Introduction

A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

Advanced Encryption Standard (AES) is now the widely adopted symmetric encryption algorithm. It was found to be at least six time faster than triple DES.

Some of the AES features that makes it a choice over 3-DES are listed below:

1. Symmetric key symmetric block cipher.
2. 128-bit data, 128/192/256-bit key.
3. Stronger and faster than Triple-DES
4. Provide full specification and design details
5. Software implementable in C and Java

Problem Statement

Given a set of ciphertext and nonces from the encryption of a set of plaintexts using the AES cipher in the randomized counter mode with a key space effectively limited to 24 bits. The key has the format (in hex): 80 00 … 00 XX XXXX, where X is an arbitrary hex number. In other words, the first 13 bytes of the key are fixed to be zeroes (except that the first bit is fixed to be "1"), and the last 3 bytes can be arbitrary.
This program aims to find the keys which matches the provided three plaintext-ciphertext pairs and have the key displayed in the hex format.

Methodology

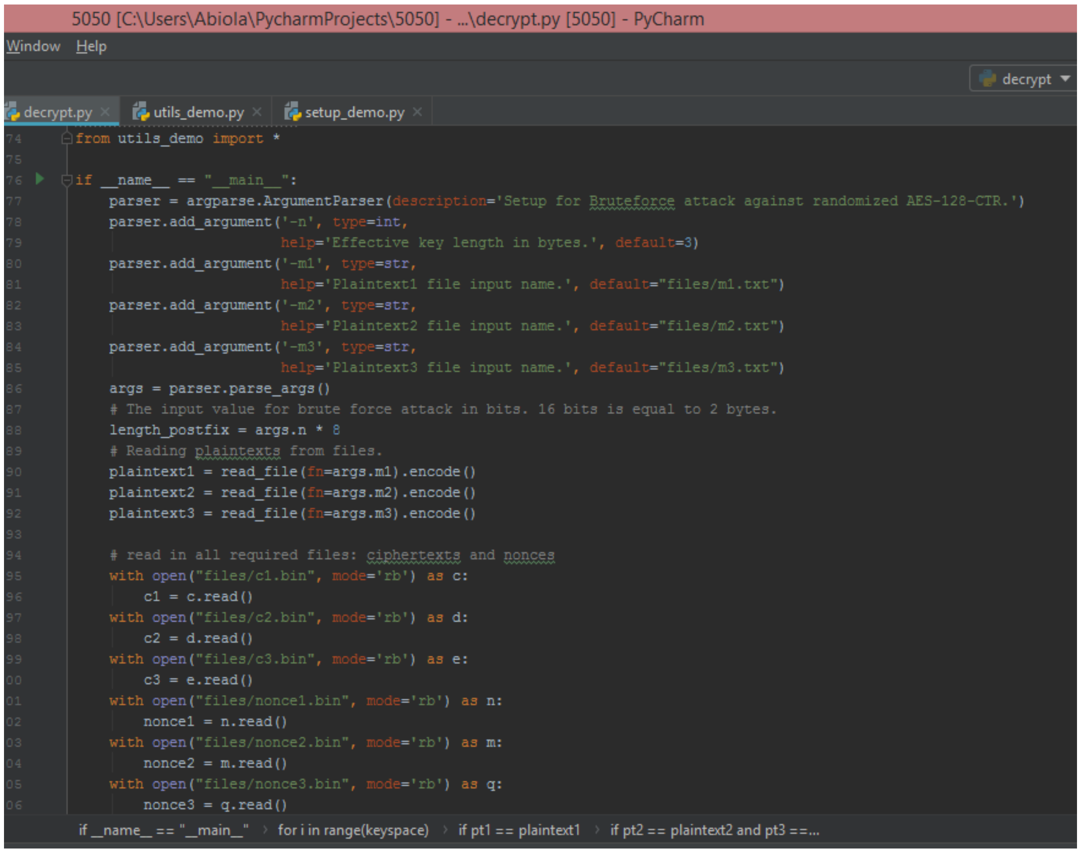
Considering the available resources, that is, cipher texts, nonces, and an information about the key space being limited to 24bits and in the format hex(80 00 … 00 XX XXXX), where the last 3 bytes is our target. A brute-force search of the key space will be carried out on the block cipher with the first cipher text and its nonce. The decryption key will be tested on the other ciphertexts, and if successful, the key is output in hexadecimal format.

