MAT-4902E
PROJECT PRESENTATION

GOLDWASSER-MICALI HOMOMORPHIC ENCRYPTION

Eymen Gül
090190746
Supervisor : Assoc. Dr. Ergün Yaraneri

What is Homomorphic Encryption (HE)?

Homomorphic encryption allows us to perform computations directly on encrypted data. This enables us to make secure calculations without decrypting the data. With the growing popularity of big data and cloud-based services today, the importance of HE is rapidly increasing.

Importance of the Goldwasser-Micali Algorithm

The Goldwasser-Micali (GM) algorithm is pivotal in cryptography as one of the earliest probabilistic public-key encryption methods. Its unique feature of generating varied ciphertexts from identical plaintexts enhances security against chosen plaintext attacks. It was also the first to offer semantic security, ensuring that no plaintext data can be inferred from its ciphertext. This algorithm set a foundation for future encryption methods, significantly impacting modern cryptography.

What was implemented in this project?

All steps of the Goldwasser-Micali algorithm were implemented. Implementation was done on the below topics.

- Legendre Symbol - formula
- Quadratic Residues
- The Miller-Rabin Test
- Prime number generation
- Key Generation
- Encryption
- Decryption

Quadratic Residues

a is a quadratic residue modulo p if for any x there is a solution to;

$x^2 = a \pmod{p}$

Legendre Symbol

$(a/p) = 1$ if a is a quadratic residue mod p

$= -1$ if a is a quadratic non-residue mod p

Formula; $a^{(p-1)/2} \pmod{p}$

Jacobi Symbol

Jacobi Symbol is the generalization of the Legendre symbol.

