

TP3 cryptographie

Arnaud Champierre de Villeneuve

M1 Cybersécurité & Management

04/06/2025

Github : Warwick001 / [Warwick001/cryptographie](https://github.com/Warwick001/cryptographie)

1 - Objectifs pédagogiques

1) L'entropie, en cryptographie, correspond au degré d'imprévisibilité d'une donnée ou d'un message. Plus elle est élevée, plus il est complexe de deviner ou de reconstituer l'information. Un mot de passe basique comme présente une faible entropie, car il suit des schémas courants, alors qu'un mot de passe généré aléatoirement, tel que "h\$D2!aK#", dispose d'une entropie supérieure. En augmentant l'entropie, on renforce efficacement la sécurité d'un système cryptographique.

2) La redondance désigne la présence d'éléments répétitifs ou prévisibles dans un message, tels que des lettres fréquemment utilisées ou des structures grammaticales classiques. Dans la cryptographie cette caractéristique peut être exploitée par les attaquants, notamment à travers l'analyse fréquentielle appliquée aux chiffrements simples comme ceux de César ou Vigenère (vu dans le TP2). Pour contrer cette faiblesse, il est essentiel d'employer des techniques de chiffrement avancées qui brouillent les schémas perceptibles, comme par exemple le principe de Claude Shannon qualifie de "confusion". Donc Réduire la redondance est ainsi un facteur clé pour améliorer la confidentialité des données.

3) L'indice de coïncidence est un outil statistique qui mesure la fréquence des lettres identiques au sein d'un texte. Il sert notamment à différencier un texte en clair d'un texte chiffré. Cet indicateur est particulièrement utile pour déterminer le type de chiffrement employé et peut également aider à retrouver la longueur de la clé dans les chiffrements polyalphabétiques comme celui de Vigenère.

4) Le chiffrement RSA fonctionne avec des blocs de données de taille fixe. Le padding, ou bourrage, est une méthode consistant à ajouter des données supplémentaires, souvent aléatoires ou formatées selon un standard, avant le chiffrement afin d'améliorer la sécurité. Sans cette protection le RSA peut être exposé à des attaques exploitant les structures répétitives ou les oracles. En intégrant du padding, on limite ces vulnérabilités et rend le message chiffré moins prévisible.

2- Entropie , Redondance et Incide de coïncidence

- 1) **L'entropie**, c'est mesurer l'incertitude ou le désordre de l'information contenue dans un texte. En python :

```
import math
from collections import Counter

def calculer_entropie(texte):
    n = len(texte)
    freqs = Counter(texte)
    return -sum((f/n) * math.log2(f/n) for f in freqs.values())
```

- 2) La **redondance** va indiquer combien d'information est prévisible. Une formule python :

```
def redondance(text, N=26):
    entropie = calculer_entropie(text)
    return 1 - (entropie / math.log2(N))
```

Sachant que H de la formule mathématique est l'entropie du **text** et **N** est le nombre de symboles possibles (26 pour l'alphabet sans accents et sans différencier les majuscules & minuscules).

- 3) Et pour finir la **coïncidence**, elle va mesurer la probabilité que deux lettres prises au hasard soient identiques.

```
def indicecoincidence(texte):
    n = len(texte)
    freqs = Counter(texte)
    numerateur = sum(f * (f - 1) for f in freqs.values())
    denominateur = n * (n - 1)
    return numerateur / denominateur if denominateur > 0 else 0
```

2 - 1 Le chiffrement de Lester Hill

Pour ce code en python, rappelons que le chiffrement de Lester Hill est un système de cryptographie symétrique basé sur l'algèbre linéaire. Il transforme des blocs de lettres en vecteurs numériques, puis applique une multiplication matricielle avec une matrice clé, le tout en arithmétique modulaire. Ce procédé permet un chiffrement par blocs plus résistant à l'analyse fréquentielle qu'un simple chiffrement monoalphabétique. Pour garantir le déchiffrement, la matrice utilisée doit être inversible modulo 26.

```
import numpy as np

def hill_encrypt(plaintext, key_matrix):
```

```

# converti le text en valeur numérique
plaintext = plaintext.upper().replace(" ", "")
plaintext_vector = [ord(char) - ord('A') for char in plaintext]
n = len(key_matrix)

while len(plaintext_vector) % n != 0:
    plaintext_vector.append(25) # Remplissage avec 'Z' (valeur 25)

# Reshape le texte brut en forme de matrice
plaintext_matrix = np.array(plaintext_vector).reshape(-1, n)

# Chiffrer en utilisant la multiplication de matrices (mod 26)
encrypted_matrix = np.dot(plaintext_matrix, key_matrix) % 26
encrypted_text = "".join(chr(int(num) + ord('A')) for num in
encrypted_matrix.flatten())

return encrypted_text

# Matrice 3x3
key_matrix = np.array([[6, 24, 1], [13, 16, 10], [20, 17, 15]])
plaintext = "Salutttt"
ciphertext = hill_encrypt(plaintext, key_matrix)
print("Encrypted text:", ciphertext)
print("Texte déchiffrer :", plaintext)

