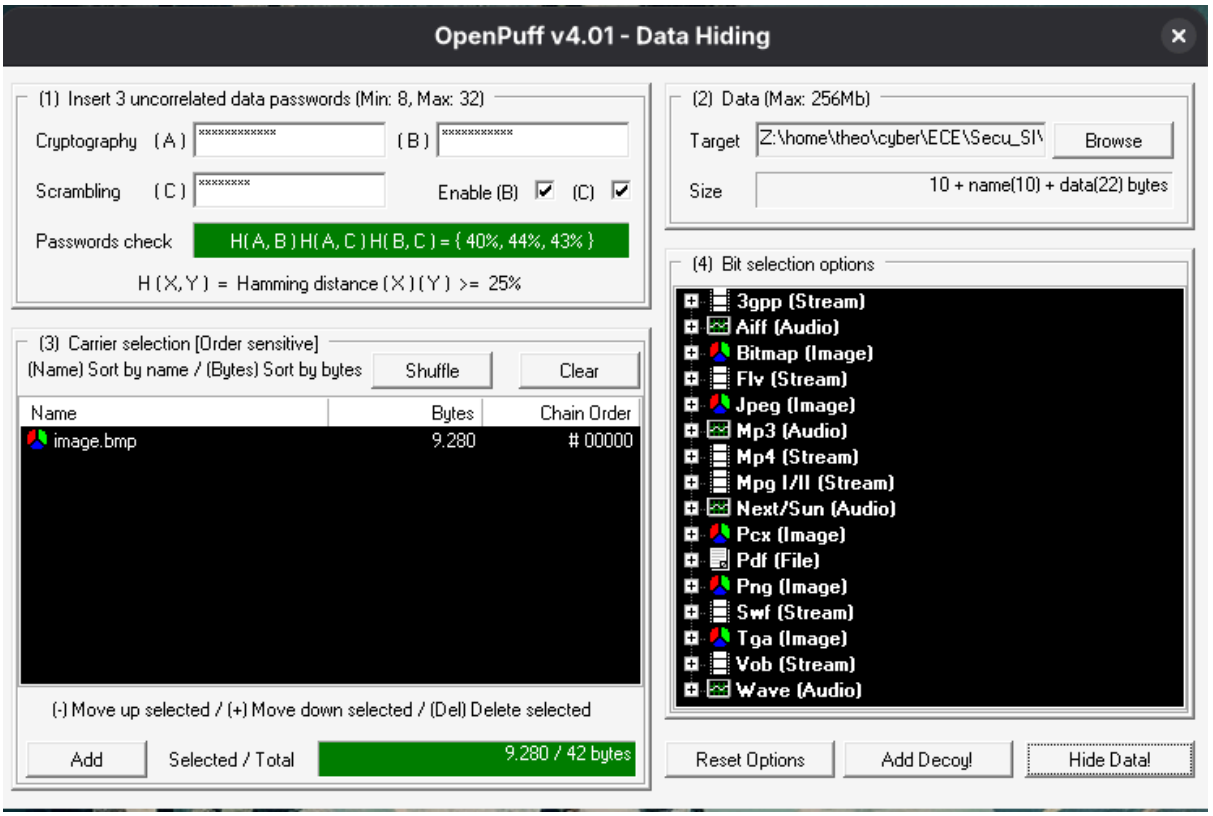
Théophile
Dutrey
Cybersécurité
Groupe 1

Lab report — OpenPuff

1) Questions / réponses sur OpenPuff

Image sans le message caché

Image avec message caché.



Config OpenPuff

Q1 — Comparer les fichiers porteurs avant/après insertion. Discrétion du marquage ?

- Visuel : aucune différence perceptible à l'œil (objectif de stéganographie → invisibilité).

- Taille : souvent quasi identique. Selon le format (JPEG/MP3/PDF…), la taille peut rester exactement la même ou varier légèrement.

- Bits modifiés : OpenPuff répartit et dissimule les bits du secret dans des zones "peu sensibles" du/des porteur(s) (p.ex. LSB/coefficients/espaces redondants), de façon adaptative et distribuée sur plusieurs fichiers si tu en fournis plusieurs.

