Chosen-plaintext Attack (CPA)

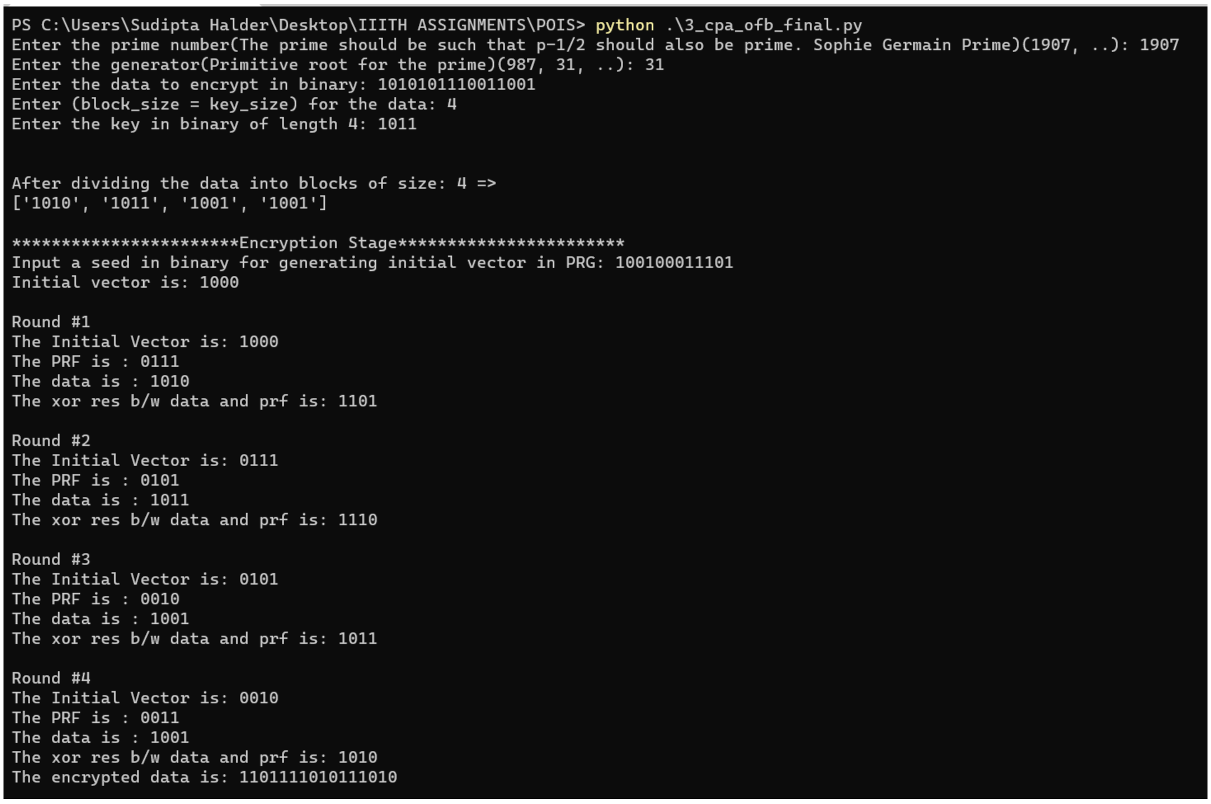
Assumptions: I've used **output feedback mode (OFB)** to design CPA.

Input format:

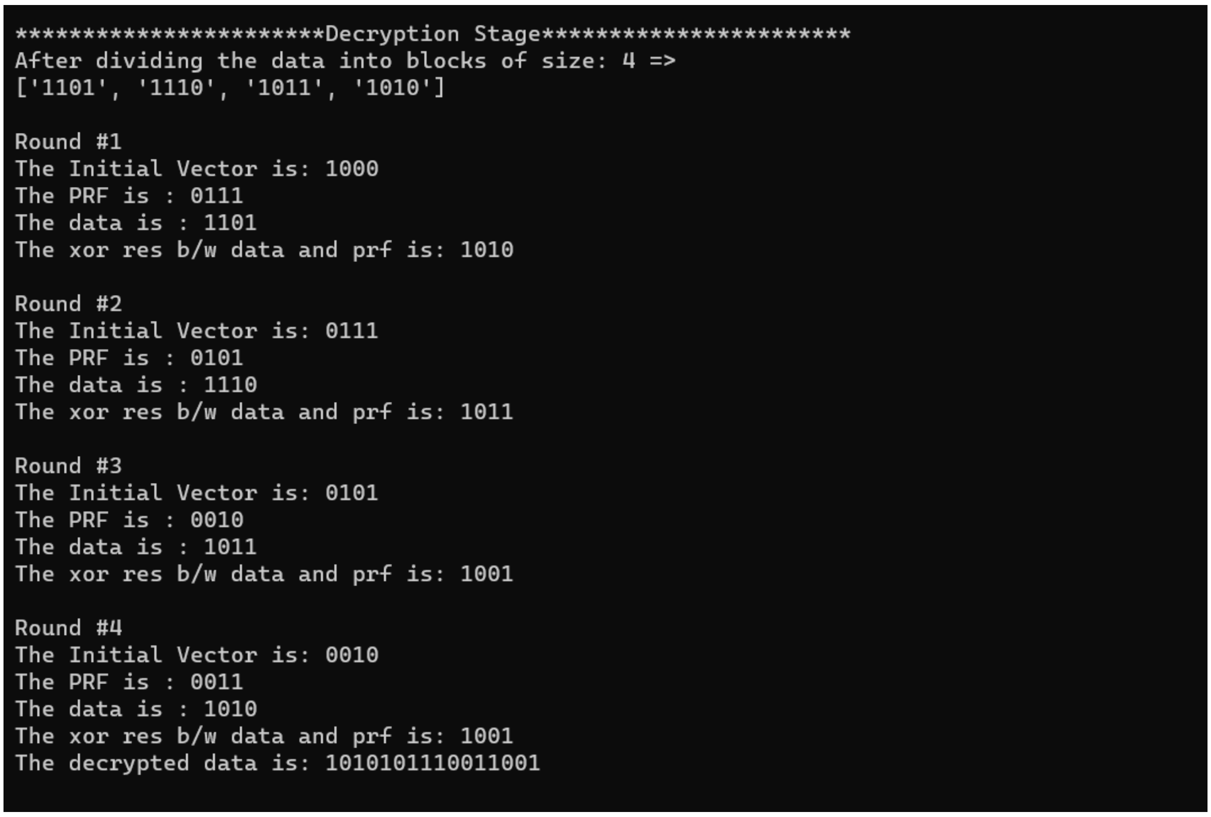
1. It will ask for a safe prime number p in integer format.
2. It will ask for a generator (primitive root) of that prime.
3. It will ask to enter the data in binary form.
4. It will then ask for block size.
5. It will then ask for a key whose length is same as block size in binary format.
6. It will also ask for seed to generate initial vector in between.

Output Format:

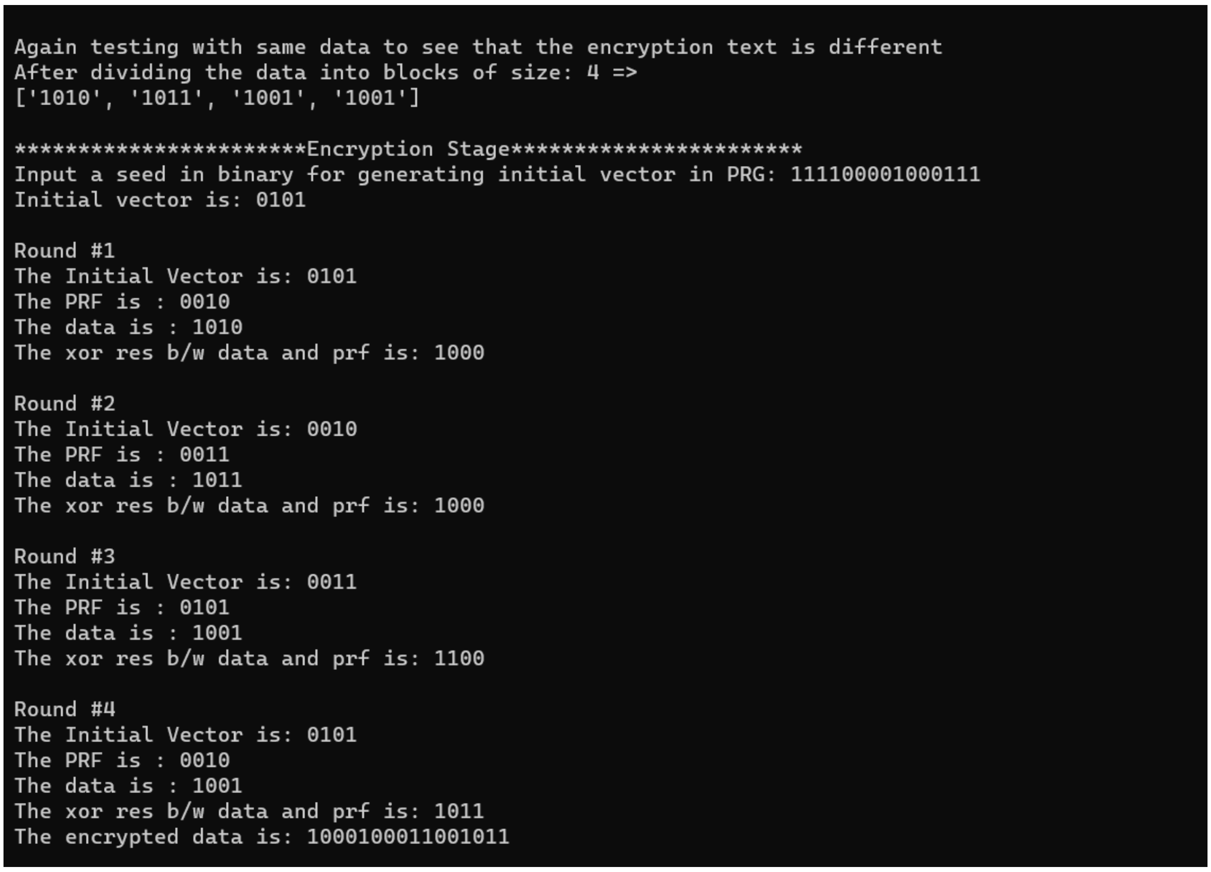
1. It will encrypt the data and decrypt the data to show that we are able to decrypt also properly.
2. It will encrypt for once more with different initial vector to show that how the encrypted data is different from the previous one upon using a different initial vector. Also, decryption will follow to show the correctness.



