UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES HOUARI BOUMEDIENE

Faculté des Sciences Informatiques

Année universitaire 2025-2026

Travail pratique N°1
Encryption Framework

Réalisé par :

MEHAIGUENE Mohamed
MELLITI Ounaissa
KASASNI Sara
SITAYEB Serine
BAGHBAGHA Omar
AOUNE SEGHIR Imene

Module : Stratégie de sécurité pour l'aide à la décision

02 Novembre 2025

Table des matières

1 Introduction

De nos jours, la sécurité des données est devenue un enjeu majeur pour les organisations et les individus. Avec l'expansion des communications numériques et l'augmentation des cyberattaques, la protection des informations sensibles représente un défi constant. Cette sécurité repose sur plusieurs piliers fondamentaux, tel que la confidentialité, l'intégrité, l'authentification et la non répudiation.

Pour répondre à ces besoins de sécurité, divers concepts ont été développés, tels que la cryptographie, qui permet de transformer des informations lisibles en données chiffrées, ou encore la stéganographie, qui offre une approche complémentaire en dissimulant l'existence même d'une communication secrète. Néanmoins, tout système de sécurité présente des vulnérabilités potentielles, d'où la nécessité de développer et de tester des solutions robustes face aux différentes formes d'attaques.

1.1 Contexte du TP

Dans ce contexte, ce projet vise à développer un framework de chiffrement intégrant trois algorithmes cryptographiques distincts. Ce framework permettra l'échange sécurisé de messages entre deux entités, tout en simulant trois types d'attaques visant à compromettre tant les messages chiffrés que les mécanismes d'authentification. En outre, une analyse comparative de chaque méthode de chiffrement sera présentée, incluant l'identification de leurs failles de sécurité et les solutions pour y remédier. Enfin, une composante de stéganographie sera intégrée pour démontrer une des techniques de dissimulation d'information.

2 Interface Utilisateur

2.1 Page d'authentification

L'utilisateur se retrouve devant une page d'authentification, où il saisit son adresse email et son mot de passe pour se connecter ou créer son compte.

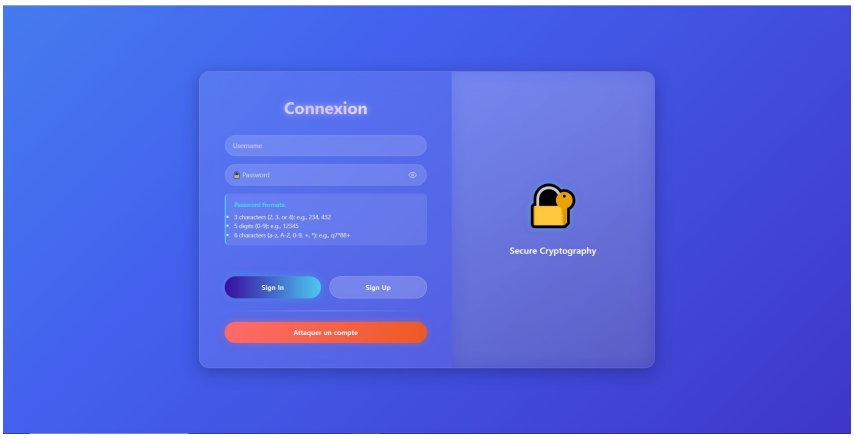

FIGURE 1 – Page d'authentification

2.2 Page d'accueil

Après authentification, il accède à la page d'accueil proposant plusieurs fonctionnalités :

- **Messages :** Ce bouton dirige l'utilisateur vers sa messagerie.
- **Algorithmes de chiffrement :** Accès aux trois systèmes implémentés (Caesar, Hill, PlayFair).
- **Attaques :** Simulation des attaques (Dictionary, Brute Force, Man-in-The-Middle).
- **Stéganographie :** Module de dissimulation d'information.

FIGURE 2 – Page d'accueil avec les différentes fonctionnalités

2.3 Page de messagerie

La page de messagerie permet à l'utilisateur de consulter ses messages envoyés et reçus, ainsi que d'envoyer de nouveaux messages chiffrés.

