Université de Sherbrooke
CeFTI

DEVOIR 1

IFT814 - Cryptographie

Professeur :
Dave **TOUCHETTE**

Travaux présentés par :
Annick **KOUASSI**
Joël **BATCHO**
Buzukira **CHABIKULI**
Fridrich Destin **MORTINIERA**

Automne 2023

Table des matières

1 CHIFFRE DE CÉSAR

Ce devoir implique trois schémas de chiffrement (à longueur fixe) et des interactions entre Alice, Bob et Eve. Voici comment nous implémentons et étudions le chiffrement de César, puis analysons les interactions entre les personnages :

1.1 Chiffrement de César (César Cipher) en Python :

Python

```python
# Fonction pour générer une clé aléatoire (décalage)
import random

def Gen1():
    return random.randint(0, 25)  # Le décalage est un nombre entre 0 et 25

# Fonction de chiffrement
def E1(k, m):
    ciphertext = ""
    for char in m:
        if char.isalpha():  # Vérifie si le caractère est une lettre
            shift = k % 26  # Assurez-vous que le décalage est dans la plage de l'alphabet
            if char.islower():
                encrypted_char = chr(((ord(char) - ord('a') + shift) % 26) + ord('a'))
            else:
                encrypted_char = chr(((ord(char) - ord('A') + shift) % 26) + ord('A'))
            ciphertext += encrypted_char
        else:
            ciphertext += char  # Gardez les caractères non alphabétiques inchangés
    return ciphertext

# Fonction de déchiffrement
def D1(k, c):
    return E1(-k, c)  # Déchiffrement en utilisant un décalage inverse

# Message initial
m = 'ceciestlemessageclairadechiffrer'

# Exécution des 3 Scénarios avec génération de clé aléatoire
for i in range(3):
```

```
    k = Gen1()  # Alice génère secrètement une clé aléatoire
    c = E1(k, m)  # Alice chiffre le message
    # Eve ne modifie pas le cryptogramme, donc c reste le même
    m_bob = D1(k, c)  # Bob déchiffre le message

# Affichage des résultats
    print("\n----------------TRACE SCENARIO",i+1,"----------------")
    print(f"Alice: k={k}, c={c}")
    print(f"Eve: c={c}")
    print(f"Bob: k={k}, m={m_bob}")
```

1.2 Analyse des intéractions :

(a) Répétons le scénario suivant trois fois de manière indépendante :

Scénario 1 :
- Alice génère secrètement k1.
- Alice chiffre m avec k1 pour obtenir c1.
- Alice envoie c1 à Bob via Eve.
- Bob déchiffre c1 avec k1 pour obtenir m.

Python

```
----------------TRACE SCENARIO 1 ----------------
Alice: k=10, c=momsocdvowocckqomvksbknomrsppbob
Eve: c=momsocdvowocckqomvksbknomrsppbob
Bob: k=10, m=ceciestlemessageclairadechiffrer
```

Scénario 2 :
- Alice génère secrètement k2.
- Alice chiffre m avec k2 pour obtenir c2.
- Alice envoie c2 à Bob via Eve.
- Bob déchiffre c2 avec k2 pour obtenir m.

Python
```
---------------TRACE SCENARIO 2 ---------------
Alice: k=0, c=ceciestlemessageclairadechiffrer
Eve: c=ceciestlemessageclairadechiffrer
Bob: k=0, m=ceciestlemessageclairadechiffrer
```

Scénario 3 :
- Alice génère secrètement k3.
- Alice chiffre m avec k3 pour obtenir c3.
- Alice envoie c3 à Bob via Eve.
- Bob déchiffre c3 avec k3 pour obtenir m.

Python
```
---------------TRACE SCENARIO 3 ---------------
Alice: k=19, c=vxvbxlmexfxlltzxvetbktwxvabyykxk
Eve: c=vxvbxlmexfxlltzxvetbktwxvabyykxk
Bob: k=19, m=ceciestlemessageclairadechiffrer
```

(b) **Eve ne peut pas déchiffrer le message m** dans chaque scénario **car elle ne connaît pas la clé de chiffrement k utilisée** par Alice pour chaque scénario.

Remarque : La sécurité du chiffrement de César repose sur le fait que si Eve ne connaît pas la clé de décalage, elle ne peut pas récupérer le message en regardant le cryptogramme. Cependant, si l'on considère qu'on est dans la langue française, Eve pourrait, par attaque en force brute, trouver la clef de déchiffrement car le nombre de possibilités est limité. Avec les résultats, Eve pourrait se rendre compte que le décalage est constant.

