CSCE 5050/4050 Programming Project 1 Report

Karxyriah Ashley and Chance Currie
Group 2

Task Description

The task at hand is to write a program that can run an exhaustive search on a block cipher. This means that we want to essentially "brute force" finding out what the key is in order to be able to use it to decrypt some encoded message. For this task, we know that the example we are given utilizes the AES-128 cipher in the randomized counter mode with a key space that is limited to only 24 bits. We are given the plaintext message, which is split into 3 different parts labeled as m1, m2, and m3. We are given the respective ciphertexts that are made from m1, m2, and m3, which are denoted as c1, c2, and c3. We are also given the nonce which is split up into 3 parts and denoted as nonce1, nonce2, and nonce3. Our initial task is to find the key, and our challenge task is to use the key that we found and nonce_c.bin to decrypt what is inside of c_c.bin.

Program Description and Algorithms

The main.py program will utilize functions that are found within Pycryptodome, a custom cryptography Python package, and functions that are provided by the professor inside of utils_demo.py.

We first begin by creating our own helper functions. The first helper function seen is a print function. As we will be printing out specific information many times, it's much more convenient to have a simple print function instead of doing many print statements over and over. The second helper function is an update key function, and this will increment our possible key value by 1 if the key we try to guess is incorrect.

Next, we want to read in the information from the text and bin files. We read in the plaintext files, the ciphertext files, and the nonce files, and we print those values to the console through the helper print function to ensure that proper file reading has occurred.

Once the files have been read in properly, we now need to begin setting values for our two key values: our full key value and our partial key value. As stated in the task description, the AES-128 cipher is used. AES-128 encrypts the plaintext by using a key length of 128 bits, so this means that our full key value must also be a length of 128 bits, and we set that in the code as the variable full_key_val. For our partial key value, in the task description, we are told that the key space is limited to 24 bits. Therefore, we need to set our partial_key_val variable to have a length of 24 bits. We then create a byte string using the 128 bit key that we have created and print out its value. Then we set the current key to be the value of the full key.

We now must begin brute forcing all key possibilities, and this is where our algorithm of finding the key comes into play. We have a set range of values that we need to check, where the

variable partial_key_val is the maximum number in the range, so we begin looping through the possible key values. In our loop, we first want to try decrypting the first ciphertext, c1, by using the decryptor_CTR function provided in utils_demo.py and the current key that we want to try decrypting with. If the decrypted messages matches the given plaintext message, which is denoted by m1, we then want to try decrypting ciphertexts c2 and c3, as the key used to find c1 is most likely the same key that will decrypt c2 and c3. With that in mind, we see if the decoded messages c2 and c3 match their given plaintext messages, and if they do, we then move onto the challenge message and decrypt it using our current key. We then print the decrypted messages and keys, write that information to some files, and the program should finish. However, if the decoded message is not the same as the given plaintext message, meaning that the current key is not the key that we need to decrypt the messages, we then use our helper key function to increment our current key by a value of 1, and we repeat the loop.

Key and Decrypted Message

The following pieces of information are the key that was found, as well as the decrypted plaintext messages including the challenge message.