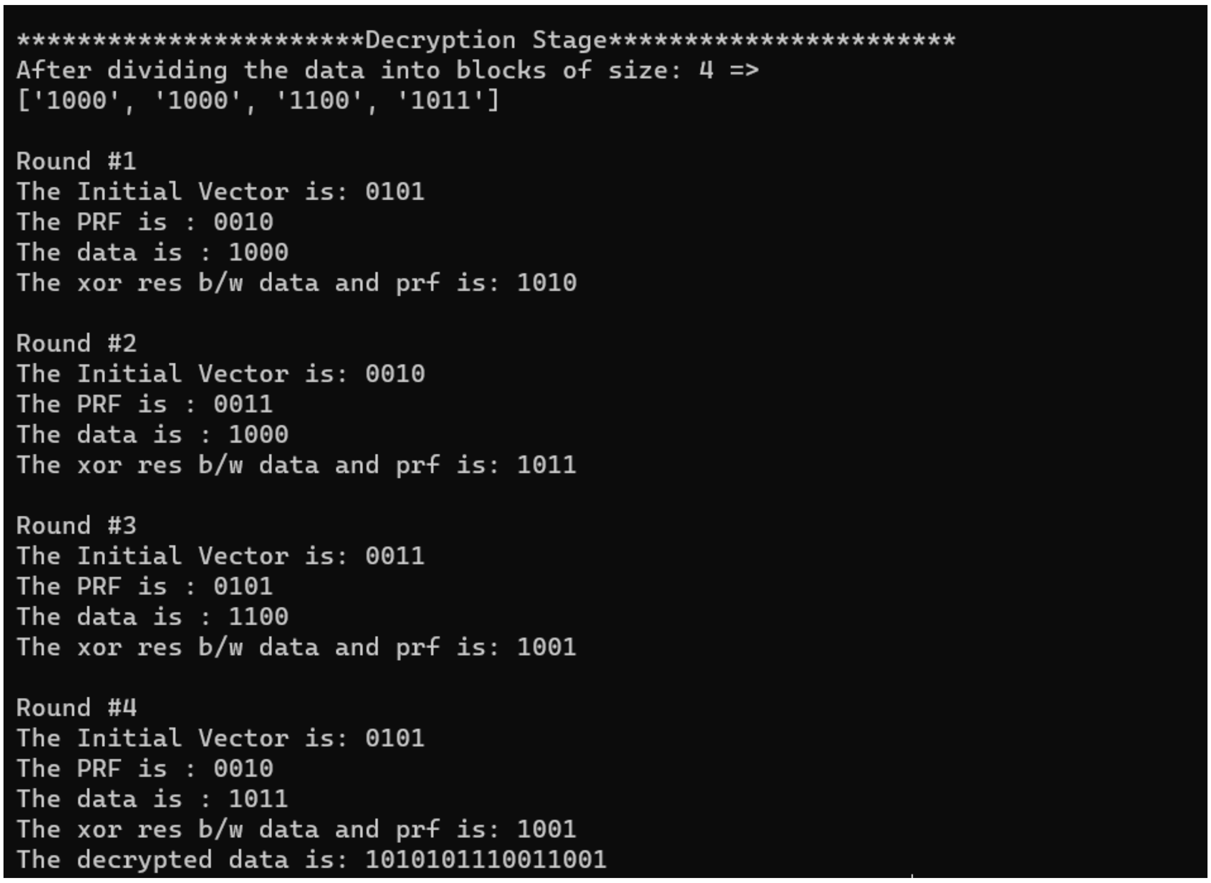
```
PS C:\Users\Sudipta Halder\Desktop\IIITH ASSIGNMENTS\POIS> python .\3_cpa_ofb_final.py
Enter the prime number(The prime should be such that p-1/2 should also be prime. Sophie Germain Prime)(1907, ..): 1907
Enter the generator(Primitive root for the prime)(987, 31, ..): 31
Enter the data to encrypt in binary: 1010101110011001
Enter (block_size = key_size) for the data: 4
Enter the key in binary of length 4: 1011


After dividing the data into blocks of size: 4 =>
['1010', '1011', '1001', '1001']

***********************Encryption Stage***********************
Input a seed in binary for generating initial vector in PRG: 100100011101
Initial vector is: 1000

Round #1
The Initial Vector is: 1000
The PRF is : 0111
The data is : 1010
The xor res b/w data and prf is: 1101

Round #2
The Initial Vector is: 0111
The PRF is : 0101
The data is : 1011
The xor res b/w data and prf is: 1110

Round #3
The Initial Vector is: 0101
The PRF is : 0010
The data is : 1001
The xor res b/w data and prf is: 1011

Round #4
The Initial Vector is: 0010
The PRF is : 0011
The data is : 1001
The xor res b/w data and prf is: 1010
The encrypted data is: 1101111010111010
```



```
***********************Decryption Stage***********************
After dividing the data into blocks of size: 4 =>
['1101', '1110', '1011', '1010']

Round #1
The Initial Vector is: 1000
The PRF is : 0111
The data is : 1101
The xor res b/w data and prf is: 1010

Round #2
The Initial Vector is: 0111
The PRF is : 0101
The data is : 1110
The xor res b/w data and prf is: 1011

Round #3
The Initial Vector is: 0101
The PRF is : 0010
The data is : 1011
The xor res b/w data and prf is: 1001

Round #4
The Initial Vector is: 0010
The PRF is : 0011
The data is : 1010
The xor res b/w data and prf is: 1001
The decrypted data is: 1010101110011001
```