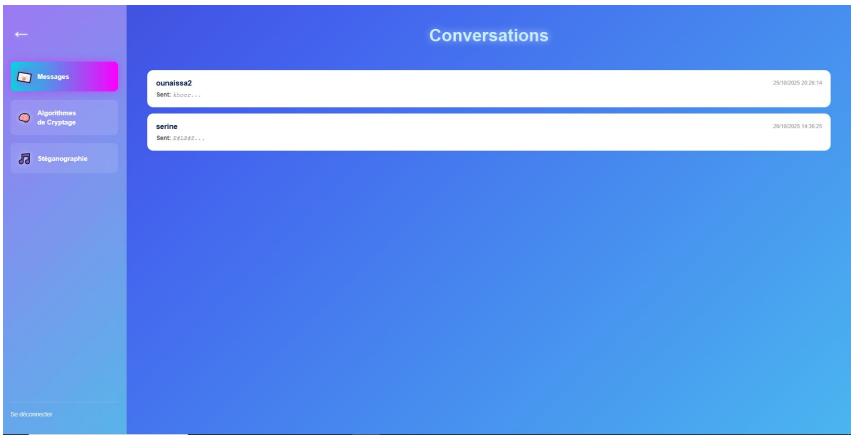

FIGURE 3 – Interface de la messagerie

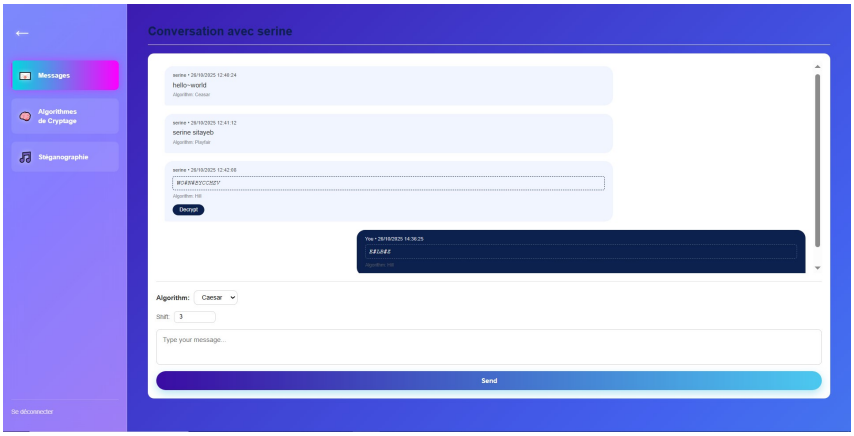

FIGURE 4 – Details de la messagerie

2.4 Page des algorithmes de chiffrement

Cette page permet de sélectionner l'algorithme de chiffrement souhaité (Caesar, Hill ou PlayFair) et de chiffrer/déchiffrer des messages.

FIGURE 5 – Interface de sélection des algorithmes de chiffrement

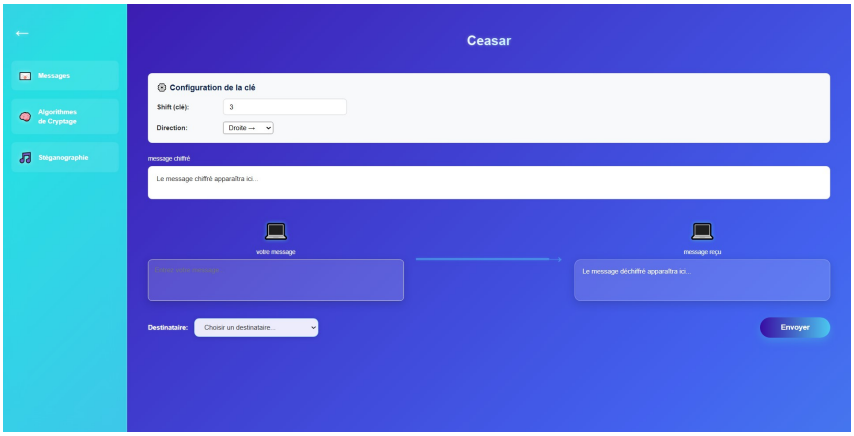

FIGURE 6 – Interface d'un algorithme de chiffrement

2.5 Page des attaques

Cette page permet de simuler différents types d'attaques sur les systèmes de chiffrement et d'authentification, tout en affichant le progres de l'attaques et les tentatives de récuperation des mots de passe.

