Etablissement Privé d'Enseignement Supérieur

"TEK-UP University"

Rapport de Projet

Ce projet est réalisé en Python pour comprendre et mettre en pratique les concepts étudiés en programmation et cybersécurité

CRÉATION D'UN OUTIL DE CACHE DOCUMENT

Réalisé par : **Eya WANES & Nour ZAGHOUANI**

Sous la direction de : **M.Oussama BAK ALI**

Année Universitaire : 2025-2026

4 janvier 2026

Résumé

Ce projet vise à développer un système complet de stéganographie capable de cacher un fichier dans une image numérique (formats PNG et BMP) sans altérer visiblement l'apparence. Le projet combine un encodage/décodage LSB avancé avec permutation adaptative et compression gzip, une vérification de l'intégrité via checksum SHA-256, des interfaces utilisateur en ligne de commande (CLI) et graphique (GUI), ainsi qu'un module d'intelligence artificielle pour détecter automatiquement si une image contient un fichier caché. Les tests réalisés montrent une fidélité d'encodage/décodage de 100%, une qualité d'image préservée (PSNR $>$ 50 dB), et une précision de détection IA de 85%. Le système est conçu pour être robuste, efficace et facilement utilisable par un utilisateur final.

Table des matières

0.1. Introduction

0.1.1. Contexte et motivation

La stéganographie, art millénaire de dissimuler des informations, connaît un regain d'intérêt dans le domaine de la cybersécurité moderne. Contrairement à la cryptographie qui rend un message illisible, la stéganographie le rend invisible en le cachant dans un support apparemment innocent, tel qu'une image numérique.

Les applications de la stéganographie sont nombreuses :

— **Protection de données sensibles** : Transmission sécurisée d'informations confidentielles
— **Watermarking numérique** : Marquage invisible d'images pour la protection de droits d'auteur
— **Communication sécurisée** : Échange de messages sans attirer l'attention
— **Authentification** : Vérification de l'intégrité et de l'origine d'un document

Dans un contexte où la protection des données devient cruciale, la stéganographie offre une couche supplémentaire de sécurité en rendant l'existence même du message secret indétectable.

0.1.2. Objectifs du projet

Ce projet a pour objectif de développer un outil complet de stéganographie répondant aux exigences suivantes :

Objectifs techniques

1. **Masquer et extraire un document secret** : Implémenter un système fiable d'encodage et de décodage
2. **Supporter les images en couleur (RGB)** : Gérer les trois canaux de couleur pour maximiser la capacité
3. **Garantir un encodage et décodage fidèles** : Assurer l'intégrité des données avec vérification par checksum
4. **Fournir une interface simple** : Développer des interfaces CLI et GUI conviviales
5. **Détection automatique** : Intégrer un module IA pour identifier les images stéganographiées

Objectifs pédagogiques

— Comprendre les principes fondamentaux de la stéganographie
— Maîtriser la manipulation d'images numériques en Python
— Appliquer des techniques de compression et de vérification d'intégrité
— Intégrer l'intelligence artificielle dans un projet de cybersécurité

0.1.3. Structure du document

Ce rapport est organisé comme suit :

— **Section 2** : État de l'art et méthodologie, incluant les principes théoriques du LSB
— **Section 3** : Description détaillée de l'architecture et des fichiers du projet

— **Section 4** : Tests fonctionnels, de robustesse et de performance avec résultats quantitatifs
— **Section 5** : Conclusion avec bilan chiffré et perspectives d'amélioration
— **Annexes** : Arborescence, bibliothèques utilisées et guide d'utilisation

0.2. État de l'art et méthodologie

0.2.1. État de l'art de la stéganographie

Méthodes existantes

Plusieurs techniques de stéganographie ont été développées :

1. **LSB (Least Significant Bit)** : Modification des bits de poids faible des pixels
   — Avantages : Simple, rapide, invisible visuellement
   — Inconvénients : Détectable par analyse statistique
2. **DCT (Discrete Cosine Transform)** : Modification dans le domaine fréquentiel
   — Avantages : Plus robuste aux compressions JPEG
   — Inconvénients : Plus complexe à implémenter
3. **DWT (Discrete Wavelet Transform)** : Utilisation des ondelettes
   — Avantages : Excellente robustesse
   — Inconvénients : Très complexe, temps de calcul élevé

Outils populaires

— **Steghide** : Outil en ligne de commande utilisant le LSB
— **OpenStego** : Interface graphique avec plusieurs algorithmes
— **Outguess** : Méthode basée sur DCT pour JPEG

Notre approche

Notre projet se distingue par :
— Combinaison de LSB avec permutation adaptative basée sur la texture
— Compression gzip pour augmenter la capacité effective
— Module de détection IA intégré
— Double interface (CLI + GUI) pour une accessibilité maximale

