atelier1

May 23, 2025

1. Implémentation manuelle de l'inverse modulaire (Algorithme d'Euclide étendu)

[1]:
```python
def inverse_modulaire(a, m):
    r0, r1 = a, m
    x0, x1 = 1, 0
    y0, y1 = 0, 1

    while r1 != 0:
        q = r0 // r1
        r0, r1 = r1, r0 - q * r1
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1

    if r0 != 1:
        raise ValueError("L'inverse modulaire n'existe pas (a et m ne sont pas␣
 ↪premiers entre eux)")

    return x0 % m

# Exemple de test
a = 17
m = 13
print("Inverse modulaire de", a, "mod", m, "est :", inverse_modulaire(a, m))
```

```
Inverse modulaire de 17 mod 13 est : 10
```

2. Calcul de l'inverse modulaire avec pow() et sympy.mod_inverse

Méthode 1 : Avec la fonction pow() (intégrée à Python)

[2]:
```python
a = 17
m = 13
try:
    inverse = pow(a, -1, m)
    print("Inverse modulaire avec pow():", inverse)
except ValueError:
    print("L'inverse modulaire n'existe pas")
```

```
Inverse modulaire avec pow(): 10
```

Méthode 2 : Avec sympy.mod_inverse

[3]:
```python
from sympy import mod_inverse

a = 17
m = 13
try:
    inverse = mod_inverse(a, m)
    print("Inverse modulaire avec sympy.mod_inverse:", inverse)
except ValueError:
    print("L'inverse modulaire n'existe pas")
```

```
Inverse modulaire avec sympy.mod_inverse: 10
```

```
3. Implémentation de SHA-1
```

```
            Étape 1 : Padding du message (pad_message)
```

[4]:
```python
def pad_message(message: bytes) -> bytes:
    original_length = len(message) * 8
    message += b'\x80'

    while (len(message) * 8) % 512 != 448:
        message += b'\x00'

    message += original_length.to_bytes(8, byteorder='big')
    return message
```

```
            Étape 2 : Initialisation des registres
```

[5]:
```python
H0 = 0x67452301
H1 = 0xEFCDAB89
H2 = 0x98BADCFE
H3 = 0x10325476
H4 = 0xC3D2E1F0
```

```
            Étape 3 : Traitement des blocs
```

Fonction de rotation à gauche

[6]:
```python
def left_rotate(n, b):
    return ((n << b) | (n >> (32 - b))) & 0xFFFFFFFF
```

Fonction de traitement d'un bloc (512 bits)

[7]:
```python
def process_block(block, H):
    W = [int.from_bytes(block[i:i+4], 'big') for i in range(0, 64, 4)]

    for i in range(16, 80):
        W.append(left_rotate(W[i-3] ^ W[i-8] ^ W[i-14] ^ W[i-16], 1))

    a, b, c, d, e = H
    for i in range(80):
        if 0 <= i <= 19:
            f = (b & c) | ((~b) & d)
            k = 0x5A827999
        elif 20 <= i <= 39:
            f = b ^ c ^ d
            k = 0x6ED9EBA1
        elif 40 <= i <= 59:
            f = (b & c) | (b & d) | (c & d)
            k = 0x8F1BBCDC
        else:
            f = b ^ c ^ d
            k = 0xCA62C1D6

        temp = (left_rotate(a, 5) + f + e + k + W[i]) & 0xFFFFFFFF
        e = d
        d = c
        c = left_rotate(b, 30)
        b = a
        a = temp

    H[0] = (H[0] + a) & 0xFFFFFFFF
    H[1] = (H[1] + b) & 0xFFFFFFFF
    H[2] = (H[2] + c) & 0xFFFFFFFF
    H[3] = (H[3] + d) & 0xFFFFFFFF
    H[4] = (H[4] + e) & 0xFFFFFFFF

    return H
```

Étape 4 : Fonction principale de hachage SHA-1

[8]:
```
def sha1(message: bytes) -> str:
    message = pad_message(message)
    H = [H0, H1, H2, H3, H4]

    for i in range(0, len(message), 64):
        block = message[i:i+64]
        H = process_block(block, H)

    return ''.join(f'{h:08x}' for h in H)
```

Étape 5 : Test

[9]:
```
if __name__ == "__main__":
    msg = b"hello"
    print("SHA-1 custom:", sha1(msg))
```

```
SHA-1 custom: aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
```