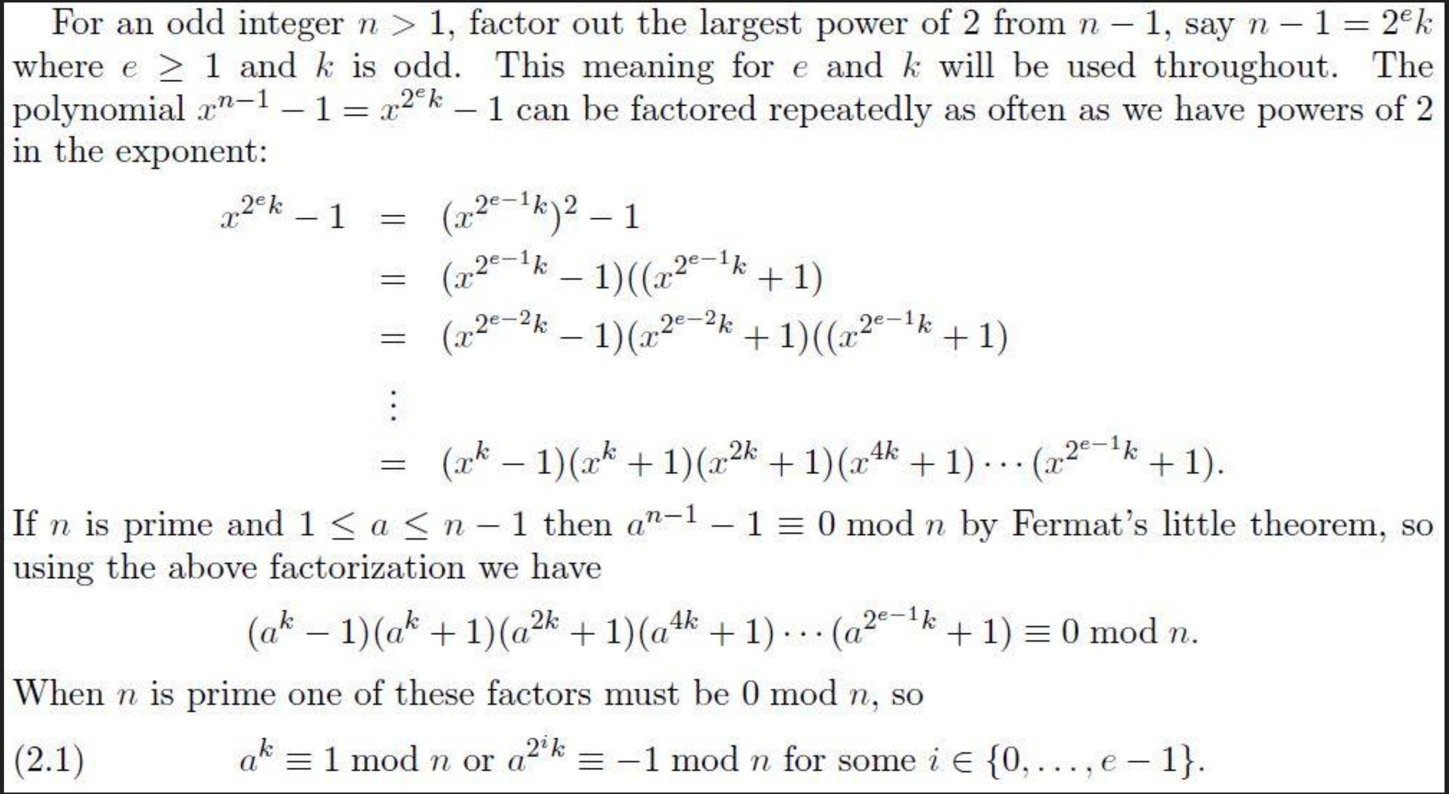
Let $a = c_1 c_2 \dots c_m$

c’s are factors of a.

$(a/p) = (c_1/p)(c_2/p)\dots(c_m/p)$ where c_i/p’s are Legendre Symbol and (a/p) is Jacobi Symbol.

Miller Rabin Test

From Fermat's Little Theorem we know if p is prime, then $a^{n-1} \equiv 1 \pmod{n}$.



For an odd integer $n > 1$, factor out the largest power of 2 from $n-1$, say $n - 1 = 2^e k$ where $e \geq 1$ and k is odd. This meaning for e and k will be used throughout. The polynomial $x^{n-1} - 1 = x^{2^e k} - 1$ can be factored repeatedly as often as we have powers of 2 in the exponent:

$$\begin{aligned} x^{2^e k} - 1 &= (x^{2^{e-1}k})^2 - 1 \\ &= (x^{2^{e-1}k} - 1)((x^{2^{e-1}k} + 1) \\ &= (x^{2^{e-2}k} - 1)(x^{2^{e-2}k} + 1)((x^{2^{e-1}k} + 1) \\ &\vdots \\ &= (x^k - 1)(x^k + 1)(x^{2k} + 1)(x^{4k} + 1) \cdots (x^{2^{e-1}k} + 1). \end{aligned}$$

If n is prime and $1 \leq a \leq n-1$ then $a^{n-1} - 1 \equiv 0 \bmod n$ by Fermat's little theorem, so using the above factorization we have

$$(a^k - 1)(a^k + 1)(a^{2k} + 1)(a^{4k} + 1) \cdots (a^{2^{e-1}k} + 1) \equiv 0 \bmod n.$$

When n is prime one of these factors must be 0 mod n, so

(2.1) $$a^k \equiv 1 \bmod n \text{ or } a^{2^i k} \equiv -1 \bmod n \text{ for some } i \in \{0, \ldots, e-1\}.$$

Goldwasser-Micali Algorithm

1- We choose two large primes (p and q) in Blum integer form. (Blum int. $p \equiv 3 \pmod 4$).

2- We choose a quadratic non-residue $(x/p) = (x/q) = -1$.

3- We compute $p \times q = n$.

Private key is (p,q)

Public key is (n,x).

Goldwasser-Micali Algorithm - Encryption

The plaintext is a string of bits $(m_1,m_2,...,m_k)$. We pick uniformly at random $y_i \in Z^*_n$.

Encryption formula is;

$$c_i \equiv y_i^2 \times x^{m_i} \pmod{n}.$$

The ciphertext generated is $(c_1,c_2,...,c_k)$, such that $c_i \in Z_n$.

