affiche clairement "Chiffrement Asymétrique", indiquant la fonctionnalité principale de cette section de l'application.
2. **Champ de Sélection de Fichier** :
   - `Bouton "Charger Fichier"` : Permet à l'utilisateur de sélectionner le fichier à chiffrer, avec une zone de texte affichant le chemin du fichier sélectionné pour confirmation.
3. **Sélection de l'Algorithme de Chiffrement** :
   - `ComboBox "Algorithme de Chiffrement"` : Permet à l'utilisateur de choisir l'algorithme de chiffrement approprié, tel que "RSA" ou "DSA".
4. **Charger la Clé Publique** :
   - `Bouton "Charger Clé Publique"` : Permet à l'utilisateur de sélectionner un fichier contenant la clé publique à utiliser pour le chiffrement, avec une zone de texte qui affiche son chemin.
5. **Champ d'Entrée de Clé Privée** :
   - `TextField "Entrer Clé Privée"` : Un champ où l'utilisateur peut saisir manuellement la clé privée, si nécessaire pour des opérations ultérieures.
6. **Bouton de Chiffrement** :
   - `Bouton "Chiffrer"` : Déclenche le processus de chiffrement des données en utilisant la clé publique et l'algorithme sélectionnés, affichant le résultat dans la zone de texte prévue à cet effet.
7. **Affichage des Résultats** :
   - `Zone de Texte` : Présente les données chiffrées ou un message d'erreur en cas de problème lors du chiffrement.

Figure 4.7: Chiifrement d'un fichier Texte avec RSA.

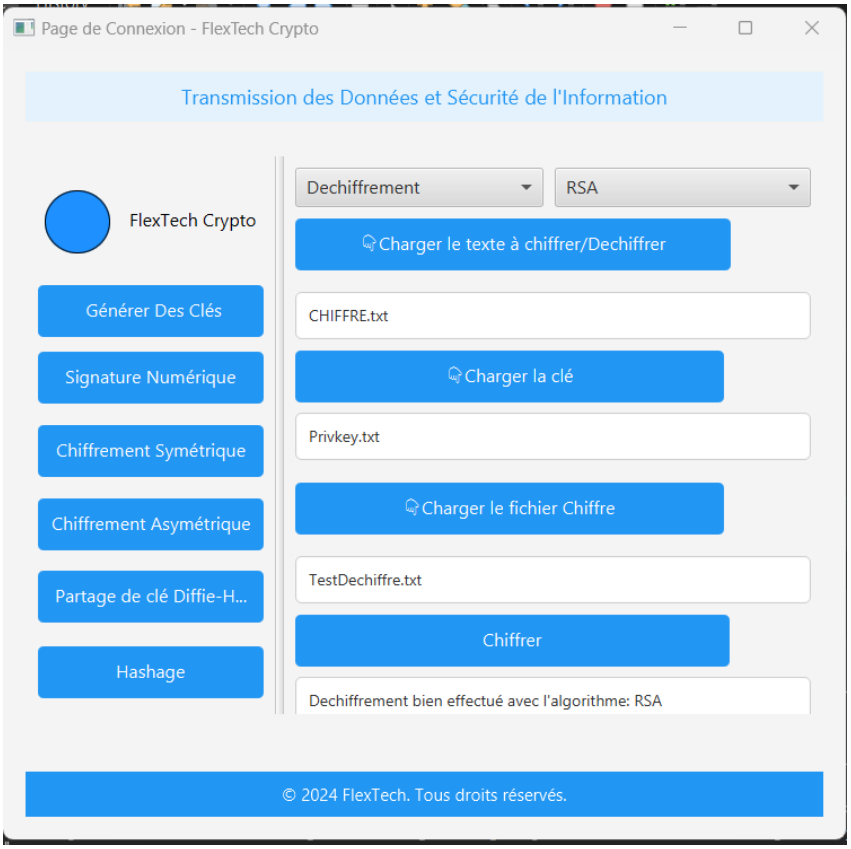

Figure 4.8: Déchiffrement d'un fichier Texte avec RSA.