The program will be written in the python programming language and the pycryptodome cryptographic library will be employed because it contains sophisticated tools that can encrypt and decrypt very fast.

For the brute-force search, we need a decryption function, this can gotten from the supplied files in the assignment folder. Necessary dependent libraries are imported and all the ciphertext and nonce files are read into different variables. We then iterate over the total key space and in each iteration we display the tested key in hexadecimal format. In each iteration, we also call the decrypt function which takes as input a ciphertext, nonce and a key, then returns a plaintext.

The plaintext returned by the decryption function is compared with the respective plaintext given in the project task. If a both decrypted plaintext and the given respective plaintext match in content, then we use the found key to decrypt other ciphertexts and also compare them with their respective given plaintexts, and if the contents are the same again, we output the found key in hexadecimal format, otherwise, the key is discarded and the next key in iteration is tested.

Results

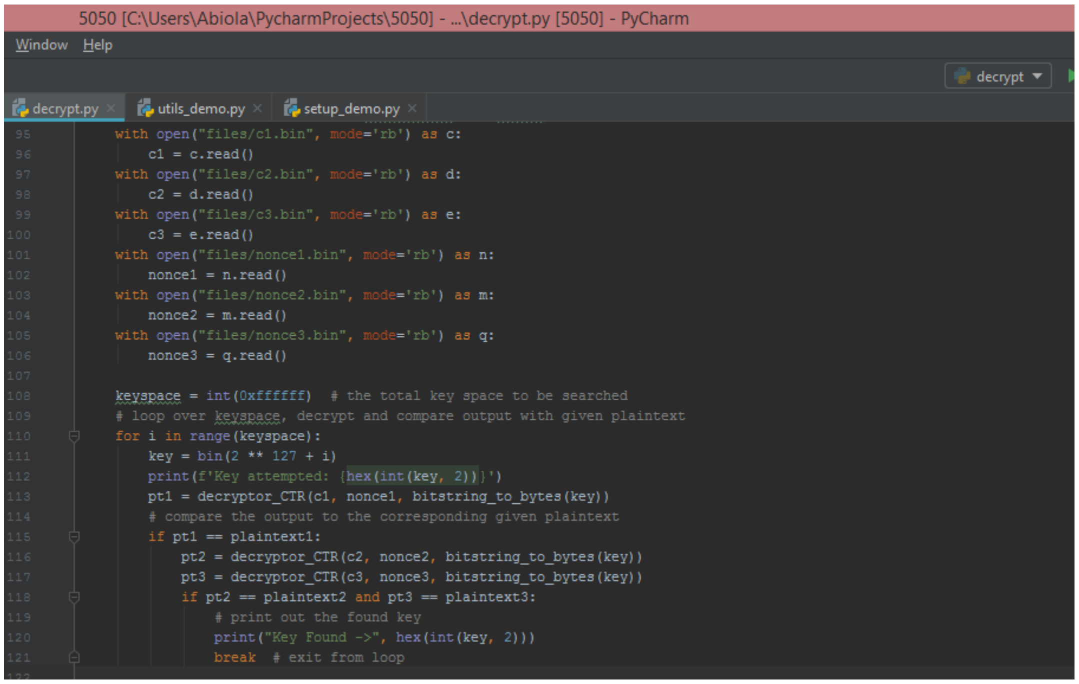

5050 [C:\Users\Abiola\PycharmProjects\5050] - ...\decrypt.py [5050] - PyCharm
Window Help
decrypt
decrypt.py
utils_demo.py
setup_demo.py
if __name__ == "__main__" › for i in range(keyspace) › if pt1 == plaintext1 › if pt2 == plaintext2 and pt3 ==...