Goldwasser-Micali Algorithm - Decryption

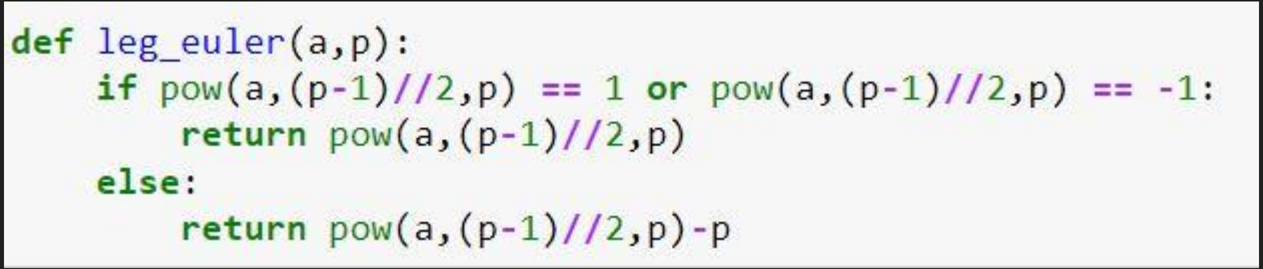
To decrypt the message, we check Jacobi Symbol (c_i/p) and (c_i/q). From formula;

$$c_i \equiv y_i^2 \times x^{m_i} \pmod{n}.$$

while y_i^2 is a quadratic residue mod n, it has no effect on Jacobi Symbol. So we can check directly with Legendre Symbol. x is quadratic non-residue and 1 is quadratic residue. So if mi = 0, (ci/p) and (ci/q) = 1. If mi = 1, (ci/p) and (ci/q) = -1. The receiver can receive decrypted message with private key with checking Legendre Symbol for each c_i.