4. Comparaison avec hashlib.sha1()

Code de comparaison :

```
[10]: import hashlib


message = b"hello"


mon_hash = sha1(message)


hashlib_hash = hashlib.sha1(message).hexdigest()


print("Mon SHA-1       :", mon_hash)
print("hashlib SHA-1   :", hashlib_hash)


print("Les deux résultats sont-ils identiques ?", mon_hash == hashlib_hash)
```

```
Mon SHA-1       : aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
hashlib SHA-1   : aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
Les deux résultats sont-ils identiques ? True
```

5. Implémentation de SHA-256 (simplifiée mais fidèle)

Étape 1 : Padding du message (SHA-256)

```
[11]: def pad_message_sha256(message: bytes) -> bytes:
    original_length = len(message) * 8
    message += b'\x80'

    while (len(message) * 8) % 512 != 448:
        message += b'\x00'

    message += original_length.to_bytes(8, byteorder='big')
    return message
```

Étape 2 : Initialisation des registres SHA-256

```
[12]: H256 = [
    0x6a09e667,
    0xbb67ae85,
    0x3c6ef372,
    0xa54ff53a,
    0x510e527f,
    0x9b05688c,
    0x1f83d9ab,
    0x5be0cd19
]
```

Étape 3 : Constantes K

[13]:
```
K256 = [
    0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
    0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
    0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
    0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
    0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
    0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
    0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
    0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
    0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
    0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
    0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
    0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
    0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
    0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
    0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
    0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
]
```

Étape 4 : Fonctions de SHA-256

[14]:
```
def right_rotate(value, bits):
    return ((value >> bits) | (value << (32 - bits))) & 0xFFFFFFFF
```

Étape 5 : Traitement d'un bloc de 512 bits

[15]:
```
def process_block_sha256(block, H):
    W = [int.from_bytes(block[i:i+4], 'big') for i in range(0, 64, 4)]

    for i in range(16, 64):
        s0 = right_rotate(W[i-15], 7) ^ right_rotate(W[i-15], 18) ^ (W[i-15] >>␣
 ↪3)
        s1 = right_rotate(W[i-2], 17) ^ right_rotate(W[i-2], 19) ^ (W[i-2] >>␣
 ↪10)
        W.append((W[i-16] + s0 + W[i-7] + s1) & 0xFFFFFFFF)

    a, b, c, d, e, f, g, h = H

    for i in range(64):
        S1 = right_rotate(e, 6) ^ right_rotate(e, 11) ^ right_rotate(e, 25)
        ch = (e & f) ^ ((~e) & g)
        temp1 = (h + S1 + ch + K256[i] + W[i]) & 0xFFFFFFFF
        S0 = right_rotate(a, 2) ^ right_rotate(a, 13) ^ right_rotate(a, 22)
        maj = (a & b) ^ (a & c) ^ (b & c)
        temp2 = (S0 + maj) & 0xFFFFFFFF

        h = g
        g = f
        f = e
        e = (d + temp1) & 0xFFFFFFFF
        d = c
        c = b
        b = a
        a = (temp1 + temp2) & 0xFFFFFFFF

    H[0] = (H[0] + a) & 0xFFFFFFFF
    H[1] = (H[1] + b) & 0xFFFFFFFF
    H[2] = (H[2] + c) & 0xFFFFFFFF
    H[3] = (H[3] + d) & 0xFFFFFFFF
    H[4] = (H[4] + e) & 0xFFFFFFFF
    H[5] = (H[5] + f) & 0xFFFFFFFF
    H[6] = (H[6] + g) & 0xFFFFFFFF
    H[7] = (H[7] + h) & 0xFFFFFFFF

    return H
```

Étape 6 : SHA-256 complet

[16]:
```
def sha256(message: bytes) -> str:
    message = pad_message_sha256(message)
    H = H256.copy()

    for i in range(0, len(message), 64):
        block = message[i:i+64]
        H = process_block_sha256(block, H)

    return ''.join(f'{h:08x}' for h in H)
```

Test final

[17]:
```
if __name__ == "__main__":
    msg = b"hello"
    print("SHA-256 custom:", sha256(msg))
```

```
SHA-256 custom: 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
```

6. Afficher chaque étape du calcul dans SHA-1

On va modifier la fonction process_block() pour imprimer les étapes de transformation : chaque itération, avec les valeurs de a, b, c, d, e, etc.

