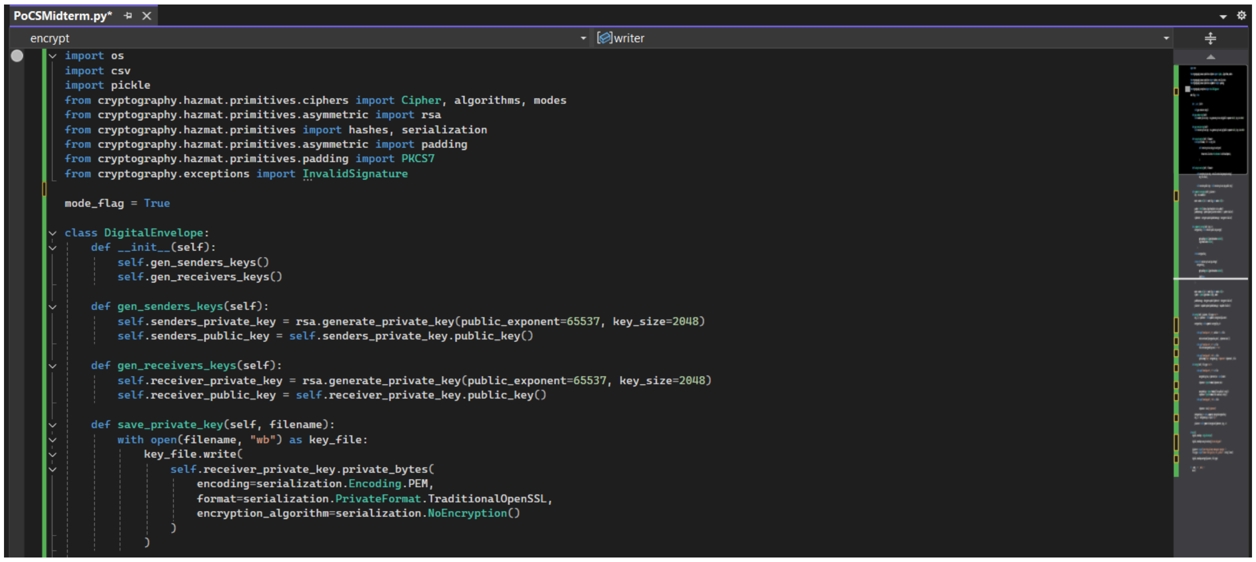
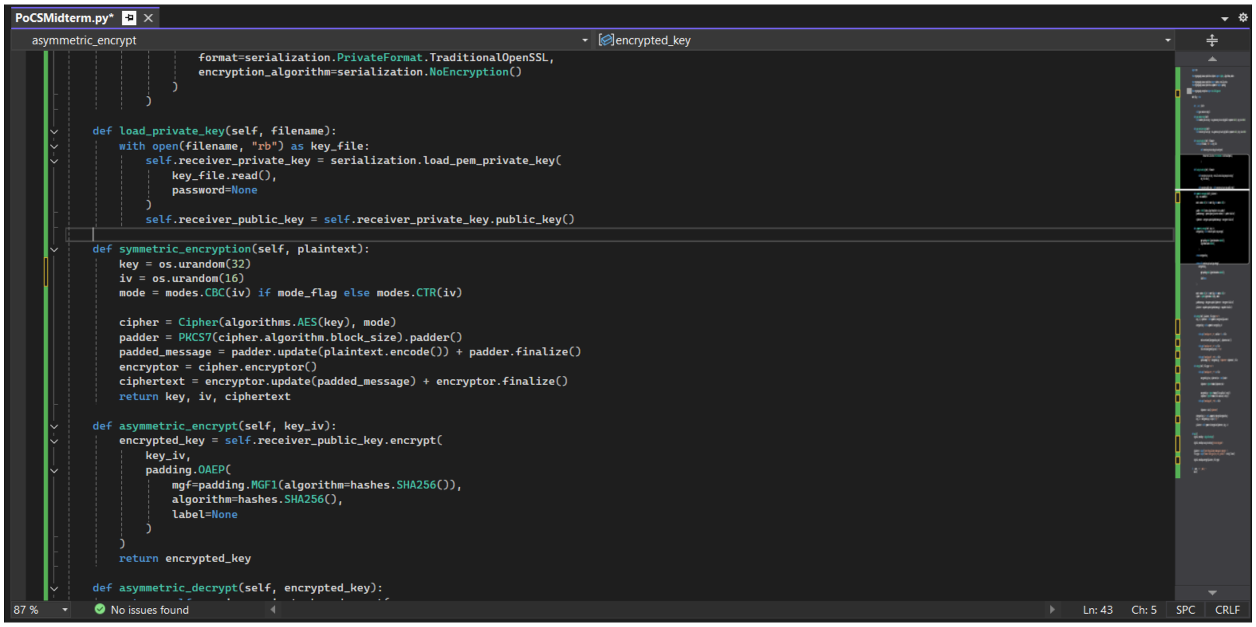
Jack Lin

