CSCE 4050/5050

Applications of Cryptography

Programming Project 1 –

Exhaustive key search

By:

Group 4

Shashidhar Kalapatapu – 11545863

Dharmateja Kollipara - 11555465

Project Task:

The objective of the project is to write a program to run the exhaustive search attack on the AES-128 Block cipher. We will be using AES-128 Cipher in the randomized Counter Mode (CTR) with the key space size as 24 bits. Provided 3 Plaintext-Ciphertext pairs, we need to find a key which satisfies these 3 pairs and use it to decrypt the challenge ciphertext.

There are two tasks that will be done in this project.

1. Find the key which satisfies the three plaintext-ciphertext pairs and write this to a file.
2. Use the key from file to decrypt the challenge ciphertext.

Project Description:

Three ciphertexts (c1.bin, c2.bin, c3.bin) and their corresponding plaintexts (m1.txt, m2.txt, m3.txt) along with their nonce files (nonce1, nonce2, nonce3) are given. These were encrypted using AES-128 cipher running in CTR mode. We were given a hint about the key. Key is in hexadecimal format. Out of the 16 bytes, the first 13 bytes are fixed but the last 3 bytes are hidden. Moreover, the leading byte is '80'.

The Key format (in Hex):

80 00 00 00 00 00 00 00 00 00 00 00 00 **XX XX XX**

We need to search for all possible combinations of the above key using exhaustive search. Each byte above can take values from '00' to 'FF'. In other words, 3 bytes is 24 bits and we will have to search 2^{24} possible combinations till we find our key. Once we find the key, we will be saving it in a file. This is our first task.

In the second task, we will decrypt another ciphertext called challenge ciphertext (c_c.bin) using the above key and its corresponding nonce (nonce_c.bin). The resulting plaintext will also be saved to a result file.

Methodology

- **Exhaustive Key Search**

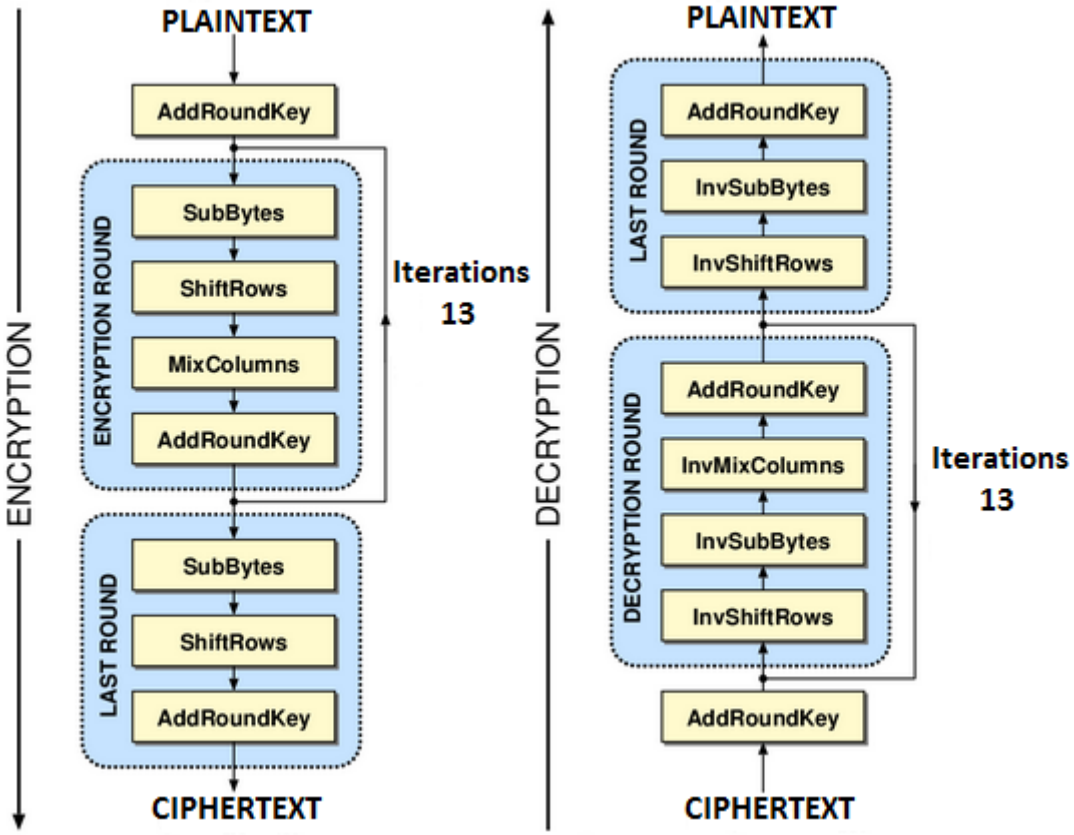
Exhaustive Key Search is a simple and straightforward approach for finding the key of a given encryption by trying every possible value [1]. This approach can be referred to as Brute Force Attack. For an attacker to perform a successful exhaustive key search, they need to have as much information as possible about the plaintext.

