Codage / Décodage du code à points

Présenté par : Zine-Elabidine Chouali
Saad Eddine Eljarida
Encadré par : Prof. Ounachad

Avril 2025

Résumé

Ce projet vise à concevoir un système innovant d'encodage et de décodage de messages en utilisant des motifs de points colorés. Nous combinons des techniques de traitement d'images et de cryptographie visuelle, en exploitant les fonctionnalités du langage Python et de bibliothèques spécialisées. Ce document présente en détail la méthodologie employée, les défis rencontrés et les perspectives d'évolution du projet.

Table des matières

Introduction générale

Le codage et le décodage de messages sont des enjeux cruciaux dans le domaine de la cryptographie. Ce projet propose une approche alternative basée sur des grilles de points colorés pour coder chaque caractère. Le texte est transformé en motifs visuels, et seuls ceux qui connaissent le code de correspondance entre les motifs et les lettres peuvent déchiffrer le message. L'objectif est de créer un système sécurisé, simple à utiliser et facile à implémenter avec Python.

Concept général

Le système repose sur un principe de cryptographie visuelle où chaque lettre ou caractère est représenté par une grille de points colorés. Les étapes principales incluent :

- La création d'une table de correspondance entre les lettres et les motifs de points.
- L'encodage du texte sous forme d'une image composée de grilles de points.
- Le décodage de l'image pour retrouver le texte original.

Cette méthode permet de transformer un message en une image apparemment aléatoire, mais qui peut être décodée grâce à la connaissance du code.

Chapitre 1

Principe général du système

1.1 Description du codage

Le système encode un texte sous forme d'une image en utilisant des grilles de points colorés pour représenter chaque lettre. Chaque lettre est traduite en un motif visuel spécifique, constitué d'une grille 3×3 où la présence, l'absence et la couleur des points représentent la lettre.

1.2 Étapes du processus

1.2.1 Préparation des correspondances

Avant de lire ou générer une image, le système établit une table de correspondance :
- À chaque schéma de points colorés correspond un caractère (lettre ou espace).
- Chaque schéma est une liste ordonnée de couleurs ou d'absences de points.

Voici un exemple de la table de correspondance utilisée :

1.2.2 Encodage du texte

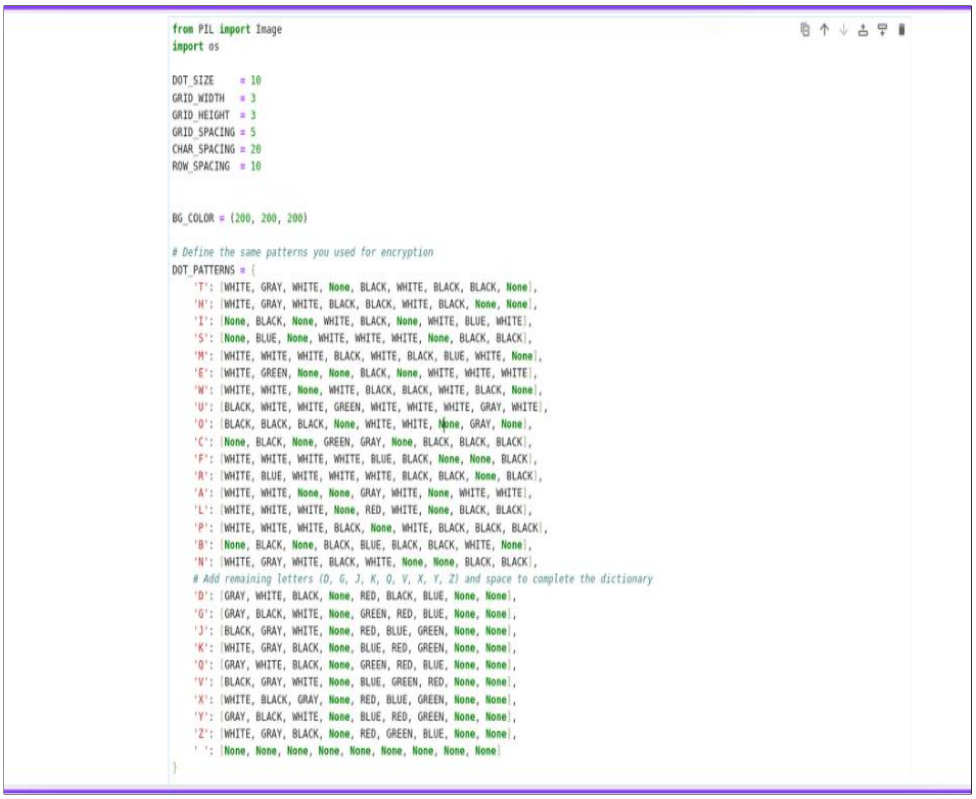
Le texte à encoder est converti en majuscules pour correspondre aux modèles existants. Une image vierge est créée, et chaque lettre est dessinée sous forme de grille 3×3 de points colorés. Les espaces entre les grilles garantissent une séparation claire.