# Midterm Coding



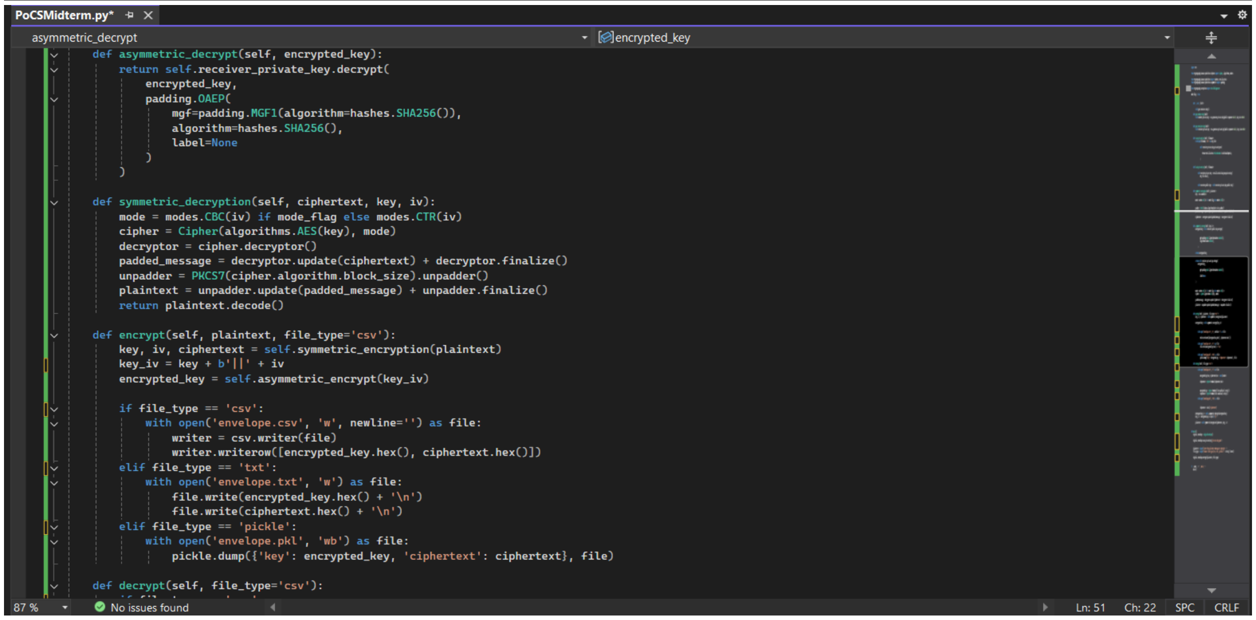
```python
import os
import csv
import pickle
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives.padding import PKCS7
from cryptography.exceptions import InvalidSignature

mode_flag = True

class DigitalEnvelope:
    def __init__(self):
        self.gen_senders_keys()
        self.gen_receivers_keys()

    def gen_senders_keys(self):
        self.senders_private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
        self.senders_public_key = self.senders_private_key.public_key()

    def gen_receivers_keys(self):
        self.receiver_private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
        self.receiver_public_key = self.receiver_private_key.public_key()

    def save_private_key(self, filename):
        with open(filename, "wb") as key_file:
            key_file.write(
                self.receiver_private_key.private_bytes(
                    encoding=serialization.Encoding.PEM,
                    format=serialization.PrivateFormat.TraditionalOpenSSL,
                    encryption_algorithm=serialization.NoEncryption()
                )
            )
```


