Rapport Détaillé des Fonctions de Cryptographie en Python

Projet de Mathématiques - SAIDI Melissa

Lycée des Mathématiques Mohand Mokhbi

Table des Matières

---

1. Introduction et Contexte

Conventions Utilisées

- **Alphabet** : 26 lettres de A à Z (indices 0 à 25)
- **Modulo 26** : Toutes les opérations sur les lettres utilisent l'arithmétique modulo 26
- **Encodage ASCII** :
  - 'A' = 65, 'Z' = 90 (majuscules)
  - 'a' = 97, 'z' = 122 (minuscules)
- **Caractères non-alphabétiques** : Conservés sans modification

Méthodes et Fonctions Python Utilisées

Cette section explique toutes les méthodes et fonctions Python intégrées utilisées dans le code de cryptographie.

1.1 `ord(caractère)` - Obtenir le code ASCII

Description : Retourne le code ASCII (valeur numérique) d'un caractère.

Syntaxe : `ord(caractère)`

Exemples :

```
ord('A')  # Retourne 65
ord('B')  # Retourne 66
ord('Z')  # Retourne 90
ord('a')  # Retourne 97
ord('z')  # Retourne 122
ord('0')  # Retourne 48
```

Utilisation dans le code :

```
m = ord(char)  # Convertit un caractère en sa valeur numérique
# Pour 'A', m = 65
# Pour 'B', m = 66
```

1.2 `chr(nombre)` - Convertir un code ASCII en caractère

Description : L'inverse de `ord()`. Convertit un code ASCII en son caractère correspondant.

Syntaxe : `chr(nombre)`

Exemples :

```
chr(65)   # Retourne 'A'
chr(66)   # Retourne 'B'
chr(90)   # Retourne 'Z'
chr(97)   # Retourne 'a'
chr(122)  # Retourne 'z'
```

Utilisation dans le code :

```
# Reconvertir une valeur numérique en caractère après chiffrement
result += chr((ord(char) - ascii_offset + shift) % 26 + ascii_offset)
```

1.3 `char.isalpha()` - Vérifier si c'est une lettre

Description : Retourne `True` si le caractère est une lettre (A-Z ou a-z), `False` sinon.

Syntaxe : `caractère.isalpha()`

Exemples :

```
'A'.isalpha()   # True
'z'.isalpha()   # True
'5'.isalpha()   # False
' '.isalpha()   # False
'!'.isalpha()   # False
'é'.isalpha()   # True (lettres accentuées aussi)
```

Utilisation dans le code :

```
for char in text:
    if char.isalpha():
        # Traiter seulement les lettres
        # Les espaces, chiffres, ponctuation sont ignorés
    else:
        result += char  # Garder tel quel
```

1.4 `char.isupper()` - Vérifier si c'est une majuscule

Description : Retourne `True` si le caractère est une lettre majuscule, `False` sinon.

Syntaxe : `caractère.isupper()`

Exemples :

```
'A'.isupper()   # True
'Z'.isupper()   # True
'a'.isupper()   # False
'z'.isupper()   # False
'5'.isupper()   # False
```

Utilisation dans le code :

```
# Déterminer l'offset ASCII selon la casse
ascii_offset = ord('A') if char.isupper() else ord('a')
# Si majuscule : ascii_offset = 65
```

```
# Si minuscule : ascii_offset = 97
```

Pourquoi c'est important : Cela permet de préserver la casse originale du texte après chiffrement.

1.5 `string.upper()` - Convertir en majuscules

Description : Retourne une copie de la chaîne avec toutes les lettres en majuscules.

Syntaxe : `chaîne.upper()`

Exemples :

```
'hello'.upper()      # 'HELLO'
'Bonjour'.upper()    # 'BONJOUR'
'CLE'.upper()        # 'CLE' (déjà en majuscules)
'abc123'.upper()     # 'ABC123'
```

Utilisation dans le code :

```
key = key.upper()  # Normaliser la clé en majuscules
# Permet de traiter 'cle', 'CLE', 'Cle' de la même façon
```

1.6 `len(objet)` - Obtenir la longueur

Description : Retourne le nombre d'éléments dans une séquence (chaîne, liste, etc.).