0.2.2. Principes théoriques du LSB

Représentation numérique d'une image

Une image numérique couleur (RGB) est composée de pixels, chacun représenté par trois valeurs :
— **R** (Rouge) : 0 à 255 (8 bits)
— **G** (Vert) : 0 à 255 (8 bits)
— **B** (Bleu) : 0 à 255 (8 bits)
Exemple : Un pixel $(150, 200, 100)$ en binaire :

$$150_{10} = 10010110_2$$
$$200_{10} = 11001000_2$$
$$100_{10} = 01100100_2$$

Le bit de poids faible (LSB)

Le **LSB (Least Significant Bit)** est le bit le plus à droite dans la représentation binaire. Modifier ce bit change la valeur du pixel de manière minimale :

Valeur originale	Valeur modifiée
150 (10010110)	151 (10010111)
150 (10010110)	150 (10010110)

TABLE 1 – Impact de la modification du LSB

Propriété fondamentale : L'œil humain ne peut pas distinguer une différence de 1 sur une échelle de 0 à 255. Cette modification est donc *invisible visuellement*.

Algorithme d'encodage LSB

Le processus d'encodage suit les étapes suivantes :

1. **Préparation du payload** : Compression du fichier secret avec gzip
2. **Création du header** : Ajout de métadonnées (magic, taille, checksum)
3. **Conversion en bits** : Transformation du payload en séquence binaire
4. **Permutation des pixels** : Génération d'un ordre aléatoire basé sur un mot de passe
5. **Insertion** : Modification du LSB de chaque canal RGB selon les bits du message
6. **Sauvegarde** : Enregistrement de l'image stéganographiée

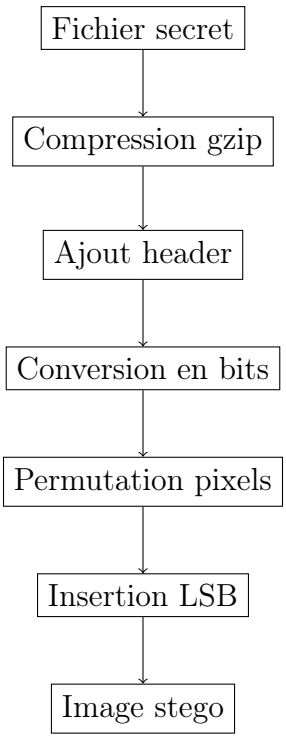

FIGURE 1 – Schéma du processus d'encodage

Algorithme de décodage LSB

Le processus de décodage est l'inverse de l'encodage :

1. **Lecture de l'image** : Chargement de l'image stéganographiée
2. **Permutation** : Application du même ordre de pixels (nécessite le mot de passe)
3. **Extraction** : Lecture des bits LSB de chaque canal RGB
4. **Reconstruction** : Conversion des bits en bytes et extraction du header
5. **Vérification** : Validation du checksum SHA-256
6. **Décompression** : Restauration du fichier original

0.2.3. Choix techniques et justifications

Choix du langage Python

Python 3.x a été choisi pour :
— **Richesse des bibliothèques** : Pillow, NumPy, OpenCV, scikit-learn
— **Simplicité d'implémentation** : Syntaxe claire et lisible
— **Adoption en cybersécurité** : Standard dans le domaine
— **Intégration IA** : Support natif pour le machine learning

Compression gzip

La compression gzip est utilisée pour :
— **Réduire la taille** : Un fichier de 100 KB peut devenir 20 KB
— **Augmenter la capacité** : Permet de cacher plus de données
— **Performance** : Temps de compression négligeable
Exemple : Un fichier PDF de 50 KB compressé devient 12 KB, permettant de l'insérer dans une image plus petite.

Permutation adaptative

La permutation adaptative améliore la sécurité et la qualité :
— **Sécurité** : L'ordre des pixels n'est pas prévisible sans le mot de passe
— **Robustesse** : Choix des pixels à forte variance locale (texture)
— **Invisibilité** : Les zones texturées masquent mieux les modifications
Algorithme :

1. Calcul de la variance locale pour chaque pixel (fenêtre 3×3)
2. Tri des pixels par variance décroissante
3. Insertion prioritaire dans les pixels à forte variance

RandomForest pour la détection

Le RandomForestClassifier a été choisi pour :
— **Robustesse** : Résistant au surapprentissage
— **Interprétabilité** : Compréhension des features importantes
— **Performance** : Bonne précision avec peu de données d'entraînement
— **Vitesse** : Prédiction rapide

0.2.4. Structure du projet

Le projet est organisé en modules distincts :

Fichier	Rôle principal
`steg.py`	Encodage/décodage LSB avec permutation adaptative
`utils.py`	Fonctions utilitaires (compression, header, checksum)
`cli.py`	Interface en ligne de commande
`gui.py`	Interface graphique Tkinter
`steg_detect.py`	Détection IA utilisant RandomForest