The time taken to complete exhaustive key search may depend on the total number of possible keys (K), the time (t) to test one key, and the number of processors (p) working in searching the keys. On average it takes about half way through the search thus making the expected run time approx. **Kt/(2p).**

- **AES-128**

Advanced Encryption Standard is a block cipher used as a standard for encrypting electronic data [2]. It is based on a design principle using substitution-permutation networks. It has a fixed block size of 128 bits and a key size of 128, 192 or 256 bits. The key size is the number of rounds of transformation it applies on the input plaintext and generates an output ciphertext.

For AES-128, it runs 10 rounds. For 192-bit or bigger keys it can be 12 -13 rounds. (Fig 1.1).

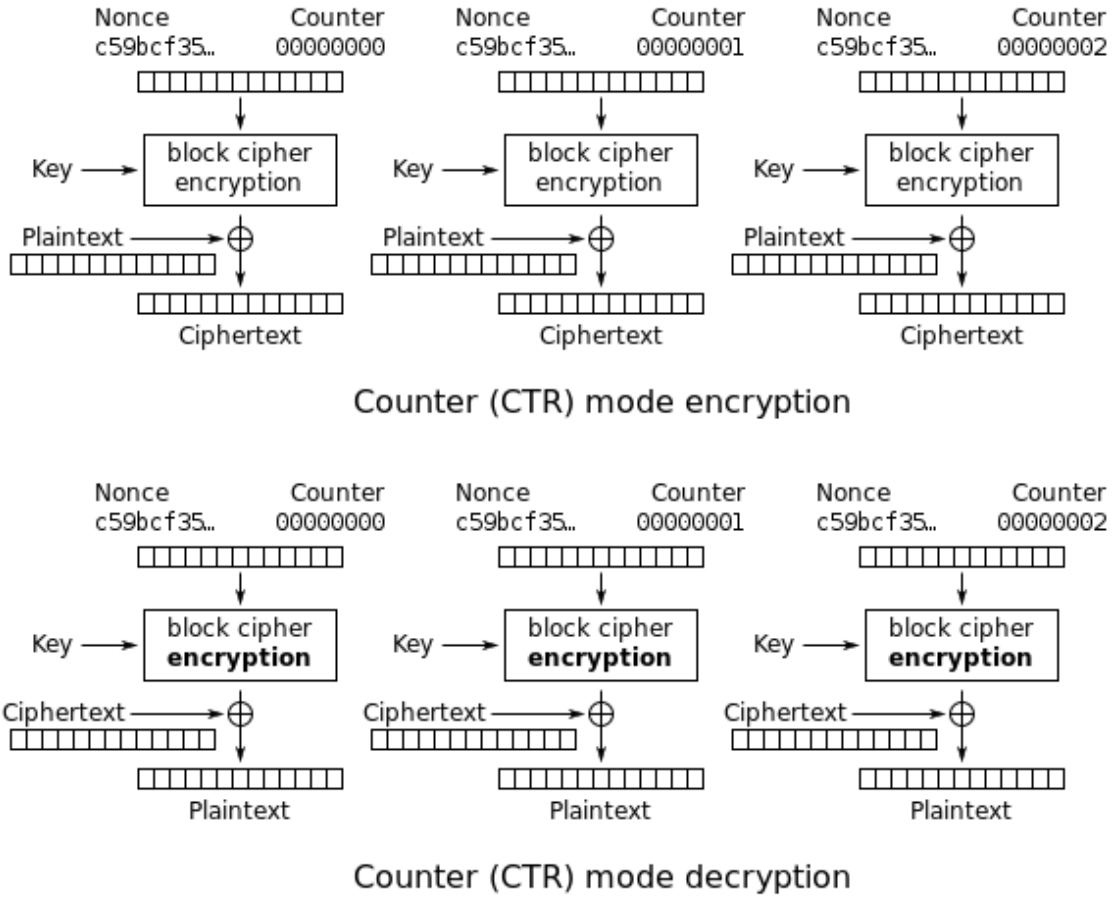


Encryption and Decryption in AES-128

(Image Source: https://www.researchgate.net/figure/Architecture-of-AES-Algorithm_fig3_301644181)

Counter (CTR) Mode:

Counter mode in AES generates the next keystream block by encrypting the successive values of a “counter”. [3] This counter is a function producing non-repeating sequences. Using this mode, a nonce and the counter value are combined with a key. This combination is then XORed with one block plaintext to get one block of ciphertext.