Version modifiée de process_block() (SHA-1) avec print() :

[18]:
```python
def process_block_verbose(block, H):
    W = [int.from_bytes(block[i:i+4], 'big') for i in range(0, 64, 4)]

    for i in range(16, 80):
        W.append(left_rotate(W[i-3] ^ W[i-8] ^ W[i-14] ^ W[i-16], 1))

    a, b, c, d, e = H

    print("\n--- Début du traitement d'un bloc ---")
    for i in range(80):
        if 0 <= i <= 19:
            f = (b & c) | ((~b) & d)
            k = 0x5A827999
        elif 20 <= i <= 39:
            f = b ^ c ^ d
            k = 0x6ED9EBA1
        elif 40 <= i <= 59:
            f = (b & c) | (b & d) | (c & d)
            k = 0x8F1BBCDC
        else:
            f = b ^ c ^ d
            k = 0xCA62C1D6

        temp = (left_rotate(a, 5) + f + e + k + W[i]) & 0xFFFFFFFF
        e = d
        d = c
        c = left_rotate(b, 30)
        b = a
        a = temp

        print(f"Étape {i:02}: a={a:08x}, b={b:08x}, c={c:08x}, d={d:08x}, e={e:
↪08x}")

    H[0] = (H[0] + a) & 0xFFFFFFFF
    H[1] = (H[1] + b) & 0xFFFFFFFF
    H[2] = (H[2] + c) & 0xFFFFFFFF
    H[3] = (H[3] + d) & 0xFFFFFFFF
    H[4] = (H[4] + e) & 0xFFFFFFFF

    print("--- Fin du bloc ---\n")
    return H
```

Utilisation dans la fonction sha1_verbose :

[19]:
```python
def sha1_verbose(message: bytes) -> str:
    message = pad_message(message)
    H = [H0, H1, H2, H3, H4]
    for i in range(0, len(message), 64):
        block = message[i:i+64]
        H = process_block_verbose(block, H)

    return ''.join(f'{h:08x}' for h in H)
```