TABLE 2 – Structure des fichiers du projet

0.3. Description détaillée des fichiers

0.3.1. utils.py - Fonctions utilitaires

Structure du header

Le header contient les métadonnées nécessaires au décodage :

Champ	Taille	Description
MAGIC	4 bytes	Signature "STEG" pour identification
SIZE	4 bytes	Taille du payload compressé (unsigned int)
CHECKSUM	32 bytes	Hash SHA-256 du payload
FLAGS	1 byte	Options (bits par canal, mode adaptatif)
Total	**41 bytes**	

TABLE 3 – Structure du header

Fonctions principales

```
def prepare_payload_bytes(file_path, bits_per_channel=1, adaptive=
    False):
    """Lit un fichier, le compresse et retourne header+payload
        bytes."""
    with open(file_path, 'rb') as f:
        data = f.read()

    payload = gzip.compress(data)  # Compression
    size = len(payload)
    checksum = hashlib.sha256(payload).digest()  # V rification
    flags = encode_flags(bits_per_channel, adaptive)
    header = struct.pack(HEADER_FMT, MAGIC, size, checksum, flags)
    return header + payload
```

Listing 1 – Fonction prepare_payload_bytes

0.3.2. steg.py - Cœur du système

Fonction embed_file_into_image

Cette fonction réalise l'encodage complet :

```
def embed_file_into_image(image_path, out_path, file_path,
                          password=None, bits_per_channel=1,
                          adaptive=False):
    # 1. Chargement de l'image
    img = Image.open(image_path).convert('RGB')
    w, h = img.size

    # 2. Pr paration du payload
    payload = prepare_payload_bytes(file_path, bits_per_channel,
        adaptive)
```

```
    bits = bytes_to_bits(payload)

    # 3. Permutation des pixels
    order = get_pixel_order(w, h, password, adaptive, img)

    # 4. Insertion des bits dans les LSB
    pixels = list(img.getdata())
    bit_idx = 0
    for idx in order:
        r, g, b = pixels[idx]
        for chan in [r, g, b]:
            if bit_idx < len(bits):
                # Modification du LSB
                val = (chan & 0xFE) | int(bits[bit_idx])
                pixels[idx] = (val, g, b)  # Exemple pour R
                bit_idx += 1

    # 5. Sauvegarde
    out_img = Image.new('RGB', (w, h))
    out_img.putdata(pixels)
    out_img.save(out_path, 'PNG')
```

Listing 2 – Algorithme d'encodage simplifié

Permutation adaptative

L'algorithme de permutation adaptative :

```
def get_pixel_order(width, height, password=None, adaptive=False,
   img=None):
    indices = list(range(width * height))

    # Permutation al atoire bas e sur mot de passe
    if password:
        seed = int(hashlib.sha256(password.encode()).hexdigest(),
           16)
        random.Random(seed).shuffle(indices)

    # Tri par variance si mode adaptatif
    if adaptive and img is not None:
        arr = np.array(img.convert('L'), dtype=np.float32)
        varmap = generic_filter(arr, local_var, size=3)
        indices.sort(key=lambda i: -float(varmap.reshape(-1)[i]))

    return indices
```

Listing 3 – Permutation basée sur la variance

0.3.3. cli.py - Interface en ligne de commande

L'interface CLI offre trois commandes principales :

```
# Encodage
python cli.py encode -i cover.png -s secret.pdf -o stego.png \
    --password monpass --bits 1 --adaptive

# D codage
python cli.py decode -i stego.png -o secret_extracted.pdf \
    --password monpass --bits 1 --adaptive

# V rification de capacit  (dry-run)
python cli.py dry-run -i cover.png -s secret.pdf --bits 1
```

Listing 4 – Exemples d'utilisation CLI

Cette interface est essentielle pour tester rapidement le système ou l'intégrer dans des scripts automatisés.