2 MASQUE JETABLE

Voici comment nous pouvons implémenter et utiliser le schéma de chiffrement à masque jetable (One-Time Pad, OTP) en Python, puis analyser les interactions entre Alice, Bob et Eve :

2.1 Schéma de chiffrement masque jetable (OTP) en Python :

Python

```python
import random

# Fonction pour générer une clé secrète k (masque jetable)
def Gen2(message_length):
    return ''.join(random.choice('01') for _ in range(message_length))

# Fonction de chiffrement
def E2(k, m):
    if len(k) != len(m):
        raise ValueError("La clé et le message doivent avoir la même
longueur")

    ciphertext = ""
    for i in range(len(m)):
        encrypted_bit = str(int(k[i]) ^ int(m[i]))  # XOR entre la clé et
le message
        ciphertext += encrypted_bit
    return ciphertext

# Fonction de déchiffrement
def D2(k, c):
    if len(k) != len(c):
        raise ValueError("La clé et le cryptogramme doivent avoir la même
longueur")

    plaintext = ""
    for i in range(len(c)):
        decrypted_bit = str(int(k[i]) ^ int(c[i]))  # XOR entre la clé et
le cryptogramme
        plaintext += decrypted_bit
    return plaintext

# Message initial
message = 'ceciestlemessageclairadechiffrer'

# Convertir le message en bits
message_bits = ''.join(format(ord(char), '08b') for char in message)

##########Exécution des 3 Scenarios d'exécution###############
for i in range(3):

    key = Gen2(len(message_bits))  # Alice génère une clé secrète
    ciphertext = E2(key, message_bits)  # Alice chiffre le message
    # Eve ne modifie pas le cryptogramme, donc ciphertext reste le même
```

```
    plaintext_bits = D2(key, ciphertext)  # Bob déchiffre le message en
bits
    plaintext = ''.join(chr(int(plaintext_bits[i:i+8], 2)) for i in
range(0, len(plaintext_bits), 8))  # Conversion en texte

    # Affichage des résultats
    print("\n---------------TRACE SCENARIO",i+1,"---------------")
    print(f"Alice: k={key}, m={message_bits}")
    print(f"Eve: c={ciphertext}")
    print(f"Bob: k={key}, m={plaintext_bits}, \n plaintext={plaintext}")
```

2.2 Analyse des interactions :

(a) Répétons le scénario suivant trois fois de manière indépendante :

Scénario 1 :
- Alice génère secrètement k1.
- Alice chiffre le message m en bits avec k1 pour obtenir c1.
- Alice envoie c1 à Bob via Eve.
- Bob déchiffre c1 avec k1 pour obtenir m en bits, puis le convertit en texte.

Python

```
---------------TRACE SCENARIO 1 ---------------
Alice:
k=000110010011011111110011111101101110101100010100010110001011001100110110
011001100000011000101100101111100010110010101010110100101110010100000110101
111010110110011100101010000011100110010100011111100001111001100111000001001
100100011011110110010011110011011111,
m=011000110110010101100011011010010110010101110011011101000110110001100101
011011010110010101110011011100110110000010110011101100101011000110110110001
100001011010010111001001100000101100100011001010110001101101000011010010110
011001100110011100100110010101110010
Eve:
c=011110100101001010010000100111111000111001100111001011001101111101010011
000010110110001101011111110011010100110111001101101101111000011001101010111
011011101100001011100011100010111111010010001010000010100011100000110010010
101011101011100111101111101100011101
Bob:
k=000110010011011111110011111101101110101100010100010110001011001100110110
011001100000011000101100101111100010110010101010110100101110010100000110101
111010110110011100101010000011100110010100011111100001111001100111000001001
100100011011110110010011110011011111,
m=011000110110010101100011011010010110010101110011011101000110110001100101
011011010110010101110011011100110110000010110011101100101011000110110110001
100001011010010111001001100000101100100011001010110001101101000011010010110
011001100110011100100110010101110010,
plaintext=ceciestlemessageclairadechiffrer
```

Scénario 2 :

- Alice génère secrètement k2.
- Alice chiffre le message m en bits avec k2 pour obtenir c2.
- Alice envoie c2 à Bob via Eve.
- Bob déchiffre c2 avec k2 pour obtenir m en bits, puis le convertit en texte.

