Configuration Manual

MSc Research Project

Master of Science in Cyber Security

Sudhanshu
Student ID: 22219471

School of Computing

National College of Ireland

Supervisor: Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sudhanshu		
Student ID:	22219471		
Programme:	Master of Science in Cyber Security	**Year:**	2023-24
Module:	MSc Research Practicum		
Lecturer:	Jawad Salahuddin		
Submission Due Date:	12th August 2024 14:00.		
Project Title:	Quantum encryption to tackle brute force attacks from quantum encryption		
Word Count:	344		
Page Count:	3		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Sudhanshu

Date: 12th August 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sudhanshu
22219471

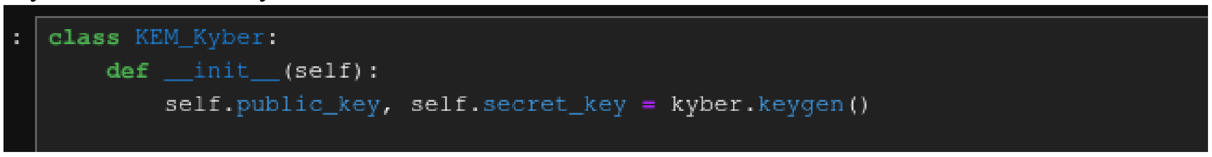
1 Introduction

In the process of finding out the best quantum encryption available for IoT devices that can with stand brute force attacks. This configuration file is what we have discovered yet. This configuration will be divided into 3 parts. First will be dedicated to Key encapsulation Mechanism (KEM) that we have used KYPER-512 and second part will be dedicated to AES-256. LAstly the third and final section will cover how they both work together.

2 KYBER-512

We have used a library called ‘kyber’ in executing this KEM. The basic working of KEM is divided into 3 parts:

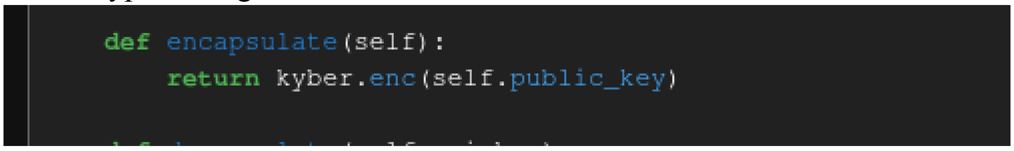
1. *Key generation:* Since KEM is a Asymmetric encryption it need to generate a Public key and a Private key.



```
class KEM_Kyber:
    def __init__(self):
        self.public_key, self.secret_key = kyber.keygen()
```

Fig 2.1: Key generation

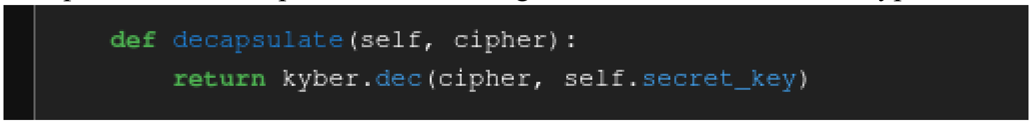
2. *Encapsulation:* Once we have a private and public key we have to make a secret key and encrypt it using KYBER-512



```
    def encapsulate(self):
        return kyber.enc(self.public_key)
```

Fig 2.2: Encapsulate

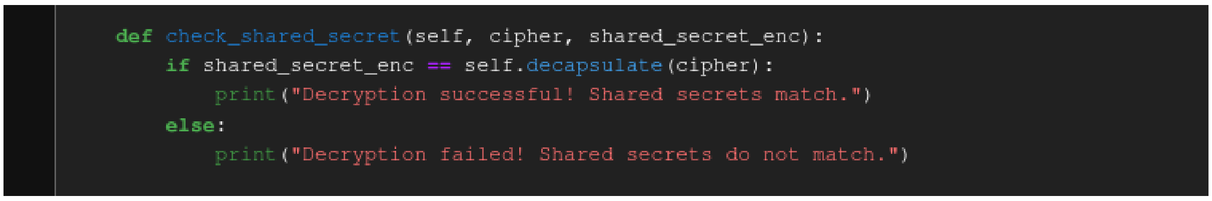
3. *Decapsulation:* Final process is receiving the shared secret and decrypt it.



```
    def decapsulate(self, cipher):
        return kyber.dec(cipher, self.secret_key)
```

Fig 2.3 Decapsulate

- We also created a function to check if the shared secret is correct or not.



