Project 1 : OTP REPORT

getKey Function:

This function loads and gets the secret key from the file key.txt from data/key.txt

```
# Function to load the secret key from a file
def getKey():
    # Define the path to the key file using os.path.join for platform-independent path joining
    key_file_path = os.path.join(os.getcwd(), 'data', 'key.txt')
    # Open the key file in read mode
    with open(key_file_path, "r") as file:
        # Read the content of the file and remove leading/trailing whitespace
        sk = file.read().strip()
    return sk
```

getPlaintext Function:

This function loads and gets the plaintext from the file plaintext.txt from data/plaintext.txt

```
# Function to load plaintext from the plaintext file
def getPlaintext():
    # Define the path to the plaintext file using os.path.join
    plaintext_file_path = os.path.join(os.getcwd(), 'data', 'plaintext.txt')
    # Initialize an empty list to store the words from the plaintext file
    words = []
    # Open the plaintext file for reading
    with open(plaintext_file_path, 'r') as file:
        # Read each line of the file into an array
        words = file.read().split()
    return words
```

getCiphertext and writeCiphertext function:

This function writes the encrypted ciphertext to the file ciphertext.txt and loads and gets the ciphertext from the ciphertext.txt file to decrypt it further.

```
# Function to load ciphertext from the ciphertext file
def getCiphertext():
    # Define the path to the ciphertext file using os.path.join
    ciphertext_file_path = os.path.join(os.getcwd(), 'data', 'ciphertext.txt')
    # Open the ciphertext file for reading
    with open(ciphertext_file_path, "r") as file:
        # Read each line of the file into an array
        array = [line.strip() for line in file]
    return array

def writeCiphertxt(ciphertext):
    # Define the path to the ciphertext file using os.path.join
    ciphertext_file_path = os.path.join(os.getcwd(), 'data', 'ciphertext.txt')
    # Open the ciphertext file for writing
    with open(ciphertext_file_path, "w") as file:
        # Write each item in the ciphertext array to the file, one per line
        for i in ciphertext:
            # Write each item in the ciphertext array to the file, one per line
            file.write(str(i) + "\n")
```

writeResults function:

This function writes the decrypted plaintext to the result.txt file in data/result.txt

```
# Function to write results to a file
def writeResults(plaintext):
    # Define the path to the plaintext file using os.path.join
    plaintext_file_path = os.path.join(os.getcwd(), 'data', 'result.txt')
    # Open the plaintext file for writing
    with open(plaintext_file_path, "w") as file:
        # Write each item in the plaintext array to the file, one per line
        for i in plaintext:
            file.write(str(i))
```

writeKey Function:

This function writes the random generated hexadecimal key to the file newkey.txt in data/newkey.txt

```
# Function to write keys to a file
def writekeys(newKey):
    # Define the path to the newkey file using os.path.join
    newkey_file_path = os.path.join(os.getcwd(), 'data', 'newkey.txt')
    # Open the newkey file for writing
    with open(newkey_file_path, "w") as file:
        # Write each item in the newkey array to the file, one per line
        file.write(str(newKey) + "\n")
```

Xor Function:

This function performs xor between two strings

```
# Function to perform XOR between two binary strings
def xor(str1, str2):
    result = ""
    # Iterate through the characters in the binary strings and perform XOR
    for i in range(len(str1)):
        if str1[i] == '0' and str2[i] == '0':
            result += '0'
        elif str1[i] == '1' and str2[i] == '1':
            result += '0'
        else:
            result += '1'
    return result
```

Otp, text_to_binary. And binary_to_text Functions:

These fuinctions perform otp encryption (xor), converts text to binary representation, and converts binary to text.

```
# Function to perform OTP (One-Time Pad) encryption
def otp(plaintext, secret_key):
    # Perform XOR between the plaintext and the secret key
    ciphertext = xor(plaintext, secret_key)
    return ciphertext

# Function to convert text to binary representation
def text_to_binary(text):
  binary_result = ""
  for char in text:
        # Convert each character to its ASCII value and then to binary
        binary_char = bin(ord(char))[2:].zfill(8)
        binary_result += binary_char
  return binary_result

# Function to convert binary representation back to text
def binary_to_text(binary_string):
    text_result = ""
    # Convert binary back to characters, assuming 8-bit characters
    for i in range(0, len(binary_string), 8):
        split = binary_string[i:i+8]
        text_result += chr(int(split, 2))
    return text_result
```

encryption Function;

This function encrypts the plaintext using the secret key using xor, and otp and writes the encrypted text to the ciphertext file.