Python

```
----------------TRACE SCENARIO 2 ----------------
Alice:
k=0011101010000001100001100101010110101100111011100000010010111010010010000111 0
011111100100000010101010110001010001011110010000001101000110001111110001011001
000101011000101111100101011001011001011110001011010010100010101110010001010100
1110101010110010110110 10,
m=0110001101100101011000110110100101100101011100110111010001101100011001010110
110101100101011100110111001101100001011001110110010101100011011011000110000101
101001011100100110000101100100011001010110001101101000011010010110011001100110
01110010011001010111 0010
Eve:
c=0101100111100110011011111100001000111100101011110111110100011000111101011000
101010000000101111001110111111001100000000111100110011000100101010100110111011100
101100000010000100110000011110100000000010000001101110101110001110000001000110010
100110001101011110101000
Bob:
k=00111010100000011000011001010101101011001110111000000100101110100100100001110
011111100100000010101010110001010001011110010000001101000110001111110001011001
000101011000101111100101011001011001011110001011010010100010101110010001010100
1110101010110010110110 10,
m=0110001101100101011000110110100101100101011100110111010001101100011001010110
110101100101011100110111001101100001011001110110010101100011011011000110000101
101001011100100110000101100100011001010110001101101000011010010110011001100110
01110010011001010111 0010,
plaintext=ceciestlemessageclairadechiffrer
```

Scénario 3 :
- Alice génère secrètement k3.
- Alice chiffre le message m en bits avec k3 pour obtenir c3.
- Alice envoie c3 à Bob via Eve.
- Bob déchiffre c3 avec k3 pour obtenir m en bits, puis le convertit en texte.

Python

```
----------------TRACE SCENARIO 3 ----------------
Alice:
k=10110000011100011101000111101010101010110110011111111101011100101010100000
00010101001111100101110100001101111001001001111000111000001010001100110011000
00001110110001100101010111011010010100100100101110011000011010010101101001 1
00000111011001011101111111101,
m=01100011011001010110001101101001011001010111001101110100011011000110010101
10110101100101011100110111001101100001011001110110010101100011011011000110000
01011010010111001001100001011001000110010101100011011010000110100101100110011
10011001110010011001010111 0010
Eve:
c=11010011000101001011001010000011110011100001010010001001000111101100110101
10100000101010111001000011000000011000010000001110101101101001010111101010 0
01010100100110101100110110110100001100000111101000101100010111011110100101 0
10011110101011000100101000 1111
Bob:
k=10110000011100011101000111101010101010110110011111111101011100101010100000
00010101001111100101110100001101111001001001111000111000001010001100110011000
00001110110001100101010111011010010100100100101110011000011010010101101001 1
00000111011001011101111111101,
m=01100011011001010110001101101001011001010111001101110100011011000110010101
10110101100101011100110111001101100001011001110110010101100011011011000110000
01011010010111001001100001011001000110010101100011011010000110100101100110011
10011001110010011001010111 0010,
plaintext=ceciestlemessageclairadechiffrer
```

(b) Pour chacune de ces trois exécutions, **Eve ne peut pas déchiffrer le message m** avec l'information qu'elle voit. **La sécurité de l'OTP repose sur le fait que la clé est aussi longue que le message et est utilisée une seule fois**. Sans la clé, Eve ne peut pas obtenir le message en regardant le cryptogramme. Le XOR est utilisé pour chiffrer et déchiffrer, et il est mathématiquement difficile de retrouver la clé à partir du cryptogramme sans information supplémentaire.

3 CHIFFRE DE CÉSAR MODIFIÉ

Voici la manière à laquelle nous implémentons et utilisons le schéma de chiffrement de César 3 en Python, puis analysons les interactions entre Alice, Bob et Eve :

