TP2 de Cryptographie

Chiffrements de César, Vigenère et Confusion (Shannon)

Yassine Gouja
Msc Cybersécurité & Management – EFREI

https://github.com/YassineTns/TP2-Cryptographie-EFREI

Année Universitaire 2024–2025

Ce projet a été réalisé dans le cadre du TP2 de Cryptographie.
Tous les codes sources sont disponibles sur le GitHub :
https://github.com/YassineTns/TP2-Cryptographie-EFREI

Table des matières

1 Introduction **3**

2 Chiffrement de César avec encodage Base64 **3**
2.1 Principe du chiffrement de César . . . 3
2.2 Fonctionnement du programme . . . 3
2.3 Extrait de code . . . 3
2.4 Exemple d'utilisation . . . 3
2.5 Conclusion César . . . 4

3 Chiffrement de Vigenère **4**
3.1 Principe du chiffrement de Vigenère . . . 4
3.2 Fonctionnement du programme . . . 4
3.3 Extrait de code . . . 4
3.4 Exemple d'utilisation . . . 4
3.5 Conclusion Vigenère . . . 4

4 Confusion de Shannon avec XOR et SHA-256 **5**
4.1 Principe de la confusion selon Claude Shannon . . . 5
4.2 Fonctionnement du programme . . . 5
4.3 Extrait de code . . . 5
4.4 Exemple d'utilisation . . . 5
4.5 Analyse de sécurité . . . 5
4.6 Limites . . . 5
4.7 Conclusion Shannon . . . 5

5 Conclusion Génerale **6**

1 Introduction

Ce rapport présente les résultats du TP2 de cryptographie réalisé dans le cadre du Master Cybersécurité & Management à l'EFREI. L'objectif de ce TP est de mettre en œuvre trois techniques fondamentales de cryptographie :

— le chiffrement de César avec encodage Base64,
— le chiffrement de Vigenère,
— la confusion selon la théorie de Shannon à l'aide de l'opération XOR et du hachage SHA-256.

Le rapport détaillera le fonctionnement de chaque méthode, la logique implémentée, des extraits de code, des résultats de tests ainsi que les conclusions associées.

2 Chiffrement de César avec encodage Base64

2.1 Principe du chiffrement de César

Le chiffrement de César repose sur un décalage fixe de chaque lettre du message clair dans l'alphabet. Dans notre implémentation, ce décalage est déterminé dynamiquement à partir de la longueur d'une clé saisie par l'utilisateur.

Ce chiffrement est ensuite renforcé par un encodage Base64, utilisé pour rendre le résultat moins lisible et masquer la structure du texte original.

2.2 Fonctionnement du programme

Le script `cesar.py` permet :
— de chiffrer un message avec la méthode de César (décalage = longueur de la clé),
— d'encoder le résultat en Base64,
— de faire l'opération inverse : décoder puis déchiffrer.

2.3 Extrait de code

```
def cesar(message, cle, mode='chiffrement'):
    decalage = len(cle)
    resultat = ""
    for char in message:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            code = (ord(char) - base + decalage) % 26 if mode == 'chiffrement' \
                else (ord(char) - base - decalage) % 26
            resultat += chr(base + code)
        else:
            resultat += char
    return resultat
```

2.4 Exemple d'utilisation

Chiffrement :
— Message : `HELLO`
— Clé : `SECURE` (longueur = 6)
— Résultat chiffré : `NKRRU`
— Résultat encodé Base64 : `TktSUlU=`

Déchiffrement :
— Message encodé : `TktSUlU=`
— Décodage Base64 : `NKRRU`
— Résultat déchiffré : `HELLO`

2.5 Conclusion César

Cette première étape du TP permet de revoir les bases de la substitution monoalphabétique. Le chiffrement de César est simple à implémenter mais insuffisant pour des cas réels. L'ajout de l'encodage Base64 permet d'obscurcir davantage les messages, bien que cela ne soit pas une méthode de chiffrement à proprement parler.

3 Chiffrement de Vigenère

3.1 Principe du chiffrement de Vigenère

Le chiffrement de Vigenère est une méthode de substitution polyalphabétique. Contrairement au chiffrement de César, qui utilise un décalage constant, Vigenère applique une série de décalages basés sur les lettres d'une clé répétée.

Chaque lettre du message est décalée en fonction de la lettre correspondante dans la clé :

$$\text{Lettre chiffrée} = (\text{Lettre claire} + \text{Décalage (clé)}) \mod 26$$

3.2 Fonctionnement du programme

Le script `vigenere.py` contient deux fonctions :

— `vigenere_chiffrement(message, cle)` : applique la méthode de chiffrement de Vigenère.
— `vigenere_dechiffrement(message, cle)` : fait l'opération inverse pour retrouver le texte clair.