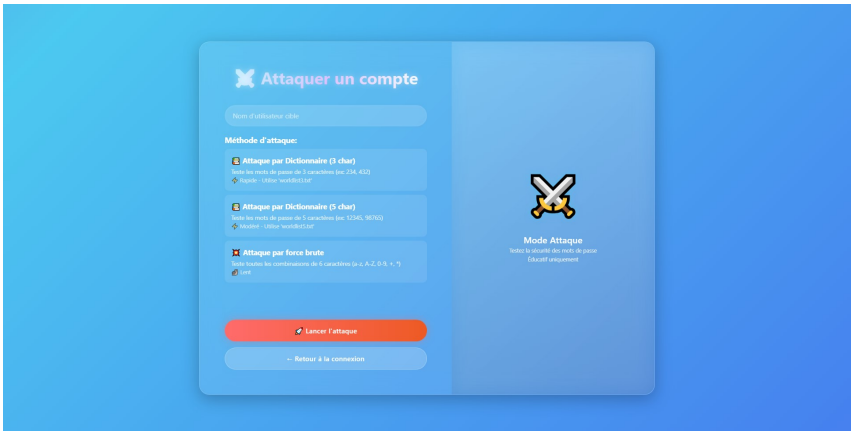

FIGURE 7 – Interface de simulation des attaques

2.6 Page de stéganographie

Cette page permet d'uploader un fichier audio, de cacher un message secret dedans, et d'extraire des messages cachés.

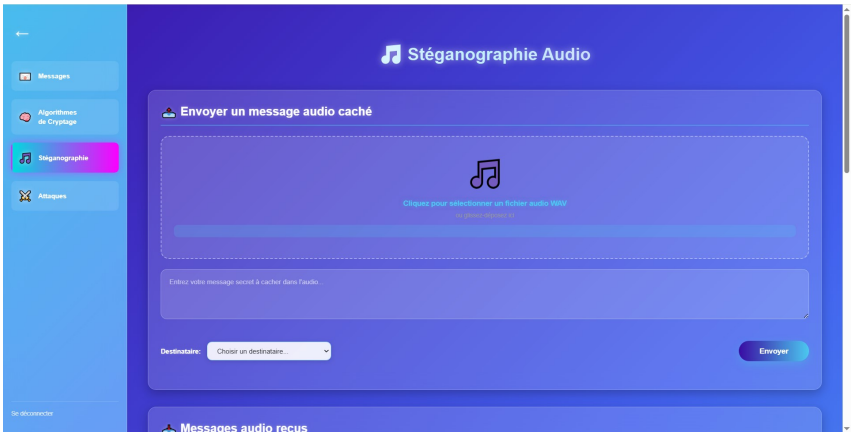

FIGURE 8 – Interface du module de stéganographie

La page affiche également les détails des fichiers audio originaux et stéganographiés, incluant une visualisation complète des modifications apportées à chaque échantillon, permettant ainsi d'observer le processus de dissimulation bit par bit.