3.1 Schéma de chiffrement de César 3 en Python :

Python

```python
import random

# Fonction pour générer une clé secrète k de longueur 32
def Gen3():
    return ''.join(random.choice('abcdefghijklmnopqrstuvwxyz') for _ in
range(32))

# Fonction de chiffrement de César de longueur 32
def E3(k, m):
    if len(k) != 32:
        raise ValueError("La clé doit avoir une longueur de 32 caractères")

    ciphertext = ""
    for i in range(len(m)):
        if m[i].isalpha():
            shift = ord(k[i]) - ord('a')
            if m[i].islower():
                new_char = chr(((ord(m[i]) - ord('a') + shift) % 26) +
ord('a'))
            else:
                new_char = chr(((ord(m[i]) - ord('A') + shift) % 26) +
ord('A'))
            ciphertext += new_char
        else:
            ciphertext += m[i]
    return ciphertext

# Fonction de déchiffrement de César de longueur 32
def D3(k, c):
    if len(k) != 32:
        raise ValueError("La clé doit avoir une longueur de 32 caractères")

    plaintext = ""
    for i in range(len(c)):
        if c[i].isalpha():
            shift = ord(k[i]) - ord('a')
            if c[i].islower():
                new_char = chr(((ord(c[i]) - ord('a') - shift) % 26) +
ord('a'))
            else:
                new_char = chr(((ord(c[i]) - ord('A') - shift) % 26) +
ord('A'))
            plaintext += new_char
        else:
            plaintext += c[i]
    return plaintext
```

```
# Message initial
message = 'ceciestlemessageclairadechiffrer'

# Exécution des 3 Scénarios avec génération de clé aléatoire
for i in range(3):
    key = Gen3()  # Alice génère une clé secrète
    ciphertext = E3(key, message)  # Alice chiffre le message
    # Eve ne modifie pas le cryptogramme, donc ciphertext reste le même
    plaintext = D3(key, ciphertext)  # Bob déchiffre le message

    # Affichage des résultats
    print("\n---------------TRACE SCENARIO",i+1,"---------------")
    print(f"Alice: k={key}, m={message}")
    print(f"Eve: c={ciphertext}")
    print(f"Bob: k={key}, m={plaintext}")
```

3.2 Analyse des interactions :

(a) Répétons le scénario suivant trois fois de manière indépendante :

Scénario 1 :
- Alice génère secrètement k1 (une clé de longueur 32).
- Alice chiffre le message m avec k1 pour obtenir c1.
- Alice envoie c1 à Bob via Eve (qui ne modifie pas le cryptogramme).
- Bob déchiffre c1 avec k1 pour obtenir m.

Python

```
---------------TRACE SCENARIO 1 ---------------
Alice: k=aejcdgaduxoikdpxmpbsjghcirbrpiso,
m=ceciestlemessageclairadechiffrer
Eve: c=cilkhytoyjsacdvboabaagkgkyjwuzwf
Bob: k=aejcdgaduxoikdpxmpbsjghcirbrpiso, m=ceciestlemessageclairadechiffrer
```

Scénario 2 :
- Alice génère secrètement k2 (une clé de longueur 32).
- Alice chiffre le message m avec k2 pour obtenir c2.
- Alice envoie c2 à Bob via Eve (qui ne modifie pas le cryptogramme).
- Bob déchiffre c2 avec k2 pour obtenir m.

Python

```
---------------TRACE SCENARIO 2 ---------------
Alice: k=shkxjrrdcylaswcswudlpeubrsrxarta,
m=ceciestlemessageclairadechiffrer
Eve: c=ulmfnjkogkpskwiwyfdtgexftzzcfixr
Bob: k=shkxjrrdcylaswcswudlpeubrsrxarta, m=ceciestlemessageclairadechiffrer
```

Scénario 3 :
- Alice génère secrètement k3 (une clé de longueur 32).
- Alice chiffre le message m avec k3 pour obtenir c3.
- Alice envoie c3 à Bob via Eve (qui ne modifie pas le cryptogramme).
- Bob déchiffre c3 avec k3 pour obtenir m.

Python

```
---------------TRACE SCENARIO 3 ---------------
Alice: k=coehovjykpnybhemsjcuqsrynluwsdyp, m=ceciestlemessageclairadechiffrer
Eve: c=esgpsncjobrqthkquucchsucpscbxucg
Bob: k=coehovjykpnybhemsjcuqsrynluwsdyp, m=ceciestlemessageclairadechiffrer
```

(b) Pour chacune de ces trois exécutions, **Eve ne peut pas déchiffrer le message m avec l'information qu'elle voit**. **La sécurité de ce schéma est similaire à celle du chiffrement de César classique car les clés sont de longueur 32 et sont générées de manière aléatoire. Cependant, une attaque de force brute reste possible si Eve obtient suffisamment de paires clair-chiffré.**

(c) Si Eve fait un décalage de 9 lettres sur le troisième symbole de c et de 18 lettres sur le quatrième symbole de c dans chacune de ces exécutions, cela signifie qu'elle modifie le cryptogramme. **Bob détectera une altération du cryptogramme et ne pourra pas déchiffrer correctement le message**. La sécurité du schéma permet de détecter les altérations, ce qui le rend plus robuste contre les attaques.