CNS LAB 4

SAHIL BONDRE: U18CO021

Write a program to implement Vernam cipher and perform the following tasks.

1) Input plain text from a file and obtain binary ASCII character code for each letter of the plaintext.
2) Randomly generate a key and obtain binary ASCII character code for each letter of the key. Your key must be equal in length to the plaintext.
3) Perform Encryption: Plaintext XOR Key.
4) Write output cipher text as hex values in a file.
5) Decryption – Read ciphertext from the file and obtain its equivalent binary character code. Perform its XOR with Key (must be in binary).
6) Output plaintext - Obtain plaintext from decrypted binary code and write in a file.

```
def recursive_read(allowed_input, message=""):
    # Recursively reads user input until input is not in allowd_input
    while True:
        user_input = input(message)
        if user_input in allowed_input:
            return user_input



def recursive_read_int(message=""):
    # Recursively reads user input until input is not in allowd_input
    while True:
        user_input = input(message)
        try:
            value = int(user_input)
            return value
```

```
        except:
            pass



def file_to_str(filename):
    try:
        with open(filename, 'r') as file:
            return file.read()
    except:
        print("Error: File not found!")
        exit(1)



def perform_encryption():
    filename = input("Enter file to be encrypted: ")
    message = file_to_str(filename)
    key = input("Enter key value: ")
    if len(key) != len(message):
        print("ERROR: Key and message length should be same")
        exit(0)
    result = ""
    hex_val = ""


    print("Message:")
    for i in message:
        print(f"{i}: {bin(ord(i))[2:].zfill(8)}")
```

```
    print()
    print("Key:")
    for i in key:
        print(f"{i}: {bin(ord(i))[2:].zfill(8)}")

    print()
    print("XOR: ")
    for i in range(len(message)):
        a = ord(message[i])
        b = ord(key[i])

        print(
            f"{bin(a)[2:]} ^ {bin(b)[2:]} = {bin(a ^ b)[2:].zfill(8)}")

        result += chr(a ^ b)
        hex_val += f"{hex(a ^ b)[2:].zfill(2)} "

    print(f"Final hex:\n{hex_val}")
    print("Saved to hex.txt")

    with open("hex.txt", "w") as f:
        f.write(hex_val)



def perform_decyption():
```

```
filename = input("Enter hex file to be decrypted: ")

message = file_to_str(filename)

hex_vals = message.split(" ")[:-1]


key = input("Enter key value: ")

if len(key) != len(hex_vals):

    print("ERROR: Key and message length should be same")

    exit(0)


result = ""


print("Hex Values:")

for i in hex_vals:

    print(f"{i}: {bin(int(i, 16))[2:].zfill(8)}")


print()

print("Key:")

for i in key:

    print(f"{i}: {bin(ord(i))[2:].zfill(8)}")


print()

print("XOR: ")

for i in range(len(hex_vals)):

    a = int(hex_vals[i], 16)

    b = ord(key[i])
```

```
        print(
            f"{bin(a)[2:].zfill(8)} ^ {bin(b)[2:]} = {bin(a ^
b)[2:].zfill(8)}")


        result += chr(a ^ b)
    print(f"Final string:\n{result}")
    print("Saved to ans.txt")


    with open("ans.txt", "w") as f:
        f.write(result)



is_encrypt = recursive_read(
    ["e", "d"], "Enter 'e' for encryption or 'd' for decryption: ") == "e"



if is_encrypt:
    perform_encryption()
else:
    perform_decyption()
```

```
PS F:\code\github.com\godcrampy\college-notes\cns\lab-04> python .\vernam.py
Enter 'e' for encryption or 'd' for decryption: e
Enter file to be encrypted: msg.txt
Enter key value: PLUTO
Message:
H: 01001000
E: 01000101
L: 01001100
L: 01001100
O: 01001111

Key:
P: 01010000
L: 01001100
U: 01010101
T: 01010100
O: 01001111

XOR:
1001000 ^ 1010000 = 00011000
1000101 ^ 1001100 = 00001001
1001100 ^ 1010101 = 00011001
1001100 ^ 1010100 = 00011000
1001111 ^ 1001111 = 00000000
Final hex:
18 09 19 18 00
Saved to hex.txt
```

```
PS F:\code\github.com\godcrampy\college-notes\cns\lab-04> python .\vernam.py
Enter 'e' for encryption or 'd' for decryption: d
Enter hex file to be decrypted: hex.txt
Enter key value: PLUTO
Hex Values:
18: 00011000
09: 00001001
19: 00011001
18: 00011000
00: 00000000

Key:
P: 01010000
L: 01001100
U: 01010101
T: 01010100
O: 01001111

XOR:
00011000 ^ 1010000 = 01001000
00001001 ^ 1001100 = 01000101
00011001 ^ 1010101 = 01001100
00011000 ^ 1010100 = 01001100
00000000 ^ 1001111 = 01001111
Final string:
HELLO
Saved to ans.txt
PS F:\code\github.com\godcrampy\college-notes\cns\lab-04>
```