Test :

```
[20]: if __name__ == "__main__":
          sha1_verbose(b"abc")
```

```
--- Début du traitement d'un bloc ---
Étape 00: a=0116fc33, b=67452301, c=7bf36ae2, d=98badcfe, e=10325476
Étape 01: a=8990536d, b=0116fc33, c=59d148c0, d=7bf36ae2, e=98badcfe
Étape 02: a=a1390f08, b=8990536d, c=c045bf0c, d=59d148c0, e=7bf36ae2
Étape 03: a=cdd8e11b, b=a1390f08, c=626414db, d=c045bf0c, e=59d148c0
Étape 04: a=cfd499de, b=cdd8e11b, c=284e43c2, d=626414db, e=c045bf0c
Étape 05: a=3fc7ca40, b=cfd499de, c=f3763846, d=284e43c2, e=626414db
Étape 06: a=993e30c1, b=3fc7ca40, c=b3f52677, d=f3763846, e=284e43c2
Étape 07: a=9e8c07d4, b=993e30c1, c=0ff1f290, d=b3f52677, e=f3763846
Étape 08: a=4b6ae328, b=9e8c07d4, c=664f8c30, d=0ff1f290, e=b3f52677
Étape 09: a=8351f929, b=4b6ae328, c=27a301f5, d=664f8c30, e=0ff1f290
Étape 10: a=fbda9e89, b=8351f929, c=12dab8ca, d=27a301f5, e=664f8c30
Étape 11: a=63188fe4, b=fbda9e89, c=60d47e4a, d=12dab8ca, e=27a301f5
Étape 12: a=4607b664, b=63188fe4, c=7ef6a7a2, d=60d47e4a, e=12dab8ca
Étape 13: a=9128f695, b=4607b664, c=18c623f9, d=7ef6a7a2, e=60d47e4a
Étape 14: a=196bee77, b=9128f695, c=1181ed99, d=18c623f9, e=7ef6a7a2
Étape 15: a=20bdd62f, b=196bee77, c=644a3da5, d=1181ed99, e=18c623f9
Étape 16: a=4e925823, b=20bdd62f, c=c65afb9d, d=644a3da5, e=1181ed99
Étape 17: a=82aa6728, b=4e925823, c=c82f758b, d=c65afb9d, e=644a3da5
Étape 18: a=dc64901d, b=82aa6728, c=d3a49608, d=c82f758b, e=c65afb9d
Étape 19: a=fd9e1d7d, b=dc64901d, c=20aa99ca, d=d3a49608, e=c82f758b
Étape 20: a=1a37b0ca, b=fd9e1d7d, c=77192407, d=20aa99ca, e=d3a49608
Étape 21: a=33a23bfc, b=1a37b0ca, c=7f67875f, d=77192407, e=20aa99ca
Étape 22: a=21283486, b=33a23bfc, c=868dec32, d=7f67875f, e=77192407
Étape 23: a=d541f12d, b=21283486, c=0ce88eff, d=868dec32, e=7f67875f
Étape 24: a=c7567dc6, b=d541f12d, c=884a0d21, d=0ce88eff, e=868dec32
Étape 25: a=48413ba4, b=c7567dc6, c=75507c4b, d=884a0d21, e=0ce88eff
Étape 26: a=be35fbd5, b=48413ba4, c=b1d59f71, d=75507c4b, e=884a0d21
Étape 27: a=4aa84d97, b=be35fbd5, c=12104ee9, d=b1d59f71, e=75507c4b
Étape 28: a=8370b52e, b=4aa84d97, c=6f8d7ef5, d=12104ee9, e=b1d59f71
Étape 29: a=c5fbaf5d, b=8370b52e, c=d2aa1365, d=6f8d7ef5, e=12104ee9
Étape 30: a=1267b407, b=c5fbaf5d, c=a0dc2d4b, d=d2aa1365, e=6f8d7ef5
Étape 31: a=3b845d33, b=1267b407, c=717eebd7, d=a0dc2d4b, e=d2aa1365
Étape 32: a=046faa0a, b=3b845d33, c=c499ed01, d=717eebd7, e=a0dc2d4b
Étape 33: a=2c0ebc11, b=046faa0a, c=cee1174c, d=c499ed01, e=717eebd7
```

```
Étape 34: a=21796ad4, b=2c0ebc11, c=811bea82, d=cee1174c, e=c499ed01
Étape 35: a=dcbbb0cb, b=21796ad4, c=4b03af04, d=811bea82, e=cee1174c
Étape 36: a=0f511fd8, b=dcbbb0cb, c=085e5ab5, d=4b03af04, e=811bea82
Étape 37: a=dc63973f, b=0f511fd8, c=f72eec32, d=085e5ab5, e=4b03af04
Étape 38: a=4c986405, b=dc63973f, c=03d447f6, d=f72eec32, e=085e5ab5