4.4 Interface de Signature Numérique

L'interface de signature numérique est conçue pour faciliter le processus de signature de documents. Elle comprend plusieurs champs clés :

1. **Champ de Sélection de Fichier** : Un bouton "Charger Fichier" permet à l'utilisateur de sélectionner le document à signer, avec une zone de texte affichant le chemin du fichier sélectionné pour confirmation.

2. **Sélection de l'Algorithme de Signature** : Une `ComboBox` permet de choisir l'algorithme de signature approprié, tel que "SHA256withRSA".

3. **Charger la Clé Privée** : Un bouton "Charger Clé Privée" permet de sélectionner le fichier contenant la clé privée, accompagné d'une zone de texte qui affiche son chemin.

4. **Bouton de Signature** : Le bouton "Signer" déclenche le processus de signature du document en utilisant la clé privée et l'algorithme sélectionnés.

5. **Affichage des Résultats** : Une zone de texte ou étiquette présente des messages de succès ou d'erreur, informant l'utilisateur du résultat de l'opération.

6. **Bouton de Vérification** : Le bouton "Vérifier la Signature" permet de contrôler la validité d'une signature existante en utilisant le fichier et la clé publique.

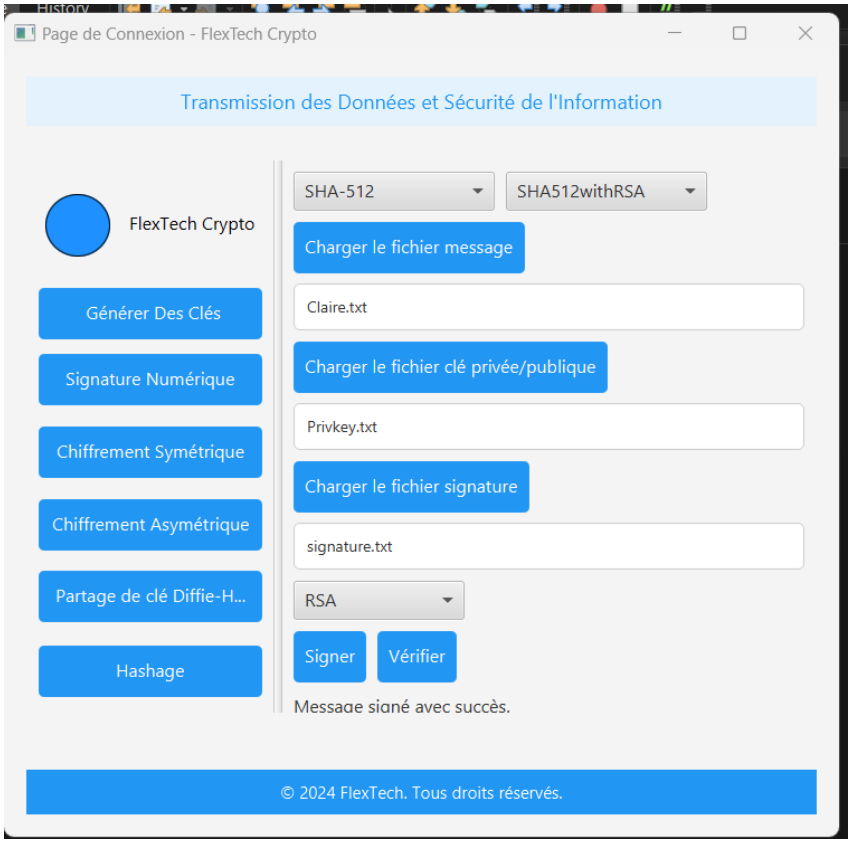

Figure 4.9: Signature Numerique.

Figure 4.10: Verification d'une signature.

4.5 Interface de Partage de Clé Diffie-Hellman

L'interface de partage de clé Diffie-Hellman est conçue pour l'échange sécurisé de clés entre deux ou trois participants. Elle comprend plusieurs éléments qui permettent à l'utilisateur de naviguer facilement dans le processus de génération et de vérification des clés.