(Image Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation)

Program Description:

Prerequisite Libraries:

Prerequisite Libraries

```
]: 1 !pip3 install pycryptodomex
   2 !pip3 install tqdm
```

```
Requirement already satisfied: pycryptodomex in c:\users\admin\anaconda3\lib\site-packages (3.17)

WARNING: You are using pip version 22.0.4; however, version 23.0.1 is available.
You should consider upgrading via the 'C:\Users\admin\anaconda3\python.exe -m pip install --upgrade pip' command.

Requirement already satisfied: tqdm in c:\users\admin\anaconda3\lib\site-packages (4.62.3)
Requirement already satisfied: colorama in c:\users\admin\anaconda3\lib\site-packages (from tqdm) (0.4.4)

WARNING: You are using pip version 22.0.4; however, version 23.0.1 is available.
You should consider upgrading via the 'C:\Users\admin\anaconda3\python.exe -m pip install --upgrade pip' command.
```

- **Pycryptodomex [4]:** Python package containing many implementations of encryption algorithms such as SHA-3, AES, Salsa20, ChaCha20 etc.
- **tqdm [5]:** python package which displays a progress bar and is implemented as a wrapper for a iterable object.

These 2 libraries are imported into our program

Imports:

Imports

```
1 from Cryptodome.Cipher import AES
2 from tqdm import tqdm
```

- **AES Class** which implements AES from the Cryptodome Cipher package.
- **Tqdm** implements the progress bar functionality.

Utility Functions:

Utility functions

```
1  # This is the encryption function. It gets a plaintext a key and returns the ciphertext and nonce.
2  def encryptor_CTR(message, key, nonce = None):
3      cipher = AES.new(key, AES.MODE_CTR)
4      ct = cipher.encrypt(message)
5      nonce = cipher.nonce
6      return nonce, ct
7
8
9  # This is the decryption funtion. It gets a ciphertext, a nonce and a key and returns the plaintext.
10 def decryptor_CTR(ctxt, nonce, key):
11     try:
12         cipher = AES.new(key, AES.MODE_CTR, nonce=nonce)
13         pt = cipher.decrypt(ctxt)
14         return pt
15     except ValueError as KeyError:
16         return None
17
18
19 # Converting string to bytes.
20 def string_to_bytes(string_value):
21     return bytearray(string_value,
```

```
18
19  # Converting string to bytes.
20  def string_to_bytes(string_value):
21      return bytearray(string_value,
22                       encoding='utf-8')
23
24  def read_file(fn):
25      f = open(fn, "r")
26      value = f.read()
27      f.close()
28      return value
29
30  def read_bytes(fn):
31      f = open(fn, "rb")
32      value = f.read()
33      f.close()
34      return value
35
36
37  def write_file(fn, value):
38      f = open(fn, "w")
39      f.write(value)
40      f.close()
41
42  def write_bytes(fn, value):
43      f = open(fn, "wb")
44      f.write(value)
45      f.close()
```

```
40      f.close()
41
42  def write_bytes(fn, value):
43      f = open(fn, "wb")
44      f.write(value)
45      f.close()
46
47
48  def bitstring_to_bytes(s):
49      v = int(s, 2)
50      b = bytearray()
51
52      while v:
53          b.append(v & 0xff)
54          v >>= 8
55      return bytes(b[::-1])
```

These functions were provided along with the problem description. We are simply using some of them. Mainly we will be using these 4 functions:

- read_file(): reads .txt files and returns the contents.

- write_file(): writes content to a .txt file.
- read_bytes(): reads .bin files and return value as a bytes object.
- write_bytes(): writes bytes object to a .bin file.

Exhaustive Key Search function:

AES-128 Exhaustive Key Search Function

In [22]:
```
def exhaustiveKeySearch(files, keyPrefix, lastXBytes):
    lengthPostfix = lastXBytes * 8
    keySpace = 2 ** lengthPostfix
    print("EXHAUSTIVE KEY SEARCH OF AES-128")
    print("--------------------------------")
    print()
    print("The First 13 bytes of Hexadecimal Key are fixed.")

    print("80 00 00 00 00 00 00 00 00 00 00 00 00 XX XX XX")
    print("For the last 3 bytes, all combinations are searched from 00 to FF")
    print()
    print("Brute Force searching through ", keySpace, "possibilities ...")
    for i in tqdm(range(keySpace), desc='Progress '):
        # Define our hex key as a 'bytes' object
        key = b'\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' + i.to_bytes(lastXBytes, byteorder="big")

        # Construct and Initialize three AES-128 Ciphers in CTR Mode using the given three nonce bin files.
        c1 = AES.new(key, AES.MODE_CTR, nonce=files[1]['nonce'])
        c2 = AES.new(key, AES.MODE_CTR, nonce=files[2]['nonce'])
        c3 = AES.new(key, AES.MODE_CTR, nonce=files[3]['nonce'])