Voici un exemple du code utilisé pour l'encodage :

1.2.3 Décodage de l'image

L'image est divisée en blocs correspondant à chaque lettre. Pour chaque bloc :
- Le programme calcule la position des points dans la grille.
- Il lit la couleur de chaque point et forme une liste correspondant au motif.
- Cette liste est comparée à la table de correspondance pour retrouver le caractère associé.

Voici un exemple du code utilisé pour le décodage :



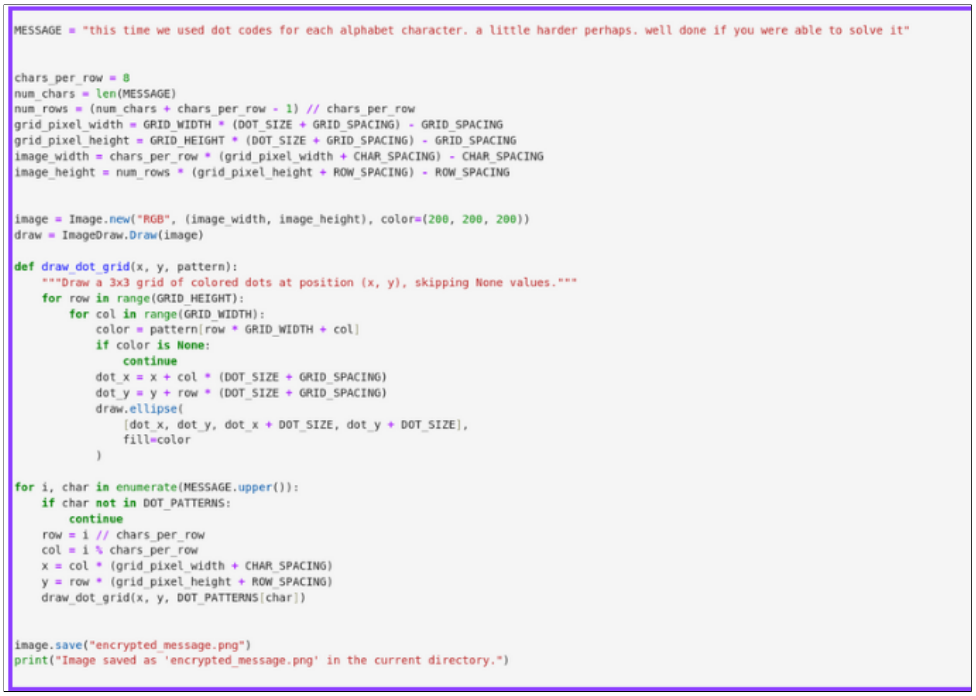
```python
from PIL import Image
import os

DOT_SIZE     = 10
GRID_WIDTH   = 3
GRID_HEIGHT  = 3
GRID_SPACING = 5
CHAR_SPACING = 20
ROW_SPACING  = 10


BG_COLOR = (200, 200, 200)

# Define the same patterns you used for encryption
DOT_PATTERNS = {
    'T': [WHITE, GRAY, WHITE, None, BLACK, WHITE, BLACK, BLACK, None],
    'H': [WHITE, GRAY, WHITE, BLACK, BLACK, WHITE, BLACK, None, None],
    'I': [None, BLACK, None, WHITE, BLACK, None, WHITE, BLUE, WHITE],
    'S': [None, BLUE, None, WHITE, WHITE, WHITE, None, BLACK, BLACK],
    'M': [WHITE, WHITE, WHITE, BLACK, WHITE, BLACK, BLUE, WHITE, None],
    'E': [WHITE, GREEN, None, None, BLACK, None, WHITE, WHITE, WHITE],
    'W': [WHITE, WHITE, None, WHITE, BLACK, BLACK, WHITE, BLACK, None],
    'U': [BLACK, WHITE, WHITE, GREEN, WHITE, WHITE, WHITE, GRAY, WHITE],
    'O': [BLACK, BLACK, BLACK, None, WHITE, WHITE, None, GRAY, None],
    'C': [None, BLACK, None, GREEN, GRAY, None, BLACK, BLACK, BLACK],
    'F': [WHITE, WHITE, WHITE, WHITE, BLUE, BLACK, None, None, BLACK],
    'R': [WHITE, BLUE, WHITE, WHITE, WHITE, BLACK, BLACK, None, BLACK],
    'A': [WHITE, WHITE, None, None, GRAY, WHITE, None, WHITE, WHITE],
    'L': [WHITE, WHITE, WHITE, None, RED, WHITE, None, BLACK, BLACK],
    'P': [WHITE, WHITE, WHITE, BLACK, None, WHITE, BLACK, BLACK, BLACK],
    'B': [None, BLACK, None, BLACK, BLUE, BLACK, BLACK, WHITE, None],
    'N': [WHITE, GRAY, WHITE, BLACK, WHITE, None, None, BLACK, BLACK],
    # Add remaining letters (D, G, J, K, Q, V, X, Y, Z) and space to complete the dictionary
    'D': [GRAY, WHITE, BLACK, None, RED, BLACK, BLUE, None, None],
    'G': [GRAY, BLACK, WHITE, None, GREEN, RED, BLUE, None, None],
    'J': [BLACK, GRAY, WHITE, None, RED, BLUE, GREEN, None, None],
    'K': [WHITE, GRAY, BLACK, None, BLUE, RED, GREEN, None, None],
    'Q': [GRAY, WHITE, BLACK, None, GREEN, RED, BLUE, None, None],
    'V': [BLACK, GRAY, WHITE, None, BLUE, GREEN, RED, None, None],
    'X': [WHITE, BLACK, GRAY, None, RED, BLUE, GREEN, None, None],
    'Y': [GRAY, BLACK, WHITE, None, BLUE, RED, GREEN, None, None],
    'Z': [WHITE, GRAY, BLACK, None, RED, GREEN, BLUE, None, None],
    ' ': [None, None, None, None, None, None, None, None, None]
}
```