1. **Choix du Nombre de Participants** :
    - `ComboBox "Participants"` : Permet à l'utilisateur de sélectionner le nombre de participants (2 ou 3) impliqués dans l'échange de clé. Ce choix affecte la disponibilité des boutons de génération de clés.
2. **Sélection de l'Algorithme** :
    - `ComboBox "Algorithme"` : Offre à l'utilisateur la possibilité de choisir l'algorithme de cryptographie à utiliser pour l'échange de clés (par exemple, "DH" ou "ECDH").
3. **Boutons de Génération de Clés** :
    - `Bouton "Générer Clé pour Alice"` : Lance le processus de génération de la clé pour Alice, créant une paire de clés (publique et privée) et stockant les résultats dans des fichiers texte.
    - `Bouton "Générer Clé pour Bob"` : Similaire au précédent, mais pour Bob.
    - `Bouton "Générer Clé pour Charlie"` : Actif uniquement lorsque trois participants sont sélectionnés, permettant la génération de la clé pour Charlie.
4. **Message d'Information** :
    - `Zone de Texte` : Affiche des messages d'état informant l'utilisateur des résultats des opérations, comme "Clé générée avec succès" ou des messages d'erreur en cas de problème.
5. **Génération de la Clé Secrète Partagée** :
    - `Bouton "Générer Clé Secrète"` : Permet de générer la clé secrète partagée une fois que les clés publiques ont été échangées entre les participants.
6. **Chargement et Comparaison des Clés Secrètes** :

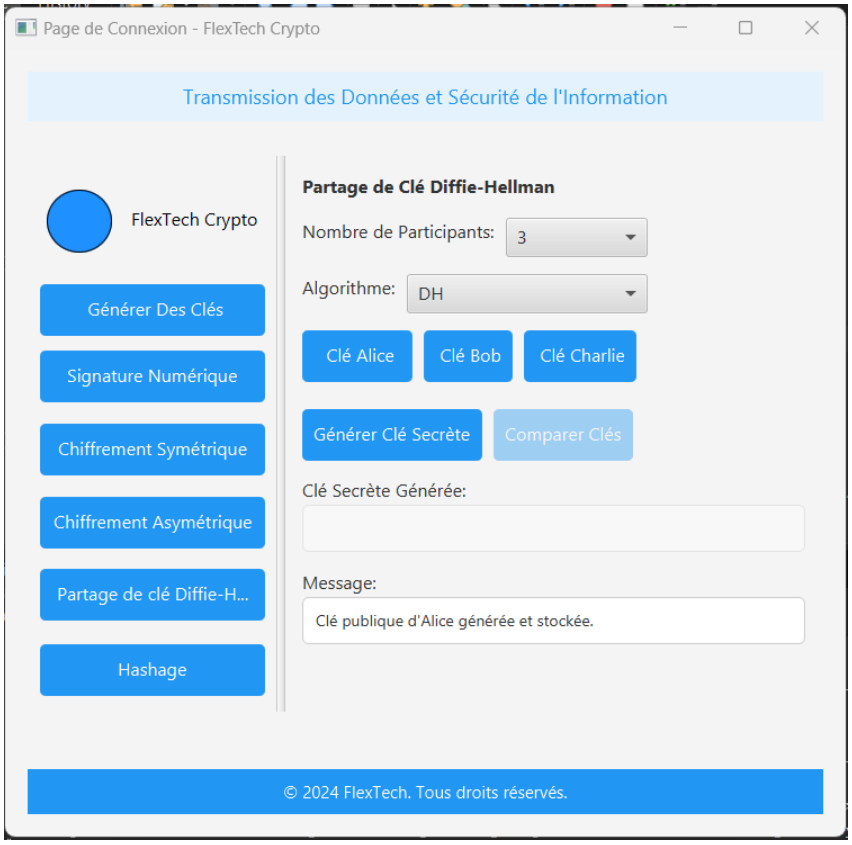

Figure 4.11: Generation de la paire de Cle DH de Alice.

Figure 4.12: Generation de la cle secrete partagee par les trois.

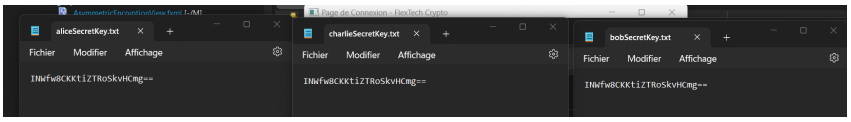

Figure 4.13: Resultat de la Generation d'une cle secrete.

