Implementation of Paillier Cryptosystem in Python

Abstract— This report presents a simple implementation of the Paillier cryptosystem in Python. We detail the algorithm's theoretical foundations and practical considerations, including dependency management and the cryptographic process. The Paillier cryptosystem is a probabilistic asymmetric algorithm for public-key cryptography. We provide a comprehensive overview of its encryption and decryption processes, key generation, and the mathematical principles underlying its security.

1. Introduction

The Paillier cryptosystem, introduced by Pascal Paillier in 1999, is renowned for its homomorphic properties, allowing specific algebraic operations to be carried out on ciphertexts and yielding results equivalent to those operations performed on the plaintext. This paper outlines a Python implementation of the Paillier cryptosystem.

2. Theoretical Background

The Paillier cryptosystem relies on the difficulty of solving composite residuosity classes. Its security is based on the decisional composite residuosity assumption (DCRA). Key aspects include:
- *Key Generation*: Involves selecting two large prime numbers and computing their product.
- *Encryption*: Uses randomization to ensure that identical plaintexts encrypt to different ciphertexts.
- *Decryption*: Retrieves the original plaintext from the ciphertext using a private key.

3. Implementation Details

3.1 Dependencies and Requirements

The implementation requires Python3 and the following libraries:
- `socket`: Provides a low-level networking interface for communication between computers over a network.
- `pickle`: Serializes and deserializes Python object structures for data persistence and transmission.
- `crypto`: Implements cryptographic algorithms and protocols for secure communication and data protection.
- `sympy`: Facilitates symbolic mathematics, allowing algebraic computations and manipulations.
- `random`: A built-in module that generates pseudo-random numbers for various applications including simulations and security.

A **`requirements.txt`** file is included for easy installation:

```
socket==3.10.0
```

```
pickle==4.0
pycryptodome==3.11.0
sympy==1.9
```

3.2 Code Implementation

Key Generation:

```
def key_generation(self, bit_length):
    self.p_bit_length = bit_length
    self.q_bit_length = bit_length

    p = number.getPrime(bit_length)
    q = number.getPrime(bit_length)

    n = p * q
    g = n + 1

    #lambda_ = math.lcm(p - 1, q - 1)
    lambda_ = int(totient(n))

    mu = mod_inverse(lambda_, n)

    self.public_key = (n, g)
    self.private_key = (lambda_, mu)
    return self.public_key, self.private_key
```

Encryption:

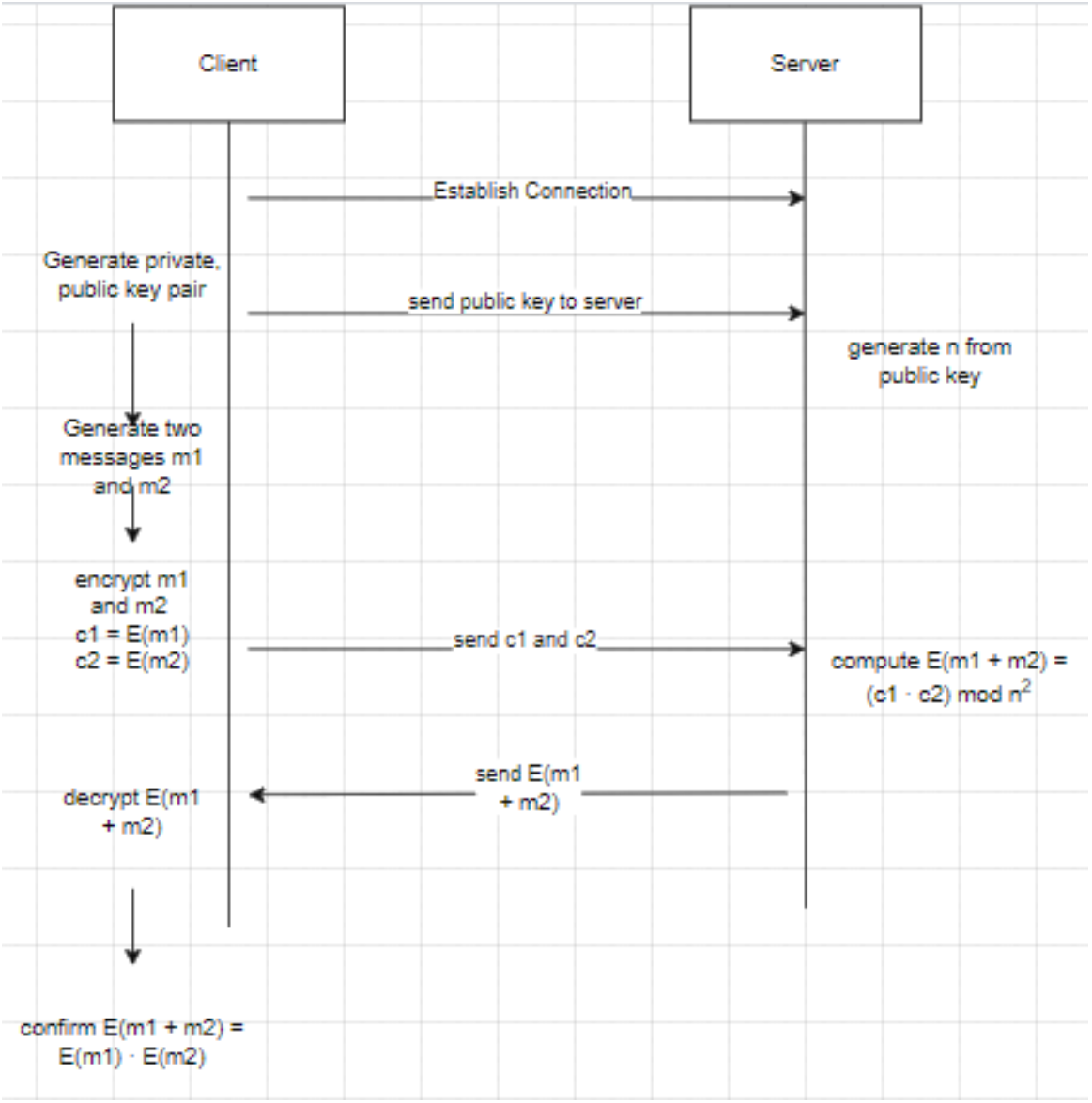
```
def encrypt(self, public_key, m):
    n, g = public_key
    nsq = n * n
    while True:
        r = random.randint(1, n - 1)
        if gcd(r, n) == 1:
            break
    c = (pow(g, m, nsq) * pow(r, n, nsq)) % nsq
    return c
```

Decryption:

```
def decrypt(self, public_key, private_key, c):
    n, g = public_key
    nsq = n * n
    lambda_val, mu = private_key
    x = pow(c, lambda_val, nsq)
    l = (x - 1) // n
    m = (l * mu) % n
    return m
```

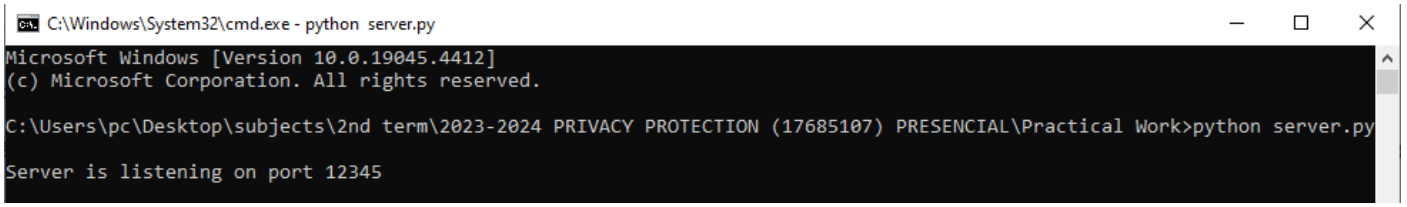
4. Demo

1) Client generates public and private keys of Paillier Cryptosystem.
2) Client sends the public key (public parameters) to the server.
3) Client encrypts the messages m1=1000 and m2=2000 and encrypts them using a public key and sends the respective ciphertexts c1 and c2 to the server.
4) Server computes (c1 . c2) mod n^2
5) Server sends the result back to the client.
6) Client decrypts the received result using its private key.
7) Client Confirms the result = 3000 which is the sum of m1 and m2.

