

PROBLEM 1 - PROJECT REPORT

Implemented a brute-force search (exhaustive search) in python on AES cipher with a key space of 128 bits where the first 13 bytes are fixed to be zeros (except that the first bit is fixed to be “1”) and the last 3 bytes (24 bits) are arbitrary which were checked by exhaustive search.

Provided with three plaintext-ciphertext pairs - “m1.txt”, “c1.bin”, “m2.txt”, “c2.bin”, “m3.txt”, “c3.bin” and the respective nonces - “nonce1.bin”, “nonce2.bin”, “nonce3.bin”.

Utils_demo.py imported an AES module from Crypto.Cipher and it consists of functions to encrypt, decrypt, read and write to files. This is imported in our program AES.py. Defaulted the number of bytes to be 3 (n in the code) because we need to check for the remaining 24 bits. And so the postfix length will be 8 times the number of bytes. Using the read_file function read all the plaintext files and encode them because they are .txt files. And by using the read_bytes function read all the ciphertext and respective nonce files.

```
if __name__ == "__main__":

    parser = argparse.ArgumentParser(description='Setup for Bruteforce attack against randomized AES-128-CTR.')
    parser.add_argument('-n', type=int,
                        help='Effective key length in bytes.', default=3)

    args = parser.parse_args()
    #The input value for brute force attack in bits. 24 bits is equal to 3 bytes.
    length_postfix = args.n * 8
    #Reading plaintexts, ciphertexts, nonces from files.
    plaintext1 = utils_demo.read_file(fn = "files/m1.txt").encode()
    plaintext2 = utils_demo.read_file(fn = "files/m2.txt").encode()
    plaintext3 = utils_demo.read_file(fn = "files/m3.txt").encode()
    ciphertext1 = utils_demo.read_bytes(fn = "files/c1.bin")
    ciphertext2 = utils_demo.read_bytes(fn = "files/c2.bin")
    ciphertext3 = utils_demo.read_bytes(fn = "files/c3.bin")
    nonce1 = utils_demo.read_bytes(fn = "files/nonce1.bin")
    nonce2 = utils_demo.read_bytes(fn = "files/nonce2.bin")
    nonce3 = utils_demo.read_bytes(fn = "files/nonce3.bin")
```

As we will have 2 raised to 24 possibilities, iterate the for loop from 0 to 2**length_postfix-1. Used bin() function to convert this integer number to binary string prefixed with 0b. Call the decryptor_CTR function with ciphertext1, nonce1, key (converted to bytes) as arguments and compare the returned plaintext with the actual plaintext in m1.txt. If it returns true, then similarly check for the second one and next the third one. Now we write this key value to a bin file, and break the loop. Imported the datetime module and computed the total time elapsed using now() function.

```

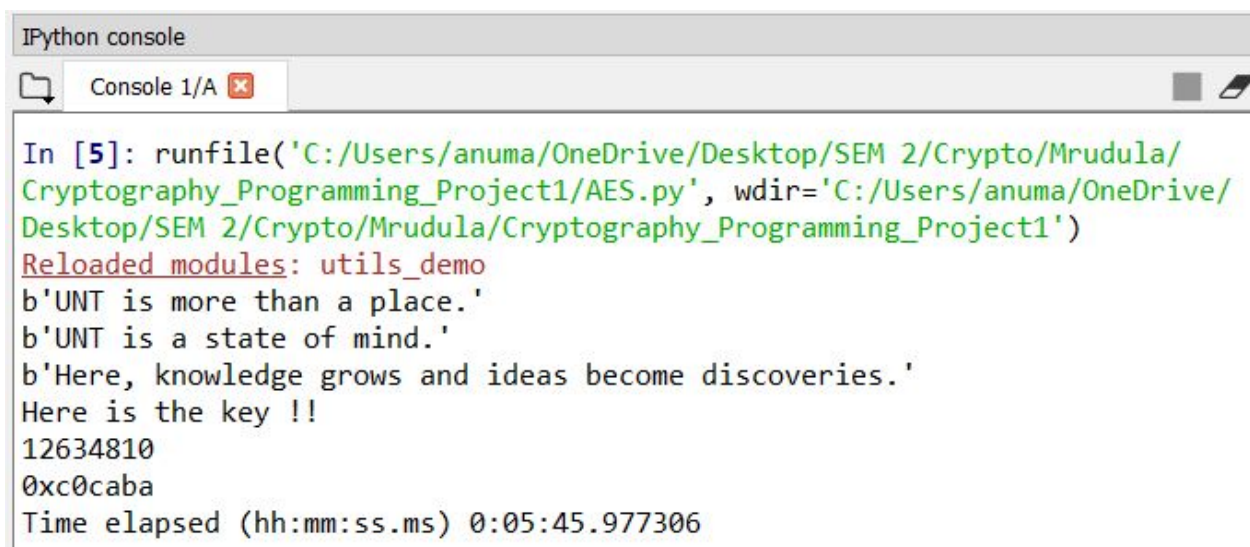
start_time = datetime.now()

for i in range(0, 2**length_postfix-1): # because we have 2 raised to 24 possibilities
    #Defining the key for decryption.
    main_key = bin(2 ** 127 + i)
    #Decrypting cipher-texts.
    ptxt1 = utils_demo.decryptor_CTR(ctxt=ciphertext1, nonce=nonce1, key=utils_demo.bitstring_to_bytes(main_key))
    if(ptxt1 == plaintext1):
        print(ptxt1)
        ptxt2 = utils_demo.decryptor_CTR(ctxt=ciphertext2, nonce=nonce2, key=utils_demo.bitstring_to_bytes(main_key))
        if(ptxt2 == plaintext2):
            print(ptxt2)
            ptxt3 = utils_demo.decryptor_CTR(ctxt=ciphertext3, nonce=nonce3, key=utils_demo.bitstring_to_bytes(main_key))
            if(ptxt3 == plaintext3):
                print(ptxt3)
                utils_demo.write_file(fn = "files/key.bin", value = main_key) # writing the key to a bin file
                print("Here is the key !!")
                print(i) # 12634810 - integer value, C0CABA - hex value is the final key
                print(hex(i))
                break
time_elapsed = datetime.now() - start_time
print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