The 24-bit key (in bits): 010111111110111001100100

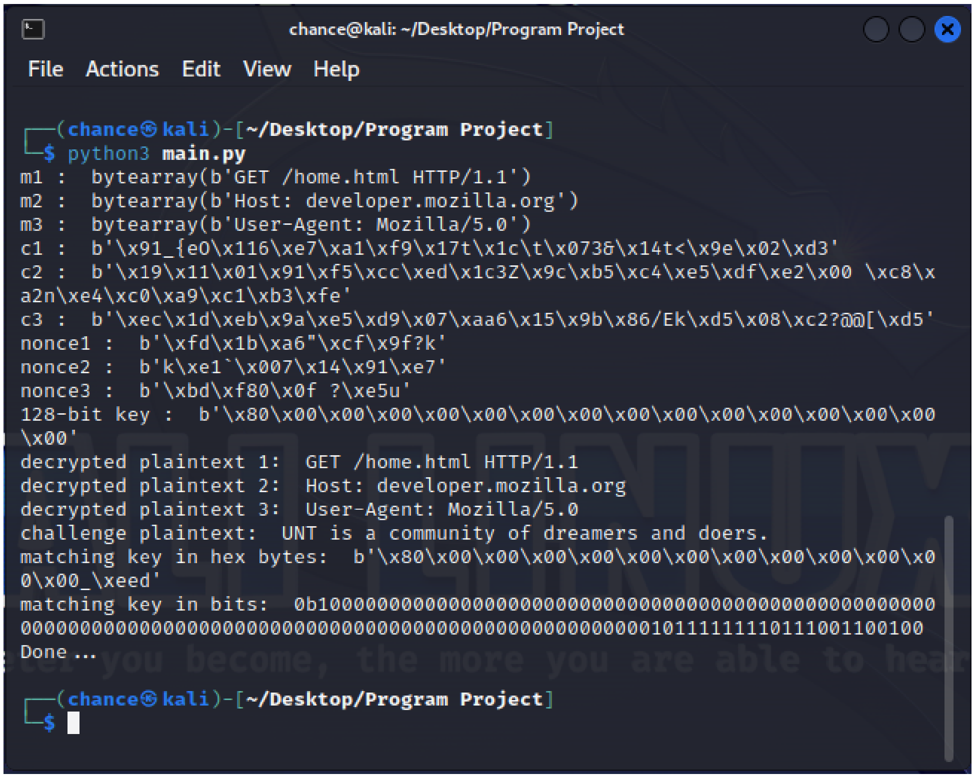
Plaintext 1: GET /home.html HTTP/1.1
Plaintext 2: Host: developer.mozilla.org
Plaintext 3: User-Agent: Mozilla/5.0
Challenge Plaintext: UNT is a community of dreamers and doers.

Screenshots

The Program is Started, Searching for the Correct Key

```
┌──(chance㉿kali)-[~/Desktop/Program Project]
└─$ python3 main.py
m1 :  bytearray(b'GET /home.html HTTP/1.1')
m2 :  bytearray(b'Host: developer.mozilla.org')
m3 :  bytearray(b'User-Agent: Mozilla/5.0')
c1 :  b'\x91_{eO\x116\xe7\xa1\xf9\x17t\x1c\t\x073&\x14t<\x9e\x02\xd3'
c2 :  b'\x19\x11\x01\x91\xf5\xcc\xed\x1c3Z\x9c\xb5\xc4\xe5\xdf\xe2\x00 \xc8\x
a2n\xe4\xc0\xa9\xc1\xb3\xfe'
c3 :  b'\xec\x1d\xeb\x9a\xe5\xd9\x07\xaa6\x15\x9b\x86/Ek\xd5\x08\xc2?@@[\xd5'
nonce1 :  b'\xfd\x1b\xa6"\xcf\x9f?k'
nonce2 :  b'k\xe1`\x007\x14\x91\xe7'
nonce3 :  b'\xbd\xf80\x0f ?\xe5u'
128-bit key :  b'\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00'
```

The program has now finished completely. We have decrypted the message, found the plaintext messages, cracked the challenge plaintext, and found the matching key


chance@kali: ~/Desktop/Program Project
File Actions Edit View Help
┌──(chance㉿kali)-[~/Desktop/Program Project]
└─$ python3 main.py
m1 : bytearray(b'GET /home.html HTTP/1.1')
m2 : bytearray(b'Host: developer.mozilla.org')
m3 : bytearray(b'User-Agent: Mozilla/5.0')
c1 : b'\x91_{eO\x116\xe7\xa1\xf9\x17t\x1c\t\x073&\x14t<\x9e\x02\xd3'
c2 : b'\x19\x11\x01\x91\xf5\xcc\xed\x1c3Z\x9c\xb5\xc4\xe5\xdf\xe2\x00 \xc8\x
a2n\xe4\xc0\xa9\xc1\xb3\xfe'
c3 : b'\xec\x1d\xeb\x9a\xe5\xd9\x07\xaa6\x15\x9b\x86/Ek\xd5\x08\xc2?@@[\xd5'
nonce1 : b'\xfd\x1b\xa6"\xcf\x9f?k'
nonce2 : b'k\xe1`\x007\x14\x91\xe7'
nonce3 : b'\xbd\xf80\x0f ?\xe5u'
128-bit key : b'\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00'
decrypted plaintext 1: GET /home.html HTTP/1.1
decrypted plaintext 2: Host: developer.mozilla.org
decrypted plaintext 3: User-Agent: Mozilla/5.0
challenge plaintext: UNT is a community of dreamers and doers.
matching key in hex bytes: b'\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00_\xeed'
matching key in bits: 0b1000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000010111111110111001100100
Done ...
┌──(chance㉿kali)-[~/Desktop/Program Project]
└─$