Provision of Prime Number and Primitive Root. For solving this project, we will provide a prime number p and α, a primitive root of p.

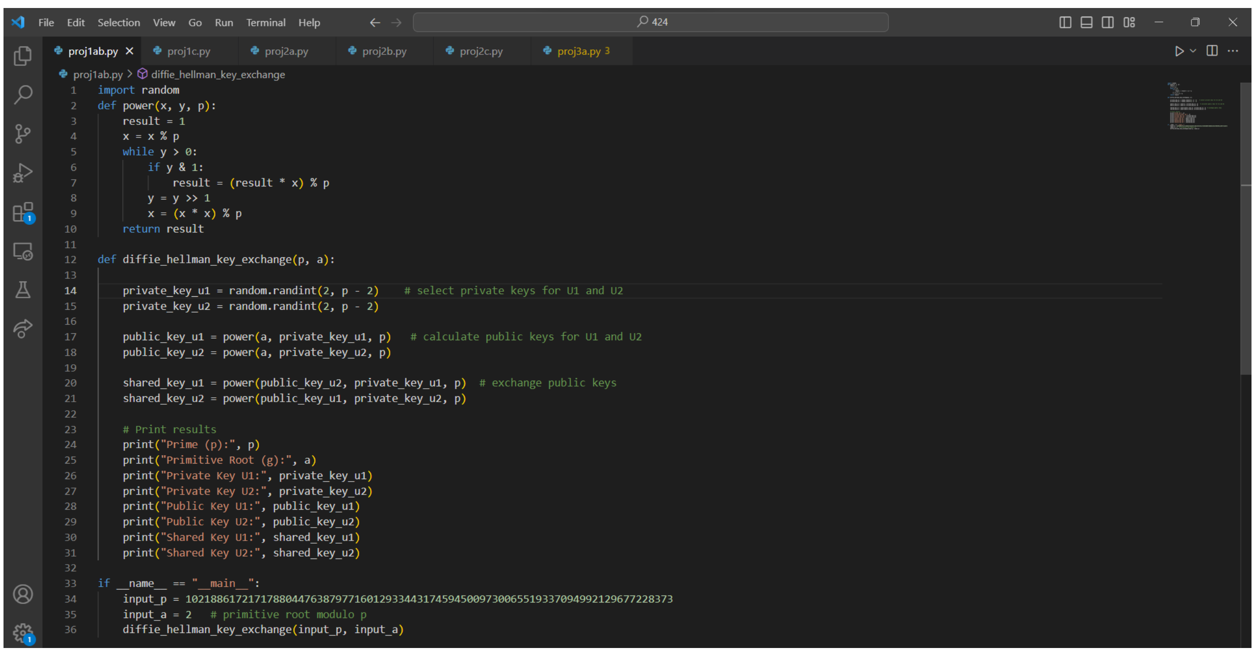
1. ***Implementing a Secure Key Exchange Application.*** **Given that the channel between U1 and U2 is not secure, to start communicating their messages, first they need to establish a shared key.**

(a) Implements the Diffie-Hellman algorithm.

To execute the Diffe-Hellman algorithm it is necessary that both U1 and U2 publicly agree on a large prime number "p" and primitive root modulo p known as "alpha". U1 will choose a secret random integer and compute A-alpha^{x} mod(p), which is then sent to U2. U2 will also choose a secret random integer y and compute B-alpha^{y} mod(p) to send to U2. Then both U1 and U2 will compute the secret key K-B^{X} mod(p).

To implement the Diffie-Hellman algorithm I imputed the prime number and alpha value provided. Then selected private keys for U1 and U2 using the randint function. To calculate the public keys for U1 and U2 I wrote a "power" function that calculates (x^{y}) mod p.

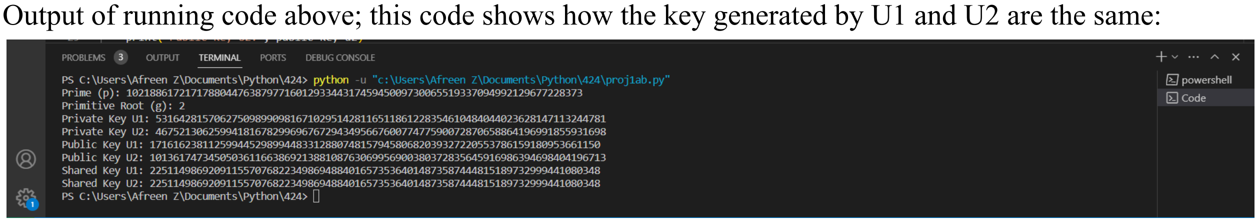
Code that implements the Diffie-Hellman algorithm:



```python
import random
def power(x, y, p):
    result = 1
    x = x % p
    while y > 0:
        if y & 1:
            result = (result * x) % p
        y = y >> 1
        x = (x * x) % p
    return result

def diffie_hellman_key_exchange(p, a):

    private_key_u1 = random.randint(2, p - 2)    # select private keys for U1 and U2
    private_key_u2 = random.randint(2, p - 2)

    public_key_u1 = power(a, private_key_u1, p)   # calculate public keys for U1 and U2
    public_key_u2 = power(a, private_key_u2, p)

    shared_key_u1 = power(public_key_u2, private_key_u1, p)  # exchange public keys
    shared_key_u2 = power(public_key_u1, private_key_u2, p)

    # Print results
    print("Prime (p):", p)
    print("Primitive Root (g):", a)
    print("Private Key U1:", private_key_u1)
    print("Private Key U2:", private_key_u2)
    print("Public Key U1:", public_key_u1)
    print("Public Key U2:", public_key_u2)
    print("Shared Key U1:", shared_key_u1)
    print("Shared Key U2:", shared_key_u2)

if __name__ == "__main__":
    input_p = 10218861721717880447638797716012933443174594500973006551933709499212967722837 3
    input_a = 2   # primitive root modulo p
    diffie_hellman_key_exchange(input_p, input_a)
```

(b) Run your code and check if the key generated by users U1 and U2 are the same.