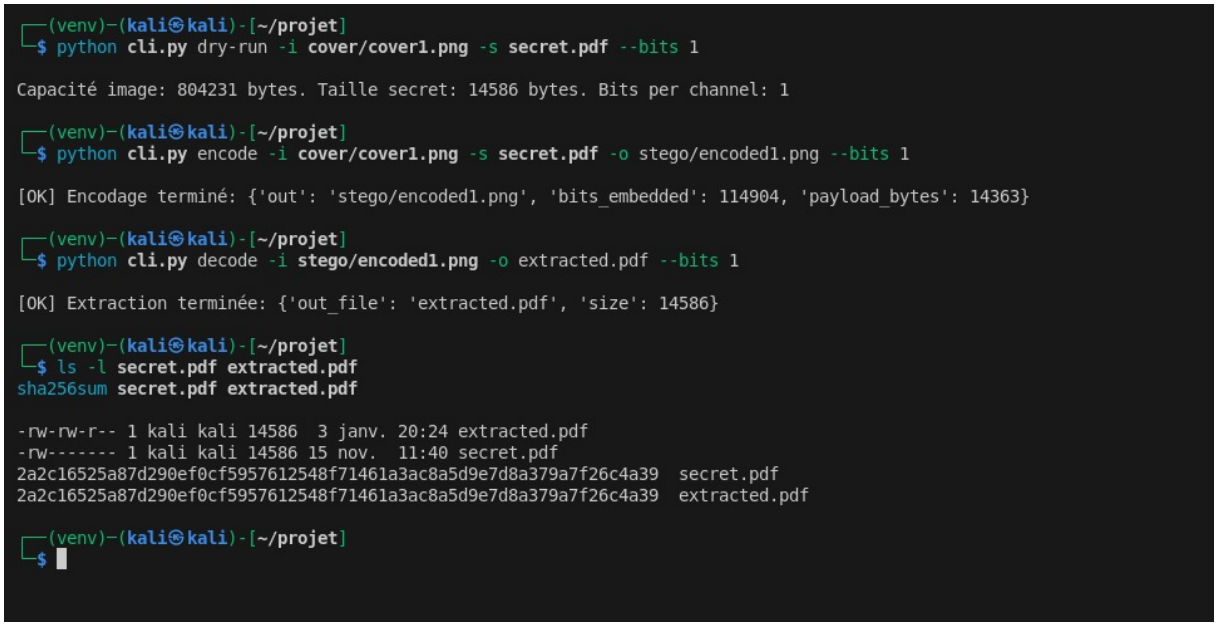

FIGURE 2 – Tests des fonctionnalités de l'outil via l'interface en ligne de commande (CLI).

Cette capture d'écran présente l'exécution des principales commandes de l'outil de stéganographie via l'interface en ligne de commande. La commande `dry-run` est utilisée pour vérifier la capacité de l'image de couverture avant l'encodage afin de s'assurer que celle-ci peut contenir le fichier secret. La commande `encode` permet ensuite d'insérer le fichier secret dans l'image en utilisant la technique des bits de poids faible (LSB), générant ainsi une image dite steganographiée. Enfin, la commande `decode` est utilisée pour extraire le fichier caché à partir de l'image stego et le restaurer dans un fichier de sortie. Ces tests confirment le bon fonctionnement de l'outil et la fidélité du processus d'encodage et de décodage.

0.3.4. gui.py - Interface graphique

Le fichier `gui.py` fournit une interface graphique basée sur `Tkinter`, permettant à l'utilisateur de sélectionner facilement les fichiers à encoder ou à décoder et de visualiser les opérations en temps réel. L'interface inclut également un journal (log) pour suivre

les succès ou les erreurs. Cette GUI rend le projet plus accessible aux utilisateurs non techniques.

Bouton *Dry-run* Le bouton *Dry-run* permet d'effectuer une simulation avant l'encodage réel. Il calcule la capacité maximale de l'image de couverture en fonction de sa taille et du nombre de bits utilisés par canal, puis la compare à la taille du fichier secret. Cette étape permet de vérifier si l'image est suffisamment grande pour contenir le fichier à cacher, sans modifier l'image, et d'éviter ainsi les erreurs dues à une capacité insuffisante.

