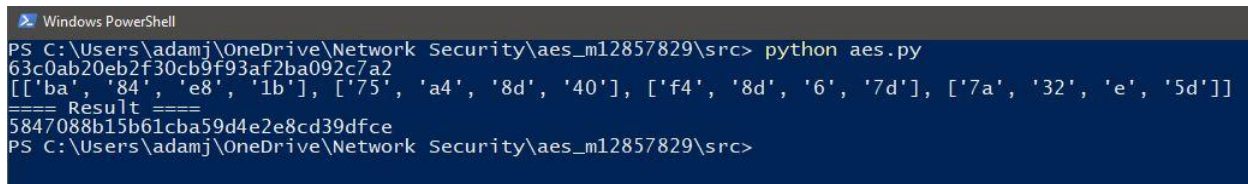


Project 1 Report

For this project, I developed a Python program that implements the 1st round of AES encryption. To run the program, open a terminal in the “./src” directory of this project. In the terminal, type “python aes.py” to run the Python script.

The input message is stored in a .txt file called “plaintext.txt” within the “./data” directory of this project. The default message is “Two One Nine Two”. Additionally, two subkeys are provided in the “subkey_example.txt” txt file in the same directory. The result of the Python script is printed to the “result.txt” txt file in the same directory.

Using the default values, the terminal output should be the following:

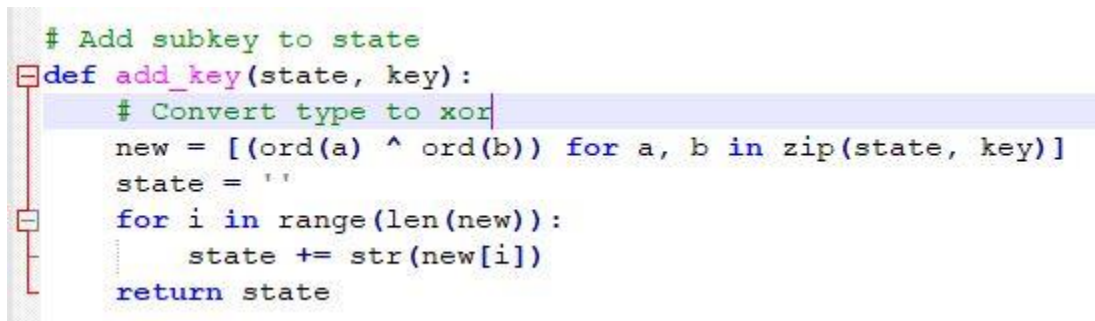


```
Windows PowerShell
PS C:\Users\adamj\OneDrive\Network Security\aes_m12857829\src> python aes.py
63c0ab20eb2f30cb9f93af2ba092c7a2
[['ba', '84', 'e8', '1b'], ['75', 'a4', '8d', '40'], ['f4', '8d', '6', '7d'], ['7a', '32', 'e', '5d']]
==== Result ====
5847088b15b61cba59d4e2e8cd39dfce
PS C:\Users\adamj\OneDrive\Network Security\aes_m12857829\src>
```

The first round of AES consists of several steps:

1. Add first subkey
2. Substitute bytes using Sbox
3. Shift rows
4. Mix columns
5. Add second subkey

First, the add_key function adds the first subkey:



```
# Add subkey to state
def add_key(state, key):
    # Convert type to xor
    new = [(ord(a) ^ ord(b)) for a, b in zip(state, key)]
    state = ''
    for i in range(len(new)):
        state += str(new[i])
    return state
```

This function takes the current state of the message and a subkey. The function performs bitwise XOR operation on each character in the state and key strings, and the result is stored in a new list called new. The new list is then converted to a string and returned.