PoCSMidterm.py*

asymmetric_encrypt | encrypted_key

```python
                    format=serialization.PrivateFormat.TraditionalOpenSSL,
                    encryption_algorithm=serialization.NoEncryption()
                )
            )

    def load_private_key(self, filename):
        with open(filename, "rb") as key_file:
            self.receiver_private_key = serialization.load_pem_private_key(
                key_file.read(),
                password=None
            )
            self.receiver_public_key = self.receiver_private_key.public_key()

    def symmetric_encryption(self, plaintext):
        key = os.urandom(32)
        iv = os.urandom(16)
        mode = modes.CBC(iv) if mode_flag else modes.CTR(iv)

        cipher = Cipher(algorithms.AES(key), mode)
        padder = PKCS7(cipher.algorithm.block_size).padder()
        padded_message = padder.update(plaintext.encode()) + padder.finalize()
        encryptor = cipher.encryptor()
        ciphertext = encryptor.update(padded_message) + encryptor.finalize()
        return key, iv, ciphertext

    def asymmetric_encrypt(self, key_iv):
        encrypted_key = self.receiver_public_key.encrypt(
            key_iv,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )
        return encrypted_key

    def asymmetric_decrypt(self, encrypted_key):
```

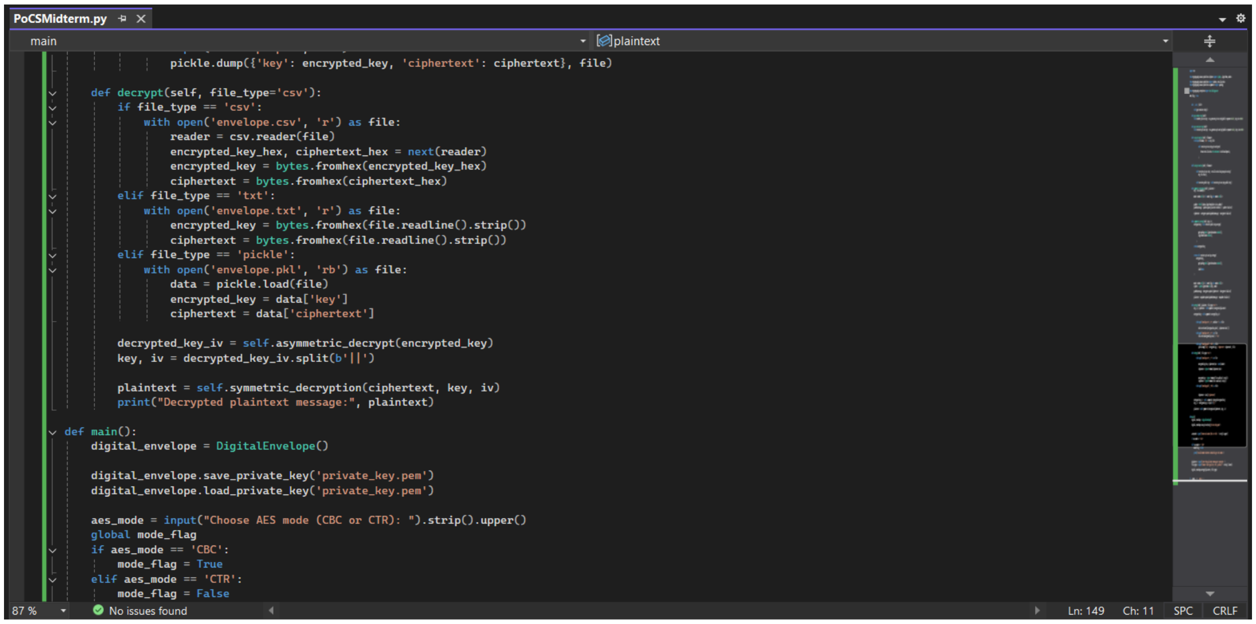
87 % No issues found Ln: 43 Ch: 5 SPC CRLF