```


This function implements the exhaustive key search attack for our AES-128 cipher. We define the keyspace and the last bytes of HEX key which we need to find out. For our case, we need to find out the last 3 bytes, so 3 x 8 = 24 bits. It is stored in lengthPostfix variable.

Then keyspace is a variable which takes 2^{24} denoting that many possible key combinations. As the last 3 bytes are arbitrary, we search through all possible combinations in the keyspace.

We define a loop where these steps are performed:

1. define a hex key of 16 bytes where first 13 are fixed, last 3 are set to the current number in the keyspace.
2. Initialize three AES cipher objects in CTR mode using the key and the respective nonce files.
3. Generate decryptions of the 3 ciphertexts from files using cipher.decrypt method.
4. Check if these 3 decryptions are equal to the respective plaintext files. If they are equal, Then the key is found.
5. Print the key and write this key to a .bin and .txt file.

Driver Code:

In the driver code, we set the path variable to the location where our ciphertexts and plaintexts files are located.

We created a files dictionary object, which contains the ciphertext, nonce, and the plaintext file name of the 3 files and the challenge files.

```
# Driver Code. Here the execution starts
if __name__ == "__main__":
    # PATH of the folder where the bin and txt files should be set here
    path = 'D:/UNT Class Work/Spring 2023/CSCE 5050 Applications of Cryptography/Programming Project 1/'
    files = {
        1: {
            'ctxt': read_bytes(path + "c1.bin"),
            'nonce': read_bytes(path + "nonce1.bin"),
            'txt': read_file(path + "m1.txt").encode()
        },

        2: {
            'ctxt': read_bytes(path + "c2.bin"),
            'nonce': read_bytes(path + "nonce2.bin"),
            'txt': read_file(path + "m2.txt").encode()
        },

        3: {
            'ctxt': read_bytes(path + "c3.bin"),
            'nonce': read_bytes(path + "nonce3.bin"),
            'txt': read_file(path + "m3.txt").encode()
        },
        "challenge": {
            'ctxt': read_bytes(path + "c_c.bin"),
            'nonce': read_bytes(path + "nonce_c.bin"),
        }
    }
```

```
        2: {
            'ctxt': read_bytes(path + "c2.bin"),
            'nonce': read_bytes(path + "nonce2.bin"),
            'txt': read_file(path + "m2.txt").encode()
        },

        3: {
            'ctxt': read_bytes(path + "c3.bin"),
            'nonce': read_bytes(path + "nonce3.bin"),
            'txt': read_file(path + "m3.txt").encode()
        },
        "challenge": {
            'ctxt': read_bytes(path + "c_c.bin"),
            'nonce': read_bytes(path + "nonce_c.bin"),
        }
    }

    # The first 13 bytes keyprefix which was already given in the problem question
    keyPrefix = b'\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

    #The last 3 bytes of our key are arbitrary. We need to find these 3 bytes
    lastXBytes = 3

    #Start our exhaustive search
    exhaustiveKeySearch(files, keyPrefix, lastXBytes)

    #Above code will run the search and write the key to a bin and text file.
```


EXHAUSTIVE KEY SEARCH OF AES-128

We define the key prefix as 13 bytes and the lastXBytes as 3 which denotes the last 3 bytes to be found.

Finally, we call our exhaustive keysearch method with these parameters.

DEMO SCREENSHOTS:

Task 1: Finding the key for the 3 ciphertext-plaintext pairs and write the key to file.

```
lastXBytes = 3

#Start our exhaustive search
exhaustiveKeySearch(files, keyPrefix, lastXBytes)

#Above code will run the search and write the key to a bin and text file.
```

```
EXHAUSTIVE KEY SEARCH OF AES-128
--------------------------------

The First 13 bytes of Hexadecimal Key are fixed.
80 00 00 00 00 00 00 00 00 00 00 00 00 XX XX XX
For the last 3 bytes, all combinations are searched from 00 to FF

Brute Force searching through  16777216 possibilities ...
Progress :  37%|████████████████                                  | 6286948/16777216 [06:01<10:02, 17407.51it/s]
Key found (in HEXADECIMAL): 8000000000000000000000005fee64
Key Written to the path:  D:/UNT Class Work/Spring 2023/CSCE 5050 Applications of Cryptography/Programming Project 1/key.bin
```