Next, substitute bytes using the Sbox:

```
# Substitute bytes using Sbox
def sub_bytes(state):
    # Replace the state matrix with values from s-box
    hex_str = bin2hex(state)
    new = ''
    sbox = [
        0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
        0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
        0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
        0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
        0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
        0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
        0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
        0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
        0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
        0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
        0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
        0xE7, 0x08, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
        0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
        0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
        0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
        0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16 ]

    for i in range(0, len(hex_str), 2):
        if (sbox[int(hex_str[i:i+2], 16)] > 16):
            new += str(hex(sbox[int(hex_str[i:i+2], 16)])[2:])
        else:
            new += "0" + str(hex(sbox[int(hex_str[i:i+2], 16)])[2:])
    return new
```

After converting the state to a hex_str, define the Sbox. The function iterates over the hex string and substitutes each hex with a value from the Sbox. The result is then returned.

Next, shift rows:

```
# Shift rows using algorithm
def shift_rows(hex_str):
    print(hex_str)
    # Shift rows in the state matrix
    shifted_state = [''] * int((len(hex_str)/2))
    shifted_state[0] = hex_str[0:2]
    shifted_state[1] = hex_str[10:12]
    shifted_state[2] = hex_str[20:22]
    shifted_state[3] = hex_str[30:32]
    shifted_state[4] = hex_str[8:10]
    shifted_state[5] = hex_str[18:20]
    shifted_state[6] = hex_str[28:30]
    shifted_state[7] = hex_str[6:8]
    shifted_state[8] = hex_str[16:18]
    shifted_state[9] = hex_str[26:28]
    shifted_state[10] = hex_str[4:6]
    shifted_state[11] = hex_str[14:16]
    shifted_state[12] = hex_str[24:26]
    shifted_state[13] = hex_str[2:4]
    shifted_state[14] = hex_str[12:14]
    shifted_state[15] = hex_str[22:24]
    new = ''
    for i in range(len(shifted_state)):
        new += shifted_state[i]
    return new
```

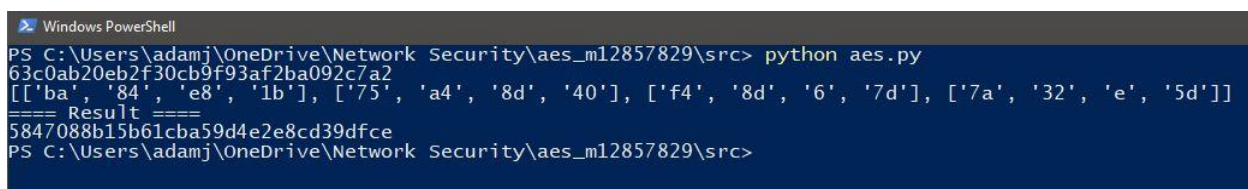
Next, define the operations to be used in mix_columns:

```
# Add all hex values together (used in mix_columns)
def addHex(s1,s2,s3,s4):
    result = str(hex(int(s1,16) ^ int(s2,16)))[2:]
    result = str(hex(int(result,16) ^ int(s3,16)))[2:]
    result = str(hex(int(result,16) ^ int(s4,16)))[2:]
    return result

# Mutiply by 2
def by2(string):
    binary = str(bin(int(string, 16)))[2:]
    result = ''
    if(len(binary) == 8):
        binaryMult2 = str(hex(int(binary[1:] + "0",16)))[2:]
        result = str(hex(int(binaryMult2,2) ^ int("1b",16)))[2:]
    else:
        binaryMult2 = str(hex(int(binary + "0",16)))[2:]
        result = str(hex(int(binaryMult2,2)))[2:]
    return result;

# Multiply by 3
def by3(string):
    by2Holder = by2(str(string))
    result = str(hex(int(str(by2Holder),16) ^ int(str(string),16)))[2:]
    return result
```

First, addHex returns the bitwise XOR sum of the four hex inputs. Next, the by2 function multiplies the hex input's binary value by 2. If the binary value is 8 bits long, the function performs a bitwise shift and then uses XOR with "1b" hex



```
Windows PowerShell
PS C:\Users\adamj\OneDrive\Network Security\aes_m12857829\src> python aes.py
63c0ab20eb2f30cb9f93af2ba092c7a2
[['ba', '84', 'e8', '1b'], ['75', 'a4', '8d', '40'], ['f4', '8d', '6', '7d'], ['7a', '32', 'e', '5d']]
==== Result ====
5847088b15b61cba59d4e2e8cd39dfce
PS C:\Users\adamj\OneDrive\Network Security\aes_m12857829\src>
```

value. If not 8 bits long, then a 0 is added as padding to the binary value and then the function performs a bitwise shift. Last, the by3 function multiplies the hex input's binary value by 3. This function first uses by2, and then uses XOR with the result of by2 and the input. The output is then returned as a hex string.