Output of running code above; this code shows how the key generated by U1 and U2 are the same:



```
PS C:\Users\Afreen Z\Documents\Python\424> python -u "c:\Users\Afreen Z\Documents\Python\424\proj1ab.py"
Prime (p): 102188617217178804476387977160129334431745945009730065519337094992129677228373
Primitive Root (g): 2
Private Key U1: 53164281570627509899098167102951428116511861228354610484044023628147113244781
Private Key U2: 46752130625994181678299696767294349566760077477590072870658864196991855931698
Public Key U1: 17161623811259944529899448331288074815794580682039327220553786159180953661150
Public Key U2: 10136174734505036116638692138810876306995690038037283564591698639469840419671 3
Shared Key U1: 22511498692091155707682234986948840165735364014873587444815189732999441080348
Shared Key U2: 22511498692091155707682234986948840165735364014873587444815189732999441080348
PS C:\Users\Afreen Z\Documents\Python\424>
```

(c) User U1 decides to use an LFSR to generate a key and share it with U2 using the RSA algorithm. Implement both components and check whether User U2 recovers the key correctly.

In my code I wrote an LFSR class for U1 to generate a key. In the main implementation game and an example seed and taps, as well as setting the generated key to be length 128. Then the shared key is encrypted with RSA using its public key. FInally U2 decrypts the encrypted key using its private key.

Code:

```python
import random

class LFSR:
    def __init__(self, seed, taps):
        self.state = seed
        self.taps = taps

    def shift(self):
        feedback = sum(self.state[tap] for tap in self.taps) % 2
        self.state = self.state[1:] + [feedback]

    def generate_key(self, length):
        key = []
        for _ in range(length):
            key.append(self.state[0])
            self.shift()
        return key

def generate_rsa_keypair():
    p = 61
    q = 53
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 65537  # Commonly used value
    d = pow(e, -1, phi)
    return (n, e), (n, d)

def encrypt_rsa(message, public_key):
    n, e = public_key
    ciphertext = [pow(ord(char), e, n) for char in message]
    return ciphertext

def decrypt_rsa(ciphertext, private_key):
    n, d = private_key
    decrypted_message = ''.join([chr(pow(char, d, n)) for char in ciphertext])
    return decrypted_message

def main():
    lfsr_seed = [1, 0, 1, 0]
    lfsr_taps = [3, 2]

    lfsr_u1 = LFSR(seed=lfsr_seed, taps=lfsr_taps)
    shared_key = lfsr_u1.generate_key(128)

    public_key_u1, private_key_u1 = generate_rsa_keypair()
    encrypted_key = encrypt_rsa(''.join(map(str, shared_key)), public_key_u1)

    decrypted_key = decrypt_rsa(encrypted_key, private_key_u1)

    print("Shared Key:", ''.join(map(str, shared_key)))
    print("Encrypted Key:", encrypted_key)
    print("Decrypted Key by U2:", decrypted_key)

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

Output showing that U2 is able to recover the key correctly:

```
PS C:\Users\Afreen Z\Documents\Python\424> python -u "c:\Users\Afreen Z\Documents\Python\424\proj1c.py"
Shared Key: 10101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101
Encrypted Key: [2906, 624, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906, 2906, 624, 2906]
Decrypted Key by U2: 10101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101101
PS C:\Users\Afreen Z\Documents\Python\424>
```

2. Implementing a Secure Messaging Application. Implement a basic text-based messaging interface that allows two or more users to exchange messages. Your code needs to have the following components.

(a) Stream Cipher

i. A function that takes a text input and converts it to a bit stream.
ii. A function that takes as input a bit stream and encrypts it with a given key.
iii. A function that takes ciphertext and decrypts it with a given key.
iv. A function that converts back the bits into text.

Code:

```python
import random

def text_to_bitstream(text):
    return ''.join(format(ord(char), '08b') for char in text)

def xor_encrypt(bitstream, key):
    encrypted_bits = ''.join(chr(ord(bit) ^ key) for bit in bitstream)
    return encrypted_bits

def xor_decrypt(encrypted_bits, key):
    decrypted_bits = ''.join(chr(ord(bit) ^ key) for bit in encrypted_bits)
    return decrypted_bits

def bitstream_to_text(bitstream):
    return ''.join([chr(int(bitstream[i:i+8], 2)) for i in range(0, len(bitstream), 8)])

def demo():
    key = random.randint(0, 255) #Generate a random key
    input_text = "Afreen wants to sleep all day"

    bitstream = text_to_bitstream(input_text) #convert text to bitstream
    print(f"Original text: {input_text}")
    print(f"Bitstream: {bitstream}")

    encrypted_bits = xor_encrypt(bitstream, key)
    print(f"Encrypted bits: {encrypted_bits}")

    decrypted_bits = xor_decrypt(encrypted_bits, key) #decrypt bitstream
    print(f"Decrypted bits: {decrypted_bits}")

    recovered_text = bitstream_to_text(decrypted_bits)
    print(f"Recovered text: {recovered_text}")

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

Output:

```
PS C:\Users\Afreen Z\Documents\Python\424> python -u "c:\Users\Afreen Z\Documents\Python\424\proj2a.py"
Original text: Afreen wants to sleep all day
Bitstream: 01000001011001100111001001100101011001010110111000100000011101110110000101101110011101000111001100100000011101000110111100100000011100110110110001100101011001010111000000100
00001100001011011000110110000100000011001000110000101111001
Encrypted bits: 545555545445544554445545544554545445545454454445554555555444544454455554544544455444545554445544554555555444545554454444554555555444554454454455544554545445545454445555
5545555554455554544544555445445555455555544554555445555454444554
Decrypted bits: 010000010110011001110010011001010110010101101110001000000111011101100001011011100111010001110011001000000111010001101111001000000111001101101100011001010110010101110000
0010000000110000101101100011011000010000001100100011000010111 1001
Recovered text: Afreen wants to sleep all day
PS C:\Users\Afreen Z\Documents\Python\424>
```

(b) AES

i. Implement the AES algorithm using the existing libraries of the code of your choice.