```python
from utils_demo import *

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)
    parser.add_argument('-m1', type=str,
                        help='Plaintext1 file input name.', default="files/m1.txt")
    parser.add_argument('-m2', type=str,
                        help='Plaintext2 file input name.', default="files/m2.txt")
    parser.add_argument('-m3', type=str,
                        help='Plaintext3 file input name.', default="files/m3.txt")
    args = parser.parse_args()
    # The input value for brute force attack in bits. 16 bits is equal to 2 bytes.
    length_postfix = args.n * 8
    # Reading plaintexts from files.
    plaintext1 = read_file(fn=args.m1).encode()
    plaintext2 = read_file(fn=args.m2).encode()
    plaintext3 = read_file(fn=args.m3).encode()

    # read in all required files: ciphertexts and nonces
    with open("files/c1.bin", mode='rb') as c:
        c1 = c.read()
    with open("files/c2.bin", mode='rb') as d:
        c2 = d.read()
    with open("files/c3.bin", mode='rb') as e:
        c3 = e.read()
    with open("files/nonce1.bin", mode='rb') as n:
        nonce1 = n.read()
    with open("files/nonce2.bin", mode='rb') as m:
        nonce2 = m.read()
    with open("files/nonce3.bin", mode='rb') as q:
        nonce3 = q.read()
```

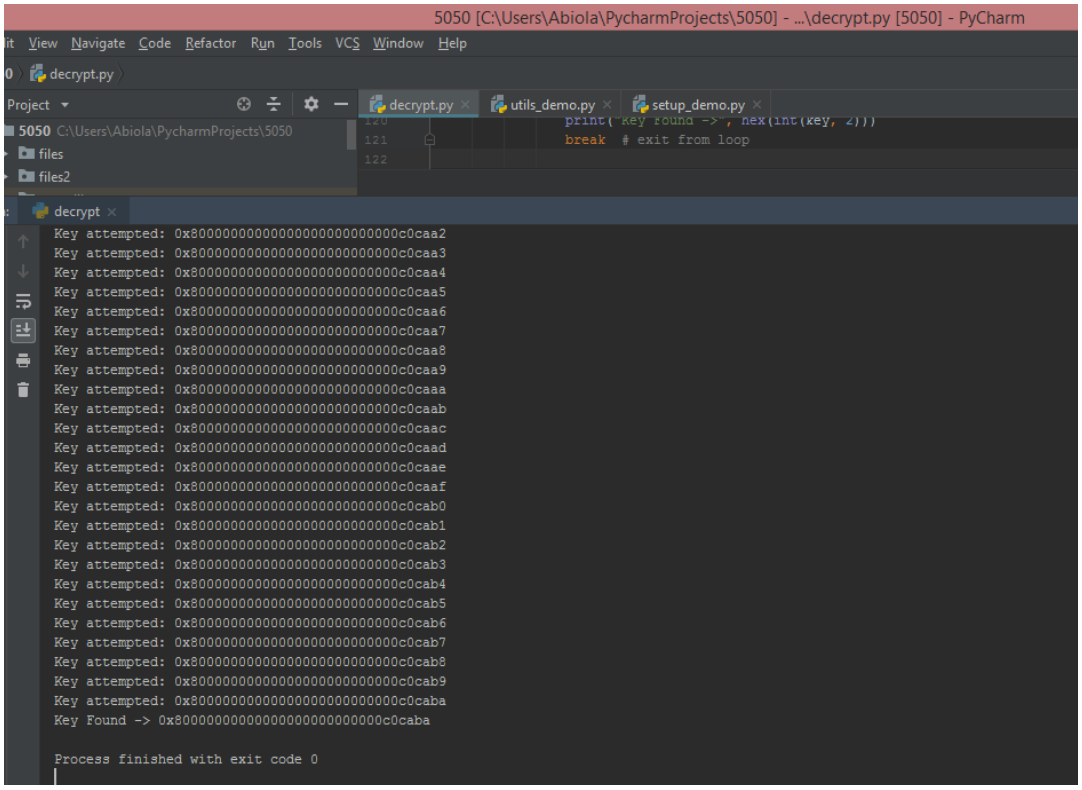
Figure 1a. The Decryption program showing the import and reading of all required files