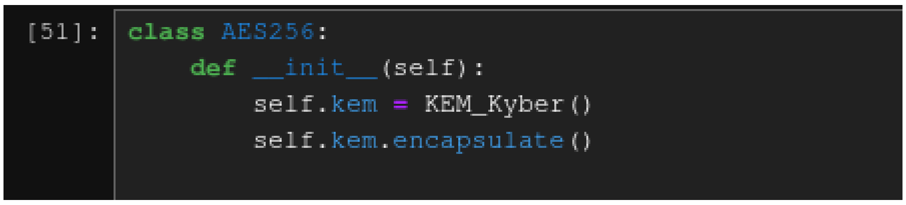
```
def check_shared_secret(self, cipher, shared_secret_enc):
    if shared_secret_enc == self.decapsulate(cipher):
        print("Decryption successful! Shared secrets match.")
    else:
        print("Decryption failed! Shared secrets do not match.")
```

Fig 2.4: Secret Check

- With no context this does not seem much. But when we implement AES we will see how important it is.

3 AES-256

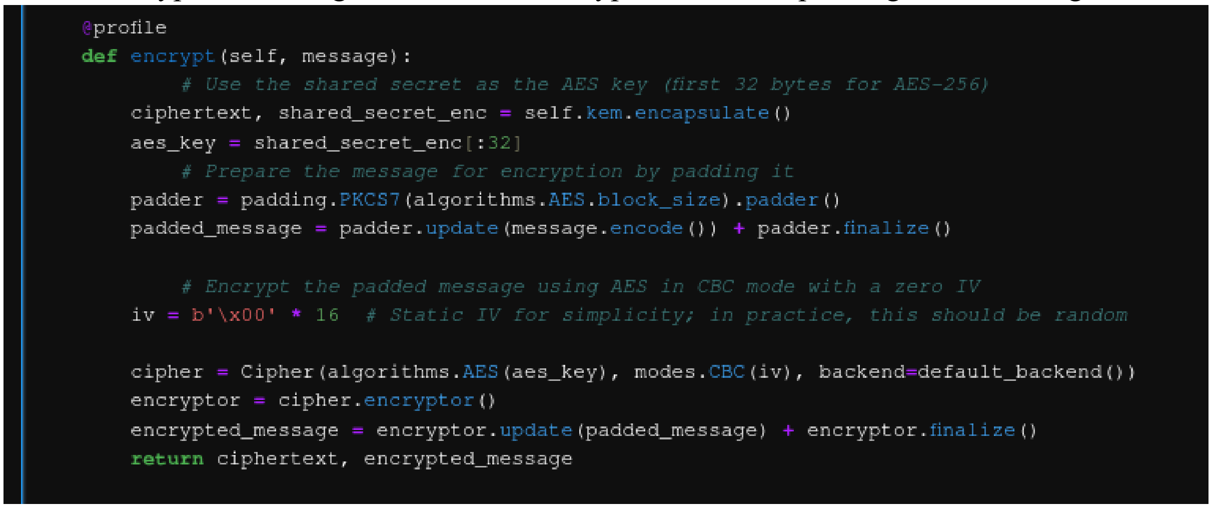
In AES we first need the key and for that we have to invoke the Kyber to exchange the key.



```
[51]: class AES256:
          def __init__(self):
              self.kem = KEM_Kyber()
              self.kem.encapsulate()
```

Fig3.1: KEM initialise

We will now have to get the shared_ssecret and the ciphertext from Kyber encapsulate. This will we used in encryption. We also use first 32 bits of shared key as our aes_key. With which we will encrypt the message. Before final encryption wee add padding to the message.



