TEAM98
P.Sai Mohith CS14B047
S.Sai Teja Reddy CS14B051

Applied Cryptography

1st March 2017

OVERVIEW

In this assignment, we will develop a block cipher that will encrypt 128-bit blocks at a time using 128-bit keys. Typical block ciphers consist of a fixed number of rounds. Each round performs the following operations:

1. **Add Round Key**
2. **Confusion Layer**
3. **Diffusion Layer**

SECTION 1

What is your cipher called?

Our cipher is called the Psi-Square Cipher.

How many rounds does your cipher have?

The cipher has 32 rounds.

Explain one round of your cipher.

One round of our cipher is similar to a round in the balanced Feistel cipher. In the Feistel box we perform **Add Round Key** using a XOR operation with the corresponding 64-bit round key. The 64-bit output is sent into 16 straight S-boxes which make up the **Confusion Layer**. In the **Diffusion Layer** we permute the 64-bit output using a predefined function given by the following 8x8 matrix which provides the required mappings:

64	56	48	40	32	24	16	8
63	55	47	39	31	23	15	7
62	54	46	38	30	22	14	6
61	53	45	37	29	21	13	5
60	52	44	36	28	20	12	4
59	51	43	35	27	19	11	3
58	50	42	34	26	18	10	2
57	49	41	33	25	17	9	1

How did you arrive at the number of rounds for your cipher?

Lucifer was a 16-round Feistel network on 128-bit blocks and 128-bit keys.This was why we chose an approximate number of 20 which could further be subject to optimization. This exact number of rounds will be obtained after cryptanalysis.

Analyzing the most deadly linear trail in our cipher, we see that 32 rounds are needed to make linear cryptanalysis more difficult than brute force.

How did you choose the s-boxes size and type?

Since we use a Feistel network in each round, we will be operating on 64-bit sub-blocks. Hence, we have 64-bit subkeys. The input to the S-boxes will be 64-bit and the output also is the same. We therefore have to use straight S-boxes. By taking a more symmetric approach we hope to make the cipher more uniform in turn making it difficult to break. We will thus be using 16 S-boxes with each being 4x4.

How did you go about choosing the s-box(es) mappings?

We will be using 4 S-boxes, each used four times. The reason we used these S-boxes is because they satisfy the balancedness property, SAC property and have high nonlinearity. All these properties are demonstrated in the next question.

S-box 1 (S1):

	0XX0	0XX1	1XX0	1XX1
Y00Y	5	14	8	12
Y01Y	6	13	0	9
Y10Y	7	4	1	11
Y11Y	2	10	15	3

S-box 2 (S2):

	0XX0	0XX1	1XX0	1XX1
Y00Y	12	1	0	13
Y01Y	10	15	3	4
Y10Y	9	2	14	7
Y11Y	6	8	5	11

S-box 3 (S3):

	0XX0	0XX1	1XX0	1XX1
Y00Y	4	11	3	12
Y01Y	2	14	9	7
Y10Y	15	0	5	10
Y11Y	8	13	6	1

S-box 4 (S4):

	0XX0	0XX1	1XX0	1XX1
Y00Y	13	9	3	8
Y01Y	15	12	14	2
Y10Y	11	5	0	1
Y11Y	7	6	4	10

For your s-box(es), explain / demonstrate how you satisfied the following properties

1.The balancedness property

2.SAC

3.Non-linearity

4.Algebraic degree (optional)

Since all the four S-boxes have numbers from 0 to 15 , they are balanced. (trivial)

We will now check SAC for the S-boxes:

For each S-box we have to compute $f(x) \oplus f(x \oplus a)$ where HW(a)=1 and check if they are balanced or not.

For this we use sac.py , (PFA) which computes the number of 1's in each S-box's $f(x) \oplus f(x \oplus a)$.

SAC is not satisfied but on an average each bit flip causes two output bits to flip.

Using the **SAGE** software for computing non linearity we get,

SBOX1: 4

SBOX2: 2

SBOX3: 2

SBOX4: 4

Similarly for algebraic degree,

SBOX1: 3

SBOX2: 3

SBOX3: 3

SBOX4: 3

Draw the linear approximation table for your s-box(es).