FIGURE 1.1 – Table de correspondance entre les lettres et les motifs de points colorés.



```
MESSAGE = "this time we used dot codes for each alphabet character. a little harder perhaps. well done if you were able to solve it"


chars_per_row = 8
num_chars = len(MESSAGE)
num_rows = (num_chars + chars_per_row - 1) // chars_per_row
grid_pixel_width = GRID_WIDTH * (DOT_SIZE + GRID_SPACING) - GRID_SPACING
grid_pixel_height = GRID_HEIGHT * (DOT_SIZE + GRID_SPACING) - GRID_SPACING
image_width = chars_per_row * (grid_pixel_width + CHAR_SPACING) - CHAR_SPACING
image_height = num_rows * (grid_pixel_height + ROW_SPACING) - ROW_SPACING


image = Image.new("RGB", (image_width, image_height), color=(200, 200, 200))
draw = ImageDraw.Draw(image)

def draw_dot_grid(x, y, pattern):
    """Draw a 3x3 grid of colored dots at position (x, y), skipping None values."""
    for row in range(GRID_HEIGHT):
        for col in range(GRID_WIDTH):
            color = pattern[row * GRID_WIDTH + col]
            if color is None:
                continue
            dot_x = x + col * (DOT_SIZE + GRID_SPACING)
            dot_y = y + row * (DOT_SIZE + GRID_SPACING)
            draw.ellipse(
                [dot_x, dot_y, dot_x + DOT_SIZE, dot_y + DOT_SIZE],
                fill=color
            )

for i, char in enumerate(MESSAGE.upper()):
    if char not in DOT_PATTERNS:
        continue
    row = i // chars_per_row
    col = i % chars_per_row
    x = col * (grid_pixel_width + CHAR_SPACING)
    y = row * (grid_pixel_height + ROW_SPACING)
    draw_dot_grid(x, y, DOT_PATTERNS[char])


image.save("encrypted_message.png")
print("Image saved as 'encrypted_message.png' in the current directory.")
```