PoCSMidterm.py*

asymmetric_decrypt | encrypted_key

```python
    def asymmetric_decrypt(self, encrypted_key):
        return self.receiver_private_key.decrypt(
            encrypted_key,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )

    def symmetric_decryption(self, ciphertext, key, iv):
        mode = modes.CBC(iv) if mode_flag else modes.CTR(iv)
        cipher = Cipher(algorithms.AES(key), mode)
        decryptor = cipher.decryptor()
        padded_message = decryptor.update(ciphertext) + decryptor.finalize()
        unpadder = PKCS7(cipher.algorithm.block_size).unpadder()
        plaintext = unpadder.update(padded_message) + unpadder.finalize()
        return plaintext.decode()

    def encrypt(self, plaintext, file_type='csv'):
        key, iv, ciphertext = self.symmetric_encryption(plaintext)
        key_iv = key + b'||' + iv
        encrypted_key = self.asymmetric_encrypt(key_iv)

        if file_type == 'csv':
            with open('envelope.csv', 'w', newline='') as file:
                writer = csv.writer(file)
                writer.writerow([encrypted_key.hex(), ciphertext.hex()])
        elif file_type == 'txt':
            with open('envelope.txt', 'w') as file:
                file.write(encrypted_key.hex() + '\n')
                file.write(ciphertext.hex() + '\n')
        elif file_type == 'pickle':
            with open('envelope.pkl', 'wb') as file:
                pickle.dump({'key': encrypted_key, 'ciphertext': ciphertext}, file)

    def decrypt(self, file_type='csv'):
```