Syntaxe : `len(objet)`

Exemples :

```
len('BONJOUR')   # 7
len('CLE')       # 3
len([1, 2, 3])   # 3
len('')          # 0
```

Utilisation dans le code :

```
# Accès cyclique à la clé
shift = ord(key[key_index % len(key)]) - ord('A')
# Si key = "CLE" (len = 3) et key_index = 5
```

```
# key_index % len(key) = 5 % 3 = 2
# On accède à key[2] = 'E'
```

1.7 `%` (Modulo) - Reste de la division

Description : L'opérateur `%` retourne le reste de la division entière.

Syntaxe : `a % b`

Exemples :

```
10 % 3    # 1  (10 = 3×3 + 1)
26 % 26   # 0
27 % 26   # 1
30 % 26   # 4
5 % 3     # 2
-1 % 26   # 25 (Python gère les négatifs)
```

Utilisation dans le code :

1. **Rester dans l'alphabet (modulo 26) :**

```
(ord(char) - ascii_offset + shift) % 26
# Si le résultat dépasse 25, on revient au début
# Exemple : 'Z' + 3 = 25 + 3 = 28
# 28 % 26 = 2 → 'C'
```

1. **Accès cyclique à la clé :**

```
key[key_index % len(key)]
# Permet de répéter la clé indéfiniment
# Indices : 0, 1, 2, 0, 1, 2, 0, 1, 2...
```

1.8 `range(start, stop, step)` - Générer une séquence de nombres

Description : Génère une séquence de nombres de `start` à `stop-1` avec un pas de `step`.

Syntaxe :

- `range(stop)` - de 0 à stop-1

- `range(start, stop)` - de start à stop-1
- `range(start, stop, step)` - avec un pas

Exemples :

```
list(range(5))         # [0, 1, 2, 3, 4]
list(range(2, 8))      # [2, 3, 4, 5, 6, 7]
list(range(3, 10, 2)) # [3, 5, 7, 9] (nombres impairs)
list(range(10, 0, -1))# [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Utilisation dans le code :

```
# Test de primalité : tester les diviseurs de 2 à √n
for i in range(2, int(math.sqrt(n)) + 1):
    if n % i == 0:
        return False

# Pour n = 61, on teste i = 2, 3, 4, 5, 6, 7
# √61 ≈ 7.81, donc int(7.81) + 1 = 8
```

1.9 `math.sqrt(n)` - Racine carrée

Description : Retourne la racine carrée d'un nombre.

Syntaxe : `math.sqrt(n)`

Exemples :

```
import math
math.sqrt(16)    # 4.0
math.sqrt(25)    # 5.0
math.sqrt(61)    # 7.810249675906654
math.sqrt(2)     # 1.4142135623730951
```

Utilisation dans le code :

```
# Optimisation du test de primalité
# On ne teste que jusqu'à √n car si n = a × b et a > √n, alors b < √n
for i in range(2, int(math.sqrt(n)) + 1):
```

Pourquoi √n suffit : Si n = 100 et n = a × b, les paires possibles sont :

- 2 × 50, 4 × 25, 5 × 20, 10 × 10
- On voit que le plus petit facteur est toujours ≤ √n

1.10 `pow(base, exp, mod)` - Exponentiation modulaire

Description : Calcule `(base^exp) % mod` de manière efficace.

Syntaxe : `pow(base, exposant, modulo)`

Exemples :

```
pow(2, 10)          # 1024 (2^10)
pow(2, 10, 1000)    # 24 (1024 % 1000)
pow(65, 17, 3233)   # Calcul rapide de 65^17 mod 3233
```

Utilisation dans le code :

```
# Chiffrement RSA
c = pow(m, e, n)  # c = m^e mod n

# Déchiffrement RSA
m = pow(c, d, n)  # m = c^d mod n
```

Avantage : Python utilise l'exponentiation rapide (square-and-multiply), ce qui permet de calculer des puissances énormes sans overflow.

1.11 `for char in message:` - Parcourir une chaîne

Description : Itère sur chaque caractère d'une chaîne de caractères.

Syntaxe : `for variable in chaîne:`

Exemples :