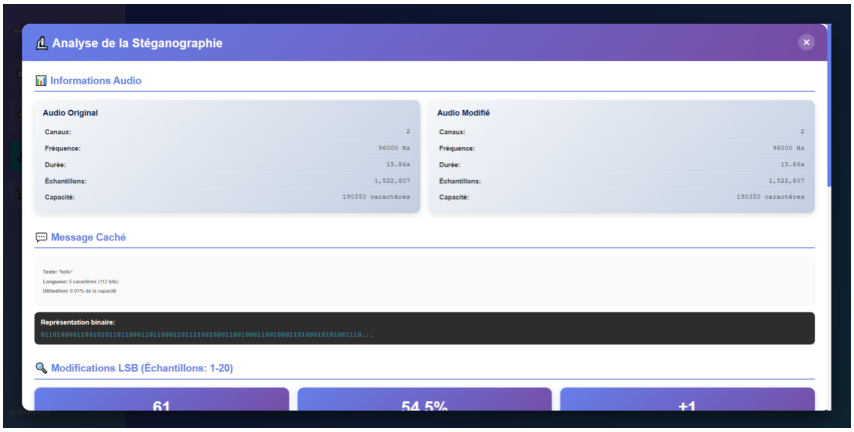

FIGURE 9 – Interface de l'analyse de stéganographie

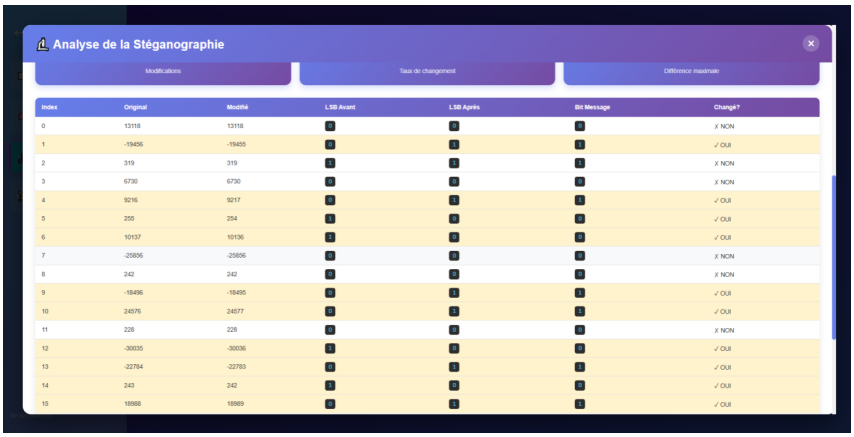

FIGURE 10 – Interface de l'analyse de stéganographie

3 Encryption/Decryption

3.1 Principe général

Un système cryptographique permet de transformer un message intelligible ou message en clair M en un message chiffré incompréhensible ou cryptogramme $C = E_{K_C}(M)$. Ce procédé est désigné sous le terme de chiffrement ou cryptage la procédure de chiffrement utilise un algorithme et une clé de chiffrement K_C. Le message chiffré peut être mis sur un support non sécurisé. Le récepteur procédera au déchiffrement du message afin de retrouver le message en clair en appliquant la transformation inverse avec une clé de déchiffrement K_d : $M = D_{K_d}[E_{K_C}(M)]$.

Les opérations de chiffrement reposent en général sur l'utilisation :

- D'un algorithme qui représente le modèle mathématique choisi ;
- D'une clé représentant le paramètre de mise en œuvre.

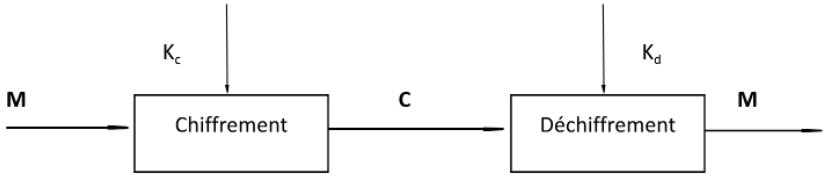

FIGURE 11 – Système de chiffrement

3.2 Chiffrement de césar

Le Chiffrement utilisé par Jules césar est la méthode de cryptographie la plus ancienne communément admise par l'Histoire. Il consiste en une substitution mono-alphabétique. Chaque lettre est remplacée ("substitution") par une seule autre, selon un certain décalage dans I 'alphabet. Il consiste à décaler les lettres de l'alphabet d'un certain rang k, qui représente le nombre de lettres à décaler. Le rang k est alors considéré comme la clé de chiffrement. Le chiffrement de César peut également être défini par une fonction de l'ensemble Z/26 Pour chaque lettre L du texte en clair qui subit un décalage de k lettres :

$$E_k(L) = (L + k) \mod 26$$

A l'inverse, pour chaque lettre C du texte chiffré résultant d'un décalage de k lettres :

$$D_k(C) = (C - k) \mod 26$$

$D_k(C)$ est la fonction de déchiffrement correspondant à et le calcul commence par 0, c'est-à-dire ; la lettre A=0.

3.2.1 Les failles du chiffrement de César

1. Pour l'alphabet standard, si $E_k(m) = m$ pour un mot m contenant au moins une lettre alors $k \equiv 0 \pmod{26}$. Donc, il n'existe pas de mots non vides qui reste identiques après le chiffrement par un k non nul.
2. Pour tout i, $m_i = m_j \Leftrightarrow c_i = c_j$. Autrement dit, la relation d'égalité entre positions est conservée par E_k. Ce qui fait que la structure de répétition est inchangée
3. Préservation de la palindromicité ; si m est un palindrome i.e. $m_i = m_{i+1-i}$ pour tout i. Alors $E_k(m)$ est également un palindrome (le mot chiffré est palindrome, mais différent du mot clair).
4. Invariance des différences relatives ; pour tous i, la différence $m_i - m_j \pmod{26}$ est égale a $c_i - c_j \pmod{26}$
5. Cas dégénérés liés à l'alphabet ; si l'on travaille avec un alphabet de taille n et que $k \equiv 0 \pmod{26}$, alors E_k est l'identité. En particulier, si n divise k, le chiffrement devient trivial. Alors l'utilisation d'un alphabet réducteur (Ex. réduction modulo 13) peut engendrer des clefs non nulles donnant l'identité.

3.3 Chiffrement de Hill

Le chiffrement publié en 1929 par Lester S. Hill est un chiffrement polygraphique, ce qui veut dire qu'on ne chiffre pas les lettres les unes après les autres, mais par paquets, le déroulement de cet algorithme se passe par quelques étapes.