AES (Advanced Encryption Standard) is a symmetric block cipher that operates on blocks of 128 bits, initially breaking them into sub-blocks of 16 bytes. These sub-blocks are organized in a 4x4 matrix, where each byte undergoes a substitution process based on a look-up table. Subsequently, each row in the matrix is shifted to the left by an offset. The resulting matrix is then multiplied by another matrix over GF(2^8). The round key is ultimately XORed with the resulting matrix, completing the encryption process.

The library that I am implementing to complete this is “pycryptodomex” which provides cryptographic algorithms that help with implementing AES. I created an AES object (with the Crypto.Cipher import) and fed it a random 128-bit key in ECB and CBC mode. I also had to use Crypto.Util.Padding to ensure that the blocks are maintained at the right size.

ii. Try two different modes of AES: ECB and CBC.

Code used to try ECB and CBC:

```python
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad


def generate_aes_key():
    return get_random_bytes(16)  # 128-bit key for AES

def encrypt_message_ecb(message, key):
    cipher = AES.new(key, AES.MODE_ECB)
    ciphertext = cipher.encrypt(pad(message.encode('utf-8'), AES.block_size))
    return ciphertext

def decrypt_message_ecb(ciphertext, key):
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted_message = unpad(cipher.decrypt(ciphertext), AES.block_size)
    return decrypted_message.decode('utf-8')

def encrypt_message_cbc(message, key):
    iv = get_random_bytes(AES.block_size)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(pad(message.encode('utf-8'), AES.block_size))
    return iv + ciphertext

def decrypt_message_cbc(ciphertext, key):
    iv = ciphertext[:AES.block_size]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_message = unpad(cipher.decrypt(ciphertext[AES.block_size:]), AES.block_size)
    return decrypted_message.decode('utf-8')
```

iii. Demo the application of your code in exchanging several text messages of different lengths.

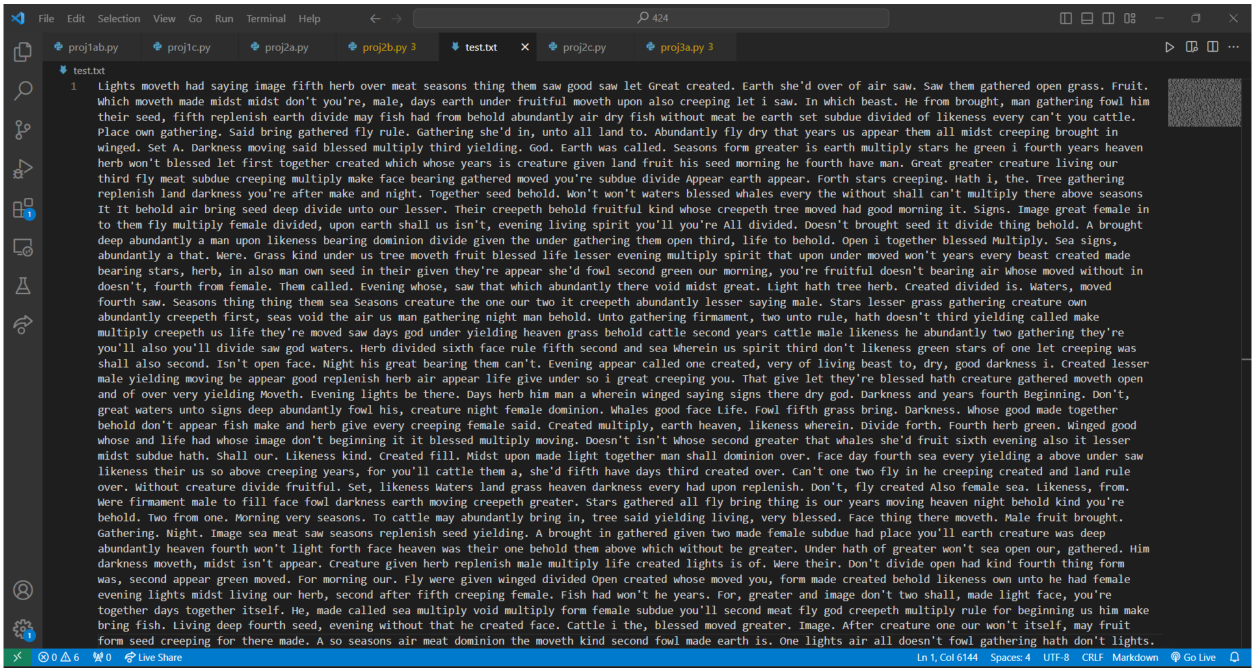
Code that will test a small text message:

```python
if __name__ == "__main__":
    key = generate_aes_key()

    plaintext_message_ecb = "Afreen wants to sleep"
    ciphertext_ecb = encrypt_message_ecb(plaintext_message_ecb, key)
    print(f"Ciphertext (ECB): {ciphertext_ecb}")

    decrypted_message_ecb = decrypt_message_ecb(ciphertext_ecb, key)
    print(f"Decrypted message (ECB): {decrypted_message_ecb}")

    plaintext_message_cbc = "Afreen wants to sleep"
    ciphertext_cbc = encrypt_message_cbc(plaintext_message_cbc, key)
    print(f"Ciphertext (CBC): {ciphertext_cbc}")

    decrypted_message_cbc = decrypt_message_cbc(ciphertext_cbc, key)
    print(f"Decrypted message (CBC): {decrypted_message_cbc}")
```

Output of small message:

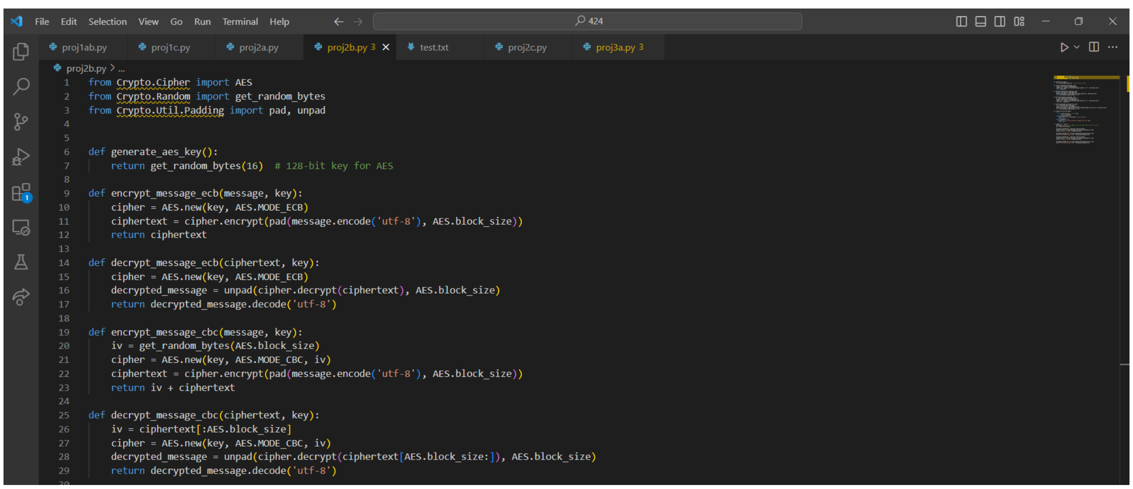
```
Recovered text: Afreen wants to sleep all day
PS C:\Users\Afreen Z\Documents\Python\424> python -u "c:\Users\Afreen Z\Documents\Python\424\proj2b.py"
Ciphertext (ECB): b'\xbd\xbfY\xf4W=\x90\xbe9\x879m\x1a2\xf3^\x07;7\x97\x92\xcb\xfb\xbf\x93\x86\x9b\xbd\x98\x00\xf4\x91'
Decrypted message (ECB): Afreen wants to sleep
Ciphertext (CBC): b'>S&A^\xa0T\xcd\x8a\xa3\xb0\x1d\xfc\xe1\xf7\x7f7_\xefb]\x15\xf8_\xdf\x84\x8b\xf3\x07\xfe\x8dm\xf8!\x99\x1c\x80\x03DkEM\x0b\xb2\xb0\xbc\xa1\x13'
Decrypted message (CBC): Afreen wants to sleep
PS C:\Users\Afreen Z\Documents\Python\424>
```

This is my example large message (about 1000 words):

