**CSCE 4050/5050 Applications of Cryptography**

**Programming Project 1 - Exhaustive key search**

**Group 9**

**By:**

**Govind Naidu Pulakhandam– 11479985**

**Nitin Varma Siruvuri - 11513261**

**Project Description:**

The main idea of the project is that we are using the AES (Advanced encryption standard in the CTR mode for the purpose of encryption and decryption of data. The AES is a symmetric key encryption technique which is widely used which uses a unique same key for both encryption and decryption.

Exhaustive Key Search: It is defined as the concept of finding the key from the given cipher to get the plain text by the means of trying multiple combination of keys to find the actual key that is used to encrypt the messages. (It can also be called as the brute force method). The time to execute the exhaustive key search is mainly dependent on the number of possible keys(k), time taken(t) to test each key and the number of processors(p). So, the approximate time would be Kt/p to test all the keys

In our program, the main objective is to analyse the plain and cipher texts to find the key that is used to encrypt the message and decrypt the messages.

**Project Given Data:** In this project to make our task simpler, we have been provided with the “util\_demo.py” and the “setup\_demo.py” files which contain methos that are used to read and write to files and a basic code on how the encryption of the files has been performed. So, we have used this information in our code as suggested to find the key that is used to encrypt all the 3 files.

The given condition is that we are provided with the 3 cipher text files namely c1.bin, c2.bin, c3.bin and the messages corresponding to the cipher texts are m1.txt, m2.txt, and m3.txt. The nonce files associated with them are nonce1.bin, nonce2.bin and nonce3.bin.

Another given information is that the key length is of 16 bytes and the first 13 bytes follow the pattern as “\x80” as the first byte and the next 12 bytes as “\x00” leaving the last 3 bytes to be arbitrary.

So the key will look like **80 00 00 00 00 00 00 00 00 00 00 00 00 xx xx xx .**

So, we need to search for the key that looks like this and the each xx in the above key format will take the values from 00 to FF.

We are having the possible 2^24 possible combinations or key combinations.

**Project Task:**

* **Task 1:** First, we need to find the key that is used to encrypt the messages and then write the key to a file.
* **Task 2:** Second, we need to read the key from the file to which the key is written in the previous task, and then also read the cipher text from the c\_c.bin file, which is the challenge cipher and also the challenge nonce from the file nonce\_c.bin and try to decrypt the message which is present in the challenge cipher.

**Algorithm Used:**

We are using the AES block cipher with CTR mode. This mode is used to encrypt and decrypt the messages of the arbitrary length.

Here, in this mode, the nonce and a counter value are combined to create a unique key for each block of the plain text. So, this unique key will be used to encrypt and decrypt the corresponding block of plain text and cipher text. The resulting text is Xored with the block to produce the output.

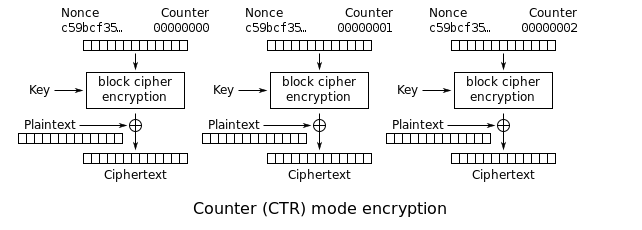


Fig Represents how the encryption with CTR works.

Diagram

Description automatically generated

Fig Represents how the decryption using CTR works.

**Program Execution and Implementation:**

**Task 1:**

**Libraries:**

Here we have used the libraries **pycryptodome** and **pycryptodomex** to perform the AES encryption and decryption functionalities.

The library **tqdm** is used to display the progress of the task in the system.

These are installed In the initial phase of the project.

Graphical user interface, text, application

Description automatically generated

The screenshot shows that the dependencies are installed in the environment.

We have used the google colab to perform the task.

**Functions defined:**

We have defined a custom function called as “read\_binary\_files” which is used to read the data from the files In the bytes object format or bytes format.

Graphical user interface, text

Description automatically generated

The above fig shows the code snippet that is defined by us.

Next, we have used the functions that are provided in the “util\_demo.py” file to help us work with the file operations like to write the data in the .bin files and also to write the data normal file in the normal text or hex format.