Next, define the mix_columns function:


```

# Mix columns
def mix_columns(hex_str):
    # Separate current state into a state matrix
    state_matrix = [(int(hex_str[0:2], 16), int(hex_str[2:4], 16), int(hex_str[4:6], 16), int(hex_str[6:8], 16)),
                    (int(hex_str[8:10], 16), int(hex_str[10:12], 16), int(hex_str[12:14], 16), int(hex_str[14:16], 16)),
                    (int(hex_str[16:18], 16), int(hex_str[18:20], 16), int(hex_str[20:22], 16), int(hex_str[22:24], 16)),
                    (int(hex_str[24:26], 16), int(hex_str[26:28], 16), int(hex_str[28:30], 16), int(hex_str[30:32], 16))]
    result = [[0 for col in range(4)] for row in range(4)]

    # Need to add together after multiplying
    mix_column_matrix = [[2, 3, 1, 1],
                        [1, 2, 3, 1],
                        [1, 1, 2, 3],
                        [3, 1, 1, 2]]

    result[0][0] = addHex(by2(hex(state_matrix[0][0][2:]), by3(hex(state_matrix[0][1][2:]), hex(state_matrix[0][2][2:]), hex(state_matrix[0][3][2:]))
    result[1][0] = addHex((hex(state_matrix[0][0][2:]), by2(hex(state_matrix[0][1][2:]), by3(hex(state_matrix[0][2][2:]), hex(state_matrix[0][3][2:]))
    result[2][0] = addHex((hex(state_matrix[0][0][2:]), (hex(state_matrix[0][1][2:]), by2(hex(state_matrix[0][2][2:]), by3(hex(state_matrix[0][3][2:]))
    result[3][0] = addHex(by3(hex(state_matrix[0][0][2:]), (hex(state_matrix[0][1][2:]), (hex(state_matrix[0][2][2:]), by2(hex(state_matrix[0][3][2:]))

    result[0][1] = addHex(by2(hex(state_matrix[1][0][2:]), by3(hex(state_matrix[1][1][2:]), hex(state_matrix[1][2][2:]), hex(state_matrix[1][3][2:]))
    result[1][1] = addHex((hex(state_matrix[1][0][2:]), by2(hex(state_matrix[1][1][2:]), by3(hex(state_matrix[1][2][2:]), hex(state_matrix[1][3][2:]))
    result[2][1] = addHex((hex(state_matrix[1][0][2:]), (hex(state_matrix[1][1][2:]), by2(hex(state_matrix[1][2][2:]), by3(hex(state_matrix[1][3][2:]))
    result[3][1] = addHex(by3(hex(state_matrix[1][0][2:]), (hex(state_matrix[1][1][2:]), (hex(state_matrix[1][2][2:]), by2(hex(state_matrix[1][3][2:]))

    result[0][2] = addHex(by2(hex(state_matrix[2][0][2:]), by3(hex(state_matrix[2][1][2:]), hex(state_matrix[2][2][2:]), hex(state_matrix[2][3][2:]))
    result[1][2] = addHex((hex(state_matrix[2][0][2:]), by2(hex(state_matrix[2][1][2:]), by3(hex(state_matrix[2][2][2:]), hex(state_matrix[2][3][2:]))
    result[2][2] = addHex((hex(state_matrix[2][0][2:]), (hex(state_matrix[2][1][2:]), by2(hex(state_matrix[2][2][2:]), by3(hex(state_matrix[2][3][2:]))
    result[3][2] = addHex(by3(hex(state_matrix[2][0][2:]), (hex(state_matrix[2][1][2:]), (hex(state_matrix[2][2][2:]), by2(hex(state_matrix[2][3][2:]))

    result[0][3] = addHex(by2(hex(state_matrix[3][0][2:]), by3(hex(state_matrix[3][1][2:]), hex(state_matrix[3][2][2:]), hex(state_matrix[3][3][2:]))
    result[1][3] = addHex((hex(state_matrix[3][0][2:]), by2(hex(state_matrix[3][1][2:]), by3(hex(state_matrix[3][2][2:]), hex(state_matrix[3][3][2:]))
    result[2][3] = addHex((hex(state_matrix[3][0][2:]), (hex(state_matrix[3][1][2:]), by2(hex(state_matrix[3][2][2:]), by3(hex(state_matrix[3][3][2:]))
    result[3][3] = addHex(by3(hex(state_matrix[3][0][2:]), (hex(state_matrix[3][1][2:]), (hex(state_matrix[3][2][2:]), by2(hex(state_matrix[3][3][2:]))

    print(result)
    new = ""
    for i in range(4):
        for j in range(4):
            if(int(str(result[j][i]), 16) < 16):
                new += "0" + str(result[j][i])
            else:
                new += result[j][i]
    return(new)

