TP : Introduction à la Cryptographie en python

Chiffrement symétrique :

1 - Donner une fonction python césar chiffre(t) qui permet de chiffrer le message t en appliquant l'algorithme de chiffrement de césar :

```python
def cesar_chiffre(t, decalage=3):
    resultat = ""
    for caractere in t:
        if caractere.isalpha():
            # détermine le code de base (majuscule ou minuscule)
            base = ord('A') if caractere.isupper() else ord('a')
            # applique le décalage modulo 26
            resultat += chr((ord(caractere) - base + decalage) % 26 + base)
        else:
            # les caractères non alphabétiques ne sont pas modifiés
            resultat += caractere
    return resultat

# Exemple d'utilisation
message = "Bonjour le Monde !"
chiffre = cesar_chiffre(message)
print("Message chiffré :", chiffre)
```

Le résultat donne :

```
❯ /usr/bin/python /home/mamadou/Documents/Cours-Sorbonne-Paris-Nord/KANAWATI/tp2/1.py
Message chiffré : Erqmrxu oh Prqgh !
```

- L'algorithme utilise un décalage fixe pour chaque lettre du message.
- Seules les lettres de l'alphabet sont modifiées, les autres caractères sont ignorés.
- Le chiffrement respecte la casse (majuscules/minuscules).
- Le décalage est appliqué sur chaque lettre pour obtenir une nouvelle lettre dans l'alphabet.

2 - Donner une fonction python césar déchiffre(m) qui permet de déchiffrer un message chiffré avec le code de césar :

```python
def vigenere_chiffre(t, cle):
    resultat = ""
    cle = cle.lower()
    index_cle = 0

    for caractere in t:
        if caractere.isalpha():
            # Décalage basé sur la lettre de la clé (0 à 25)
            decalage = ord(cle[index_cle % len(cle)]) - ord('a')
            base = ord('A') if caractere.isupper() else ord('a')
            chiffre = chr((ord(caractere) - base + decalage) % 26 + base)
            resultat += chiffre
            index_cle += 1
        else:
            # Les caractères non alphabétiques ne sont pas modifiés
            resultat += caractere

    return resultat

# Exemple d'utilisation
message = "Bonjour le Monde !"
cle = "SECURITE"
chiffre = vigenere_chiffre(message, cle)
print("Message chiffré :", chiffre)
```

Le résultat donne :

```
❯ /usr/bin/python /home/mamadou/Documents/Cours-Sorbonne-Paris-Nord/KANAWATI/tp2/2.py
Message chiffré : Tspdfck pw Qqhum !
```

- Le déchiffrement suit le même principe que le chiffrement, mais inverse le décalage.
- Chaque lettre du message chiffré est ramenée en arrière dans l'alphabet.
- La casse est conservée, et les caractères spéciaux restent inchangés.
- Le résultat est le message original avant chiffrement.

3 - Donner une fonction python vigenère chiffre(t) qui permet de chiffrer le message t en appliquant l'algorithme de chiffrement de Vigenère.

```python
def vigenere_chiffre(t, cle):
    resultat = ""
    cle = cle.lower()
    index_cle = 0

    for caractere in t:
        if caractere.isalpha():
            # Décalage basé sur la lettre de la clé (0 à 25)
            decalage = ord(cle[index_cle % len(cle)]) - ord('a')
            base = ord('A') if caractere.isupper() else ord('a')
            chiffre = chr((ord(caractere) - base + decalage) % 26 + base)
            resultat += chiffre
            index_cle += 1
        else:
            # Les caractères non alphabétiques ne sont pas modifiés
            resultat += caractere

    return resultat

# Exemple d'utilisation
message = "Bonjour le Monde !"
cle = "SECURITE"
chiffre = vigenere_chiffre(message, cle)
print("Message chiffré :", chiffre)
```

Le résultat donne :

```
❯ /usr/bin/python /home/mamadou/Documents/Cours-Sorbonne-Paris-Nord/KANAWATI/tp2/3.py
Message chiffré : Tspdfck pw Qqhum !
```

- L'algorithme utilise un décalage différent pour chaque lettre, basé sur la clé.
- La clé est répétée autant que nécessaire pour couvrir tout le message.
- On différencie majuscules et minuscules, mais la clé est toujours en minuscules pour simplifier les calculs.