5050 [C:\Users\Abiola\PycharmProjects\5050] - ...\decrypt.py [5050] - PyCharm
Window Help
decrypt
decrypt.py
utils_demo.py
setup_demo.py

```python
    with open("files/c1.bin", mode='rb') as c:
        c1 = c.read()
    with open("files/c2.bin", mode='rb') as d:
        c2 = d.read()
    with open("files/c3.bin", mode='rb') as e:
        c3 = e.read()
    with open("files/nonce1.bin", mode='rb') as n:
        nonce1 = n.read()
    with open("files/nonce2.bin", mode='rb') as m:
        nonce2 = m.read()
    with open("files/nonce3.bin", mode='rb') as q:
        nonce3 = q.read()

    keyspace = int(0xffffff)  # the total key space to be searched
    # loop over keyspace, decrypt and compare output with given plaintext
    for i in range(keyspace):
        key = bin(2 ** 127 + i)
        print(f'Key attempted: {hex(int(key, 2))}')
        pt1 = decryptor_CTR(c1, nonce1, bitstring_to_bytes(key))
        # compare the output to the corresponding given plaintext
        if pt1 == plaintext1:
            pt2 = decryptor_CTR(c2, nonce2, bitstring_to_bytes(key))
            pt3 = decryptor_CTR(c3, nonce3, bitstring_to_bytes(key))
            if pt2 == plaintext2 and pt3 == plaintext3:
                # print out the found key
                print("Key Found ->", hex(int(key, 2)))
                break  # exit from loop
```

Figure 1b. The Decryption program showing the loop code over the key space


5050 [C:\Users\Abiola\PycharmProjects\5050] - ...\decrypt.py [5050] - PyCharm
it View Navigate Code Refactor Run Tools VCS Window Help
0 decrypt.py
Project
decrypt.py
utils_demo.py
setup_demo.py
5050 C:\Users\Abiola\PycharmProjects\5050
files
files2
decrypt

```
print("Key Found ->", hex(int(key, 2)))
break  # exit from loop
```

```
Key attempted: 0x8000000000000000000000000c0caa2
Key attempted: 0x8000000000000000000000000c0caa3
Key attempted: 0x8000000000000000000000000c0caa4
Key attempted: 0x8000000000000000000000000c0caa5
Key attempted: 0x8000000000000000000000000c0caa6
Key attempted: 0x8000000000000000000000000c0caa7
Key attempted: 0x8000000000000000000000000c0caa8
Key attempted: 0x8000000000000000000000000c0caa9
Key attempted: 0x8000000000000000000000000c0caaa
Key attempted: 0x8000000000000000000000000c0caab
Key attempted: 0x8000000000000000000000000c0caac
Key attempted: 0x8000000000000000000000000c0caad
Key attempted: 0x8000000000000000000000000c0caae
Key attempted: 0x8000000000000000000000000c0caaf
Key attempted: 0x8000000000000000000000000c0cab0
Key attempted: 0x8000000000000000000000000c0cab1
Key attempted: 0x8000000000000000000000000c0cab2
Key attempted: 0x8000000000000000000000000c0cab3
Key attempted: 0x8000000000000000000000000c0cab4
Key attempted: 0x8000000000000000000000000c0cab5
Key attempted: 0x8000000000000000000000000c0cab6
Key attempted: 0x8000000000000000000000000c0cab7
Key attempted: 0x8000000000000000000000000c0cab8
Key attempted: 0x8000000000000000000000000c0cab9
Key attempted: 0x8000000000000000000000000c0caba
Key Found -> 0x8000000000000000000000000c0caba

Process finished with exit code 0
```

Figure 2. The tested keys and the Found Key shown at the bottom

From Figure two above, it can be seen that after searching through the key space, the program was able to successfully detect the decryption key as **0x80000000000000000000000000000c0caba.**

Conclusion

We were able to successfully show that given a set of ciphertext and nonces, with an information about the key space, it is possible to use a brute-force search over the key space to decrypt ciphertexts from an encryption block cipher.