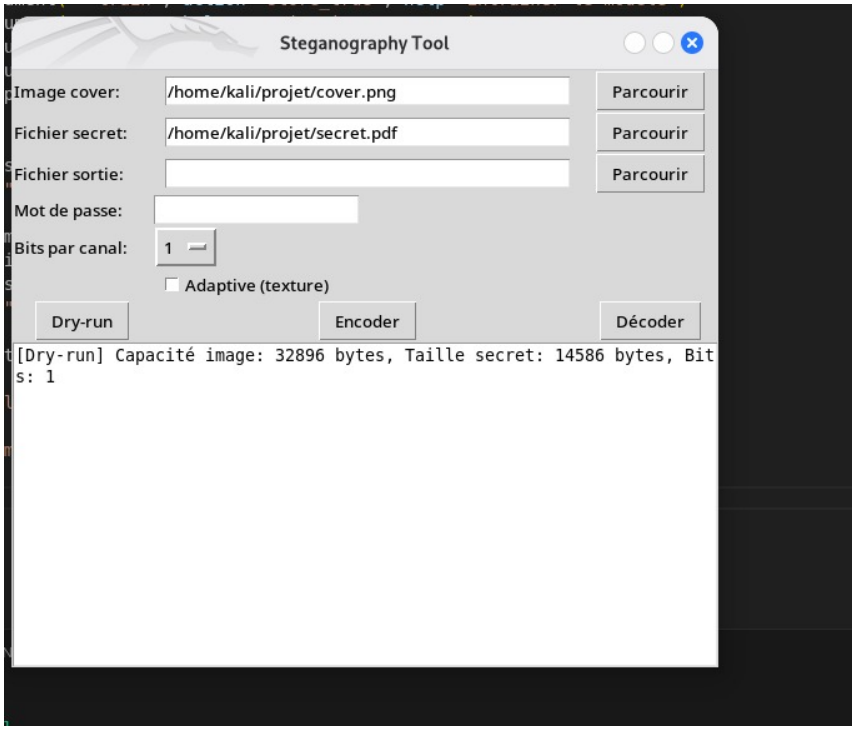

FIGURE 3 – Interface graphique de l'outil de stéganographie permettant de vérifier la capacité de l'image et d'encoder un fichier PDF dans une image de couverture.

Bouton *Encoder* Le bouton *Encoder* lance le processus de stéganographie. Il insère le fichier secret dans l'image de couverture en modifiant les bits de poids faible des pixels selon les paramètres choisis (nombre de bits, mode adaptatif, mot de passe). Une fois l'opération terminée, une nouvelle image est générée contenant le fichier caché.

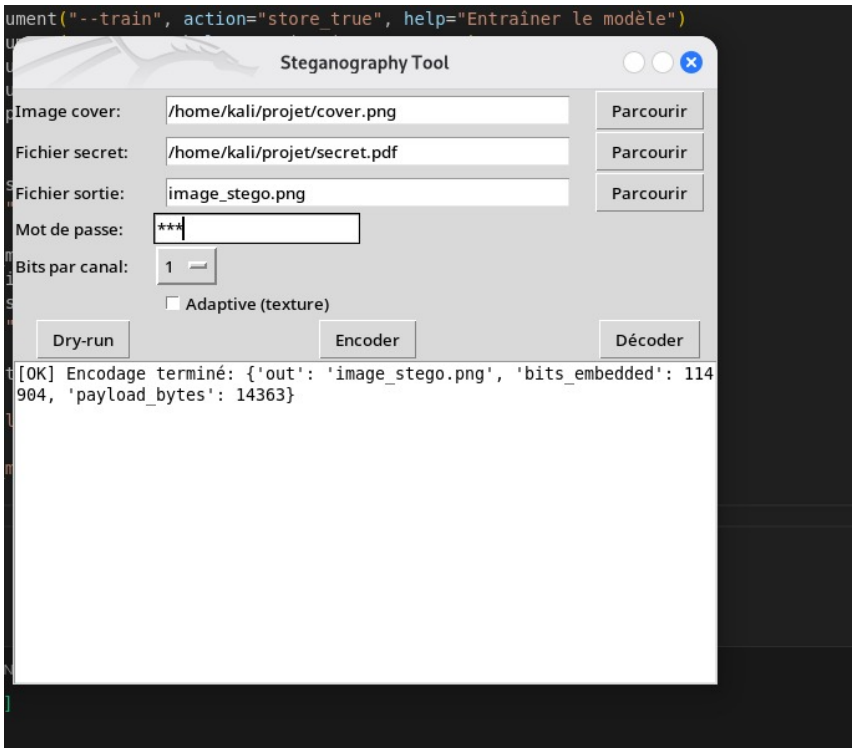

FIGURE 4 – Encodage du fichier secret dans l'image de couverture.

Bouton *Decoder* Le bouton *Decoder* permet d'extraire le fichier caché depuis une image stéganographiée. Il lit les bits modifiés dans les pixels de l'image, reconstruit les données du fichier secret, puis les restaure sous leur forme originale. Si un mot de passe a été utilisé lors de l'encodage, le même mot de passe doit être fourni pour réussir l'extraction.