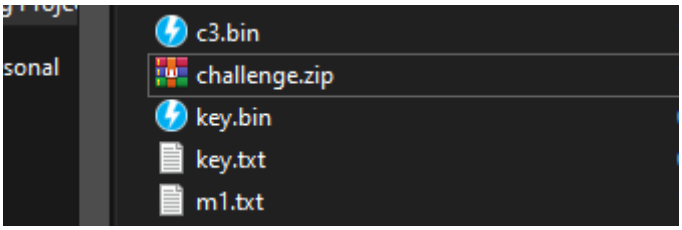
We call the exhaustive key search function in the driver code. This will loop through the key space, testing all possible combinations of keys and when it finds a key which matches with all the 3 ciphertext-plaintext pairs, it writes this to a file and the loop breaks.

As shown in the screenshot, Key was found after checking 6286947 values.

```
Key found (in HEXADECIMAL): 8000000000000000000000005fee64
```

KEY (in HEX): **8000000000000000000000005fee64**

We write this key to a .bin file and .txt file



Task 2: Program to decrypt the Challenge Cipher Text

Short Program to Decrypt the Challenge Cipher Text

```
|:  1  # Program to Decrypt the Challenge Cipher Text
    2  print("Reading Key from the path: \n", path + "key.bin")
    3  print()
    4  key = read_bytes(fn = path + "key.bin")
    5
    6  print("Reading Challenge Nonce from the path: \n", path + "nonce_c.bin")
    7  print()
    8
    9  print("Reading Challenge Cipher Text from the path: \n", path + "c_c.bin")
   10  print()
   11
   12  print("Decrypting Challenge plaintext using the key ... ")
   13  print()
   14
   15  challengeCipher = AES.new(key, AES.MODE_CTR, nonce=files["challenge"]['nonce'])
   16  challengePlaintext = challengeCipher.decrypt(files["challenge"]['ctxt'])
   17
   18  print("Decrypted Challenge Plaintext: \n", challengePlaintext)
   19  print()
   20
   21  print("Writing Challenge Plaintext file to this path ...\n", path + "c_m.txt")
   22  write_file(fn = path + "c_m.txt", value = str(challengePlaintext))
```

```
20
21 print("Writing Challenge Plaintext file to this path ...\n", path + "c_m.txt")
22 write_file(fn = path + "c_m.txt", value = str(challengePlaintext))
```

```
Reading Key from the path:
 D:/UNT Class Work/Spring 2023/CSCE 5050 Applications of Cryptography/Programming Project 1/key.bin

Reading Challenge Nonce from the path:
 D:/UNT Class Work/Spring 2023/CSCE 5050 Applications of Cryptography/Programming Project 1/nonce_c.bin

Reading Challenge Cipher Text from the path:
 D:/UNT Class Work/Spring 2023/CSCE 5050 Applications of Cryptography/Programming Project 1/c_c.bin

Decrypting Challenge plaintext using the key ...

Decrypted Challenge Plaintext:
 b'UNT is a community of dreamers and doers.'

Writing Challenge Plaintext file to this path ...
 D:/UNT Class Work/Spring 2023/CSCE 5050 Applications of Cryptography/Programming Project 1/c_m.txt
```

```
In [ ]: 1
```

We have written a small program which decrypts the Challenge cipher text file. It loads the challenge ciphertext bin file, its corresponding nonce file and also the ‘key.bin’ file which we created earlier.

Using these 3 files, we create a new AES cipher object and store it in challengeCipher. Then we call the decrypt method from this challengeCipher variable.

DECRYPTED PLAINTEXT MESSAGE:

After decryption, we get the plaintext as:

```
Decrypted Challenge Plaintext:
 b'UNT is a community of dreamers and doers.'
```

PLAINTEXT: **“UNT is a community of dreamers and doers**

References:

[1] Exhaustive Key Search – [https://link.springer.com/referenceworkentry/10.1007/0-387-23483-7_147]

[2] AES – [https://en.wikipedia.org/wiki/Advanced_Encryption_Standard]

[3] CTR Block cipher mode of operation

[https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation]

[4] PyCryptoDomex Documentation – [https://pypi.org/project/pycryptodomex/]

[5] tqdm Documentation – [https://tqdm.github.io/]

Submission Details:

The submission will include these

1. Project report .docx file
2. Code Files .zip file (This includes the .ipynb file, key .bin file, decrypted plaintext file and the other files given with the problem)