To run this: **python src/prj_1.py**

```python
# Function to perform encryption
def encryption():
    # Load the secret key from a file
    sk = getKey()
    # Load the plaintext from a file
    m = getPlaintext()
    # Initialize an empty list to store the ciphertext
    ciphertext = []
    # Print the ciphertext text
    print("CIPHERTEXT")
    # Loop through each element in the plaintext
    for i in range(len(m)):
        # Convert the plaintext to binary representation
        binary_text = text_to_binary(m[i])
        # Check if the binary text has the same length as the secret key
        if len(binary_text) != len(sk):
            # If not, print an error message and continue to the next element
            print("error: length is incorrect!")
            continue
        else:
            # Encrypt the binary plaintext using OTP (One-Time Pad)
            enc = otp(binary_text, sk)
            print(enc)
            # Append the encrypted text to the ciphertext list
            ciphertext.append(enc)
    # Write the ciphertext to a file
    writeCiphertxt(ciphertext)
```

OUTPUT for encryption function:

```
PS C:\Users\muhib\OneDrive - University of Cincinnati\Desktop\Fall 2023\Data Security\o
tp_m14056283> python .\src\prj_1.py
CIPHERTEXT
00110111110011111001000101111101
```

decryption Function:

This function decrypts the ciphertext using the secret using xor, and writes the decrypted text to result.txt in data/result.txt

To run this: **python src/prj_1.py**

```
# Function to perform decryption
def decryption():
    # Load the secret key from a file
    sk = getKey()
    # Load the ciphertext from a file
    ciphertext = getCiphertext()
    # Initialize an empty list to store the decrypted plaintext
    plaintext = []
    # Print the Decrypted text
    print("DECRYPTED TEXT")
    # Loop through each element in the ciphertext
    for i in range(len(ciphertext)):
        # Check if the length of the ciphertext is the same as the secret key
        if len(ciphertext[i]) != len(sk):
            print("error: length is incorrect!")
            continue
        else:
            # Decrypt the ciphertext using XOR with the secret key and convert back to
            dec = binary_to_text(xor(ciphertext[i], sk))
            print(dec)
            # Append the decrypted text to the plaintext list
            plaintext.append(dec)
    # Write the decrypted plaintext to a file
    writeResults(plaintext)
```

OUTPUT for decryption function:

```
PS C:\Users\muhib\OneDrive - University of Cincinnati\Desktop\Fall 2023\Data Security\o
tp_m14056283> python .\src\prj_1.py
DECRYPTED TEXT
bear
```

keygen Function:

This function generates a key which is in hexadecimal, between1 to 128 and writes to newkey.txt in data/newkey.txt

To run this: **python src/keyGen.py -l 128** (or number of bits in multiple of 8)

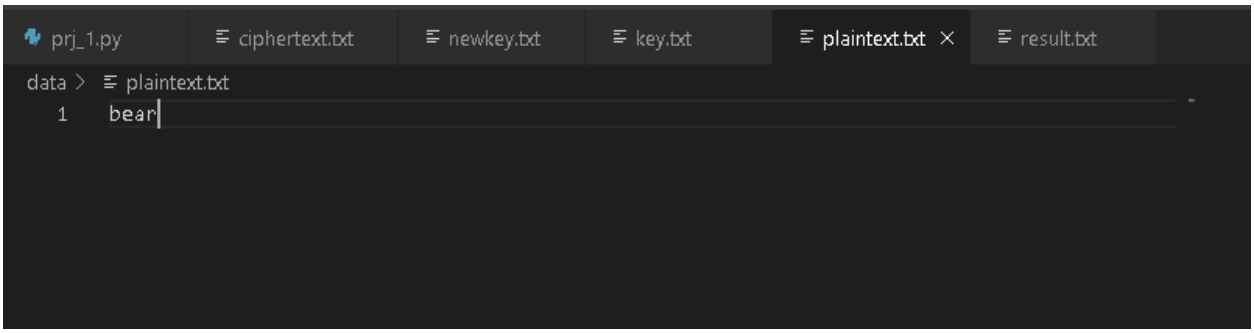
```
# Function to parse command line arguments
def parseArgs(args):
    parser = argparse.ArgumentParser()
    # Add the lambda argument
    parser.add_argument('-l', '--lambda1', type=int, default=128, help='enter lambda')
    opts, _ = parser.parse_known_args(args)
    return opts

# Function to generate a secret key
def generate_secret_key(bits):
    # Check if the number of bits is a multiple of 8
    if bits % 8 != 0:
        raise ValueError("Number of bits must be a multiple of 8")
    num_bytes = bits // 8
    # Generate a random secret key as a hexadecimal string
    key = secrets.token_hex(num_bytes)
    return key

# Function to generate a secret key
def KeyGen(opts):
    # Get the number of bits from the command line
    lambda_bits = opts.lambda1
    # Check if the number of bits is between 1 and 128
    if 1 <= lambda_bits <= 128:
        # Generate a random secret key as a hexadecimal string
        key=generate_secret_key(lambda_bits)
        writekeys(key)
```

