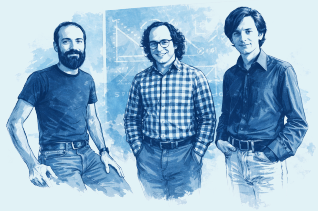
Sécurisation des communications

Chiffrement RSA en Python - Correction

02

Rivest, Shamir, Adleman



Le chiffrement RSA, inventé en 1977 par Ron Rivest, Adi Shamir et Leonard Adleman, est l'un des premiers systèmes de cryptographie à clé publique. Contrairement au chiffrement de Vigenère, il repose sur une paire de clés : une clé publique pour chiffrer et une clé privée pour déchiffrer.
La sécurité de RSA repose sur la difficulté de factoriser le produit de deux grands nombres premiers.
Ce système est encore largement utilisé aujourd'hui pour sécuriser les communications sur Internet (HTTPS, signatures numériques, etc.).

Principe du RSA

Génération des clés & chiffrement.

1. Choisir deux nombres premiers p et q
2. Calculer $n = p \times q$
3. Calculer $\varphi = (p-1) \times (q-1)$
4. Choisir e tel que $\text{pgcd}(e, \varphi) = 1$
5. Trouver d tel que $e \times d \equiv 1 \pmod{\varphi}$

Clé publique : (e, n)
Chiffrement : $c = m^e \mod n$

Clé privée : (d, n)
Déchiffrement : $m = c^d \mod n$

Partie 1 — Génération des clés

Question 1 — `est_premier(n)`

Écrire une fonction **`est_premier(n)`** qui renvoie **`True`** si **`n`** est un nombre premier, **`False`** sinon.

Exemples

```
>>> est_premier(2)
True
>>> est_premier(15)
False
>>> est_premier(61)
True
```

Correction

Correction

```
def est_premier(n):
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

Question 2 — `pgcd(a, b)`

Écrire une fonction **récursive** **`pgcd(a, b)`** qui calcule le plus grand commun diviseur de **`a`** et **`b`** à l'aide de l'algorithme d'Euclide.

- **Condition d'arrêt :** $\text{pgcd}(a, 0) = a$
- **Relation de récurrence :** $\text{pgcd}(a, b) = \text{pgcd}(b,\ a \bmod b)$

Exemples

```
>>> pgcd(17, 3120)
1
>>> pgcd(12, 8)
4
```

Correction

Correction

```
def pgcd(a, b):
    if b == 0:
        return a
    return pgcd(b, a % b)
```

Question 3 — `inverse_modulaire(e, phi)`

Écrire une fonction `inverse_modulaire(e, phi)` qui renvoie l'entier d tel que $(e \times d) \bmod \varphi = 1$.
Indication : on peut tester toutes les valeurs de d de 2 à $\varphi - 1$.

Exemples

```
>>> inverse_modulaire(17, 3120)
2753
>>> 17 * 2753 % 3120
1
```

Correction

Correction

```
def inverse_modulaire(e, phi):
    for d in range(2, phi):
        if (e * d) % phi == 1:
            return d
```

Question 4 — `generer_cles(p, q, e)`

En utilisant les fonctions précédentes, écrire une fonction `generer_cles(p, q, e)` qui renvoie la **clé publique** (e, n) et la **clé privée** (d, n) sous forme d'un tuple de deux tuples. La fonction doit **vérifier** avec `assert` que :

— `p` et `q` sont des nombres premiers

— `e` est premier avec φ (sinon `e` n'est pas inversible)

Exemples

```
>>> generer_cles(61, 53, 17)
((17, 3233), (2753, 3233))
>>> generer_cles(7, 13, 5)
((5, 91), (29, 91))
>>> generer_cles(10, 53, 17)
AssertionError: p n'est pas premier
```

Correction

Correction

```
def generer_cles(p, q, e):
    assert est_premier(p), "p n'est pas premier"
    assert est_premier(q), "q n'est pas premier"
    n = p * q
    phi = (p - 1) * (q - 1)
    assert pgcd(e, phi) == 1, "e et phi ne sont pas premiers entre eux"
    d = inverse_modulaire(e, phi)
    return (e, n), (d, n)
```

Partie 2 — Chiffrement et déchiffrement

Clés utilisées dans ce TP

Dans la suite du TP, on pourra tester les fonctions de chifremment à l'aide des clés suivantes :
On choisit p et q tel que $p = 61$ et $q = 53$.

Génération des clés :

— $n = 61 \times 53 = 3233$

- $\varphi = 60 \times 52 = 3120$
- $e = 17$ (clé publique : $(17,\ 3233)$)
- $d = 2753$ (clé privée : $(2753,\ 3233)$)

Chaque lettre est représentée par son code ASCII : `ord('A')` = 65, `ord('B')` = 66, etc.

Question 5 — `chiffre_lettre(lettre, e, n)`

Écrire une fonction qui chiffre une lettre en appliquant la formule $c = m^e \bmod n$, où m est le code ASCII de la lettre.
Indication : `pow(m, e, n)` calcule $m^e \bmod n$ efficacement en Python.

Exemples

```
>>> chiffre_lettre('A', 17, 3233)
2790
>>> chiffre_lettre('N', 17, 3233)
3165
```

Correction

Correction

```
def chiffre_lettre(lettre, e, n):
    return pow(ord(lettre), e, n)
```

Question 6 — `chiffre_message(message, e, n)`

Écrire une fonction qui chiffre un message complet. Elle renvoie une **liste** d'entiers.

Texte	N	S	I
ASCII	78	83	73
Chiffré	3165	2680	1486

Exemples

```
>>> chiffre_message("NSI", 17, 3233)
[3165, 2680, 1486]
>>> chiffre_message("RSA", 17, 3233)
[1859, 2680, 2790]
```

Correction

Correction

```
def chiffre_message(message, e, n):
    resultat = []
    for lettre in message:
        resultat.append(chiffre_lettre(lettre, e, n))
    return resultat
```

Question 7 — `dechiffre_code(code, d, n)`

Écrire une fonction qui déchiffre un entier c en appliquant $m = c^d \bmod n$, puis renvoie le caractère correspondant.

Exemples

```
>>> dechiffre_code(2790, 2753, 3233)
'A'
>>> dechiffre_code(3165, 2753, 3233)
'N'
```

Correction

Correction

```
def dechiffre_code(code, d, n):
    return chr(pow(code, d, n))
```

Question 8 — `dechiffre_message(message, d, n)`

Écrire une fonction qui déchiffre une liste d'entiers chiffrés et renvoie le message en clair.

Chiffré	3165	2680	1486
ASCII	78	83	73
Texte	N	S	I

Exemples

```
>>> dechiffre_message([3165, 2680, 1486], 2753, 3233)
'NSI'
>>> dechiffre_message([1859, 2680, 2790], 2753, 3233)
'RSA'
```

Correction

Correction

```
def dechiffre_message(message, d, n):
    resultat = ""
    for c in message:
        resultat += dechiffre_code(c, d, n)
    return resultat
```