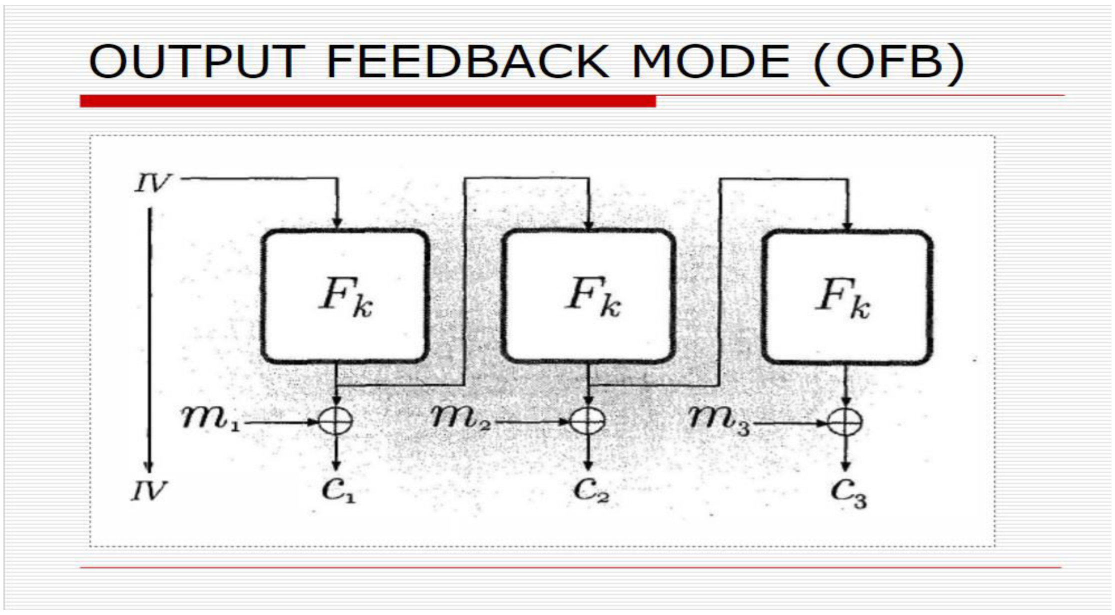
```
Again testing with same data to see that the encryption text is different
After dividing the data into blocks of size: 4 =>
['1010', '1011', '1001', '1001']

***********************Encryption Stage***********************
Input a seed in binary for generating initial vector in PRG: 111100001000111
Initial vector is: 0101

Round #1
The Initial Vector is: 0101
The PRF is : 0010
The data is : 1010
The xor res b/w data and prf is: 1000

Round #2
The Initial Vector is: 0010
The PRF is : 0011
The data is : 1011
The xor res b/w data and prf is: 1000

Round #3
The Initial Vector is: 0011
The PRF is : 0101
The data is : 1001
The xor res b/w data and prf is: 1100

Round #4
The Initial Vector is: 0101
The PRF is : 0010
The data is : 1001
The xor res b/w data and prf is: 1011
The encrypted data is: 1000100011001011
```



```
**********************Decryption Stage***********************
After dividing the data into blocks of size: 4 =>
['1000', '1000', '1100', '1011']

Round #1
The Initial Vector is: 0101
The PRF is : 0010
The data is : 1000
The xor res b/w data and prf is: 1010

Round #2
The Initial Vector is: 0010
The PRF is : 0011
The data is : 1000
The xor res b/w data and prf is: 1011

Round #3
The Initial Vector is: 0011
The PRF is : 0101
The data is : 1100
The xor res b/w data and prf is: 1001

Round #4
The Initial Vector is: 0101
The PRF is : 0010
The data is : 1011
The xor res b/w data and prf is: 1001
The decrypted data is: 1010101110011001
```

Working Flow:

1. Upon getting the input, the function `generate_cpa_ofb(prime, generator, data, block_size, key)` will be called.
2. It will then divide the data in blocks of given block-size. After that, it will generate a random vector of the length = block-size. It will then iterate through the data blocks.
3. For each data block, first pass the initial vector through the PRF and obtain a prf. Then calculate xor between data block and the obtained prf. The result will the corresponding ciphertext-block for that corresponding data block. For the next iteration the obtained prf will be used as initial vector. For better understanding refer to the below pic of OFB mode. Now, concatenating the ciphertext-blocks, we'll get the actual ciphertext.



OUTPUT FEEDBACK MODE (OFB)

4. In the decryption stage the initial vector taken in encryption stage is passed along with the ciphertext and block-size. There the ciphertext is decrypted in the same way as OFB. We then check whether the initial data and the decrypted data same or not.
5. We then run the encryption and decryption once again with a different initial vector to show that the encryption data is different this time which is the main agenda of CPA that the encrypted data should be different every time for the same data.