1. D'abord, l'assignation numérique ; chaque lettre de l'alphabet est convertie en un nombre.
2. Ensuite, le choix de la clé ; la clé du chiffrement est une matrice carrée K de taille $n \times n$ ou n est la taille des blocs de lettres que l'on souhaite chiffrer (conditions : pour que le déchiffrement soit possible la matrice clé doit être inversible modulo m et son déterminant doit être premier avec m avec m la taille de l'alphabet choisi pour le chiffrement).
3. Puis, on passe a la division en blocs ; le message clair est divisé en blocs de n lettres, si la dernière n'a pas un bloc complet on complète par une lettre arbitraire 'X' ou 'Z', et chaque bloc de de n lettres est converti en un vecteur colonne P en utilisant l'assignation numérique.
4. Enfin, le calcul du chiffrement ; le chiffrement de chaque vecteur bloc de texte clair P en son vecteur bloc de texte chiffré C est effectué comme suit : $C = K \cdot P \pmod{m}$

Pour $n = 2$ et $m = 26$ on a :

$$\begin{pmatrix} C_k \\ C_{k+1} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} P_k \\ P_{k+1} \end{pmatrix} \pmod{26}$$

Pour déchiffrer le principe est le même que le chiffrement, on prend les lettres deux par deux. Puis on les multiplie par la matrice de déchiffrement.

$$P_{k+1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} C_{k+1} \pmod{26}$$

3.3.1 Failles de l'algorithme de chiffrement de Hill

Bien que cet algorithme soit plus complexe que le chiffrement de César, certaines situations particulières conduisent à ses failles internes, c'est-à-dire des cas ou le chiffrement « échoue » à produire un texte réellement transformé.

1. **Matrice identité :** si la matrice clé K est équivalente à la matrice identité modulo n :

$$K \equiv I \pmod{n}$$
$$C = K \cdot P \equiv I \cdot P \equiv P \pmod{n}$$

2. **Matrice de faible ordre :** si la matrice K vérifie $K^m \equiv I \pmod{n}$
3. **Existence de vecteurs propres de valeurs propre 1 :** un bloc P est un vecteur propre de la matrice K si ; $K \cdot P \equiv P \pmod{n}$ alors ce bloc ne subit aucune modification apres le chiffrement $C \equiv P \pmod{n}$

4. **Matrice non inversible modulo n :** si le déterminant de la matrice K n'est pas inversible modulo n, i.e. $\text{pgcd}(\det(K), n) \neq 1$ dans ce cas, le déchiffrement devient impossible et certaines combinaisons de blocs peuvent se chiffrer en elles-mêmes ou produire les mêmes résultats

3.3.2 Exemples

1. Matrice identité :

Soit $K = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Mot : « ME »

Valeurs : M = 12, E = 4

$$P = \begin{pmatrix} 12 \\ 4 \end{pmatrix}$$

$$C = K \cdot P = P \Rightarrow \text{le chiffrement ne modifie rien.}$$

2. Vecteurs propres de valeur propre 1 :

$K = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ et considérons $P = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ (correspondant a « BA »)

Les valeurs propres de la matrice K :

$$\det \begin{pmatrix} 1-\lambda & 1 \\ 0 & 1-\lambda \end{pmatrix} = (1-\lambda)^2 \Rightarrow \lambda = \pm 1$$

$$C = K \cdot P = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = P \Rightarrow \text{la paire de lettres reste inchangée.}$$

3. Si le mot en clair contient la paire de lettres « aa » qui correspond a $P = \begin{pmatrix} 0 \\ 0 \end{pmatrix}^t$, le bloc ne sera pas chiffré car $K \cdot P = P \ \forall K \in M(\{0, .., 26\})$

3.4 Chiffrement de Playfair

Le chiffrement de Playfair fut la première technique utilisable en pratique de chiffrement par substitution polygrammique. Il fut imaginé en 1854 par Charles Wheatsone, mais porte le nom de Lord Playfair. Le chiffre de Playfair utilise un tableau de 5x5 lettres, contenant un mot clé. Pour construire le tableau, on devait d'abord le remplir avec les lettres du mots clé en ignorant les doublons, puis le compléter avec les autres lettres de l'alphabet dans l'ordre (soit en omettant la lettre Q, soit en occupant une même case pour les lettres I et J). Pour chiffrer un message, il faut prendre les lettres 2 par 2 et appliquer les règles suivantes en fonction de la position des lettres dans la table :

— Si les 2 lettres sont identiques (ou si la taille du mot est impaire ce qui laisse un bloc incomplet a la fin) on ajoute une lettre arbitraire pour séparer les 2 ou pour compléter le bloc,
— Si les lettres se trouvent sur la même ligne de la table, il faut les remplacer par celles se trouvant immédiatement a leur droite (en bouclant sur la gauche sur le bord est atteint),
— Si les lettres apparaissent sur la même colonne, les remplacer par celles qui sont juste en dessous (en bouclant par le haut si le bas de la table est atteint),
— Sinon, si elles se trouvent sur les bords d'un rectangle on les remplace par celles se trouvant sur la même ligne mais dans le coin opposé du rectangle défini par la paire originale.