SBOX1:

[8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 4 4 0 4 0 0 -4 0 0 0 0]
[0 0 2 2 -2 -2 0 0 0 -4 2 -2 -2 2 0 -4]
[0 -4 2 -2 2 2 0 0 0 -4 -2 2 -2 -2 0 0]
[0 2 4 -2 -2 0 2 4 -2 0 -2 0 0 2 0 2]
[0 2 0 2 2 0 -2 4 2 0 2 0 -4 -2 0 2]
[0 2 -2 0 0 2 -2 0 -2 -4 0 -2 2 0 -4 2]
[0 -2 2 0 0 2 -2 0 -2 4 0 -2 -2 0 -4 -2]
[0 2 -2 -4 -4 2 -2 0 2 0 0 2 -2 0 0 -2]
[0 -2 -2 0 0 -2 -2 0 2 0 -4 -2 -2 4 0 2]
[0 -2 0 2 -2 0 2 0 2 0 2 4 0 2 -4 2]
[0 -2 0 2 -2 4 -2 0 -2 0 2 0 0 2 4 2]
[0 -4 -2 -2 -2 -2 0 4 0 0 2 -2 2 -2 0 0]
[0 0 2 -2 -2 -2 0 -4 0 0 2 -2 -2 -2 0 4]
[0 0 0 -4 4 0 0 0 0 0 4 0 0 4 0 0]
[0 0 4 0 0 0 -4 0 4 0 0 0 4 0 0 0]

SBOX2:

[8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 2 0 -2 0 -2 0 2 0 -2 0 2 0 2 0 6]
[0 0 2 -2 0 0 2 -2 0 0 -2 2 4 4 2 -2]
[0 2 2 4 0 -2 2 0 -4 2 2 0 0 2 -2 0]
[0 0 2 2 0 0 -2 -2 0 0 2 2 4 -4 2 2]
[0 2 -2 4 4 2 -2 0 0 -2 -2 0 0 2 2 0]
[0 0 4 0 0 0 0 4 0 0 0 -4 0 0 4 0]
[0 2 0 2 -4 2 0 -2 4 2 0 -2 0 2 0 2]
[0 2 0 -2 2 0 2 -4 -2 0 -2 -4 0 -2 0 2]
[0 0 0 0 -2 -2 -2 -2 -2 2 -2 2 -4 0 4 0]
[0 -2 2 0 2 -4 -4 -2 2 0 0 -2 0 2 -2 0]
[0 -4 2 2 -2 2 0 0 -2 -2 -4 0 0 0 -2 2]
[0 -2 -2 0 -2 0 0 -2 -2 -4 4 -2 0 2 2 0]
[0 4 2 -2 -2 2 -4 0 -2 -2 0 0 0 0 -2 -2]
[0 2 0 2 -2 -4 2 0 2 -4 -2 0 0 -2 0 -2]
[0 0 -4 0 -2 -2 -2 2 -2 2 -2 -2 4 0 0 0]

SBOX3:

[8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 2 2 -2 -2 -2 -2 -6 2]
[0 0 0 4 0 0 -4 0 0 0 4 0 0 0 0 4]
[0 -4 0 0 -4 0 0 0 2 -2 2 2 2 -2 -2 -2]
[0 0 -2 2 0 -4 -2 -2 0 0 -2 2 0 4 -2 -2]
[0 0 2 -2 4 0 -2 -2 2 2 0 4 2 -2 0 0]
[0 0 2 2 0 -4 -2 2 0 0 -2 -2 0 -4 2 -2]
[0 4 -2 2 0 0 2 2 2 -2 0 4 -2 -2 0 0]
[0 2 0 2 0 2 0 2 -2 4 2 0 2 0 -2 -4]
[0 2 0 2 0 2 0 -6 0 -2 0 -2 0 -2 0 -2]
[0 -2 0 2 0 -2 4 -2 2 4 2 0 -2 0 2 0]
[0 2 0 -2 -4 -2 0 -2 -4 2 0 2 0 -2 0 2]
[0 2 -2 -4 0 -2 -2 0 2 0 4 -2 -2 0 0 -2]
[0 2 2 0 -4 2 -2 0 4 2 -2 0 0 2 2 0]
[0 -2 2 0 0 2 -2 0 -2 0 0 2 -6 0 0 -2]
[0 2 6 0 0 -2 2 0 0 -2 2 0 0 2 -2 0]