3.3 Extrait de code

```
def vigenere_chiffrement(message, cle):
    resultat = ""
    cle = cle.upper()
    message = message.upper()
    i = 0
    for char in message:
        if char.isalpha():
            decalage = ord(cle[i % len(cle)]) - ord('A')
            nouveau = chr((ord(char) - ord('A') + decalage) % 26 + ord('A'))
            resultat += nouveau
            i += 1
        else:
            resultat += char
    return resultat
```

3.4 Exemple d'utilisation

Chiffrement :
— Message : `BONJOUR`
— Clé : `CLE`
— Résultat chiffré : `DSPMWYV`
Déchiffrement :
— Message chiffré : `DSPMWYV`
— Clé : `CLE`
— Résultat déchiffré : `BONJOUR`

3.5 Conclusion Vigenère

Le chiffrement de Vigenère renforce considérablement la sécurité par rapport à César. Grâce à l'usage d'une clé variable, il rend l'analyse fréquentielle beaucoup plus difficile. Il s'agit d'un excellent exemple de chiffrement polyalphabétique utilisé jusqu'au 19e siècle avant d'être cassé.

4 Confusion de Shannon avec XOR et SHA-256

4.1 Principe de la confusion selon Claude Shannon

Claude Shannon, le père fondateur de la théorie de l'information, a introduit les notions de **confusion** et **diffusion** comme propriétés fondamentales d'un bon système cryptographique :

— La **confusion** consiste à rendre la relation entre la clé et le texte chiffré aussi complexe que possible.

— La **diffusion** a pour but de répartir l'information du message clair sur l'ensemble du message chiffré.

Dans ce TP, nous avons implémenté la confusion à l'aide de deux outils :

— l'opérateur logique `XOR`, appliqué entre le message et la clé,

— la fonction de hachage `SHA-256`, pour produire un condensé du résultat.

4.2 Fonctionnement du programme

Le programme `shannon.py` applique un XOR caractère par caractère entre le message et la clé, puis calcule le condensé avec SHA-256.

4.3 Extrait de code

```
def xor_bytes(message: str, key: str) -> bytes:
    message_bytes = message.encode()
    key_bytes = key.encode()
    key_repeated = (key_bytes * (len(message_bytes) // len(key_bytes) + 1))[:len(message_bytes)]
    return bytes([mb ^ kb for mb, kb in zip(message_bytes, key_repeated)])

def main():
    message = input("Entrez un message : ")
    key = input("Entrez une clé : ")
    xor_result = xor_bytes(message, key)
    print("Résultat du XOR (hexadécimal) :", xor_result.hex())
    print("SHA-256 du résultat XOR :", hashlib.sha256(xor_result).hexdigest())
```

4.4 Exemple d'utilisation

— **Message :** `HELLO`

— **Clé :** `EFREI`

— **XOR (hex) :** `0d031e0906`

— **SHA-256 :** `b9ba7a93cb2b4249a9ea9537a9d3f737...`

4.5 Analyse de sécurité

Le XOR permet d'introduire de la confusion, mais il est symétrique : la même opération permet de retrouver le message d'origine. Ce n'est donc pas suffisant seul. L'ajout de SHA-256 rend le résultat non réversible, renforçant la sécurité en cas de stockage du haché (mot de passe, identifiant unique...).

4.6 Limites

— Le XOR n'est sécurisé que si la clé est aléatoire, de même longueur que le message et utilisée une seule fois (One-Time Pad).

— SHA-256 est irréversible, mais ne chiffre pas : il hache, donc il n'est pas possible de retrouver le message à partir du condensé.

4.7 Conclusion Shannon

Cette partie nous a permis d'expérimenter le concept de confusion, clé de la sécurité selon Claude Shannon. En combinant une opération XOR avec un hachage SHA-256, nous avons pu observer comment un simple message peut être transformé en une empreinte unique et non réversible. Cette technique est fondamentale dans les systèmes modernes de chiffrement et d'intégrité des données.

5 Conclusion Génerale

Ce TP nous a permis d'approfondir notre compréhension des principes fondamentaux de la cryptographie, à travers trois méthodes distinctes :

- le chiffrement de César, un algorithme classique basé sur un simple décalage dans l'alphabet,
- le chiffrement de Vigenère, qui introduit une clé répétée pour rendre le chiffrement plus complexe,
- et enfin la confusion selon Shannon, utilisant des opérateurs logiques (`XOR`) combinés à une fonction de hachage (`SHA-256`).

Chacune de ces techniques met en avant des concepts cryptographiques essentiels tels que la substitution, la confusion, la diffusion et la sécurité des mots de passe.

La mise en œuvre de ces outils nous a permis de renforcer notre compréhension :

- de la logique des algorithmes de chiffrement,
- de l'importance des clés et de leur gestion,
- des limites des algorithmes historiques face aux exigences de sécurité modernes.

Références

[1] Claude E. Shannon, *Communication Theory of Secrecy Systems*, Bell System Technical Journal, 1949.

[2] Bruce Schneier, *Applied Cryptography*, John Wiley & Sons, 1996.

[3] Python Software Foundation, *hashlib — Secure hashes and message digests*, `https://docs.python.org/3/library/hashlib.html`

[4] Python Software Foundation, *base64 — Base16, Base32, Base64, Base85 Data Encodings*, `https://docs.python.org/3/library/base64.html`