3.4.1 Failles du chiffrement Playfair

Raisonnant mathématiquement, soit M la matrice de chiffrement de Playfair, de taille 5x5 ou les indices $j \in \{1, 2, 3, 4, 5\}$.

— L_1 et L_2 : la paire de lettres claires à chiffrer.
— C_1 et C_2 : la paire de lettres chiffrées résultante.
— (i_x, j_{x1}) et (i_x, j_{x2}) : les coordonnées de L_1 et L_2 dans M.

Hypothèse de chiffrement : L_1 et L_2 sont sur la même ligne i_x, donc : $L_1 = M(i_x, j_{x1})$ et $L_2 = M(i_x, j_{x2})$ avec $j_{x1} \neq j_{x2}$.

Règle de chiffrement : chaque lettre est remplacée par la lettre immédiatement a sa droite (cycliquement) :

$$C_1 = M(i_x, (j_{x1} \mod 5) + 1) \text{ et } C_2 = M(i_x, (j_{x2} \mod 5) + 1)$$

Condition d'Échec du Chiffrement : L'échec du chiffrement se produit si la paire chiffrée est identique à la paire claire :

$$C_1 = L_1 \Rightarrow M(i_x, (j_{x1} \mod 5) + 1) = M(i_x, j_{x1})$$
$$C_2 = L_2 \Rightarrow M(i_x, (j_{x2} \mod 5) + 1) = M(i_x, j_{x2})$$

Pour que l'échec se produise pour toutes les paires de lettres sur la ligne i_x, la condition nécessaire sur la matrice M est :

$$\forall j \in \{1, 2, 3, 4, 5\}, M(i_x, (j \mod 5) + 1) = M(i_x, j)$$

Cette condition formelle signifie que tous les éléments de la ligne i_x sont identiques. Soit k la valeur commune à cette ligne :

$$M(i_x, 1) = M(i_x, 2) = M(i_x, 3) = M(i_x, 4) = M(i_x, 5) = k$$

Ce qui enfreint les conditions de l'algorithme de Playfair. Et c'est le même raisonnement pour les lettres situées sur la même colonne ou sur les bords d'un rectangle. Ce qui fait que pour qu'une paire de lettres reste inchangée après l'application de l'algorithme il faut que la matrice clé contient des doublons (matrice non valide).

4 Passwords et Attacks

4.1 Les attaques (dictionnaire + force brute)

Les attaques par mot de passe s'appuient sur deux approches fondamentales. L'attaque par dictionnaire teste une liste préétablie de mots ou variantes (mots de passe fuités, mots courants, combinaisons simples). Elle est particulièrement efficace contre des mots de passe faibles ou réutilisés. L'attaque par force brute énumère exhaustivement toutes les combinaisons possibles d'un alphabet donné jusqu'à une longueur maximale.

Soit s la taille de l'alphabet et L la longueur maximale. L'espace de recherche total est $N = \sum_{i=1}^{L} s^i = \frac{s^{L+1}-s}{s-1}$. La difficulté de casser un mot de passe augmente exponentiellement avec L et polynomialement (en pratique linéairement) avec s pour des petits changements. En pratique, l'espérance d'essais pour un mot de passe aléatoire est $\frac{N}{2}$. Le temps nécessaire s'obtient en divisant cette espérance par le nombre d'essais par seconde réalisables par l'attaquant.

$$T \approx \frac{N}{2r}$$

Où r est le nombre d'essais par seconde que peut effectuer l'attaquant (performances dépendant du matériel et de la méthode).

4.2 Protection par authentification

Pour éviter les tentatives d'attaques sur les mots de passe par force brute ou par dictionnaire, nous avons implémenté une limite stricte de tentatives de connexion. L'utilisateur ne peut effectuer que trois tentatives de connexion consécutives avant que son compte ne soit temporairement verrouillé.

Cette mesure permet de ralentir considérablement les attaques automatisées. En effet, un attaquant ne peut tester qu'un nombre très limité de combinaisons (trois) avant d'être bloqué, rendant les attaques par force brute ou par dictionnaire pratiquement inefficaces. Alors qu'une attaque non protégée pourrait tester des milliers de mots de passe par seconde, notre système limite drastiquement cette capacité.

4.3 Protection (hachage de mot de passe, salt)

Un hash (ou fonction de hachage) transforme une entrée de taille arbitraire (mot de passe, fichier, etc.) en une sortie de taille fixe (souvent exprimée en hexadécimal).