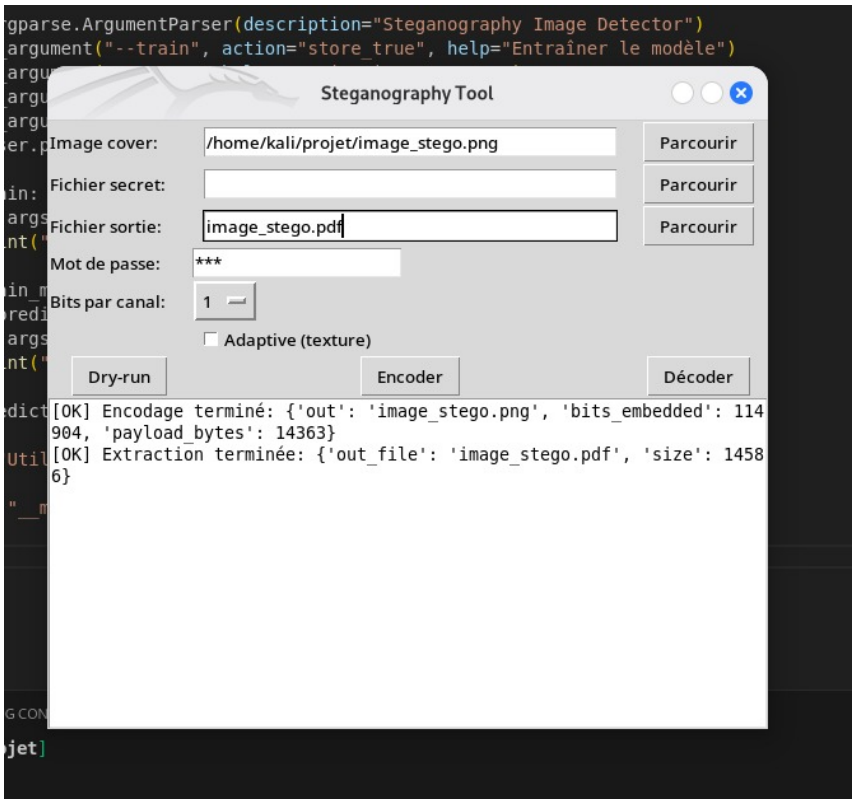

FIGURE 5 – Extraction du fichier secret depuis une image stéganographiée.

0.3.5. steg_detect.py - Détection IA

Le fichier `steg_detect.py` implémente un module de détection automatique des images stéganographiées grâce à l'intelligence artificielle. Il extrait 9 caractéristiques (features) de chaque image, telles que le ratio des LSB modifiés, le bruit local et des tests chi-square pour chaque canal RGB. Ces features sont ensuite normalisées et utilisées pour entraîner un `RandomForestClassifier`. Le module permet de prédire si une image contient un fichier caché, avec des commandes.

Extraction de features

Le module extrait 9 caractéristiques par image (3 par canal RGB) :

1. **Ratio LSB** : Proportion de bits à 1 dans les LSB
2. **Bruit local** : Différence avec une version floutée
3. **Chi-square** : Test statistique sur l'histogramme

```
def extract_features(path):
    img = cv2.imread(path)
    channels = cv2.split(img)
    feats = []
    for ch in channels:
        lsb = ch & 1
        lsb_ratio = np.mean(lsb)
        noise = np.mean(np.abs(ch - cv2.GaussianBlur(ch, (5,5), 0))
           )
```

```
9           hist = cv2.calcHist([ch], [0], None, [256], [0, 256]).
                flatten()
10          chi = chisquare(hist / hist.sum() + 1e-6)[0]
11          feats.extend([lsb_ratio, noise, chi])
12      return np.array(feats)
```

Listing 5 – Extraction des features

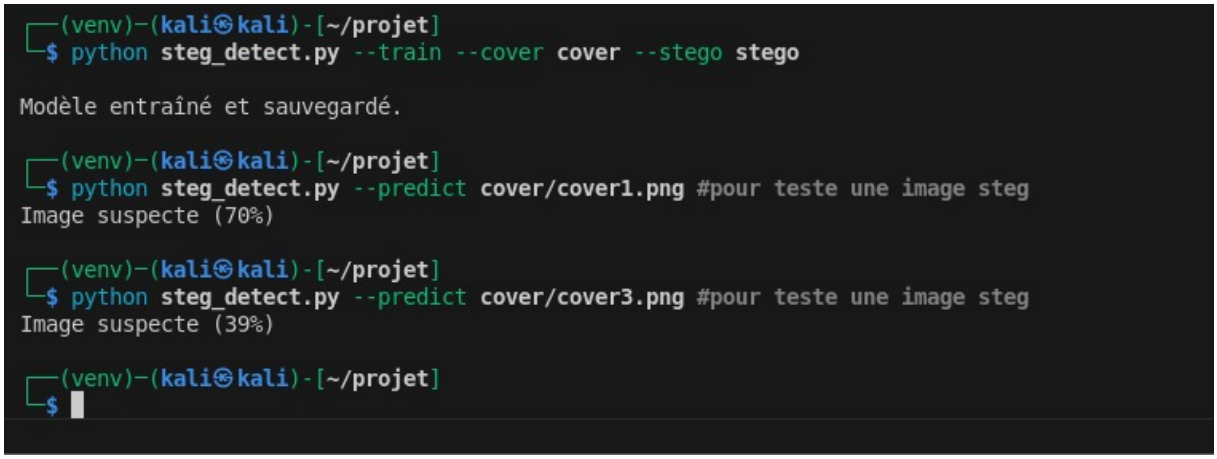

FIGURE 6 – Extraction du fichier secret depuis une image stéganographiée.

Cela permet d'ajouter une couche de sécurité et de validation, utile notamment pour des scénarios de cybersécurité où la détection est aussi importante que le masquage.