```

This function uses the above three functions to mix the columns using AES's algorithms. First, it separates the hex_str input into a state matrix. An empty result list is also defined to store the final result. The mix_column_matrix is also defined for reference, but is not used within the function. The function then calculates the values of the result matrix using addHex, by2, and by3. After calculating the final result matrix, the function converts it to a string and returns it. Finally, after mixing the columns, call the add_key function one more time to add the second subkey:

```

# Read the plaintext from file
with open("../data/plaintext.txt", "r") as f:
    message = f.read().strip()

# Convert message to ASCII values
ascii_values = [ord(c) for c in message]

# Convert ASCII values to binary
binary_str = ''.join([bin(c)[2:].zfill(8) for c in ascii_values])

# Read the subkeys
with open("../data/subkey_example.txt", "r") as f:
    subkeys = [hex2bin(line.strip()) for line in f]

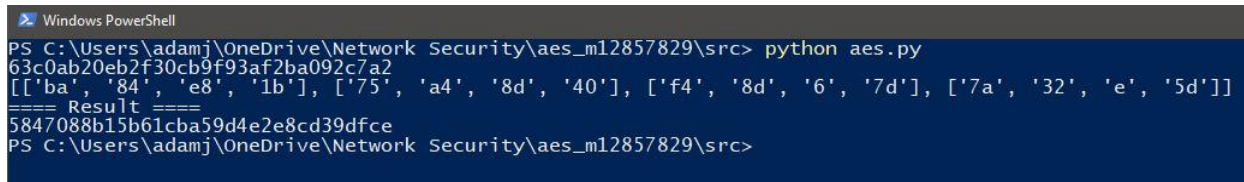
# Add first subkey
binary_str = add_key(binary_str, subkeys[0])

# Round 1
hex_str = sub_bytes(binary_str)
hex_str = shift_rows(hex_str)
hex_str = mix_columns(hex_str) # Need to finish mix_columns
bin_str = add_key(hex2bin(hex_str), subkeys[1])
print("==== Result ====")
result = bin2hex(bin_str)
print(result)

# Write result to file
with open("../data/result.txt", "w") as f:
    f.write(result)

```

Print the result to the terminal and write the result to the txt file.



```

Windows PowerShell
PS C:\Users\adamj\OneDrive\Network Security\aes_m12857829\src> python aes.py
63c0ab20eb2f30cb9f93af2ba092c7a2
[['ba', '84', 'e8', '1b'], ['75', 'a4', '8d', '40'], ['f4', '8d', '6', '7d'], ['7a', '32', 'e', '5d']]
==== Result ====
5847088b15b61cba59d4e2e8cd39dfce
PS C:\Users\adamj\OneDrive\Network Security\aes_m12857829\src>

```

Using the default values, the output is **5847088b15b61cba59d4e2e8cd39dfce**