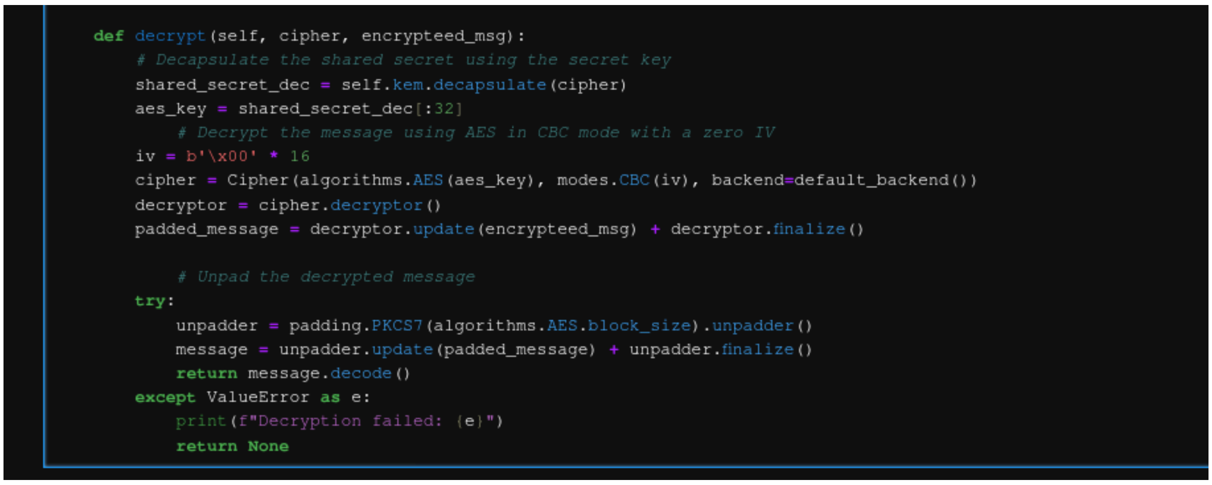
```
@profile
def encrypt(self, message):
        # Use the shared secret as the AES key (first 32 bytes for AES-256)
    ciphertext, shared_secret_enc = self.kem.encapsulate()
    aes_key = shared_secret_enc[:32]
        # Prepare the message for encryption by padding it
    padder = padding.PKCS7(algorithms.AES.block_size).padder()
    padded_message = padder.update(message.encode()) + padder.finalize()

        # Encrypt the padded message using AES in CBC mode with a zero IV
    iv = b'\x00' * 16  # Static IV for simplicity; in practice, this should be random

    cipher = Cipher(algorithms.AES(aes_key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    encrypted_message = encryptor.update(padded_message) + encryptor.finalize()
    return ciphertext, encrypted_message
```

Fig 3.2: AES encryption

once Encrypted we also have to make a method to decrypt.For that we will have to decapsulate the shared_secret using the cipher text. Once we get the shared secret we will use the first 32 bits of that as our aes_key like we did during encryption. Wee will decrypt the message and then decrypt the message.