```
for char in "ABC":
    print(char)
# Affiche :
# A
# B
# C
```

```
for lettre in "Bonjour":
    print(lettre, ord(lettre))
# B 66
# o 111
# n 110
# ...
```

Utilisation dans le code :

```
for char in message:
    m = ord(char)            # Valeur ASCII du caractère
    c = pow(m, e, n)         # Chiffrer
    encrypted.append(str(c))
```

1.12 `liste.append(élément)` - Ajouter à une liste

Description : Ajoute un élément à la fin d'une liste.

Syntaxe : `liste.append(élément)`

Exemples :

```
nombres = []
nombres.append(1)    # [1]
nombres.append(2)    # [1, 2]
nombres.append(3)    # [1, 2, 3]

texte = []
texte.append("Hello")
texte.append("World")
# ['Hello', 'World']
```

Utilisation dans le code :

```
encrypted = []
for char in message:
    c = pow(m, e, n)
    encrypted.append(str(c))  # Ajouter chaque nombre chiffré
# encrypted = ['2790', '2304', '1256', ...]
```

1.13 `" ".join(liste)` - Joindre une liste en chaîne

Description : Concatène les éléments d'une liste en une chaîne, séparés par le délimiteur spécifié.

Syntaxe : `délimiteur.join(liste)`

Exemples :

```
" ".join(['2790', '2304'])     # '2790 2304'
"-".join(['a', 'b', 'c'])      # 'a-b-c'
"".join(['H', 'e', 'l', 'l', 'o'])  # 'Hello'
", ".join(['pomme', 'banane']) # 'pomme, banane'
```

Utilisation dans le code :

```
return " ".join(encrypted)
# Transforme ['2790', '2304', '1256'] en '2790 2304 1256'
```

1.14 `string.split()` - Diviser une chaîne

Description : Divise une chaîne en liste de sous-chaînes selon un délimiteur.

Syntaxe : `chaîne.split(délimiteur)`

Exemples :

```
'2790 2304 1256'.split()       # ['2790', '2304', '1256']
'a-b-c'.split('-')             # ['a', 'b', 'c']
'pomme,banane'.split(',')      # ['pomme', 'banane']
'  hello  world  '.split()     # ['hello', 'world']
```

Utilisation dans le code :

```
numbers = encrypted_message.strip().split()
# '2790 2304 1256' devient ['2790', '2304', '1256']
for num in numbers:
    c = int(num)  # Convertir chaque nombre
```

1.15 `string.strip()` - Supprimer les espaces

Description : Supprime les espaces blancs au début et à la fin d'une chaîne.

Syntaxe : `chaîne.strip()`

Exemples :

```
'  hello  '.strip()      # 'hello'
'\n\ttest\n'.strip()     # 'test'
'2790 2304 '.strip()     # '2790 2304'
```

Utilisation dans le code :

```
numbers = encrypted_message.strip().split()
# Nettoie les espaces avant de diviser
```

1.16 `int(string)` - Convertir en entier

Description : Convertit une chaîne de caractères en nombre entier.

Syntaxe : `int(chaîne)`

Exemples :

```
int('42')       # 42
int('2790')     # 2790
int('-5')       # -5
int('3.14')     # Erreur! Utiliser float() d'abord
```

Utilisation dans le code :

```
for num in numbers:
    c = int(num)       # Convertir '2790' en 2790
    m = pow(c, d, n)   # Calculer avec le nombre
```

Exemple Complet : Décomposition d'une Ligne de Code

```
result += chr((ord(char) - ascii_offset + shift) % 26 + ascii_offset)
```

Décomposition étape par étape pour char = 'B', shift = 3 :

Étape	Opération	Valeur
1	`char.isupper()`	True
2	`ascii_offset = ord('A')`	65
3	`ord(char)`	ord('B') = 66
4	`ord(char) - ascii_offset`	66 - 65 = 1
5	`... + shift`	1 + 3 = 4
6	`... % 26`	4 % 26 = 4
7	`... + ascii_offset`	4 + 65 = 69
8	`chr(...)`	chr(69) = 'E'

Résultat : 'B' chiffré avec décalage 3 donne 'E'

---

2. Chiffrement de César

2.1 Description

Le chiffrement de César est une technique de substitution où chaque lettre du message est remplacée par une lettre située à un nombre fixe de positions plus loin dans l'alphabet.

2.2 Formule Mathématique

Chiffrement :