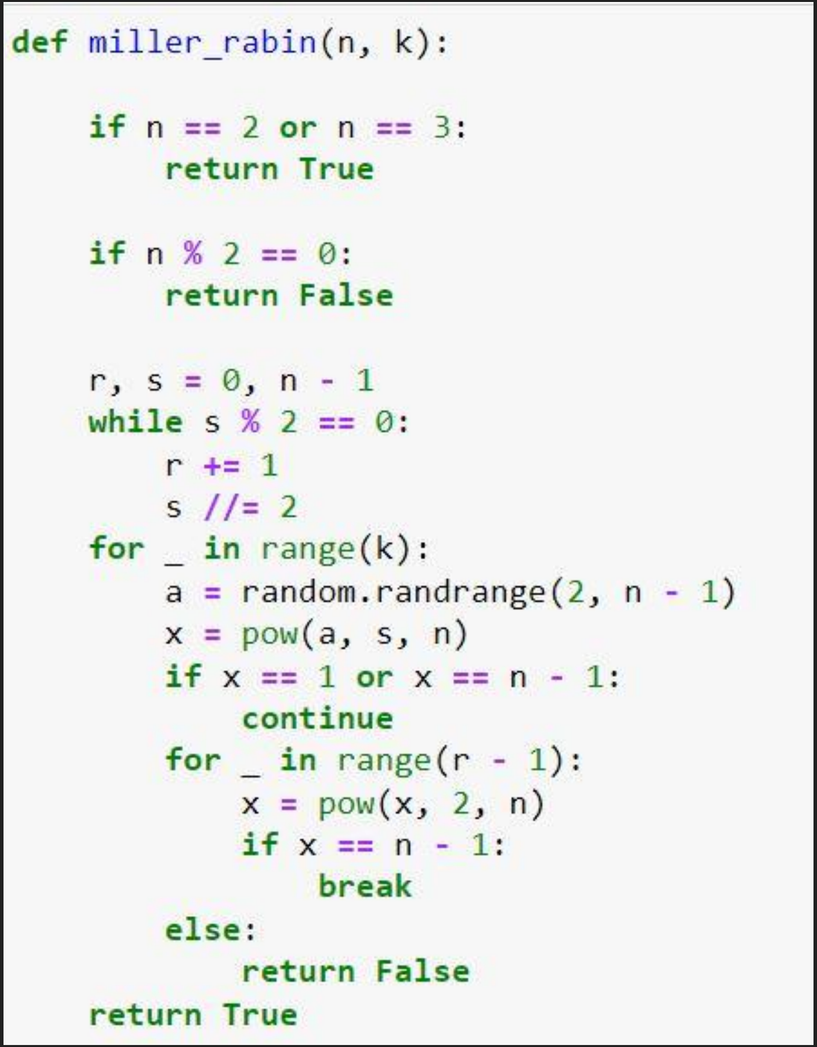
Codes

Legendre Formula



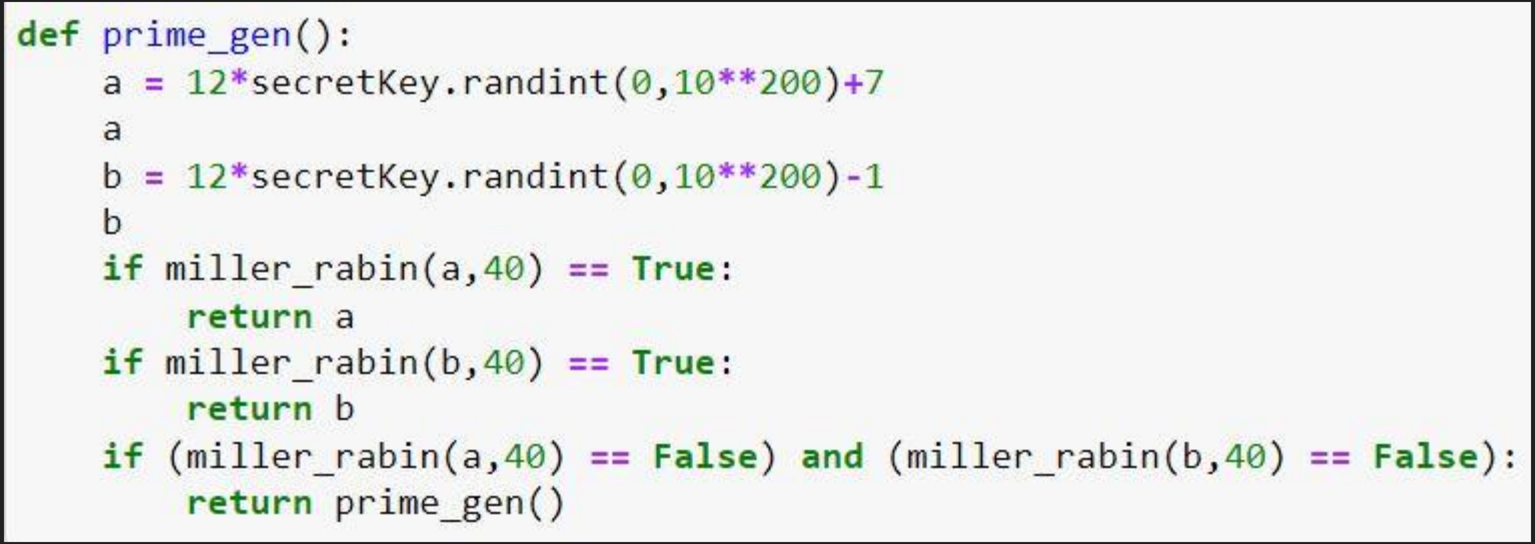
```python
def leg_euler(a,p):
    if pow(a,(p-1)//2,p) == 1 or pow(a,(p-1)//2,p) == -1:
        return pow(a,(p-1)//2,p)
    else:
        return pow(a,(p-1)//2,p)-p
```