- Empreintes/metadata : le hash du fichier change forcément (contenu modifié), mais les métadonnées visibles (EXIF, etc.) ne sont pas nécessairement altérées.

- Conclusion : très discret visuellement, mais détectable par des analyses statistiques spécialisées (stégano-analyse) ou en comparant des empreintes.

Q2 — À quoi sert le bouton "Add decoy" ?

- Il ajoute un leurre (un faux secret).

- Utilité : négation plausible. Si l'on te force à "donner le mot de passe", tu fournis celui du decoy : on peut extraire un contenu crédible mais sans importance, tandis que le vrai secret reste protégé derrière la combinaison correcte (mots de passe + ordre des porteurs).

Q3 — Cacher un gros secret dans plusieurs fichiers porteurs puis le récupérer

- Oui, OpenPuff fractionne le secret et le répartit sur plusieurs porteurs (images, PDF, audio, etc.).

- Ordre des fichiers : oui, l'ordre compte. Lors de la récupération (*Unhide*), tu dois fournir exactement la même liste dans le même ordre. Un seul porteur manquant, déplacé ou incorrect → récupération impossible (propriété de sécurité d'OpenPuff).

Q4 — Réaliser un watermark puis le vérifier. But du watermarking ?

- Un filigrane numérique sert à prouver la paternité, marquer un contenu (copyright/licence), et/ou détecter la falsification (intégrité).

- On distingue des filigranes robustes (résistent à des transformations) et fragiles (cassent au moindre changement → détection de retouche).

2) Steganography program

Code 1 (hide_image.py):

```
from PIL import Image

# Charger les deux images
container = Image.open("container.bmp")
secret = Image.open("secret.bmp")

# Vérifier que les tailles correspondent
if container.size != secret.size:
   raise ValueError("Les deux images doivent avoir la même taille !")

# Créer une nouvelle image
result = Image.new("RGB", container.size)

# Parcourir chaque pixel
for x in range(container.size[0]):
   for y in range(container.size[1]):
       # Pixels container et secret
       c_pixel = container.getpixel((x, y))
       s_pixel = secret.getpixel((x, y))

       new_pixel = []
       for i in range(3):  # pour R, G, B
           # Garder les 4 bits de poids fort du container
           c_bits = c_pixel[i] & 0b11110000
           # Prendre les 4 bits forts du secret et les mettre en faible
           s_bits = s_pixel[i] >> 4
           # Fusionner
           new_pixel.append(c_bits | s_bits)

       result.putpixel((x, y), tuple(new_pixel))

# Sauvegarde en BMP
result.save("stego.bmp")
print("Image cachée générée : stego.bmp")
```

container.bmp:

secret.bmp

stego.bmp (résultat)

Code 2 (extract_image.py):

```python
from PIL import Image
from pathlib import Path

STEGO_PATH = "stego.bmp"                # image contenant le secret
OUTPUT_PATH = "secret_extracted.bmp"    # image sortie reconstituée
BITS = 4                                # nb de bits LSB utilisés


def extract_secret(stego_path: str, out_path: str, bits: int = 4) ->
None:
    if bits <= 0 or bits >= 8:
        raise ValueError("BITS doit être entre 1 et 7 (recommandé: 4).")

    stego = Image.open(stego_path).convert("RGB")
    w, h = stego.size
    secret = Image.new("RGB", (w, h))

    lsb_mask = (1 << bits) - 1      # ex: 4 -> 0b1111
    shift = 8 - bits                # ex: 4 -> on décale à gauche de 4

    for y in range(h):
        for x in range(w):
            r, g, b = stego.getpixel((x, y))
            r_out = (r & lsb_mask) << shift
            g_out = (g & lsb_mask) << shift
            b_out = (b & lsb_mask) << shift
            secret.putpixel((x, y), (r_out, g_out, b_out))

    Path(out_path).parent.mkdir(parents=True, exist_ok=True)
    secret.save(out_path, format="BMP")

if __name__ == "__main__":
    extract_secret(STEGO_PATH, OUTPUT_PATH, BITS)
    print(f"[+] Secret extrait -> {OUTPUT_PATH} (BITS={BITS})")
```

stego.bmp

secret_extracted.bmp