```
L' = (L + décalage) mod 26
```

Déchiffrement :

```
L = (L' - décalage) mod 26
```

Où :

- L = position de la lettre originale (0-25)
- L' = position de la lettre chiffrée (0-25)
- décalage = clé de chiffrement (nombre entier)

2.3 Fonction `caesar_encrypt(text, shift)`

```
def caesar_encrypt(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            # Déterminer le point de départ ASCII (majuscule ou minuscule)
            ascii_offset = ord('A') if char.isupper() else ord('a')
            # Appliquer le décalage avec modulo 26
            result += chr((ord(char) - ascii_offset + shift) % 26 + ascii_
        else:
            # Garder les caractères non-alphabétiques
            result += char
    return result
```

Algorithme pas à pas :

1. Parcourir chaque caractère du texte
2. Si c'est une lettre :
    - Calculer l'offset ASCII (65 pour majuscule, 97 pour minuscule)
    - Convertir la lettre en position (0-25)
    - Ajouter le décalage
    - Appliquer modulo 26 pour rester dans l'alphabet
    - Reconvertir en caractère ASCII
3. Sinon, garder le caractère tel quel
4. Retourner le résultat

2.4 Fonction `caesar_decrypt(text, shift)`

```
def caesar_decrypt(text, shift):
    return caesar_encrypt(text, -shift)
```

Le déchiffrement utilise simplement un décalage négatif.

2.5 Exemple Complet

Entrée :

- Texte : `"BONJOUR"`
- Décalage : `3`

Calcul détaillé :

Lettre	Position	+ Décalage	Mod 26	Résultat
B	1	1 + 3 = 4	4	E
O	14	14 + 3 = 17	17	R
N	13	13 + 3 = 16	16	Q
J	9	9 + 3 = 12	12	M
O	14	14 + 3 = 17	17	R
U	20	20 + 3 = 23	23	X
R	17	17 + 3 = 20	20	U

Sortie : `"ERQMRXU"`

Vérification du déchiffrement :

- Entrée : `"ERQMRXU"`, Décalage : `3`
- Sortie : `"BONJOUR"` ✓

---

3. Chiffrement de Vigenère

3.1 Description

Le chiffrement de Vigenère est une amélioration du César utilisant un mot-clé au lieu d'un décalage fixe. Chaque lettre du mot-clé définit le décalage à appliquer.

3.2 Formule Mathématique

Chiffrement :

```
L'[i] = (L[i] + K[i mod len(K)]) mod 26
```

Déchiffrement :

```
L[i] = (L'[i] - K[i mod len(K)]) mod 26
```

Où :

- L[i] = i-ème lettre du message
- K[i] = i-ème lettre de la clé (cyclique)
- len(K) = longueur de la clé

3.3 Fonction `vigenere_encrypt(text, key)`

```
def vigenere_encrypt(text, key):
    if not key:
        return text
    result = ""
    key = key.upper()
    key_index = 0
    for char in text:
        if char.isalpha():
            ascii_offset = ord('A') if char.isupper() else ord('a')
            # Calculer le décalage basé sur la lettre de la clé
            shift = ord(key[key_index % len(key)]) - ord('A')
            result += chr((ord(char) - ascii_offset + shift) % 26 + ascii_
            key_index += 1
        else:
            result += char
    return result
```

Algorithme pas à pas :

1. Convertir la clé en majuscules
2. Initialiser un index pour la clé à 0
3. Pour chaque caractère du texte :
    - Si c'est une lettre :
        - Obtenir la lettre de la clé (cyclique)
        - Calculer le décalage (position de la lettre de clé)
        - Appliquer le décalage comme dans César

- Incrémenter l'index de la clé
- Sinon, garder le caractère

4. Retourner le résultat

3.4 Fonction `vigenere_decrypt(text, key)`