87 % No issues found Ln: 51 Ch: 22 SPC CRLF

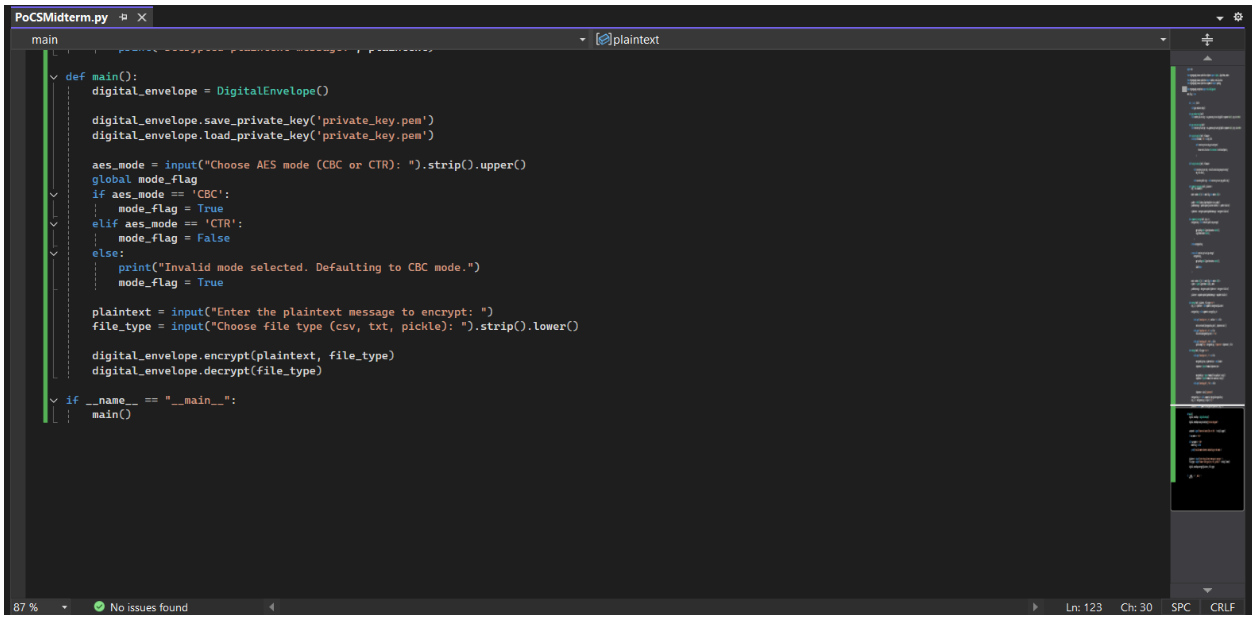


PoCSMidterm.py

main | plaintext

```python
                pickle.dump({'key': encrypted_key, 'ciphertext': ciphertext}, file)

    def decrypt(self, file_type='csv'):
        if file_type == 'csv':
            with open('envelope.csv', 'r') as file:
                reader = csv.reader(file)
                encrypted_key_hex, ciphertext_hex = next(reader)
                encrypted_key = bytes.fromhex(encrypted_key_hex)
                ciphertext = bytes.fromhex(ciphertext_hex)
        elif file_type == 'txt':
            with open('envelope.txt', 'r') as file:
                encrypted_key = bytes.fromhex(file.readline().strip())
                ciphertext = bytes.fromhex(file.readline().strip())
        elif file_type == 'pickle':
            with open('envelope.pkl', 'rb') as file:
                data = pickle.load(file)
                encrypted_key = data['key']
                ciphertext = data['ciphertext']

        decrypted_key_iv = self.asymmetric_decrypt(encrypted_key)
        key, iv = decrypted_key_iv.split(b'||')

        plaintext = self.symmetric_decryption(ciphertext, key, iv)
        print("Decrypted plaintext message:", plaintext)

def main():
    digital_envelope = DigitalEnvelope()

    digital_envelope.save_private_key('private_key.pem')
    digital_envelope.load_private_key('private_key.pem')

    aes_mode = input("Choose AES mode (CBC or CTR): ").strip().upper()
    global mode_flag
    if aes_mode == 'CBC':
        mode_flag = True
    elif aes_mode == 'CTR':
        mode_flag = False
```