4.6 Interface de Hachage

L'interface de hachage est conçue pour générer des hachages de fichiers textes en utilisant différentes algorithmes. Elle comprend plusieurs éléments clés comme le hash sans cle,le mac et le HMac

1. **Champ de Sélection de Fichier** :
    - `Bouton "Charger Fichier"` : Permet à l'utilisateur de sélectionner le fichier à hacher, avec une zone de texte affichant le chemin du fichier sélectionné pour confirmation.
2. **Sélection de l'Algorithme de Hachage** :
    - `ComboBox "Algorithme de Hachage"` : Permet à l'utilisateur de choisir l'algorithme de hachage approprié, tel que "SHA-256", "SHA-512" ou "SHA-1".
3. **Charger la Clé** :
    - `Bouton "Charger Clé"` : Permet à l'utilisateur de sélectionner un fichier clé, si l'algorithme de hachage nécessite une clé (comme HMAC), accompagné d'une zone de texte qui affiche son chemin.
4. **Champ d'Entrée de Clé** :
    - `TextField "Entrer Clé"` : Un champ où l'utilisateur peut saisir manuellement la clé à utiliser pour le hachage, actif uniquement si HMAC ou MAC est sélectionné.
5. **Bouton de Hachage** :
    - `Bouton "Hacher"` : Déclenche le processus de hachage des données en utilisant l'algorithme et la clé sélectionnés, affichant le résultat dans la zone de texte prévue à cet effet.
6. **Affichage des Résultats** :
    - `Zone de Texte` : Présente le hachage généré ou un message d'erreur en cas de problème lors du hachage.

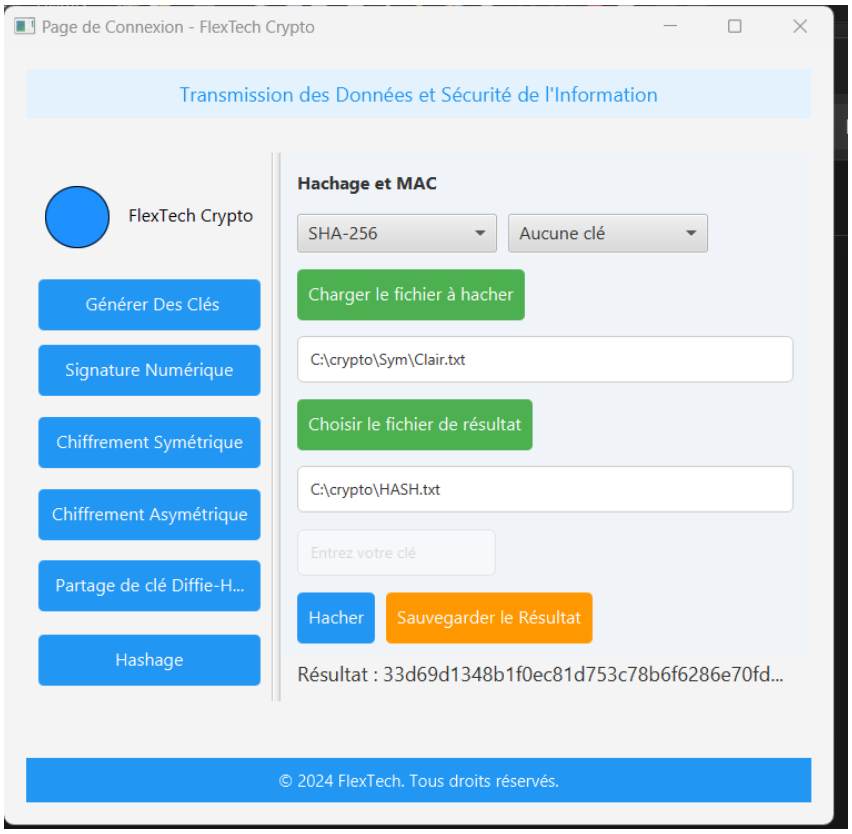

Figure 4.14: Hashage d'un fichier texte .