Miller Rabin Test Code



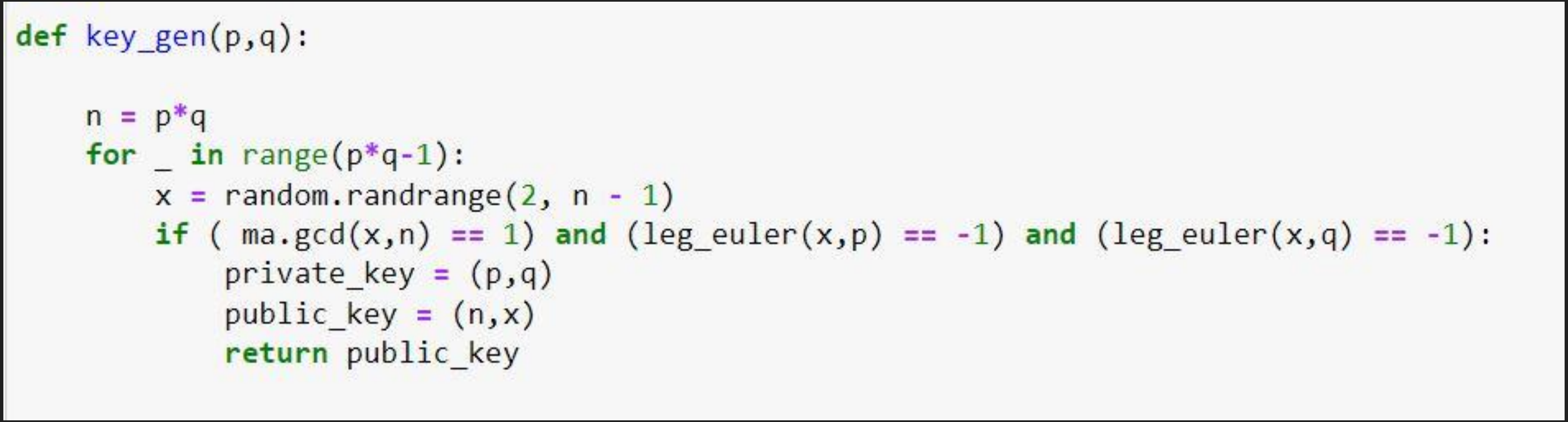
```python
def miller_rabin(n, k):

    if n == 2 or n == 3:
        return True

    if n % 2 == 0:
        return False

    r, s = 0, n - 1
    while s % 2 == 0:
        r += 1
        s //= 2
    for _ in range(k):
        a = random.randrange(2, n - 1)
        x = pow(a, s, n)
        if x == 1 or x == n - 1:
            continue
        for _ in range(r - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False
    return True
```