87 % | No issues found | Ln: 149 Ch: 11 SPC CRLF

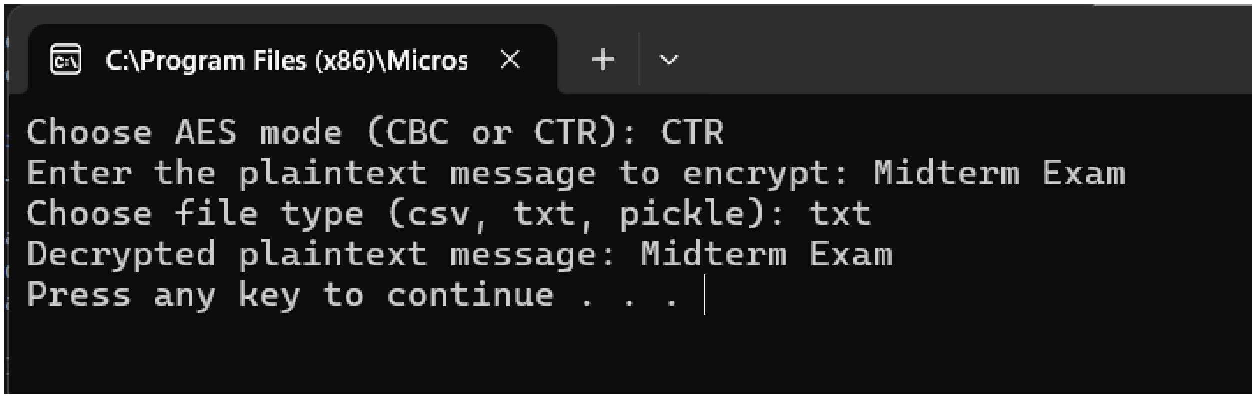


PoCSMidterm.py

main | plaintext

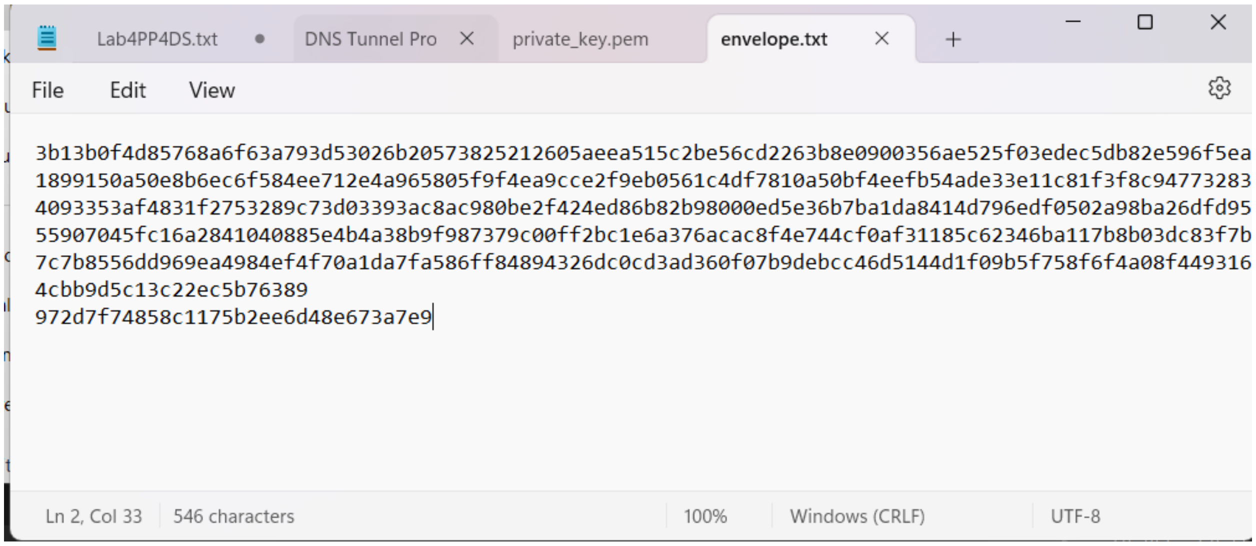
```python
def main():
    digital_envelope = DigitalEnvelope()

    digital_envelope.save_private_key('private_key.pem')
    digital_envelope.load_private_key('private_key.pem')

    aes_mode = input("Choose AES mode (CBC or CTR): ").strip().upper()
    global mode_flag
    if aes_mode == 'CBC':
        mode_flag = True
    elif aes_mode == 'CTR':
        mode_flag = False
    else:
        print("Invalid mode selected. Defaulting to CBC mode.")
        mode_flag = True

    plaintext = input("Enter the plaintext message to encrypt: ")
    file_type = input("Choose file type (csv, txt, pickle): ").strip().lower()

    digital_envelope.encrypt(plaintext, file_type)
    digital_envelope.decrypt(file_type)

if __name__ == "__main__":
    main()
```

87 % | No issues found | Ln: 123 Ch: 30 SPC CRLF



C:\Program Files (x86)\Micros

```
Choose AES mode (CBC or CTR): CTR
Enter the plaintext message to encrypt: Midterm Exam
Choose file type (csv, txt, pickle): txt
Decrypted plaintext message: Midterm Exam
Press any key to continue . . .
```



```
3b13b0f4d85768a6f63a793d53026b20573825212605aeea515c2be56cd2263b8e0900356ae525f03edec5db82e596f5ea
1899150a50e8b6ec6f584ee712e4a965805f9f4ea9cce2f9eb0561c4df7810a50bf4eefb54ade33e11c81f3f8c94773283
4093353af4831f2753289c73d03393ac8ac980be2f424ed86b82b98000ed5e36b7ba1da8414d796edf0502a98ba26dfd95
55907045fc16a2841040885e4b4a38b9f987379c00ff2bc1e6a376acac8f4e744cf0af31185c62346ba117b8b03dc83f7b
7c7b8556dd969ea4984ef4f70a1da7fa586ff84894326dc0cd3ad360f07b9debcc46d5144d1f09b5f758f6f4a08f449316
4cbb9d5c13c22ec5b76389
972d7f74858c1175b2ee6d48e673a7e9
```