```

Résultat :

```

PS C:\Users\arnau\OneDrive\Bureau\EFREI\M1\Cryptographie> & C:/Users/arnau/AppData/
Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/arnau/OneDrive/Bureau/EFREI/M1
/Cryptographie/code/lester(TP3).py"
Encrypted text: QVBTPBDPM
Texte déchiffrer : Salutttt

```

2 – 2 Le chiffrement affine

```

def chiffrement_affine(text, a, b):
    text = text.upper().replace(" ", "")
    chiffre = ""
    for c in text:
        if c.isalpha():
            x = ord(c) - ord('A')
            y = (a * x + b) % 26
            chiffre += chr(y + ord('A'))

    return chiffre

```

2- 3 AES, RSA

- 1) AES, algorithme de chiffrement symétrique :

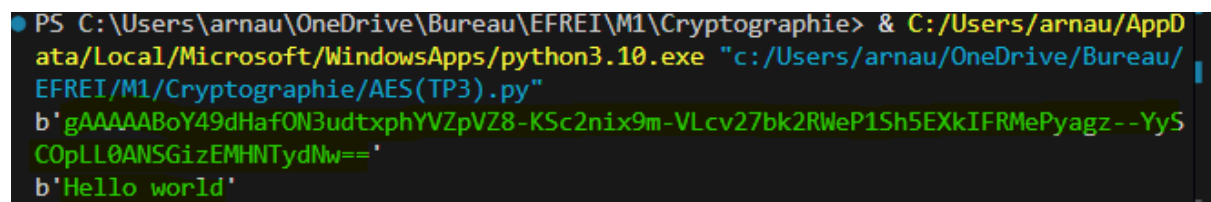
```
from cryptography.fernet import Fernet

cle = Fernet.generate_key()
f = Fernet(cle)

message = b"Hello world"
chiffre = f.encrypt(message)
dechiffre = f.decrypt(chiffre)

print (chiffre)
print (dechiffre)
```

Résultat :



```
PS C:\Users\arnau\OneDrive\Bureau\EFREI\M1\Cryptographie> & C:/Users/arnau/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/arnau/OneDrive/Bureau/EFREI/M1/Cryptographie/AES(TP3).py"
b'gAAAABoY49dHafON3udtxphYVZpVZ8-KSc2nix9m-VLcv27bk2RWeP1Sh5EXkIFRMePyagz--YySCOpLL0ANSGizEMHNTydNw=='
b'Hello world'
```

- 2) RSA, est un algorithme de chiffrement qui utilise une clé publique et une clé privée. Le nombre de bits pour la génération de clé RSA va de 128 à 4096 et etc. Pour cet exercice j'ai décidé de prendre 2048 bits, c'est une taille de génération de clé qui est sécurisée et ni trop longue à chiffrer / déchiffrer.

```
import rsa

# Génération de clés publique/privée
(public_key, private_key) = rsa.newkeys(2048)

# Message chiffrer
message = "Hello la cryptoo".encode()

# chiffrement avec la clé publique
chiffre = rsa.encrypt(message, public_key)

# Déchiffrement avec la clé privée
dechiffre = rsa.decrypt(chiffre, private_key)

print("Chiffré :", chiffre)
print("Déchiffré :", dechiffre.decode())
```

Résultat :

```
PS C:\Users\arnau\OneDrive\Bureau\EFREI\M1\Cryptographie> & C:/Users/arnau/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/arnau/OneDrive/Bureau/EFREI/M1/Cryptographie/RSA(TP3).py"
Chiffré : b'\x07\xab\xcf1\x1a\x9d\xdaJDE\\\x05\xdb2\x9e\x08\x91ZMA3\x87=T\x840R\xb9K\x90a\x8e\xc7^\x13\xfcM\xb5\x066\x07\x99\x9f\x8bzb\x09\xfb\x00\xbc\xca \xa8Z\xbd\x05\x1cojA1\x95\x837\xb3)\x99\x01\xec=\xd2\x87\xd7+I1\xd04\xae\xa1a\xf3\xd5\x98{\x85P\t\xcc@!\xb6\x97/C&\xb1\xb9. \xbc\x1f\xe4:\xf0ng\xf2\xce)\x1d\xb4\x08h\x19\xe7\x04\xf5r\x91D\x8cQ$\x9b\xb4\xea\x7f\x91\xa04\xf65\xbb4pHr\x17`bEp\rE>gA\xfb\x81\rR\x7Y\xdbC\x94\xd50G\x93\x8c\x89\xa7\xc5q\x1c\xc4@s\xc4\x98d\xcf\x0cb\x7f\x97\xcd\x8dw\xab\x1c\x91\x06\xcf\xa3\x0e, \xc5\x16\xa8\xbe\xfb9\xa2; \xcd\xeb\x90"d; \x00\xa4\xf3\x19K\xcb\xe3; \xf8G\xa4\xc4w\x82\xaaI\x8d\xc2\xaf\x18J\x82, \x90\x92\x01. \xfab\xdd\xf2\x1e\x14e\x80<FM\x065\x1f\x8d\xdd?T\x10+0\xee\x0f\x82'
Déchiffré : Hello la cryptoo
```