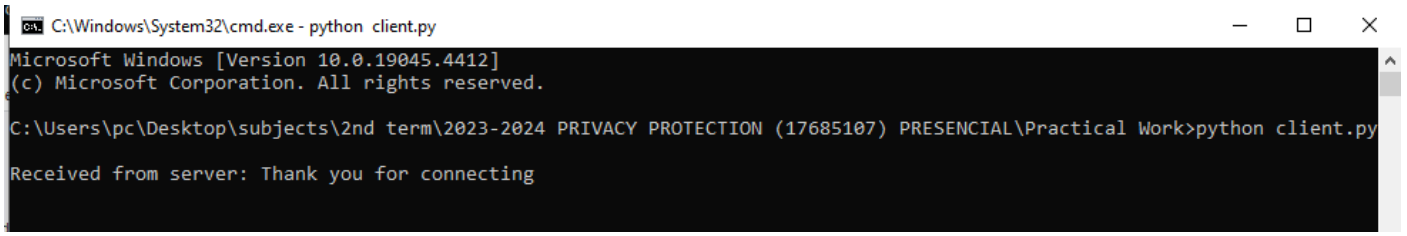


5. Example Usage

 # 1 Start the server by running `python server.py`



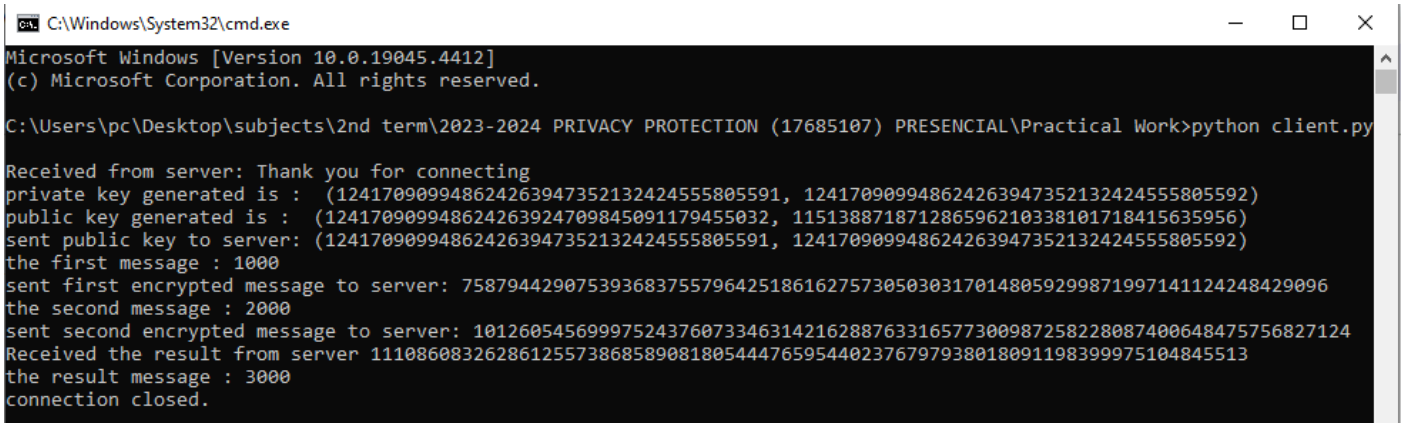
 # 2 Start the client by running `python client.py`



 # 3 Note the output in both windows

Server

Client



6. Conclusion

This report outlines the theoretical background and practical implementation of the Paillier cryptosystem in Python. The implementation demonstrates the cryptosystem's probabilistic nature and homomorphic properties, providing a foundational understanding for further exploration in cryptographic applications.

7. References

- Paillier, P. (1999). Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. Eurocrypt.
- Wikipedia contributors. (2023). Paillier cryptosystem. Wikipedia, The Free Encyclopedia.