```
def vigenere_decrypt(text, key):
    if not key:
        return text
    result = ""
    key = key.upper()
    key_index = 0
    for char in text:
        if char.isalpha():
            ascii_offset = ord('A') if char.isupper() else ord('a')
            # Soustraire le décalage au lieu de l'ajouter
            shift = ord(key[key_index % len(key)]) - ord('A')
            result += chr((ord(char) - ascii_offset - shift) % 26 + ascii_
            key_index += 1
        else:
            result += char
    return result
```

3.5 Exemple Complet

Entrée :

- Texte : `"MATHEMATIQUES"`
- Clé : `"CLE"`

Table de correspondance de la clé :

- C = 2
- L = 11
- E = 4

Calcul détaillé :

Lettre	Position	Clé	Décalage	Calcul	Mod 26	Résultat
M	12	C	2	12 + 2	14	O

A	0	L	11	0 + 11	11	L
T	19	E	4	19 + 4	23	X
H	7	C	2	7 + 2	9	J
E	4	L	11	4 + 11	15	P
M	12	E	4	12 + 4	16	Q
A	0	C	2	0 + 2	2	C
T	19	L	11	19 + 11	4 (30 mod 26)	E
I	8	E	4	8 + 4	12	M
Q	16	C	2	16 + 2	18	S
U	20	L	11	20 + 11	5 (31 mod 26)	F
E	4	E	4	4 + 4	8	I
S	18	C	2	18 + 2	20	U

Sortie : `"OLXJPQCEMSFIU"`

---

4. Chiffrement RSA

4.1 Introduction au RSA

RSA (Rivest-Shamir-Adleman) est un système de cryptographie asymétrique utilisant deux clés : une clé publique pour chiffrer et une clé privée pour déchiffrer.

4.2 Fonctions Auxiliaires

4.2.1 Test de Primalité : `is_prime(n)`

```
def is_prime(n):
```

```
    if n < 2:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True
```

Algorithme :

1. Si n < 2, retourner False
2. Tester tous les diviseurs de 2 à √n
3. Si un diviseur est trouvé, n n'est pas premier
4. Sinon, n est premier

Exemple :

- `is_prime(61)` → True (61 n'a aucun diviseur entre 2 et 7)
- `is_prime(60)` → False (60 = 2 × 30)

4.2.2 Plus Grand Commun Diviseur : `gcd(a, b)`

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
```

Algorithme d'Euclide :

1. Tant que b ≠ 0 :
    - Remplacer (a, b) par (b, a mod b)
2. Retourner a

Exemple :

```
gcd(48, 18)
→ (18, 48 mod 18) = (18, 12)
→ (12, 18 mod 12) = (12, 6)
→ (6, 12 mod 6) = (6, 0)
→ Résultat : 6
```

4.2.3 Inverse Modulaire : `mod_inverse(e, phi)`

```
def mod_inverse(e, phi):
    def extended_gcd(a, b):
        if a == 0:
            return b, 0, 1
        gcd, x1, y1 = extended_gcd(b % a, a)
        x = y1 - (b // a) * x1
        y = x1
        return gcd, x, y

    _, x, _ = extended_gcd(e % phi, phi)
    return (x % phi + phi) % phi
```

Principe mathématique : Trouver d tel que : $e \times d \equiv 1 \pmod{\varphi}$

Utilise l'algorithme d'Euclide étendu pour trouver les coefficients de Bézout.