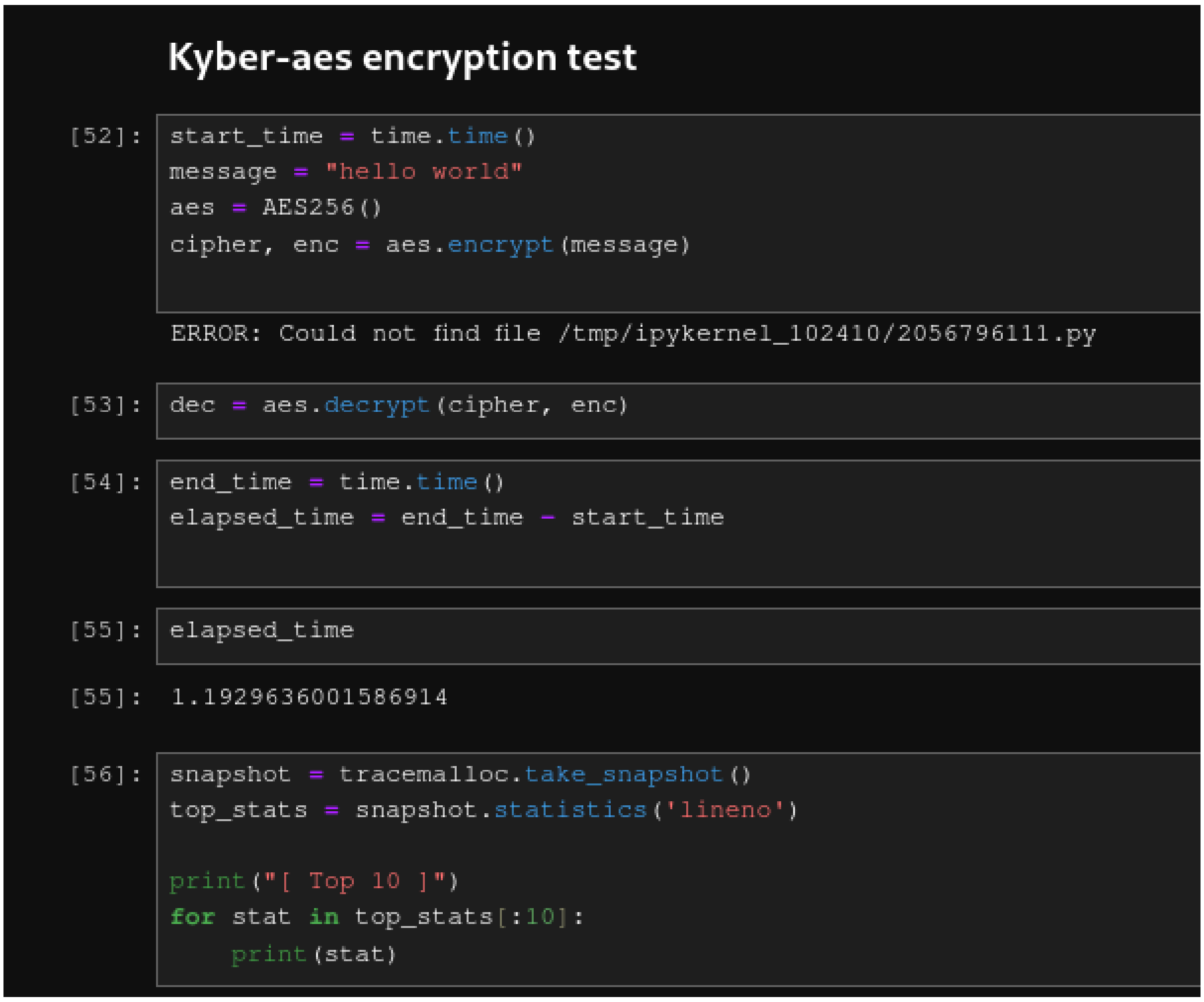


```python
def decrypt(self, cipher, encrypteed_msg):
    # Decapsulate the shared secret using the secret key
    shared_secret_dec = self.kem.decapsulate(cipher)
    aes_key = shared_secret_dec[:32]
        # Decrypt the message using AES in CBC mode with a zero IV
    iv = b'\x00' * 16
    cipher = Cipher(algorithms.AES(aes_key), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    padded_message = decryptor.update(encrypteed_msg) + decryptor.finalize()

        # Unpad the decrypted message
    try:
        unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
        message = unpadder.update(padded_message) + unpadder.finalize()
        return message.decode()
    except ValueError as e:
        print(f"Decryption failed: {e}")
        return None
```

Fig 3.3: Decryption

4 Working



Kyber-aes encryption test

```
[52]: start_time = time.time()
      message = "hello world"
      aes = AES256()
      cipher, enc = aes.encrypt(message)

      ERROR: Could not find file /tmp/ipykernel_102410/2056796111.py

[53]: dec = aes.decrypt(cipher, enc)

[54]: end_time = time.time()
      elapsed_time = end_time - start_time

[55]: elapsed_time

[55]: 1.1929636001586914

[56]: snapshot = tracemalloc.take_snapshot()
      top_stats = snapshot.statistics('lineno')

      print("[ Top 10 ]")
      for stat in top_stats[:10]:
          print(stat)
```

Fig 4.1: Testing