Prime Generator Function



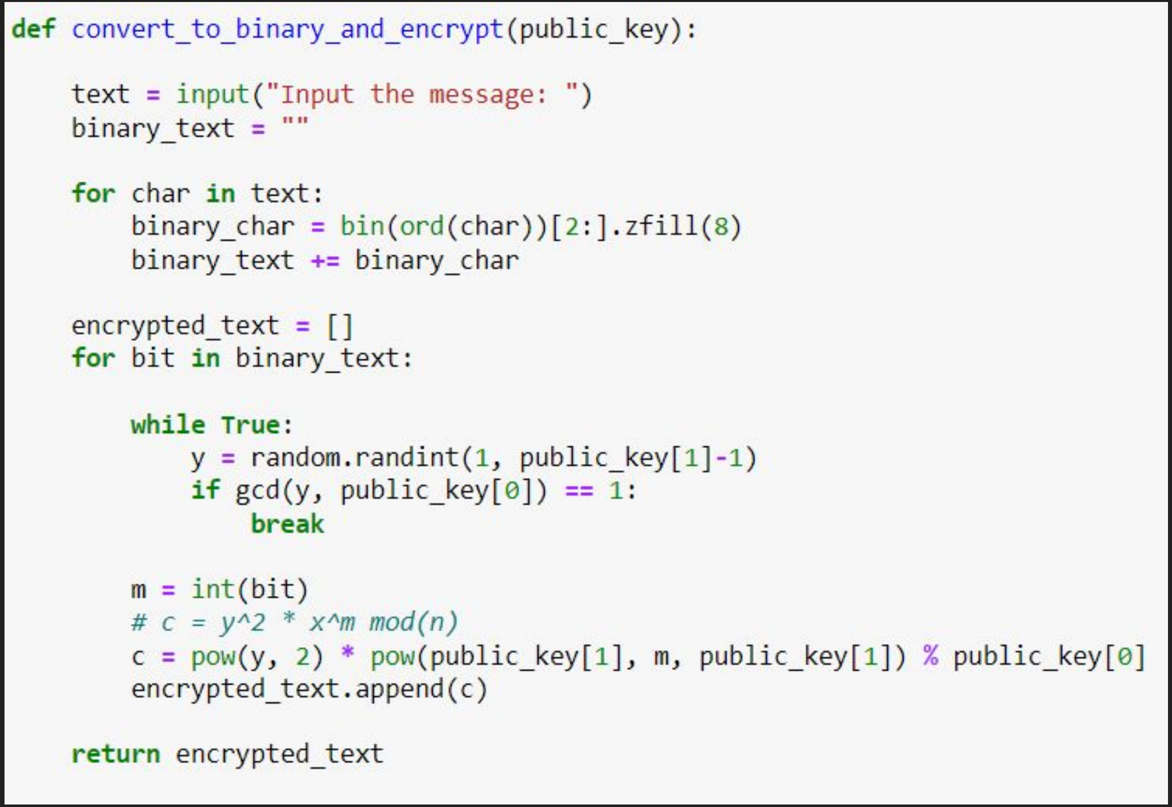
```python
def prime_gen():
    a = 12*secretKey.randint(0,10**200)+7
    a
    b = 12*secretKey.randint(0,10**200)-1
    b
    if miller_rabin(a,40) == True:
        return a
    if miller_rabin(b,40) == True:
        return b
    if (miller_rabin(a,40) == False) and (miller_rabin(b,40) == False):
        return prime_gen()
```