Étape 39: a=32de1cba, b=4c986405, c=f718e5cf, d=03d447f6, e=f72eec32
Étape 40: a=fc87dedf, b=32de1cba, c=53261901, d=f718e5cf, e=03d447f6
Étape 41: a=970a0d5c, b=fc87dedf, c=8cb7872e, d=53261901, e=f718e5cf
Étape 42: a=7f193dc5, b=970a0d5c, c=ff21f7b7, d=8cb7872e, e=53261901
Étape 43: a=ee1b1aaf, b=7f193dc5, c=25c28357, d=ff21f7b7, e=8cb7872e
Étape 44: a=40f28e09, b=ee1b1aaf, c=5fc64f71, d=25c28357, e=ff21f7b7
Étape 45: a=1c51e1f2, b=40f28e09, c=fb86c6ab, d=5fc64f71, e=25c28357
Étape 46: a=a01b846c, b=1c51e1f2, c=503ca382, d=fb86c6ab, e=5fc64f71
Étape 47: a=bead02ca, b=a01b846c, c=8714787c, d=503ca382, e=fb86c6ab
Étape 48: a=baf39337, b=bead02ca, c=2806e11b, d=8714787c, e=503ca382
Étape 49: a=120731c5, b=baf39337, c=afab40b2, d=2806e11b, e=8714787c
Étape 50: a=641db2ce, b=120731c5, c=eebce4cd, d=afab40b2, e=2806e11b
Étape 51: a=3847ad66, b=641db2ce, c=4481cc71, d=eebce4cd, e=afab40b2
Étape 52: a=e490436d, b=3847ad66, c=99076cb3, d=4481cc71, e=eebce4cd
Étape 53: a=27e9f1d8, b=e490436d, c=8e11eb59, d=99076cb3, e=4481cc71
Étape 54: a=7b71f76d, b=27e9f1d8, c=792410db, d=8e11eb59, e=99076cb3
Étape 55: a=5e6456af, b=7b71f76d, c=09fa7c76, d=792410db, e=8e11eb59
Étape 56: a=c846093f, b=5e6456af, c=5edc7ddb, d=09fa7c76, e=792410db
Étape 57: a=d262ff50, b=c846093f, c=d79915ab, d=5edc7ddb, e=09fa7c76
Étape 58: a=09d785fd, b=d262ff50, c=f211824f, d=d79915ab, e=5edc7ddb
Étape 59: a=3f52de5a, b=09d785fd, c=3498bfd4, d=f211824f, e=d79915ab
Étape 60: a=d756c147, b=3f52de5a, c=4275e17f, d=3498bfd4, e=f211824f
Étape 61: a=548c9cb2, b=d756c147, c=8fd4b796, d=4275e17f, e=3498bfd4
Étape 62: a=b66c020b, b=548c9cb2, c=f5d5b051, d=8fd4b796, e=4275e17f
Étape 63: a=6b61c9e1, b=b66c020b, c=9523272c, d=f5d5b051, e=8fd4b796
Étape 64: a=19dfa7ac, b=6b61c9e1, c=ed9b0082, d=9523272c, e=f5d5b051
Étape 65: a=101655f9, b=19dfa7ac, c=5ad87278, d=ed9b0082, e=9523272c
Étape 66: a=0c3df2b4, b=101655f9, c=0677e9eb, d=5ad87278, e=ed9b0082
Étape 67: a=78dd4d2b, b=0c3df2b4, c=4405957e, d=0677e9eb, e=5ad87278
Étape 68: a=497093c0, b=78dd4d2b, c=030f7cad, d=4405957e, e=0677e9eb
Étape 69: a=3f2588c2, b=497093c0, c=de37534a, d=030f7cad, e=4405957e
Étape 70: a=c199f8c7, b=3f2588c2, c=125c24f0, d=de37534a, e=030f7cad
Étape 71: a=39859de7, b=c199f8c7, c=8fc96230, d=125c24f0, e=de37534a
Étape 72: a=edb42de4, b=39859de7, c=f0667e31, d=8fc96230, e=125c24f0
Étape 73: a=11793f6f, b=edb42de4, c=ce616779, d=f0667e31, e=8fc96230
Étape 74: a=5ee76897, b=11793f6f, c=3b6d0b79, d=ce616779, e=f0667e31
Étape 75: a=63f7dab7, b=5ee76897, c=c45e4fdb, d=3b6d0b79, e=ce616779
Étape 76: a=a079b7d9, b=63f7dab7, c=d7b9da25, d=c45e4fdb, e=3b6d0b79
Étape 77: a=860d21cc, b=a079b7d9, c=d8fdf6ad, d=d7b9da25, e=c45e4fdb
Étape 78: a=5738d5e1, b=860d21cc, c=681e6df6, d=d8fdf6ad, e=d7b9da25
Étape 79: a=42541b35, b=5738d5e1, c=21834873, d=681e6df6, e=d8fdf6ad
--- Fin du bloc ---
```