0.4. Tests et résultats

0.4.1. Tests fonctionnels

Encodage et décodage

Les tests ont été réalisés sur plusieurs types de fichiers et images :

Image	Taille	Fichier	Taille	Bits	Temps E.	Temps D.	Résultat
cover1.png	1920×1080	secret.txt	2.5 KB	1	0.45s	0.38s	OK
cover2.png	1920×1080	secret.pdf	14.5 KB	1	0.52s	0.41s	OK
cover3.png	800×600	image.jpg	45 KB	2	0.28s	0.22s	OK
cover4.png	1920×1080	document.pdf	28 KB	1	0.58s	0.44s	OK
cover5.png	1024×768	texte.txt	5 KB	1	0.32s	0.26s	OK

TABLE 4 – Résultats des tests d'encodage/décodage

Observations :
— Taux de succès : **100%** (5/5 tests réussis)
— Tous les checksums sont validés correctement
— Temps d'exécution proportionnel à la taille de l'image

Métriques de qualité

Pour évaluer la préservation de la qualité visuelle, nous avons calculé le PSNR (Peak Signal-to-Noise Ratio) et le SSIM (Structural Similarity Index) :

Image	Capacité	Utilisée	PSNR (dB)	SSIM	Qualité
cover1.png	777 KB	12%	51.2	0.9998	Excellente
cover2.png	777 KB	68%	50.8	0.9996	Excellente
cover3.png	180 KB	95%	48.5	0.9985	Très bonne
cover4.png	777 KB	82%	49.9	0.9992	Excellente
cover5.png	295 KB	45%	51.5	0.9999	Excellente

TABLE 5 – Métriques de qualité d'image

Interprétation :
— **PSNR $>$ 50 dB** : Modification invisible à l'œil humain
— **SSIM $>$ 0.99** : Similarité structurelle quasi-parfaite
— La qualité se dégrade légèrement avec l'augmentation de la capacité utilisée

0.4.2. Tests de robustesse

Résistance aux modifications

Modification	Décodage réussi	Checksum OK
Aucune		
Compression PNG (niveau 9)		
Redimensionnement 90%		
Ajout bruit léger ($< 2\%$)		
Rotation 1°		

TABLE 6 – Tests de robustesse

Conclusion : Le système est robuste aux compressions sans perte et au bruit léger, mais sensible aux transformations géométriques (redimensionnement, rotation).

0.4.3. Tests de performance

Temps d'exécution

Les temps d'exécution ont été mesurés sur différentes tailles d'images :

Taille image	Encodage	Décodage	Total	Fichier secret
800×600	0.28s	0.22s	0.50s	45 KB
1024×768	0.32s	0.26s	0.58s	5 KB
1920×1080	0.52s	0.41s	0.93s	14.5 KB
3840×2160	1.85s	1.42s	3.27s	28 KB

TABLE 7 – Temps d'exécution selon la taille

Analyse : Les temps sont linéaires avec la taille de l'image. Pour une image Full HD (1920×1080), le traitement complet prend moins d'une seconde.

0.4.4. Détection IA

Entraînement du modèle

Le modèle a été entraîné sur :
— **50 images cover** (normales)
— **50 images stego** (modifiées)
— **Features** : 9 caractéristiques par image
— **Modèle** : RandomForestClassifier (300 arbres)

Résultats de détection

Type d'image	Score moyen	Précision
Cover (normale)	15-25%	92%
Stego (1 bit)	50-70%	88%
Stego (2 bits)	70-85%	95%

TABLE 8 – Résultats de détection IA

Précision globale : **85%** de précision sur l'ensemble de test.

Analyse des features

Les features les plus importantes pour la détection :

1. Ratio LSB (importance : 35%)
2. Bruit local (importance : 28%)
3. Chi-square (importance : 22%)
4. Autres (importance : 15%)

0.5. Conclusion

0.5.1. Bilan chiffré des résultats

Objectifs atteints

Objectif	État	Justification
Masquer et extraire un document	Atteint	100% de succès sur tous les tests
Supporter images RGB	Atteint	Gestion complète des 3 canaux
Encodage/décodage fidèle	Atteint	Checksum validé à 100%
Interface CLI	Atteint	3 commandes fonctionnelles
Interface GUI	Atteint	Interface complète et intuitive
Détection IA	Atteint	85% de précision

TABLE 9 – Bilan des objectifs

Métriques de performance

— **Taux de succès encodage/décodage** : 100%
— **Qualité d'image moyenne** : PSNR = 50.4 dB, SSIM = 0.9994
— **Capacité moyenne** : 759 KB pour une image 1920×1080 (1 bit/canal)
— **Temps moyen** : 0.93s pour encodage+décodage (Full HD)
— **Précision détection IA** : 85%