The method write\_bytes is used for writing data in binary format and the method write\_file is used to write data in normal or hex format.

The below are the code snippets supporting the same.

Graphical user interface, text, application

Description automatically generated

Next, the required files are given in the zip format, so we are unzip the contents to work with the same.

Graphical user interface, text, application, email

Description automatically generated

Once all the contents of the file are unzipped, then proceed to the execution of the main code section.

The program initially what it does is we import the dependencies from the library to the code.

Then, we read the data from the cipher text files into the variables cipher\_text\_c1 from file c1.bin, cipher\_text\_c2 from file c2.bin and to variable cipher\_text\_c3 from file c3.bin.,

Then the message contents form the files m1.txt, m2.txt and m3.txt are read to variables message\_text\_m1, message\_text\_m2, message\_text\_m3 respectively.

Then the nonce keys for the cipher texts are read to variables nonce1, nonce2 and nonce3 from the files nonce1.bin, nonce.bin and nonce3.bin respectively.

Text

Description automatically generated

The above screenshot represents, the code sniped to perform the operation.

Later, the key length and the key space are defined, the key length was given as 3 because, we are appending the already given 13 bytes length key to the key and only the remaining values are only being checked for the actual key. There is 2^24 key space.

Next, we are writing a loop to iterate the whole key space to find for the key in each iteration.

And for each iteration, inside the loop, we are defining the key that is used to encrypt the message texts to compare with the cipher texts. If all the 3 messages are encrypted and the corresponding cipher texts are matched to the cipher texts already given in the program, we consider that to be the final key that is used to encrypt all the messages.

The variables c1, c2 and c3 are the ciphers that are defined unique to encrypt the message m1 m2 and m3 respectively using their respective nonce. The AES is used with CTR mode to perform this encryption.

The below code represents the defining of the cipher.

Text, timeline

Description automatically generated

Later, we encrypt the message text read from the file with the ciphers to generate the cipher texts and then compare the generated cipher text in each iteration to the initial cipher text that is read from the file. If both are a match for all the 3 files, then the single key that is used to encrypt and all the three messages is found.

The below code shows the code implementation for the above said statement.

Text

Description automatically generated

Once the match is occurred, the key is written to the file called as “**key.bin**” in binary object format.

We have also copied the key to a file “**key\_in\_text.txt**” as a backup of the key and this is easier to read with the human eye for our understanding.

The key for this is found within 37% of the search limit of the key space.

The screenshot for the execution is as follows:

Graphical user interface, text, application, email

Description automatically generated

From the figure we can see that the key is found.

The key in hex is: “ **800000000000000000000000005fee64** “

**Task 2:**

Now, that key is copied to “key.bin” file, we are now asked to write another code snippet or program, where in we need to read the key from this file and the cipher text from the “c\_c.bin” file and its associated nonce key from “nonce\_c.bin” file.

To do the following task, we have written the following code snippet.

Graphical user interface, text, application

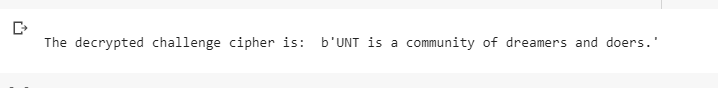
Description automatically generated

Challenge cipher text, nonce and the key are read from the files to the variables challenge\_cipher\_text, challenge\_nonce and the key variable respectively.

Then the cipher cc is created using the CTR mode.

Then later the ciphertext is decrypted using the key and the cipher cc that is created.

When the decryption is done, the resulting output message is obtained as follows.



So upon decrypting the encrypted cipher text present in the “c\_c.bin” file, the message obtained is:

**UNT is a community of dreamers and doers.**

So, with this the assigned two tasks for the project are done successfully.

**References:**

<https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation>

<https://tqdm.github.io/>

<https://link.springer.com/referenceworkentry/10.1007/0-387-23483-7_147>

<https://pypi.org/project/pycryptodomex/>

**Submission Contents/Details:**

In the submission, the following files are included:

* Project report in .docx format
* Zip file containing all the required files given in the challenge along with the **key.bin** file, **key\_in\_text.txt** file, Python .ipynb file containing both the codes( please run the code from starting block in colab or jupyter notebook ) and also a pdf document of the .ipynb file code demonstrating the outputs along with the code.