SBOX4:

[8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 -2 -2 4 -2 0 0 2 0 2 2 4 2 0 0 -2]
[0 -4 4 0 4 0 0 4 0 0 0 0 0 0 0 0]
[0 2 -2 0 2 0 -4 2 0 2 -2 0 2 0 4 2]
[0 0 0 0 0 0 0 0 -4 0 -4 0 4 0 -4 0]
[0 -2 -2 -4 -2 0 0 2 0 -2 2 0 2 -4 0 2]
[0 0 0 0 0 0 0 0 0 0 -4 4 -4 -4 0 0]
[0 -2 2 0 -2 0 -4 -2 4 -2 -2 0 2 0 0 -2]
[0 -4 0 0 -4 0 0 0 -2 2 -2 -2 -2 2 2 2]
[0 -2 -2 0 2 4 0 -2 -2 0 0 -2 0 -2 2 -4]
[0 0 0 -4 0 0 -4 0 -2 2 2 2 -2 2 -2 -2]
[0 2 2 0 -2 -4 0 2 -2 0 0 -2 0 -2 2 -4]
[0 0 4 0 0 0 0 -4 -2 2 2 2 2 -2 2 2]
[0 2 2 0 -2 4 0 2 2 4 0 -2 0 -2 -2 0]
[0 0 0 -4 0 0 4 0 2 2 -2 2 2 2 2 -2]
[0 2 2 0 -2 4 0 2 -2 -4 0 2 0 2 2 0]

Draw the differential distribution table for your s-box(es).

SBOX1:

```
[16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  2  2  0  0  0  2  2  2  4  2  0  0  0]
[ 0  0  0  4  0  4  0  0  4  0  0  0  0  0  4  0]
[ 0  2  2  0  2  0  2  0  4  0  0  0  2  2  0  0]
[ 0  0  2  0  2  0  0  4  0  2  4  0  0  0  0  2]
[ 0  2  0  4  0  0  2  0  0  2  0  0  2  2  0  2]
[ 0  4  2  0  2  0  0  4  0  2  0  0  0  0  0  2]
[ 0  0  2  2  0  0  0  0  2  0  2  4  2  0  0  2]
[ 0  0  2  0  2  0  4  0  0  2  0  0  0  4  0  2]
[ 0  2  0  0  0  4  2  0  0  2  0  0  2  2  0  2]
[ 0  4  0  2  0  2  0  4  2  0  0  0  0  0  2  0]
[ 0  0  0  0  2  2  0  0  0  2  2  4  2  0  2  0]
[ 0  0  2  2  2  2  0  0  2  2  0  0  0  0  2  2]
[ 0  0  2  0  0  2  0  0  0  0  2  4  2  0  2  2]
[ 0  0  0  0  0  0  4  4  0  0  4  0  0  4  0  0]
[ 0  2  2  0  2  0  2  0  0  0  0  0  2  2  4  0]
```

SBOX2:

```
[16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  2  0  2  0  2  0  2  0  4  4  0]
[ 0  0  0  2  0  0  2  0  0  2  2  2  2  0  2  2]
[ 0  2  2  2  4  2  0  0  0  0  0  2  0  0  2  0]
[ 0  0  0  2  0  2  2  2  0  0  2  0  2  0  2  2]
[ 0  2  2  2  0  0  0  2  4  2  0  0  0  0  2  0]
[ 0  0  0  4  0  2  2  0  0  2  2  0  0  4  0  0]
[ 0  0  0  0  4  0  2  2  4  0  2  2  0  0  0  0]
[ 0  0  0  4  0  2  0  2  0  2  0  2  4  0  0  0]
[ 0  4  0  0  0  0  0  0  0  0  0  0  4  4  4  0]
[ 0  0  2  0  0  2  0  0  2  2  2  0  2  0  0  4]
[ 0  2  4  0  0  0  2  4  2  0  0  0  0  0  0  2]
[ 0  0  2  0  2  2  2  0  0  2  0  0  2  0  0  4]
[ 0  2  4  0  2  0  0  0  0  0  2  4  0  0  0  2]
[ 0  0  0  0  2  2  4  0  2  2  4  0  0  0  0  0]
[ 0  4  0  0  2  0  0  2  2  0  0  2  0  4  0  0]
```