4 - Donner une fonction python vigenere déchiffre(t) qui permet de déchiffrer un message M chiffré avec le code de Vigenere.

```python
def vigenere_dechiffre(t, cle):
    resultat = ""
    cle = cle.lower()
    index_cle = 0

    for caractere in t:
        if caractere.isalpha():
            decalage = ord(cle[index_cle % len(cle)]) - ord('a')
            base = ord('A') if caractere.isupper() else ord('a')
            dechiffre = chr((ord(caractere) - base - decalage) % 26 + base)
            resultat += dechiffre
            index_cle += 1
        else:
            resultat += caractere

    return resultat

# Exemple d'utilisation
message_chiffre = "Thptsmv pi Hrcvii !"  # ← Chiffré avec la clé "SECURITE"
cle = "SECURITE"

print("Message déchiffré :", vigenere_dechiffre(message_chiffre, cle))
```

Le résultat donne :

```
❯ /usr/bin/python /home/mamadou/Documents/Cours-Sorbonne-Paris-Nord/KANAWATI/tp2/4.py
Message déchiffré : Bdnzbec lq Dpieap !
```

- Chaque lettre est décalée vers l'arrière selon la lettre correspondante de la clé.
- La clé est répétée autant que nécessaire pour couvrir tout le message.
- On garde les majuscules et minuscules, mais la clé est transformée en minuscules pour simplifier les calculs.
- Les caractères non alphabétiques (espaces et ponctuation) sont inchangés.

Chiffrement Asymétrique :

1 - Donner une fonction python sign(t) qui permet de signer un message t en utilisant l'algorithme RSA.

```python
d = 2753
n = 3233
e = 17

def sign_char(c, d, n):
    m_int = ord(c)  # convertit un caractère en entier (ASCII)
    signature = pow(m_int, d, n)
    return signature

def verify_char(c, signature, e, n):
    m_int = ord(c)
    m_from_sig = pow(signature, e, n)
    return m_int == m_from_sig

message = "HELLO"
signatures = [sign_char(c, d, n) for c in message]

print("Message :", message)
print("Signatures :", signatures)

valid = all(verify_char(c, sig, e, n) for c, sig in zip(message, signatures))
print("Signature valide !" if valid else "Signature invalide.")
```

Le résultat donne :

```
/usr/bin/python /home/mamadou/Documents/Cours-Sorbonne-Paris-Nord/KANAWATI/tp2/Asymétrique/1.py
Message : HELLO
Signatures : [1048, 1318, 818, 818, 2165]
Signature valide !
```

- Le message est signé caractère par caractère car chaque lettre (son code ASCII) est un entier plus petit que le modulo n.
- La signature de chaque lettre est calculée avec la clé privée `(d, n)` par la formule `signature = (lettre^d) mod n`.
- Pour vérifier, on utilise la clé publique `(e, n)` et on calcule `(signature^e) mod n` ; si ça correspond au code ASCII de la lettre, la signature est valide.

- Cette méthode simple fonctionne avec de petits modules RSA et des messages courts

2 - Donner une fonction sig valide(c) qui permet de vérifier si un document signé est valide.

```python
# Clés RSA (faibles pour l'exemple)
e = 17
d = 2753
n = 3233

# Fonction de signature caractère par caractère
def sign_char(c, d, n):
    return pow(ord(c), d, n)

# Fonction de vérification
def sig_valide(c):
    message, signatures = c
    if len(message) != len(signatures):
        return False
    for caractere, sig in zip(message, signatures):
        if ord(caractere) != pow(sig, e, n):
            return False
    return True

# Signature du message "HI"
message = "HI"
signatures = [sign_char(c, d, n) for c in message]
document = (message, signatures)

print("Message :", message)
print("Signatures :", signatures)
print("Signature valide :", sig_valide(document))
```

Le résultat donne :

```
❯ /usr/bin/python /home/mamadou/Documents/Cours-Sorbonne-Paris-Nord/KANAWATI/tp2/Asymétrique/2.py
Message : HI
Signatures : [1048, 565]
Signature valide : True
```

- Chaque caractère du message est signé individuellement avec la clé privée `(d, n)` : `signature = (ord(c) ^ d) mod n`.
- La fonction `sig_valide(document)` vérifie que chaque signature donne bien le bon caractère après déchiffrement avec la clé publique `(e, n)` : `(signature ^ e) mod n == ord(caractère)`.

- Si toutes les lettres sont bien vérifiées, la signature est jugée valide (`True`), sinon invalide (`False`).