Affichage des étapes internes de SHA-256

Version process_block_sha256_verbose() :

[21]:
```python
def process_block_sha256_verbose(block, H):
    W = [int.from_bytes(block[i:i+4], 'big') for i in range(0, 64, 4)]

    for i in range(16, 64):
        s0 = right_rotate(W[i-15], 7) ^ right_rotate(W[i-15], 18) ^ (W[i-15] >>␣
↪3)
        s1 = right_rotate(W[i-2], 17) ^ right_rotate(W[i-2], 19) ^ (W[i-2] >>␣
↪10)
        W.append((W[i-16] + s0 + W[i-7] + s1) & 0xFFFFFFFF)

    a, b, c, d, e, f, g, h = H

    print("\n--- Début du traitement d’un bloc SHA-256 ---")
    for i in range(64):
        S1 = right_rotate(e, 6) ^ right_rotate(e, 11) ^ right_rotate(e, 25)
        ch = (e & f) ^ ((~e) & g)
        temp1 = (h + S1 + ch + K256[i] + W[i]) & 0xFFFFFFFF
        S0 = right_rotate(a, 2) ^ right_rotate(a, 13) ^ right_rotate(a, 22)
        maj = (a & b) ^ (a & c) ^ (b & c)
        temp2 = (S0 + maj) & 0xFFFFFFFF

        h = g
        g = f
        f = e
        e = (d + temp1) & 0xFFFFFFFF
        d = c
        c = b
        b = a
        a = (temp1 + temp2) & 0xFFFFFFFF

        print(f"Étape {i:02}: a={a:08x} b={b:08x} c={c:08x} d={d:08x} e={e:08x}␣
↪f={f:08x} g={g:08x} h={h:08x}")

    H[0] = (H[0] + a) & 0xFFFFFFFF
    H[1] = (H[1] + b) & 0xFFFFFFFF
    H[2] = (H[2] + c) & 0xFFFFFFFF
    H[3] = (H[3] + d) & 0xFFFFFFFF
    H[4] = (H[4] + e) & 0xFFFFFFFF
    H[5] = (H[5] + f) & 0xFFFFFFFF
    H[6] = (H[6] + g) & 0xFFFFFFFF
    H[7] = (H[7] + h) & 0xFFFFFFFF

    print("--- Fin du bloc ---\n")
    return H
```