Key Generator Function



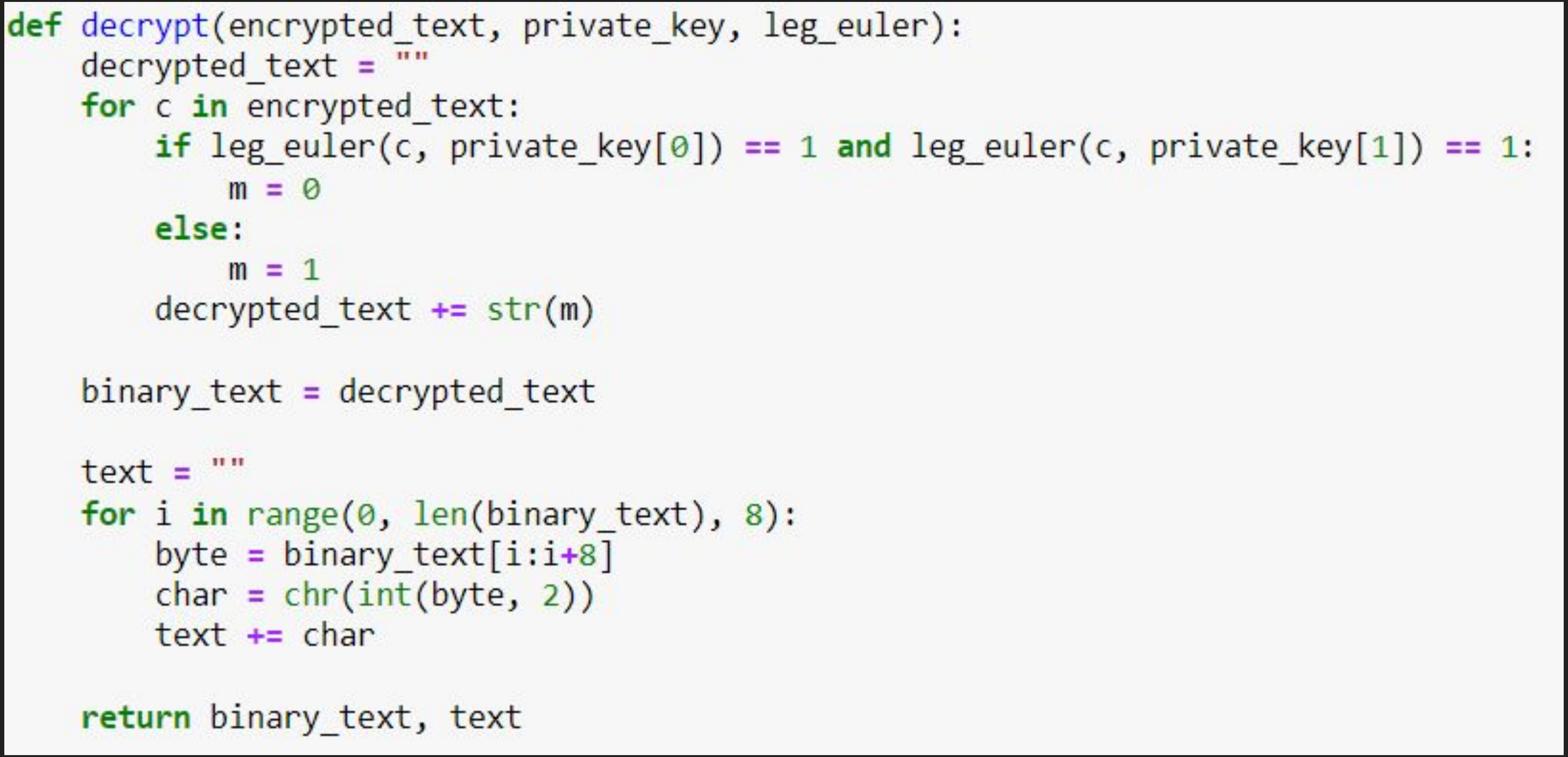
```python
def key_gen(p,q):

    n = p*q
    for _ in range(p*q-1):
        x = random.randrange(2, n - 1)
        if ( ma.gcd(x,n) == 1) and (leg_euler(x,p) == -1) and (leg_euler(x,q) == -1):
            private_key = (p,q)
            public_key = (n,x)
            return public_key
```

Encryption



```python
def convert_to_binary_and_encrypt(public_key):

    text = input("Input the message: ")
    binary_text = ""

    for char in text:
        binary_char = bin(ord(char))[2:].zfill(8)
        binary_text += binary_char

    encrypted_text = []
    for bit in binary_text:

        while True:
            y = random.randint(1, public_key[1]-1)
            if gcd(y, public_key[0]) == 1:
                break

        m = int(bit)
        # c = y^2 * x^m mod(n)
        c = pow(y, 2) * pow(public_key[1], m, public_key[1]) % public_key[0]
        encrypted_text.append(c)

    return encrypted_text
```

Decryption



```python
def decrypt(encrypted_text, private_key, leg_euler):
    decrypted_text = ""
    for c in encrypted_text:
        if leg_euler(c, private_key[0]) == 1 and leg_euler(c, private_key[1]) == 1:
            m = 0
        else:
            m = 1
        decrypted_text += str(m)

    binary_text = decrypted_text

    text = ""
    for i in range(0, len(binary_text), 8):
        byte = binary_text[i:i+8]
        char = chr(int(byte, 2))
        text += char

    return binary_text, text
```

Conclusion

The Goldwasser-Micali (GM) algorithm has brought about a significant revolution in the field of cryptography by introducing semantic security for the first time. It enhances security by producing a different ciphertext with each encryption operation and provides increased protection against chosen plaintext attacks. However, an issue with the GM algorithm is that the ciphertext can be quite large, especially when encrypting large data sets, bringing along challenges related to storage and data transfer. Despite this, the GM algorithm is a fundamental building block in cryptography and has paved the way for the development of modern encryption methods.