FIGURE 1.2 – Capture d'écran du code source pour l'encodage du texte.

```
PATTERN_TO_CHAR = { tuple(v): k for k, v in DOT_PATTERNS.items() }


def extract_dot_grid(img, x, y):

    pat = []
    for ry in range(GRID_HEIGHT):
        for rx in range(GRID_WIDTH):
            dot_x = x + rx * (DOT_SIZE + GRID_SPACING)
            dot_y = y + ry * (DOT_SIZE + GRID_SPACING)
            cx = dot_x + DOT_SIZE // 2
            cy = dot_y + DOT_SIZE // 2
            pixel = img.getpixel((cx, cy))
            if pixel == BG_COLOR:
                pat.append(None)
            else:
                pat.append(pixel)
    return tuple(pat)


def decrypt_image(path="encrypted_message.png"):
    img = Image.open(path).convert("RGB")
    W, H = img.size

    grid_w = GRID_WIDTH  * (DOT_SIZE + GRID_SPACING) - GRID_SPACING
    grid_h = GRID_HEIGHT * (DOT_SIZE + GRID_SPACING) - GRID_SPACING

    cols = (W + CHAR_SPACING) // (grid_w + CHAR_SPACING)
    rows = (H + ROW_SPACING) // (grid_h + ROW_SPACING)

    result = []
    for ry in range(rows):
        for cx in range(cols):
            x = cx * (grid_w + CHAR_SPACING)
            y = ry * (grid_h + ROW_SPACING)
            if x + grid_w > W or y + grid_h > H:
                continue
            pat = extract_dot_grid(img, x, y)
            result.append(PATTERN_TO_CHAR.get(pat, '?'))

    # join and lowercase
    return "".join(result).lower()
```

FIGURE 1.3 – Capture d’écran du code source pour le décodage de l’image.

Chapitre 2

Réalisation pratique

2.1 Outils utilisés

Le projet repose sur l'utilisation du langage de programmation Python, qui est particulièrement adapté pour ce type de tâche grâce à sa simplicité et à la richesse de ses bibliothèques. Pour manipuler les images et les couleurs, nous utilisons la bibliothèque Pillow , une bibliothèque populaire en Python dédiée au traitement d'images. Pillow permet de créer, modifier et analyser des images pixel par pixel, ce qui est essentiel pour encoder et décoder les grilles de points colorés.

Un dictionnaire nommé DOT PATTERNS a été créé pour stocker les correspondances entre les lettres de l'alphabet et leurs motifs de points respectifs. Chaque clé du dictionnaire représente une lettre ou un caractère (par exemple, 'A', 'B', 'C', etc.), et chaque valeur associée est une liste ordonnée représentant les couleurs des points dans une grille 3×3.

Chapitre 3

Résultats obtenus

3.1 Performances du système

Les résultats montrent que l'encodage et le décodage fonctionnent correctement pour les lettres et les espaces. Cependant, des erreurs surviennent lorsque l'image subit une dégradation (compression, bruit). De plus, les symboles non définis dans la table de correspondance sont ignorés.

Voici un exemple de résultat obtenu après l'encodage :