Fonction sha256_verbose() complète :

[22]:
```
def sha256_verbose(message: bytes) -> str:
    message = pad_message_sha256(message)
    H = H256.copy()

    for i in range(0, len(message), 64):
        block = message[i:i+64]
        H = process_block_sha256_verbose(block, H)

    return ''.join(f'{h:08x}' for h in H)
```

Test :

[23]:
```
if __name__ == "__main__":
    sha256_verbose(b"abc")
```

```
--- Début du traitement d’un bloc SHA-256 ---
Étape 00: a=5d6aebcd b=6a09e667 c=bb67ae85 d=3c6ef372 e=fa2a4622 f=510e527f
g=9b05688c h=1f83d9ab
Étape 01: a=5a6ad9ad b=5d6aebcd c=6a09e667 d=bb67ae85 e=78ce7989 f=fa2a4622
g=510e527f h=9b05688c
Étape 02: a=c8c347a7 b=5a6ad9ad c=5d6aebcd d=6a09e667 e=f92939eb f=78ce7989
g=fa2a4622 h=510e527f
Étape 03: a=d550f666 b=c8c347a7 c=5a6ad9ad d=5d6aebcd e=24e00850 f=f92939eb
g=78ce7989 h=fa2a4622
Étape 04: a=04409a6a b=d550f666 c=c8c347a7 d=5a6ad9ad e=43ada245 f=24e00850
g=f92939eb h=78ce7989
Étape 05: a=2b4209f5 b=04409a6a c=d550f666 d=c8c347a7 e=714260ad f=43ada245
g=24e00850 h=f92939eb
Étape 06: a=e5030380 b=2b4209f5 c=04409a6a d=d550f666 e=9b27a401 f=714260ad
g=43ada245 h=24e00850
Étape 07: a=85a07b5f b=e5030380 c=2b4209f5 d=04409a6a e=0c657a79 f=9b27a401
g=714260ad h=43ada245
Étape 08: a=8e04ecb9 b=85a07b5f c=e5030380 d=2b4209f5 e=32ca2d8c f=0c657a79
g=9b27a401 h=714260ad
Étape 09: a=8c87346b b=8e04ecb9 c=85a07b5f d=e5030380 e=1cc92596 f=32ca2d8c
g=0c657a79 h=9b27a401
Étape 10: a=4798a3f4 b=8c87346b c=8e04ecb9 d=85a07b5f e=436b23e8 f=1cc92596
g=32ca2d8c h=0c657a79
Étape 11: a=f71fc5a9 b=4798a3f4 c=8c87346b d=8e04ecb9 e=816fd6e9 f=436b23e8
g=1cc92596 h=32ca2d8c
Étape 12: a=87912990 b=f71fc5a9 c=4798a3f4 d=8c87346b e=1e578218 f=816fd6e9
g=436b23e8 h=1cc92596
Étape 13: a=d932eb16 b=87912990 c=f71fc5a9 d=4798a3f4 e=745a48de f=1e578218
g=816fd6e9 h=436b23e8
Étape 14: a=c0645fde b=d932eb16 c=87912990 d=f71fc5a9 e=0b92f20c f=745a48de
g=1e578218 h=816fd6e9
```

```
Étape 15: a=b0fa238e b=c0645fde c=d932eb16 d=87912990 e=07590dcd f=0b92f20c
g=745a48de h=1e578218
Étape 16: a=21da9a9b b=b0fa238e c=c0645fde d=d932eb16 e=8034229c f=07590dcd
g=0b92f20c h=745a48de
Étape 17: a=c2fbd9d1 b=21da9a9b c=b0fa238e d=c0645fde e=846ee454 f=8034229c
g=07590dcd h=0b92f20c
Étape 18: a=fe777bbf b=c2fbd9d1 c=21da9a9b d=b0fa238e e=cc899961 f=846ee454
g=8034229c h=07590dcd
Étape 19: a=e1f20c33 b=fe777bbf c=c2fbd9d1 d=21da9a9b e=b0638179 f=cc899961
g=846ee454 h=8034229c
Étape 20: a=9dc68b63 b=e1f20c33 c=fe777bbf d=c2fbd9d1 e=8ada8930 f=b0638179
g=cc899961 h=846ee454
Étape 21: a=c2606d6d b=9dc68b63 c=e1f20c33 d=fe777bbf e=e1257970 f=8ada8930
g=b0638179 h=cc899961
Étape 22: a=a7a3623f b=c2606d6d c=9dc68b63 d=e1f20c33 e=49f5114a f=e1257970
g=8ada8930 h=b0638179
Étape 23: a=c5d53d8d b=a7a3623f c=c2606d6d d=9dc68b63 e=aa47c347 f=49f5114a
g=e1257970 h=8ada8930
Étape 24: a=1c2c2838 b=c5d53d8d c=a7a3623f d=c2606d6d e=2823ef91 f=aa47c347
g=49f5114a h=e1257970
Étape 25: a=cde8037d b=1c2c2838 c=c5d53d8d d=a7a3623f e=14383d8e f=2823ef91
g=aa47c347 h=49f5114a
Étape 26: a=b62ec4bc b=cde8037d c=1c2c2838 d=c5d53d8d e=c74c6516 f=14383d8e
g=2823ef91 h=aa47c347
Étape 27: a=77d37528 b=b62ec4bc c=cde8037d d=1c2c2838 e=edffbff8 f=c74c6516
g=14383d8e h=2823ef91
Étape 28: a=363482c9 b=77d37528 c=b62ec4bc d=cde8037d e=6112a3b7 f=edffbff8
g=c74c6516 h=14383d8e
Étape 29: a=a0060b30 b=363482c9 c=77d37528 d=b62ec4bc e=ade79437 f=6112a3b7