SBOX3:

[16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 2 0 2 0 0 0 0 2 0 2 8]
[0 0 0 2 0 2 2 2 0 0 2 4 0 2 0 0]
[0 0 2 0 4 2 0 0 2 2 2 0 2 0 0 0]
[0 0 0 2 0 0 6 0 0 0 2 4 0 0 0 2]
[0 2 0 0 4 0 2 0 2 4 0 0 0 0 0 2]
[0 0 0 0 0 2 2 0 0 0 0 0 4 6 2 0]
[0 2 6 4 0 0 0 0 0 2 2 0 0 0 0 0]
[0 0 0 0 0 0 0 4 0 2 4 2 2 0 2 0]
[0 0 0 0 0 6 0 2 4 2 0 2 0 0 0 0]
[0 4 2 0 0 0 0 2 0 2 0 0 2 4 0 0]
[0 0 4 2 0 0 2 0 2 0 0 0 0 2 4 0]
[0 6 0 0 2 0 0 0 0 0 2 0 4 0 0 2]
[0 0 0 6 2 0 0 0 2 0 0 0 0 0 4 2]
[0 2 2 0 2 0 2 4 0 0 2 2 0 0 0 0]
[0 0 0 0 2 2 0 0 4 2 0 2 0 2 2 0]

SBOX4:

[16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 4 0 2 2 0 0 0 0 0 0 2 2 0 4 0]
[0 0 2 2 2 2 0 0 0 0 2 2 2 2 0 0]
[0 4 2 0 0 2 4 0 0 0 2 0 0 2 0 0]
[0 0 0 2 0 0 2 0 4 2 4 0 2 0 0 0]
[0 0 4 0 2 0 2 0 4 2 0 2 0 0 0 0]
[0 0 2 2 2 0 0 2 0 2 2 0 0 0 2 2]
[0 0 2 0 0 0 0 2 0 2 2 2 2 0 2 2]
[0 4 0 2 2 0 0 0 0 0 0 2 2 0 4 0]
[0 0 4 0 0 4 0 0 0 0 4 0 0 4 0 0]
[0 0 0 2 2 0 0 4 0 0 0 2 2 0 0 4]
[0 4 0 0 0 0 4 4 0 0 0 0 0 0 0 4]
[0 0 0 0 2 0 2 0 4 2 0 2 0 4 0 0]
[0 0 0 2 0 4 2 0 4 2 0 0 2 0 0 0]
[0 0 0 2 2 2 0 2 0 2 0 0 0 2 2 2]
[0 0 0 0 0 2 0 2 0 2 0 2 2 2 2 2]

How did you choose the diffusion layer for your cipher?

We choose the permutation such that the output of an S-Box affects the input of four different S-Boxes in the next layer. Keeping this in mind, the permutation we chose was,