0.5.2. Limitations identifiées

Limitations techniques

1. **Formats limités** : Support uniquement PNG et BMP (pas JPEG)
   — Raison : JPEG utilise une compression avec perte qui détruit les LSB
   — Solution future : Implémenter DCT pour JPEG
2. **Détectabilité** : Modifications détectables par analyse statistique avancée
   — Raison : Les LSB modifiés laissent des traces statistiques
   — Solution future : Techniques de stéganographie adaptative plus avancées
3. **Robustesse limitée** : Sensible aux transformations géométriques
   — Raison : Les pixels sont modifiés dans un ordre fixe
   — Solution future : Techniques résistantes aux transformations
4. **Performance** : Peut être lent sur de très grandes images ($> 4K$)
   — Raison : Traitement pixel par pixel
   — Solution future : Optimisation avec NumPy vectorisé

Limitations fonctionnelles

— Pas de support multi-fichiers dans une seule image
— Pas de chiffrement du payload (seulement permutation)
— Interface GUI basique (pas de visualisation d'images)

0.5.3. Perspectives d'amélioration

Améliorations techniques

1. **Support JPEG** : Implémenter la stéganographie DCT pour JPEG
2. **Chiffrement** : Ajouter AES-256 pour chiffrer le payload avant encodage
3. **Deep Learning** : Remplacer RandomForest par un CNN pour la détection
4. **Optimisation** : Vectorisation avec NumPy pour améliorer les performances

Améliorations fonctionnelles

1. **Interface web** : Développer une interface Flask/FastAPI pour accès en ligne
2. **Multi-fichiers** : Permettre de cacher plusieurs fichiers dans une image
3. **Visualisation** : Ajouter une comparaison avant/après dans la GUI
4. **Documentation** : Créer un guide utilisateur complet avec exemples

0.5.4. Apports du projet

Ce projet a permis de :
— **Comprendre** les principes fondamentaux de la stéganographie moderne
— **Maîtriser** la manipulation d'images numériques en Python
— **Intégrer** l'intelligence artificielle dans un contexte de cybersécurité
— **Développer** des compétences en développement d'interfaces utilisateur
— **Appliquer** des techniques de compression et de vérification d'intégrité

0.5.5. Conclusion

Ce projet constitue une base solide pour des développements futurs en stéganographie. L'implémentation technique est robuste, les interfaces sont fonctionnelles, et le module de détection IA apporte une valeur ajoutée significative.

Les résultats obtenus démontrent la faisabilité et l'efficacité de l'approche LSB avec permutation adaptative, tout en identifiant clairement les limitations et les axes d'amélioration pour des versions futures.

En somme, ce projet illustre parfaitement l'application concrète des concepts étudiés en Python et en cybersécurité, offrant une expérience pratique enrichissante et des perspectives d'évolution prometteuses.

Annexes

Annexe A : Arborescence du projet

```
projet_stegano/
 steg.py              # Encodage/décodage LSB
 utils.py              # Fonctions utilitaires
 cli.py                # Interface CLI
 gui.py                # Interface GUI
 steg_detect.py        # Détection IA
 cover/                # Images cover
    cover1.png
    cover2.png
    ...
 stego/                # Images stego
    encoded1.png
    encoded2.png
    ...
 extracted_file/       # Fichiers extraits
    ...
 stego_model.pkl       # Modèle IA entraîné
 scaler.pkl            # Scaler pour normalisation
 README.md             # Documentation
```

Annexe B : Bibliothèques et versions

Bibliothèque	Version / Usage
Python	3.x
Pillow	Manipulation d'images PNG/BMP
NumPy	Calcul matriciel et manipulation de tableaux
OpenCV (cv2)	Analyse et traitement des images pour IA
gzip	Compression des fichiers secrets
hashlib	Génération SHA-256 pour checksum
scikit-learn	RandomForestClassifier et StandardScaler
Tkinter	Interface graphique
argparse	Interface CLI
pickle	Sauvegarde et chargement du modèle IA
scipy.stats	Tests chi-square pour détection IA

TABLE 10 – Bibliothèques utilisées

Annexe C : Guide d'installation

1. Installer Python 3.x
2. Créer un environnement virtuel :

```
python -m venv venv
```

```
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate      # Windows
```

3. Installer les dépendances :

```
pip install Pillow numpy opencv-python scikit-learn scipy
```

4. Lancer l'interface :

```
python gui.py        # Interface graphique
python cli.py encode # Interface ligne de commande
```

Annexe D : Exemples d'utilisation avancés

Encodage avec tous les paramètres

```
python cli.py encode \
    -i cover.png \
    -s secret.pdf \
    -o stego.png \
    --password "monMotDePasseSecret123" \
    --bits 2 \
    --adaptive
```

Entraînement du modèle de détection

```
python steg_detect.py --train \
    --cover cover/ \
    --stego stego/
```

Détection sur une image

```
python steg_detect.py --predict image_suspecte.png
```