g=edffbff8 h=c74c6516
Étape 30: a=ea992a22 b=a0060b30 c=363482c9 d=77d37528 e=0109ab3a f=ade79437
g=6112a3b7 h=edffbff8
Étape 31: a=73b33bf5 b=ea992a22 c=a0060b30 d=363482c9 e=ba591112 f=0109ab3a
g=ade79437 h=6112a3b7
Étape 32: a=98e12507 b=73b33bf5 c=ea992a22 d=a0060b30 e=9cd9f5f6 f=ba591112
g=0109ab3a h=ade79437
Étape 33: a=fe604df5 b=98e12507 c=73b33bf5 d=ea992a22 e=59249dd3 f=9cd9f5f6
g=ba591112 h=0109ab3a
Étape 34: a=a9a7738c b=fe604df5 c=98e12507 d=73b33bf5 e=085f3833 f=59249dd3
g=9cd9f5f6 h=ba591112
Étape 35: a=65a0cfe4 b=a9a7738c c=fe604df5 d=98e12507 e=f4b002d6 f=085f3833
g=59249dd3 h=9cd9f5f6
Étape 36: a=41a65cb1 b=65a0cfe4 c=a9a7738c d=fe604df5 e=0772a26b f=f4b002d6
g=085f3833 h=59249dd3
Étape 37: a=34df1604 b=41a65cb1 c=65a0cfe4 d=a9a7738c e=a507a53d f=0772a26b
g=f4b002d6 h=085f3833
Étape 38: a=6dc57a8a b=34df1604 c=41a65cb1 d=65a0cfe4 e=f0781bc8 f=a507a53d
g=0772a26b h=f4b002d6
```

```
Étape 39: a=79ea687a b=6dc57a8a c=34df1604 d=41a65cb1 e=1efbc0a0 f=f0781bc8
g=a507a53d h=0772a26b
Étape 40: a=d6670766 b=79ea687a c=6dc57a8a d=34df1604 e=26352d63 f=1efbc0a0
g=f0781bc8 h=a507a53d
Étape 41: a=df46652f b=d6670766 c=79ea687a d=6dc57a8a e=838b2711 f=26352d63
g=1efbc0a0 h=f0781bc8
Étape 42: a=17aa0dfe b=df46652f c=d6670766 d=79ea687a e=decd4715 f=838b2711
g=26352d63 h=1efbc0a0
Étape 43: a=9d4baf93 b=17aa0dfe c=df46652f d=d6670766 e=fda24c2e f=decd4715
g=838b2711 h=26352d63
Étape 44: a=26628815 b=9d4baf93 c=17aa0dfe d=df46652f e=a80f11f0 f=fda24c2e
g=decd4715 h=838b2711
Étape 45: a=72ab4b91 b=26628815 c=9d4baf93 d=17aa0dfe e=b7755da1 f=a80f11f0
g=fda24c2e h=decd4715
Étape 46: a=a14c14b0 b=72ab4b91 c=26628815 d=9d4baf93 e=d57b94a9 f=b7755da1
g=a80f11f0 h=fda24c2e
Étape 47: a=4172328d b=a14c14b0 c=72ab4b91 d=26628815 e=fecf0bc6 f=d57b94a9
g=b7755da1 h=a80f11f0
Étape 48: a=05757ceb b=4172328d c=a14c14b0 d=72ab4b91 e=bd714038 f=fecf0bc6
g=d57b94a9 h=b7755da1
Étape 49: a=f11bfaa8 b=05757ceb c=4172328d d=a14c14b0 e=6e5c390c f=bd714038
g=fecf0bc6 h=d57b94a9
Étape 50: a=7a0508a1 b=f11bfaa8 c=05757ceb d=4172328d e=52f1ccf7 f=6e5c390c
g=bd714038 h=fecf0bc6
Étape 51: a=886e7a22 b=7a0508a1 c=f11bfaa8 d=05757ceb e=49231c1e f=52f1ccf7
g=6e5c390c h=bd714038
Étape 52: a=101fd28f b=886e7a22 c=7a0508a1 d=f11bfaa8 e=529e7d00 f=49231c1e
g=52f1ccf7 h=6e5c390c
Étape 53: a=f5702fdb b=101fd28f c=886e7a22 d=7a0508a1 e=9f4787c3 f=529e7d00
g=49231c1e h=52f1ccf7
Étape 54: a=3ec45cdb b=f5702fdb c=101fd28f d=886e7a22 e=e50e1b4f f=9f4787c3
g=529e7d00 h=49231c1e
Étape 55: a=38cc9913 b=3ec45cdb c=f5702fdb d=101fd28f e=54cb266b f=e50e1b4f
g=9f4787c3 h=529e7d00
Étape 56: a=fcd1887b b=38cc9913 c=3ec45cdb d=f5702fdb e=9b5e906c f=54cb266b
g=e50e1b4f h=9f4787c3
Étape 57: a=c062d46f b=fcd1887b c=38cc9913 d=3ec45cdb e=7e44008e f=9b5e906c
g=54cb266b h=e50e1b4f
Étape 58: a=ffb70472 b=c062d46f c=fcd1887b d=38cc9913 e=6d83bfc6 f=7e44008e
g=9b5e906c h=54cb266b
Étape 59: a=b6ae8fff b=ffb70472 c=c062d46f d=fcd1887b e=b21bad3d f=6d83bfc6
g=7e44008e h=9b5e906c
Étape 60: a=b85e2ce9 b=b6ae8fff c=ffb70472 d=c062d46f e=961f4894 f=b21bad3d
g=6d83bfc6 h=7e44008e
Étape 61: a=04d24d6c b=b85e2ce9 c=b6ae8fff d=ffb70472 e=948d25b6 f=961f4894
g=b21bad3d h=6d83bfc6
Étape 62: a=d39a2165 b=04d24d6c c=b85e2ce9 d=b6ae8fff e=fb121210 f=948d25b6
g=961f4894 h=b21bad3d
```

```
Étape 63: a=506e3058 b=d39a2165 c=04d24d6c d=b85e2ce9 e=5ef50f24 f=fb121210
g=948d25b6 h=961f4894
--- Fin du bloc ---
```