Propriétés attendues d'une bonne fonction de hachage cryptographique :

- **Déterministe :** même entrée → même sortie.
- **One-way (à sens unique) :** il est (pratiquement) impossible d'inverser la fonction pour retrouver l'entrée à partir du hash.
- **Résistance aux collisions :** difficile de trouver deux entrées distinctes qui donnent le même hash.

— **Sensibilité aux petites modifications :** une petite modification de l'entrée modifie totalement la sortie (effet avalanche).

Mathématiquement, si H est la fonction de hachage, pour un mot de passe p on calcule $h = H(p)$. L'objectif est que pour un attaquant il soit difficile de trouver p connaissant h.

Un salt est une valeur aléatoire (généralement différente pour chaque utilisateur) qui est combinée au mot de passe avant le hachage. On stocke le salt en clair avec le hash.

But pratique du salt :

— Empêcher les tables arc-en-ciel (rainbow tables) : ces tables précalculées de hash→mot de passe deviennent inutiles si chaque mot de passe est salé différemment.
— Rendre chaque hash unique même si deux utilisateurs ont le même mot de passe.
— Forcer l'attaquant à recomputer les tentatives pour chaque compte (pas un seul pré-calcul général).

Formellement, au lieu de stocker $H(p)$ on stocke $H(h\|s)$ ou s est le salt et le $\|$ la concaténation.

4.3.1 Pourquoi on a utilisé un slow hash ?

Les attaques modernes (force brute, dictionnaires accélérés par GPU/ASIC) sont très rapides. Un slow hash augmente le temps nécessaire pour tester chaque candidate de mot de passe en introduisant des itérations coûteuses en calcul.

— Si une attaque brute force prend T secondes par tentative sans stretching, et que tu utilises k itérations, le temps par tentative devient approximativement $k \times T$.
— Ainsi on augmente le coût total pour l'attaquant linéairement en k (ou plus, selon l'algorithme), tout en gardant un coût acceptable côté serveur si k est choisi prudemment.

Dans ce TP, l'algorithme utilisé combine un salt aléatoire (16 octets générés par une source cryptographiquement sûre) et une fonction itérative personnalisée. La fonction applique des opérations XOR, rotations et multiplications sur des mots de 32 bits, répétées « iterations » fois. Cette approche introduit un facteur de ralentissement utile pendant la vérification.

5 Stéganographie

5.1 Principe

La stéganographie est l'art de dissimuler un message secret dans un support anodin, de manière à ce que l'existence même de ce message passe inaperçue. Contrairement à la cryptographie qui rend le message illisible, la stéganographie cache son existence : le fichier porteur (image, audio, vidéo ou texte) semble parfaitement normal à première vue.

Dans notre projet, nous avons implémenté la stéganographie par audio. Celle-ci consiste à cacher le message secret directement dans les échantillons du fichier audio.

5.2 Fonctionnement

L'utilisateur saisit en entrée le message secret ainsi que le fichier audio porteur. Le programme génère alors un fichier audio modifié contenant le message caché, que le destinataire pourra extraire par la suite.

La dissimulation du message secret dans le support audio se déroule en plusieurs étapes via différentes fonctions :

5.2.1 Étape 1 : Conversion du message en binaire (fonction `text_to_binary`)

Le programme prend le message texte secret et le convertit en son équivalent en code ASCII, puis en binaire. Par exemple, le caractère 'H' (code ASCII 72) devient `01001000` en binaire.

5.2.2 Étape 2 : Traitement du fichier audio (fonction `hide_in_audio`)

Le programme lit le fichier audio (notre implémentation traite les fichiers au format WAV). Un fichier audio est constitué d'une suite de valeurs numériques appelées **échantillons**, chacune représentant l'amplitude du son à un instant donné.

Le programme récupère plusieurs paramètres importants :

- Le nombre d'échantillons totaux
- La largeur des échantillons (sample width) en bytes
- Le taux d'échantillonnage (framerate) en Hz
- Le nombre de canaux (mono ou stéréo)

Ces échantillons sont ensuite convertis en un tableau de nombres manipulables.

5.2.3 Étape 3 : Modification des bits de poids faible (LSB)

Le programme utilise la technique du **LSB (Least Significant Bit)** pour cacher le message. Pour chaque bit du message binaire, il modifie le bit de poids faible (le dernier bit) d'un échantillon audio. Cette modification est imperceptible à l'oreille humaine car elle ne change la valeur de l'échantillon que de ± 1.