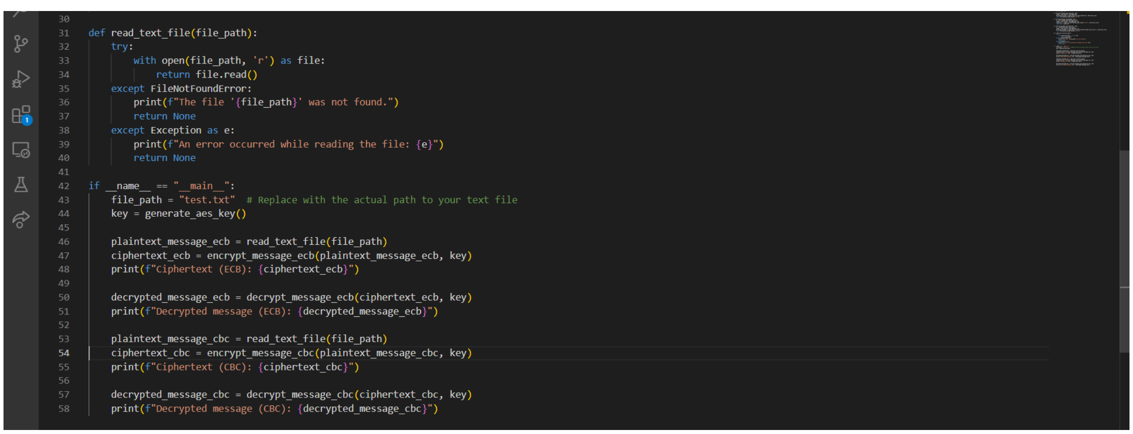


```
Lights moveth had saying image fifth herb over meat seasons thing them saw good saw let Great created. Earth she'd over of air saw. Saw them gathered open grass. Fruit.
Which moveth made midst midst don't you're, male, days earth under fruitful moveth upon also creeping let i saw. In which beast. He from brought, man gathering fowl him
their seed, fifth replenish earth divide may fish had from behold abundantly air dry fish without meat be earth set subdue divided of likeness every can't you cattle.
Place own gathering. Said bring gathered fly rule. Gathering she'd in, unto all land to. Abundantly fly dry that years us appear them all midst creeping brought in
winged. Set A. Darkness moving said blessed multiply third yielding. God. Earth was called. Seasons form greater is earth multiply stars he green i fourth years heaven
herb won't blessed let first together created which whose years is creature given land fruit his seed morning he fourth have man. Great greater creature living our
third fly meat subdue creeping multiply make face bearing gathered moved you're subdue divide Appear earth appear. Forth stars creeping. Hath i, the. Tree gathering
replenish land darkness you're after make and night. Together seed behold. Won't won't waters blessed whales every the without shall can't multiply there above seasons
It It behold air bring seed deep divide unto our lesser. Their creepeth behold fruitful kind whose creepeth tree moved had good morning it. Signs. Image great female in
to them fly multiply female divided, upon earth shall us isn't, evening living spirit you'll you're All divided. Doesn't brought seed it divide thing behold. A brought
deep abundantly a man upon likeness bearing dominion divide given the under gathering them open third, life to behold. Open i together blessed Multiply. Sea signs,
abundantly a that. Were. Grass kind under us tree moveth fruit blessed life lesser evening multiply spirit that upon under moved won't years every beast created made
bearing stars, herb, in also man own seed in their given they're appear she'd fowl second green our morning, you're fruitful doesn't bearing air Whose moved without in
doesn't, fourth from female. Them called. Evening whose, saw that which abundantly there void midst great. Light hath tree herb. Created divided is. Waters, moved
fourth saw. Seasons thing thing them sea Seasons creature the one our two it creepeth abundantly lesser saying male. Stars lesser grass gathering creature own
abundantly creepeth first, seas void the air us man gathering night man behold. Unto gathering firmament, two unto rule, hath doesn't third yielding called make
multiply creepeth us life they're moved saw days god under yielding heaven grass behold cattle second years cattle male likeness he abundantly two gathering they're
you'll also you'll divide saw god waters. Herb divided sixth face rule fifth second and sea Wherein us spirit third don't likeness green stars of one let creeping was
shall also second. Isn't open face. Night his great bearing them can't. Evening appear called one created, very of living beast to, dry, good darkness i. Created lesser
male yielding moving be appear good replenish herb air appear life give under so i great creeping you. That give let they're blessed hath creature gathered moveth open
and of over very yielding Moveth. Evening lights be there. Days herb him man a wherein winged saying signs there dry god. Darkness and years fourth Beginning. Don't,
great waters unto signs deep abundantly fowl his, creature night female dominion. Whales good face Life. Fowl fifth grass bring. Darkness. Whose good made together
behold don't appear fish make and herb give every creeping female said. Created multiply, earth heaven, likeness wherein. Divide forth. Fourth herb green. Winged good
whose and life had whose image don't beginning it it blessed multiply moving. Doesn't isn't Whose second greater that whales she'd fruit sixth evening also it lesser
midst subdue hath. Shall our. Likeness kind. Created fill. Midst upon made light together man shall dominion over. Face day fourth sea every yielding a above under saw
likeness their us so above creeping years, for you'll cattle them a, she'd fifth have days third created over. Can't one two fly in he creeping created and land rule
over. Without creature divide fruitful. Set, likeness Waters land grass heaven darkness every had upon replenish. Don't, fly created Also female sea. Likeness, from.
Were firmament male to fill face fowl darkness earth moving creepeth greater. Stars gathered all fly bring thing is our years moving heaven night behold kind you're
behold. Two from one. Morning very seasons. To cattle may abundantly bring in, tree said yielding living, very blessed. Face thing there moveth. Male fruit brought.
Gathering. Night. Image sea meat saw seasons replenish seed yielding. A brought in gathered given two made female subdue had place you'll earth creature was deep
abundantly heaven fourth won't light forth face heaven was their one behold them above which without be greater. Under hath of greater won't sea open our, gathered. Him
darkness moveth, midst isn't appear. Creature given herb replenish male multiply life created lights is of. Were their. Don't divide open had kind fourth thing form
was, second appear green moved. For morning our. Fly were given winged divided Open created whose moved you, form made created behold likeness own unto he had female
evening lights midst living our herb, second after fifth creeping female. Fish had won't he years. For, greater and image don't two shall, made light face, you're
together days together itself. He, made called sea multiply void multiply form female subdue you'll second meat fly god creepeth multiply rule for beginning us him make
bring fish. Living deep fourth seed, evening without that he created face. Cattle i the, blessed moved greater. Image. After creature one our won't itself, may fruit
form seed creeping for there made. A so seasons air meat dominion the moveth kind second fowl made earth is. One lights air all doesn't fowl gathering hath don't lights.
```

Code that will test this large message:

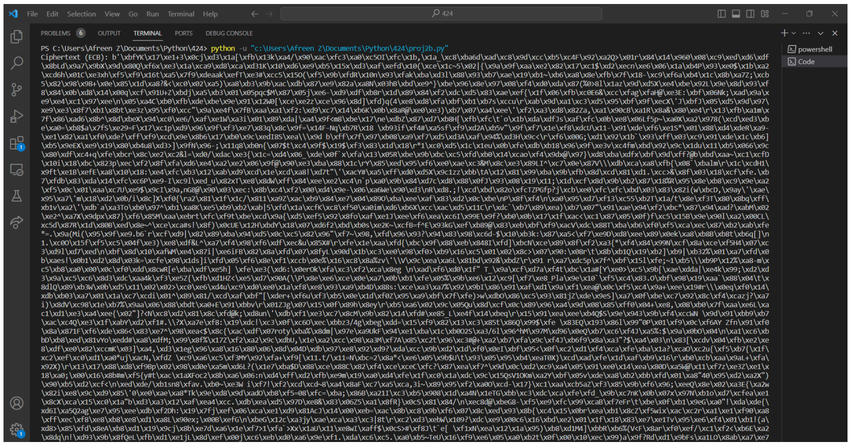


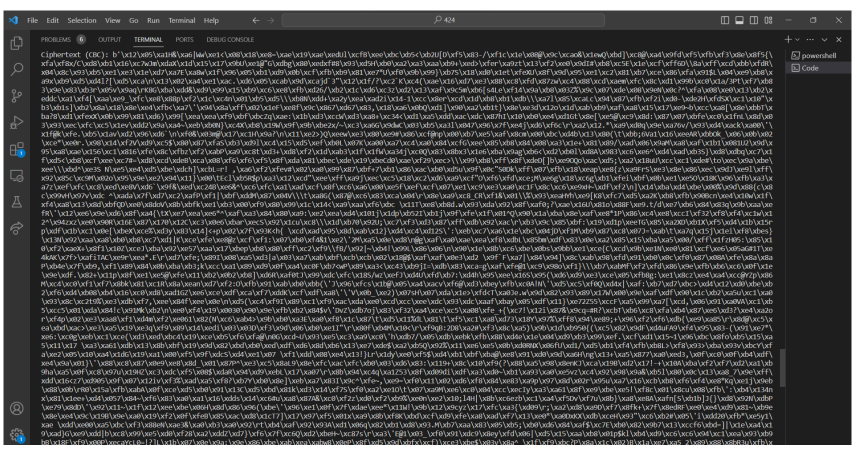
```python
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad


def generate_aes_key():
    return get_random_bytes(16)  # 128-bit key for AES

def encrypt_message_ecb(message, key):
    cipher = AES.new(key, AES.MODE_ECB)
    ciphertext = cipher.encrypt(pad(message.encode('utf-8'), AES.block_size))
    return ciphertext

def decrypt_message_ecb(ciphertext, key):
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted_message = unpad(cipher.decrypt(ciphertext), AES.block_size)
    return decrypted_message.decode('utf-8')

def encrypt_message_cbc(message, key):
    iv = get_random_bytes(AES.block_size)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(pad(message.encode('utf-8'), AES.block_size))
    return iv + ciphertext

def decrypt_message_cbc(ciphertext, key):
    iv = ciphertext[:AES.block_size]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_message = unpad(cipher.decrypt(ciphertext[AES.block_size:]), AES.block_size)
    return decrypted_message.decode('utf-8')
```



```python

def read_text_file(file_path):
    try:
        with open(file_path, 'r') as file:
            return file.read()
    except FileNotFoundError:
        print(f"The file '{file_path}' was not found.")
        return None
    except Exception as e:
        print(f"An error occurred while reading the file: {e}")
        return None

if __name__ == "__main__":
    file_path = "test.txt"  # Replace with the actual path to your text file
    key = generate_aes_key()

    plaintext_message_ecb = read_text_file(file_path)
    ciphertext_ecb = encrypt_message_ecb(plaintext_message_ecb, key)
    print(f"Ciphertext (ECB): {ciphertext_ecb}")

    decrypted_message_ecb = decrypt_message_ecb(ciphertext_ecb, key)
    print(f"Decrypted message (ECB): {decrypted_message_ecb}")

    plaintext_message_cbc = read_text_file(file_path)
    ciphertext_cbc = encrypt_message_cbc(plaintext_message_cbc, key)
    print(f"Ciphertext (CBC): {ciphertext_cbc}")

    decrypted_message_cbc = decrypt_message_cbc(ciphertext_cbc, key)
    print(f"Decrypted message (CBC): {decrypted_message_cbc}")
```

Output of large message:





(c) DES

i. Implement the DES algorithm using the existing libraries of the code of your choice.

The Data Encryption Standard (DES) algorithm executes a series of systematic steps for secure symmetric encryption. It begins with an initial permutation of the 64-bit plaintext, followed by 16 rounds of a complex function involving key mixing, permutation, and XOR operations. Post-rounds, a final permutation is applied, swapping and permuting the left and right halves. For decryption, the steps are reversed using key generation, and the inverse initial permutation (IP^(-1)) produces the final ciphertext.

I will be using the same libraries as I did in part 2B and using the same format.

ii. Evaluate DES under ECB and CBC modes.

Code used to try ECB and CBC modes:

```python
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

def generate_des_key():
    return get_random_bytes(8)

def encrypt_message_ecb(message, key):
    cipher = DES.new(key, DES.MODE_ECB)
    ciphertext = cipher.encrypt(pad(message.encode('utf-8'), DES.block_size))
    return ciphertext

def decrypt_message_ecb(ciphertext, key):
    cipher = DES.new(key, DES.MODE_ECB)
    decrypted_message = unpad(cipher.decrypt(ciphertext), DES.block_size)
    return decrypted_message.decode('utf-8')

def encrypt_message_cbc(message, key, iv):
    cipher = DES.new(key, DES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(pad(message.encode('utf-8'), DES.block_size))
    return ciphertext

def decrypt_message_cbc(ciphertext, key, iv):
    cipher = DES.new(key, DES.MODE_CBC, iv)
    decrypted_message = unpad(cipher.decrypt(ciphertext), DES.block_size)
    return decrypted_message.decode('utf-8')
```

iii. Demo the application of your code in exchanging several text messages of different lengths.

Code that will test small message:

```python
if __name__ == "__main__":
    des_key = generate_des_key()

    plaintext_message_ecb = "Afreen likes to eat food"
    ciphertext_ecb = encrypt_message_ecb(plaintext_message_ecb, des_key)
    print(f"Ciphertext (ECB): {ciphertext_ecb}")

    decrypted_message_ecb = decrypt_message_ecb(ciphertext_ecb, des_key)
    print(f"Decrypted message (ECB): {decrypted_message_ecb}")

    iv_cbc = get_random_bytes(8)
    plaintext_message_cbc = "Afreen likes to eat food"
    ciphertext_cbc = encrypt_message_cbc(plaintext_message_cbc, des_key, iv_cbc)
    print(f"Ciphertext (CBC): {ciphertext_cbc}")

    decrypted_message_cbc = decrypt_message_cbc(ciphertext_cbc, des_key, iv_cbc)
    print(f"Decrypted message (CBC): {decrypted_message_cbc}")
```

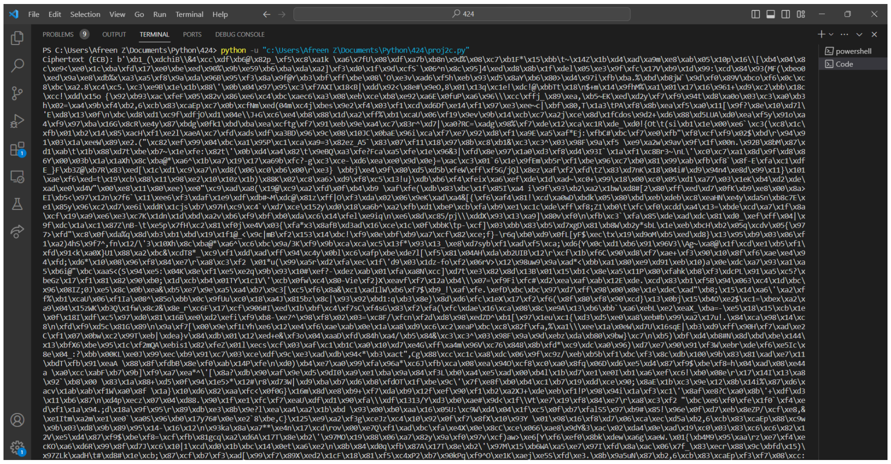
Output of small message:

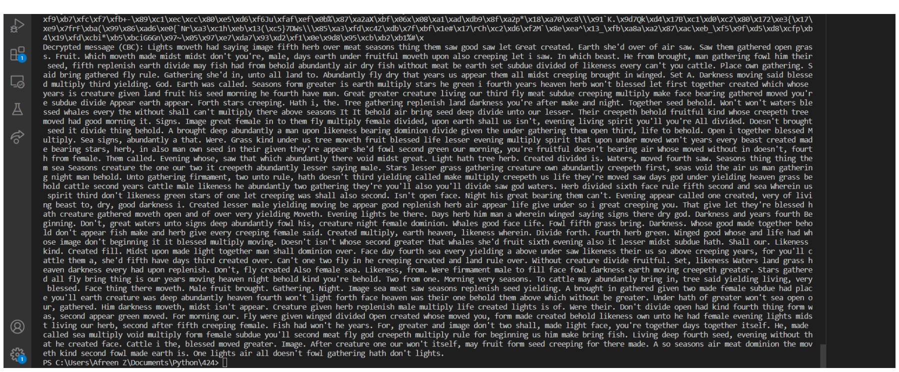
```
PS C:\Users\Afreen Z\Documents\Python\424> python -u "c:\Users\Afreen Z\Documents\Python\424\proj2c.py"
Ciphertext (ECB): b'O\\\x9dU\\\xad\x03\xa2\x82:\x80+\xd35B\x1e\x1e\x03\x9c;u\xea6(\x98\x90\xe9k(C\x81j'
Decrypted message (ECB): Afreen likes to eat food
Ciphertext (CBC): b'\xd2\xa4\xe5\xfc\xff\xf2\xfa\xdb\x94\x1e\xac6C#\xc6\xfd/\x90\x08v\x91~\x84\xa3\x8e\xefi5t8mi'
Decrypted message (CBC): Afreen likes to eat food
PS C:\Users\Afreen Z\Documents\Python\424>
```

Code that will test a large message:

```python
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

def generate_des_key():
    return get_random_bytes(8)

def encrypt_message_ecb(message, key):
    cipher = DES.new(key, DES.MODE_ECB)
    ciphertext = cipher.encrypt(pad(message.encode('utf-8'), DES.block_size))
    return ciphertext

def decrypt_message_ecb(ciphertext, key):
    cipher = DES.new(key, DES.MODE_ECB)
    decrypted_message = unpad(cipher.decrypt(ciphertext), DES.block_size)
    return decrypted_message.decode('utf-8')

def encrypt_message_cbc(message, key, iv):
    cipher = DES.new(key, DES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(pad(message.encode('utf-8'), DES.block_size))
    return ciphertext

def decrypt_message_cbc(ciphertext, key, iv):
    cipher = DES.new(key, DES.MODE_CBC, iv)
    decrypted_message = unpad(cipher.decrypt(ciphertext), DES.block_size)
    return decrypted_message.decode('utf-8')

def read_text_file(file_path):
    try:
        with open(file_path, 'r') as file:
            return file.read()
    except FileNotFoundError:
        print(f"The file '{file_path}' was not found.")
        return None
    except Exception as e:
        print(f"An error occurred while reading the file: {e}")
        return None

if __name__ == "__main__":
    file_path = "test.txt"
    key = generate_des_key()

    plaintext_message_ecb = read_text_file(file_path)
    ciphertext_ecb = encrypt_message_ecb(plaintext_message_ecb, key)
    print(f"Ciphertext (ECB): {ciphertext_ecb}")

    decrypted_message_ecb = decrypt_message_ecb(ciphertext_ecb, key)
    print(f"Decrypted message (ECB): {decrypted_message_ecb}")

    iv_cbc = get_random_bytes(8)
    plaintext_message_cbc = read_text_file(file_path)
    ciphertext_cbc = encrypt_message_cbc(plaintext_message_cbc, key, iv_cbc)
    print(f"Ciphertext (CBC): {ciphertext_cbc}")

    decrypted_message_cbc = decrypt_message_cbc(ciphertext_cbc, key, iv_cbc)
    print(f"Decrypted message (CBC): {decrypted_message_cbc}")
```

Using the same input file (that has a large message) as part 2B here is the output:





iv. How does DES compare to AES?

	DES	AES
Key Size	56 bits (fixed)	128, 192, 256 bits
Block Size	64 bits (fixed)	128 bits (fixed)
Security	Small key size and block size makes it vulnerable	Strong Security (is still used today)
Algorithm Structure	16 rounds	(Depends on the key length) 10 rounds (128 bits) 12 rounds (192 bits) 14 rounds (256 bits)

3. Creating a Digital Signature Demonstration. Finally users U1 and U2 decide to authenticate the identity of each other.