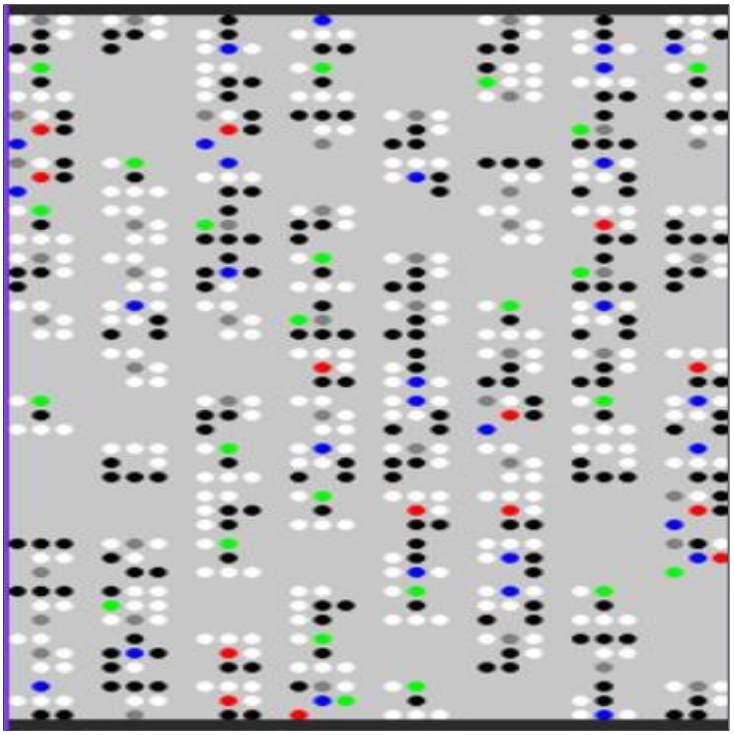


FIGURE 3.1 – Image obtenue après encodage d'un texte.

Voici un exemple de résultat obtenu après le décodage :

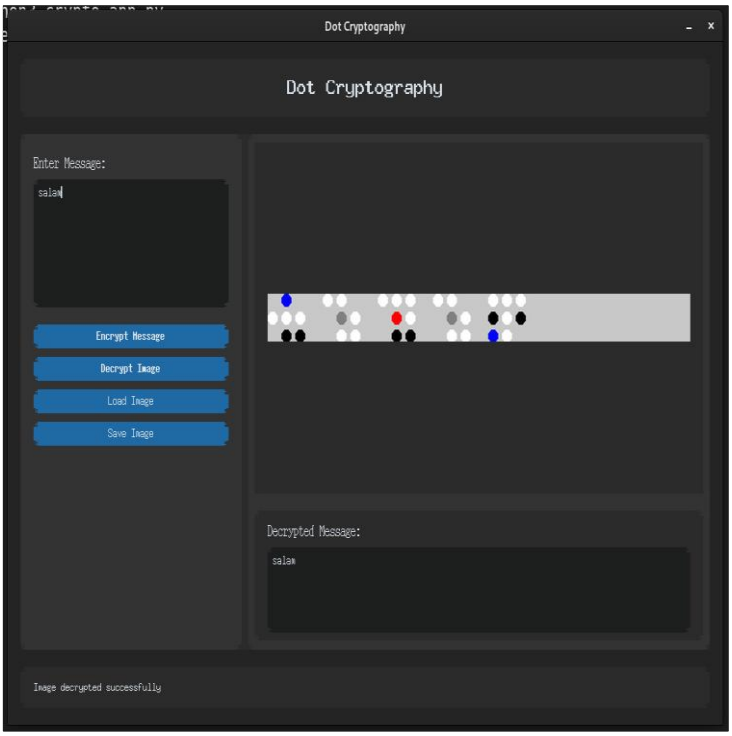


FIGURE 3.2 – Texte reconstruit après décodage de l'image.

3.2 Limitations

Le système présente plusieurs limitations qui doivent être prises en compte pour garantir son bon fonctionnement :

- **Précision** : Nous devons échantillonner le centre exact de chaque point pour obtenir la bonne couleur.
- **Correspondance de Motifs** : Le motif doit correspondre exactement pour qu'un caractère soit reconnu.
- **Qualité d'Image** : Toute distorsion dans l'image pourrait briser le décryptage.
- **Gestion des espaces et caractères non supportés** :
- Si une grille ne correspond à aucun modèle connu, elle est traitée comme un espace ou ignorée.
- Les caractères non pris en charge sont également gérés pour éviter des erreurs de décodage.

Chapitre 4

Conclusion et perspectives

Ce projet a démontré la faisabilité d'un système simple d'encodage et de décodage basé sur des grilles de points colorés. Des améliorations sont nécessaires pour inclure des symboles supplémentaires et optimiser la qualité de l'image pour un décodage plus robuste. Les prochaines étapes incluent l'intégration d'un algorithme de correction d'erreurs et l'exploration de méthodes de compression adaptées.

Références

- https://www.fbi.gov/news/stories/can-you-crack-a-code
- https://www.geeksforgeeks.org/python-pillow-tutorial/
- https://realpython.com/image-processing-with-the-python-pillow-library/

Annexes

Annexe A : Code source

Code source du programme de génération d'image et de décodeur. Le code complet est disponible sur GitHub : https://github.com/votre-repo/nom-du-projet.