OUTPUT for keygen function:

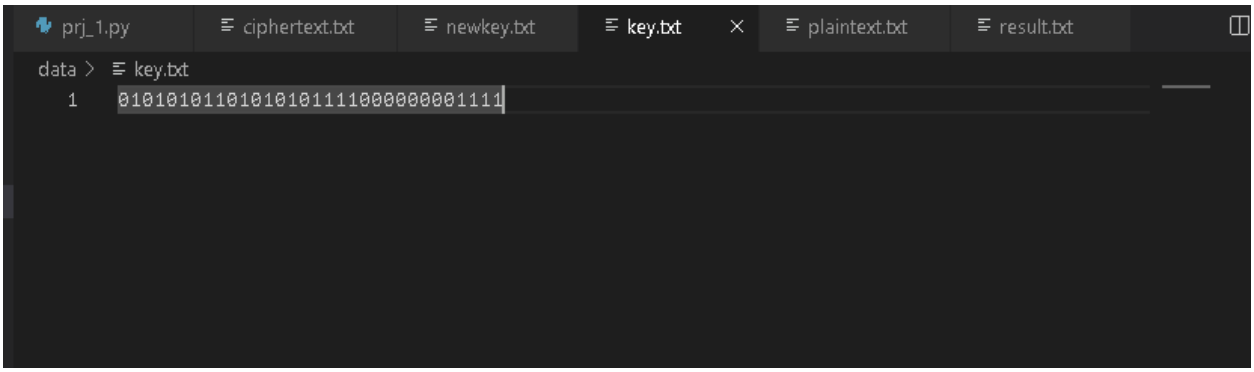
```
PS C:\Users\muhib\OneDrive - University of Cincinnati\Desktop\Fall 2023\Data Security\o
tp_m14056283> python .\src\prj_1.py -l 128
New Key:  9ad6cef9152c69e5156cc081af21a380
```

OUTPOUT OF ALL TEXT FILES:

plaintext.txt

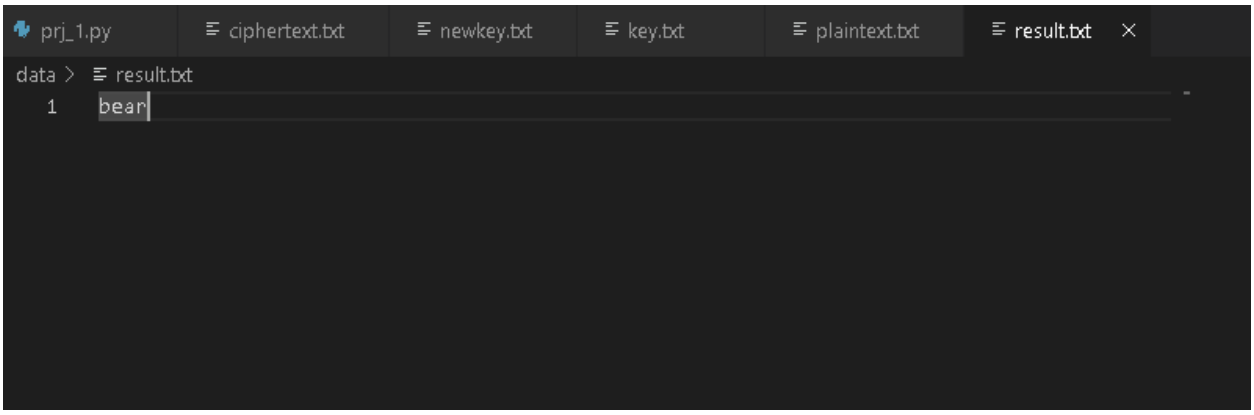


key.txt (secret key)



Ciphertext.txt

Result.txt



Newkey.txt