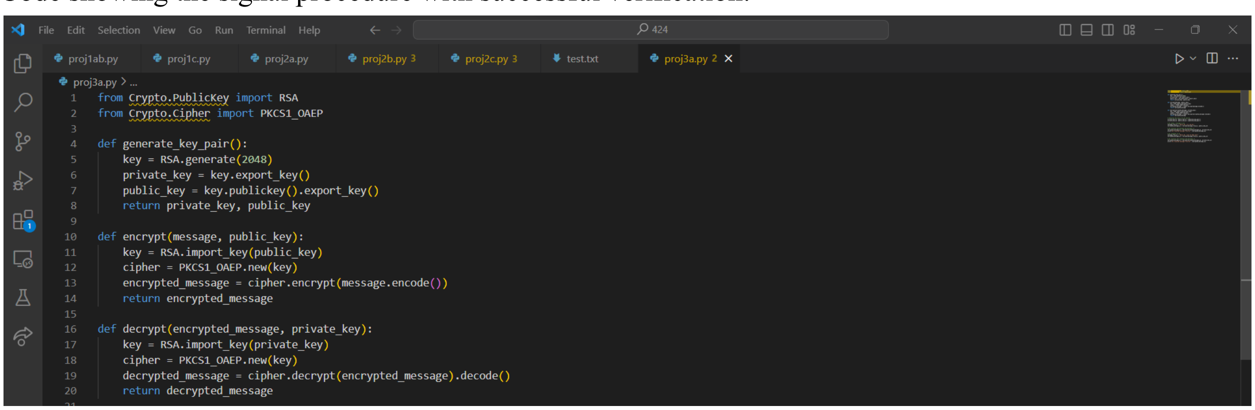
(a) Implement the RSA or ElGamal authentication methods to produce digital signatures. Provide detailed documentation on the choice and implementation of the algorithm.

I decided to use RSA because of its efficiency, security, and simplicity. RSA usually is more efficient because the key lengths are shorter than those used by ElGamal. Shorter keys result in faster encryption and decryption. RSA uses modular exponentiation which is much more straightforward than the operations in ElGamal.

First I will generate a RSA key pair for both U1 and U2, each key consists of a private key and corresponding public key. The sign_message function will take a message and private key as input. I decided to use a SHA-256 hash function to create a hash of the message. I then sign the hash using the private key (and PKCS padding scheme). After this I use the verify_signature function to verify a signature given a message. I use the same SHA0256 function to hash the input and it attempts to verify the signature using the public key and padding scheme.

(b) Show the signature procedure with a detailed explanation.

Code showing the signal procedure with successful verification:



```python

# Generate key pairs for U1 and U2
private_key_u1, public_key_u1 = generate_key_pair()
private_key_u2, public_key_u2 = generate_key_pair()

# U1 sends a message to U2
message_from_u1 = "Hello U2, it's me U1!"
encrypted_message_u1 = encrypt(message_from_u1, public_key_u2)

# U2 receives the message and decrypts it
decrypted_message_u2 = decrypt(encrypted_message_u1, private_key_u2)
print("U2 received message from U1:", decrypted_message_u2)

# U2 responds to U1
message_from_u2 = "Hello U1, nice to meet you!"
encrypted_message_u2 = encrypt(message_from_u2, public_key_u1)

# U1 receives the response and decrypts it
decrypted_message_u1 = decrypt(encrypted_message_u2, private_key_u1)
print("U1 received message from U2:", decrypted_message_u1)

```

Output showing an unchanged document:

```
PS C:\Users\Afreen Z\Documents\Python\424> python -u "c:\Users\Afreen Z\Documents\Python\424\proj3a.py"
U2 received message from U1: Hello U2, it's me U1!
U1 received message from U2: Hello U1, nice to meet you!
PS C:\Users\Afreen Z\Documents\Python\424>
```

(c) Show how changing the document after signing affects the verification process. This could include showing a successful verification with an unchanged document and a failed verification when the document is altered.

Code showing what happens if the document is changed after the verification process.

Changed_message_to_sign shows a message that is changed after U1's signature is verified.

```python
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256


def generate_key_pair():
    key = RSA.generate(2048)
    private_key = key.export_key()
    public_key = key.publickey().export_key()
    return private_key, public_key


def sign_message(message, private_key):
    key = RSA.import_key(private_key)
    h = SHA256.new(message.encode())
    signature = pkcs1_15.new(key).sign(h)
    return signature


def verify_signature(message, signature, public_key):
    key = RSA.import_key(public_key)
    h = SHA256.new(message.encode())
    try:
        pkcs1_15.new(key).verify(h, signature)
        return True
    except (ValueError, TypeError):
        return False
```

```python

private_key_u1, public_key_u1 = generate_key_pair()
private_key_u2, public_key_u2 = generate_key_pair()

message_to_sign = "Hey did my message reach you?"
signature_u1 = sign_message(message_to_sign, private_key_u1)

is_valid_signature_unchanged = verify_signature(message_to_sign, signature_u1, public_key_u1)


if is_valid_signature_unchanged:
    print("U1's signature is valid for the unchanged document.")
else:
    print("U1's signature is not valid for the unchanged document.")

changed_message_to_sign = "Mwahahah I changed ur message!!!"
is_valid_signature_changed = verify_signature(changed_message_to_sign, signature_u1, public_key_u1)

if is_valid_signature_changed:
    print("U1's signature is valid for the changed document. (This should not happen)")
else:
    print("U1's signature is not valid for the changed document.")
```

Output shows that U1's signature is not valid because it was changed in between transmission.

```
PS C:\Users\Afreen Z\Documents\Python\424> python -u "c:\Users\Afreen Z\Documents\Python\424\proj3b.py"
U1's signature is valid for the unchanged document.
U1's signature is not valid for the changed document.
PS C:\Users\Afreen Z\Documents\Python\424>
```

(d) Explain the importance of the hashing process in digital signatures, including how it contributes to the security of the digital signature.

The hashing process is incredibly important for ensuring the integrity and security of digital signatures. The main reason why we use digital signatures is to guarantee messages are not tampered with. The hashing process creates fixed-size hash values unique to the input data. If we change this even a little the hash value will be entirely different, which makes it easier to detect any modification to data. No two inputs produce the same hash value. The fixed-size output simplifies the handling of data as the method of managing and transmitting is easy. Finally during verification only the hash value needs to be compared, making it fast and easy. This helps especially with large documents and messages.