```

As a result we got C0CABA (hex value) as the last 3 bytes of the key (integer value - 12634810).

It took around 5 min to find the respective key for the given three plaintext-ciphertext pairs.

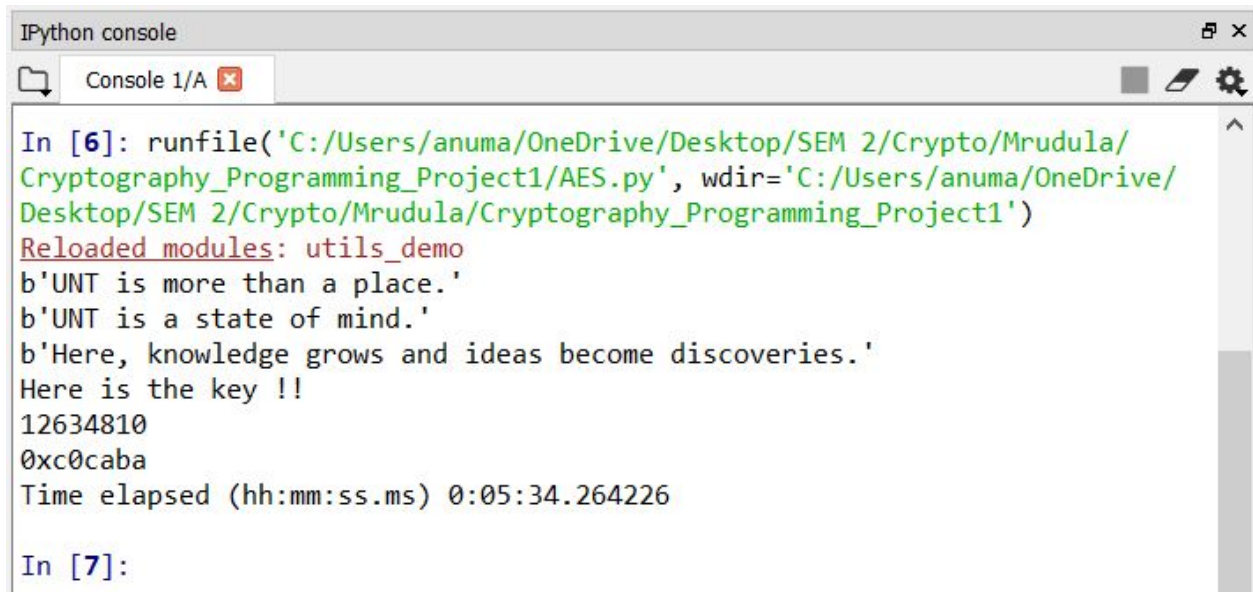


```

IPython console
Console 1/A

In [5]: runfile('C:/Users/anuma/OneDrive/Desktop/SEM 2/Crypto/Mrudula/
Cryptography_Programming_Project1/AES.py', wdir='C:/Users/anuma/OneDrive/
Desktop/SEM 2/Crypto/Mrudula/Cryptography_Programming_Project1')
Reloaded modules: utils_demo
b'UNT is more than a place.'
b'UNT is a state of mind.'
b'Here, knowledge grows and ideas become discoveries.'
Here is the key !!
12634810
0xc0caba
Time elapsed (hh:mm:ss.ms) 0:05:45.977306

```



The screenshot shows an IPython console window with a title bar 'IPython console' and a tab 'Console 1/A'. The console displays the following output for the command 'In [6]: runfile(...)':

```
In [6]: runfile('C:/Users/anuma/OneDrive/Desktop/SEM 2/Crypto/Mrudula/
Cryptography_Programming_Project1/AES.py', wdir='C:/Users/anuma/OneDrive/
Desktop/SEM 2/Crypto/Mrudula/Cryptography_Programming_Project1')
Reloaded modules: utils_demo
b'UNT is more than a place.'
b'UNT is a state of mind.'
b'Here, knowledge grows and ideas become discoveries.'
Here is the key !!
12634810
0xc0caba
Time elapsed (hh:mm:ss.ms) 0:05:34.264226

In [7]:
```

In the output console we have printed just the last 24 bits of the key because the first 104 bits are fixed. But we have written the complete key to a binary file created within the files folder.