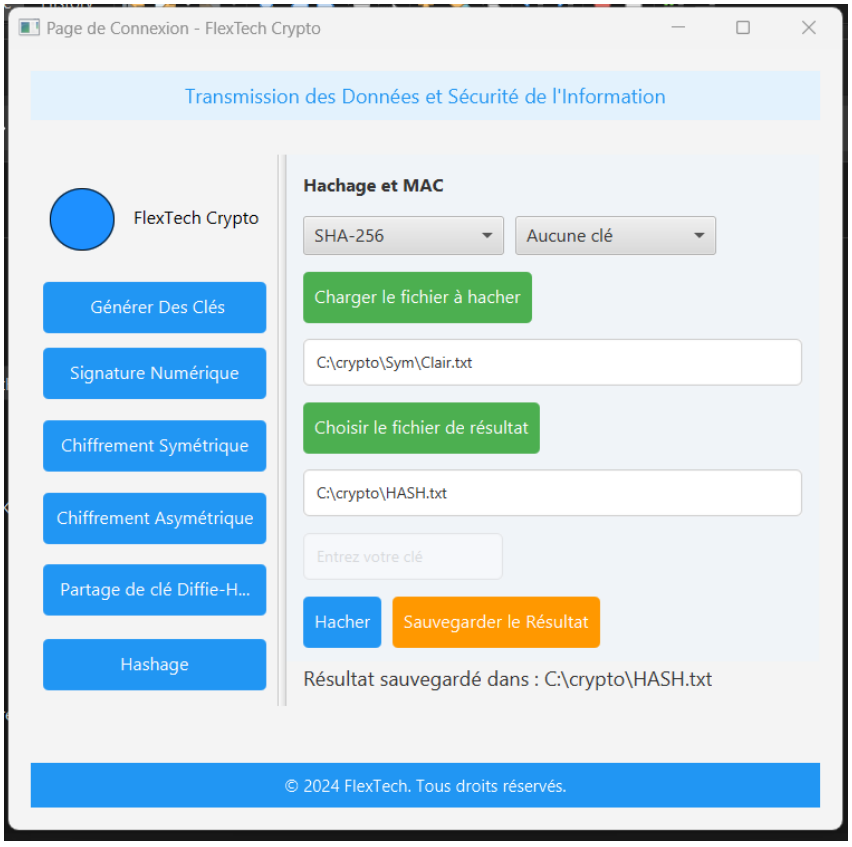

Figure 4.15: Sauvegarde du hash.

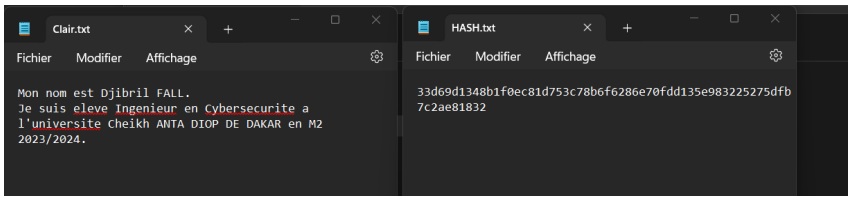

Figure 4.16: Lecture du Hash dans le fichier genere.

Figure 4.17: Hashage avec mot de Passe: HMAC.

Chapter 5

Conclusion

Ce mémoire a présenté le développement d'une application de cryptographie qui offre plusieurs fonctionnalités pour sécuriser les données.

Bibliography

[1] Dr Demba SOW, *Cours Crypto-Java Master 2 2023/2024 TDSI UCAD FST DMI*,

[2] Kensoft , *Setup JavaFX , https://www.youtube.com/watch?v=6E4IkTuvUCI*,

[3] JAVAFX, *Documentation JavaFx , https://openjfx.io/*,

[4] SceneBuilder, *SceneBuilder,https://gluonhq.com/products/scene-builder/*,

[5] ChatGpt, *ChatGpt,https://chatgpt.com*,

[6] GitHub , *GitHub,https://github.com/Papus-Jubus/ProjetCrypto.git*,