Exemple :

- e = 65537, φ = 3120
- `mod_inverse(65537, 3120)` calcule d tel que 65537 × d ≡ 1 (mod 3120)

4.3 Génération des Clés RSA : `generate_rsa_keys(p, q)`

```
def generate_rsa_keys(p, q):
    if not (is_prime(p) and is_prime(q)):
        return None, None, None, None

    n = p * q                    # Module RSA
    phi = (p - 1) * (q - 1)      # Indicatrice d'Euler

    e = 65537                    # Exposant public standard
    if gcd(e, phi) != 1:
        for candidate in range(3, phi, 2):
            if gcd(candidate, phi) == 1:
                e = candidate
                break

    d = mod_inverse(e, phi)      # Exposant privé
    return n, e, d, phi
```

Algorithme pas à pas :

1. **Vérification** : p et q doivent être premiers
2. **Calcul de n** : n = p × q (module RSA)
3. **Calcul de φ(n)** : φ = (p-1) × (q-1) (indicatrice d'Euler)
4. **Choix de e** : Exposant public tel que gcd(e, φ) = 1
5. **Calcul de d** : Inverse modulaire de e modulo φ

Clés générées :

- Clé publique : (e, n)
- Clé privée : (d, n)

4.4 Chiffrement RSA : `rsa_encrypt(message, e, n)`

```
def rsa_encrypt(message, e, n):
    encrypted = []
    for char in message:
        m = ord(char)           # Valeur ASCII du caractère
        c = pow(m, e, n)        # c = m^e mod n
        encrypted.append(str(c))
    return " ".join(encrypted)
```

Formule mathématique :

```
c = m^e mod n
```

Où :

- m = valeur numérique du message (code ASCII)
- e = exposant public
- n = module RSA
- c = message chiffré

4.5 Déchiffrement RSA : `rsa_decrypt(encrypted_message, d, n)`

```
def rsa_decrypt(encrypted_message, d, n):
    try:
        numbers = encrypted_message.strip().split()
        decrypted = ""
        for num in numbers:
            c = int(num)
            m = pow(c, d, n)     # m = c^d mod n
            decrypted += chr(m)
```

```
            return decrypted
        except:
            return "Erreur de déchiffrement"
```

Formule mathématique :

```
m = c^d mod n
```

Pourquoi ça fonctionne (Théorème d'Euler) :

```
c^d = (m^e)^d = m^(e×d) = m^(1 + k×φ) = m × (m^φ)^k ≡ m × 1^k ≡ m (mod n)
```

4.6 Exemple Complet RSA

Paramètres :

- p = 61 (premier)
- q = 53 (premier)

Génération des clés :

```
n = 61 × 53 = 3233
φ = (61-1) × (53-1) = 60 × 52 = 3120
e = 65537 → gcd(65537, 3120) = 1 ✓
d = mod_inverse(65537, 3120) = 2753
```

Clés :

- Clé publique : (e=65537, n=3233)
- Clé privée : (d=2753, n=3233)

Chiffrement du message "AB" :

Caractère	ASCII (m)	Calcul c = m^e mod n	Résultat
A	65	65^65537 mod 3233	2790
B	66	66^65537 mod 3233	2304

Message chiffré : `"2790 2304"`

Déchiffrement :

Chiffré (c)	Calcul m = c^d mod n	ASCII	Caractère
2790	2790^2753 mod 3233	65	A
2304	2304^2753 mod 3233	66	B

Message déchiffré : `"AB"` ✓

---

5. Conclusion

Comparaison des Méthodes

Critère	César	Vigenère	RSA
Type	Symétrique	Symétrique	Asymétrique
Clé	Nombre (1-25)	Mot	Deux clés (publique/privée)
Sécurité	Très faible (26 clés)	Moyenne	Très forte
Complexité	O(n)	O(n)	O(n × log(e))
Usage	Pédagogique	Pédagogique	Réel (communications sécurisées)

Limites de l'Implémentation

1. **César et Vigenère** :
   - Vulnérables à l'analyse fréquentielle
   - Adaptés uniquement à des fins pédagogiques
2. **RSA (version pédagogique)** :
   - Utilise des nombres premiers petits (non sécurisé)
   - En production, p et q doivent avoir 1024+ bits chacun
   - Chiffre caractère par caractère (inefficace)

- En production, on utilise RSA pour échanger une clé symétrique

- `app.py` : Implémentation principale avec gestion de la casse
- `flask_api.py` : Version alternative (convertit tout en majuscules)

Document généré pour le projet de Cryptographie et Mathématiques Lycée des Mathématiques
Mohand Mokhbi