Exemple :

- Échantillon original : $24567 = 0101111111110111_2$ en binaire
- Si le bit du message = 0, le LSB devient $0 \rightarrow 24566$
- Si le bit du message = 1, le LSB devient $1 \rightarrow 24567$

5.2.4 Étape 4 : Sauvegarde du fichier modifié

Les échantillons modifiés sont sauvegardés dans un nouveau fichier WAV. Ce fichier diffère légèrement de l'original au niveau des données, mais reste indiscernable à l'écoute.

5.3 Extraction du message

Les fonctions `binary_to_text` et `extract_from_audio` effectuent le processus inverse pour extraire le message secret du fichier audio :

1. **Lecture du fichier stéganographié :** Le programme lit les échantillons du fichier audio modifié
2. **Extraction des LSB :** Pour chaque échantillon, le programme récupère le bit de poids faible
3. **Reconstruction du message binaire :** Les bits extraits sont concaténés pour reformer le message en binaire
4. **Conversion en texte :** Le message binaire est regroupé par paquets de 8 bits (1 octet = 1 caractère) et reconverti en texte en utilisant la table ASCII

Un délimiteur `###END###` est utilisé pour identifier la fin du message caché et éviter d'extraire du bruit aléatoire.

5.4 Implémentation

```
def hide_in_audio(input_path, secret_message, output_path):
    audio = wave.open(input_path,'rb')
    n_channels=audio.getnchannels()
    sample_width = audio.getsampwidth()
    framerate = audio.getframerate()
    n_frames=audio.getnframes()

    frames=audio.readframes(n_frames)
    audio.close()

    audio_data = np.frombuffer(frames, dtype=np.int16).copy()
    #audio entiers 16 bits, format standard des audios cd
    message_with_end = secret_message + "###END###"
    binary_message = text_to_binary(message_with_end)
    if len(binary_message)>len(audio_data):
        raise ValueError("ce message est trop long pour l'audio")

    for i in range (len (binary_message)):
        audio_data[i] = (audio_data[i] & ~1) | int(binary_message[i])  #
   modification du lsb de chaque echantillon audio avec le bit du
   message

    stego_audio=wave.open(output_path, 'wb')
    stego_audio.setnchannels(n_channels)
    stego_audio.setsampwidth(sample_width)
    stego_audio.setframerate(framerate)
    stego_audio.writeframes(audio_data.tobytes())
    stego_audio.close()

    return True
```

Listing 1 – Fonction hide_in_audio

5.5 Avantages et limites

5.5.1 Avantages

— Le fichier audio reste parfaitement audible
— La modification est indétectable sans analyse approfondie
— Capacité de stockage importante (un bit par échantillon)

5.5.2 Limites

— Sensible à la compression (MP3 détruit le message caché)
— Nécessite un fichier audio non compressé (WAV)
— La taille du message est limitée par le nombre d'échantillons disponibles

6 Évaluation des Performances

6.1 Tests unitaires

Des tests ont été effectués pour chaque composant du framework.

6.2 Comparaison des algorithmes

Algorithme	Sécurité	Viabilité	Complexité
Caesar	Faible	Très rapide	O(n)
Hill	Moyenne	Rapide	$O(n^2)$
PlayFair	Moyenne	Moyenne	O(n)

TABLE 1 – Comparaison des algorithmes de chiffrement

7 Conclusion

Ce projet nous a permis d'explorer les fondamentaux de la sécurité informatique à travers l'implémentation d'un framework de chiffrement complet intégrant trois algorithmes cryptographiques (Caesar, Hill et PlayFair) ainsi qu'un module de stéganographie audio.

7.1 Principaux acquis

L'implémentation pratique nous a confrontés aux défis réels de la cryptographie. La réalisation du module de stéganographie par la technique LSB s'est révélée particulièrement instructive, démontrant qu'il est possible de dissimuler des informations de manière imperceptible dans des fichiers audio. Les simulations d'attaques ont mis en évidence

l'importance de la complexité des mots de passe et les vulnérabilités des systèmes de sécurité.

7.2 Compétences développées

Ce travail pratique nous a permis de maîtriser Python pour l'implémentation d'algorithmes cryptographiques, de comprendre la structure interne des fichiers audio, et de développer nos compétences en analyse de sécurité et en travail collaboratif.

7.3 Perspectives

Ce framework pourrait être amélioré en intégrant des algorithmes modernes (AES, RSA), en étendant la stéganographie aux images et vidéos, et en développant une interface utilisateur plus intuitive.

En conclusion, ce projet nous a offert une vision concrète de la cybersécurité et souligné son importance cruciale dans notre société numérique. Les connaissances acquises constituent une base solide pour nos futurs projets en sécurité informatique.