Homework Number: HW05

Name: Raghava Vivekananda Panchagnula

ECN Login: rpanchag

Due Date: 2/20/2024

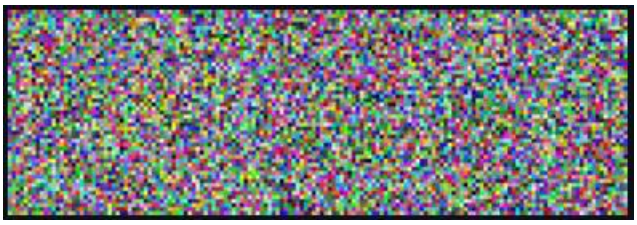
Commands used to run Homework 5:

```
(.venv) vivek@vivek-Inspiron-13-5378:~/Files/coursework/ECE-40400/Homework/HW05$ make test
python3 AES.py -e message.txt key.txt encrypted.txt
python3 AES.py -d encrypted.txt key.txt decrypted.txt
python3 AES.py -i image.ppm key.txt enc_image.ppm
python3 AES.py -r 5 key.txt random_numbers.txt
python3 AES.py -i enc_image.ppm key.txt test.ppm
```

Ignore hw4 run steps, they do not affect the results of HW 5. The test.ppm step exists to check that the final output is the same as the input, because of how ctr mode encryption and decryption work. So since I got the original helicopter back, I know my code is right.

Runtime is approx. 180s on an Intel i7 7500u laptop processor.

Output for Problem 1:



Output for Problem 2:

331374527193731622526773163027689011175

262633037080229609278739248627548891 87

621388110439928640615094882415799550 8

317525806849049200816126045738729418009

240080400546264647934751409092776671804

Code Explanation:

- block_encrypt

    - I made a new function called block_encrypt that only performs the 14 rounds of the AES encryption algorithm and returns the resultant bitvector as an output. It works the same way that my function from homework 4 works, so for further detail please refer to that.
- ctr_aes_image
    - This function reads the first 3 lines of the input image and writes it to the output as is.
    - It then reads the whole file again as a bv, discards the first 112 bits which is what the length of the header was, and then begins the encryption process. Te process to find these 112 bits is as in my code, where I write down the header as a list, merge it as bytes, and make a bv out of it to figure out its length.
    - In this process, first, it takes the initial value and encrypts it with the roundkeys. It then xors it with the 128 bits from the image. It then increments the iv by 1 and adds the result from the xor to the file and moves on to the next 128 bits. The main algorithm itself is only 4 lines, the rest is supporting code to read and write the input image and encoded image respectively.
- x931
    - This function takes the dt value to create an “encoder” variable that I will use for subsequent operations. I then, for each of the numbers we have to create and output, I take the encoder variable I made, xor it with the seed, and then generate the random number by inputting it into my block_encrypt function. I then generate a new seed by xoring the encoder value with the previously created random number and then putting it through the block_encrypt function again. I then write the generated random values to a new file.