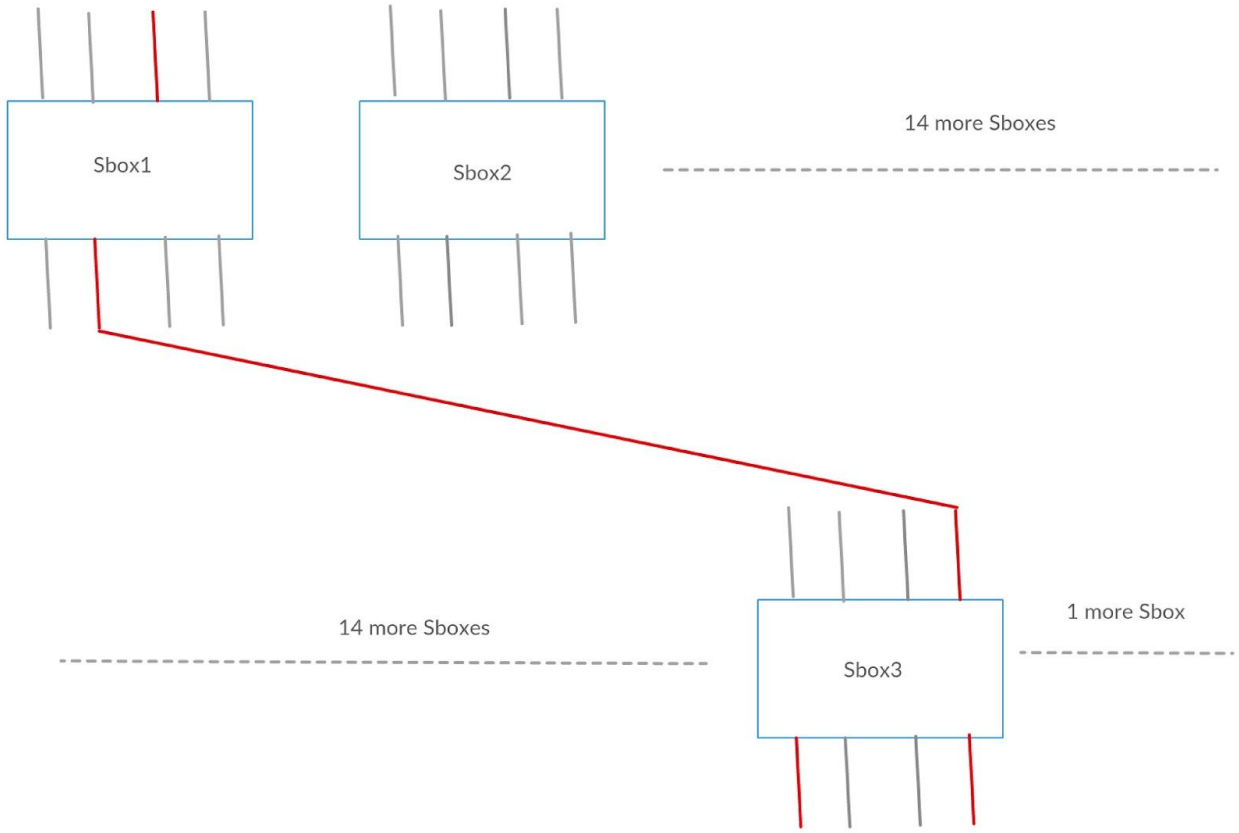
64	56	48	40	32	24	16	8
63	55	47	39	31	23	15	7
62	54	46	38	30	22	14	6
61	53	45	37	29	21	13	5
60	52	44	36	28	20	12	4
59	51	43	35	27	19	11	3
58	50	42	34	26	18	10	2
57	49	41	33	25	17	9	1

How many rounds would it take to obtain complete diffusion? Show minimum rounds needed for diffuse completely when the i-th bit in the plaintext is toggled, where i can vary from 0 to 127.

In our S-boxes, each bit flip on an average flips two bits in the output.(can be calculated using sac.py). In the first round there is no diffusion. In the next round, since the property of s-box, 2 bits are flipped, which in turn results in 4 bit flips in the next round and so on. We can see that, it takes 8 rounds for the complete diffusion to occur.However, we cannot be 100% sure that the permutation diffuses the change to different s-boxes, therefore this estimate may be off by a couple of rounds.Therefore we can say that we need a minimum of 8 rounds to achieve complete diffusion. Note that if i is greater than 63, it will take one round lesser to achieve diffusion.

SECTION-2

Show formally and with figures and code, the most deadly linear trail in your cipher. Ensure that an attacker would find linear cryptanalysis more difficult than brute force.



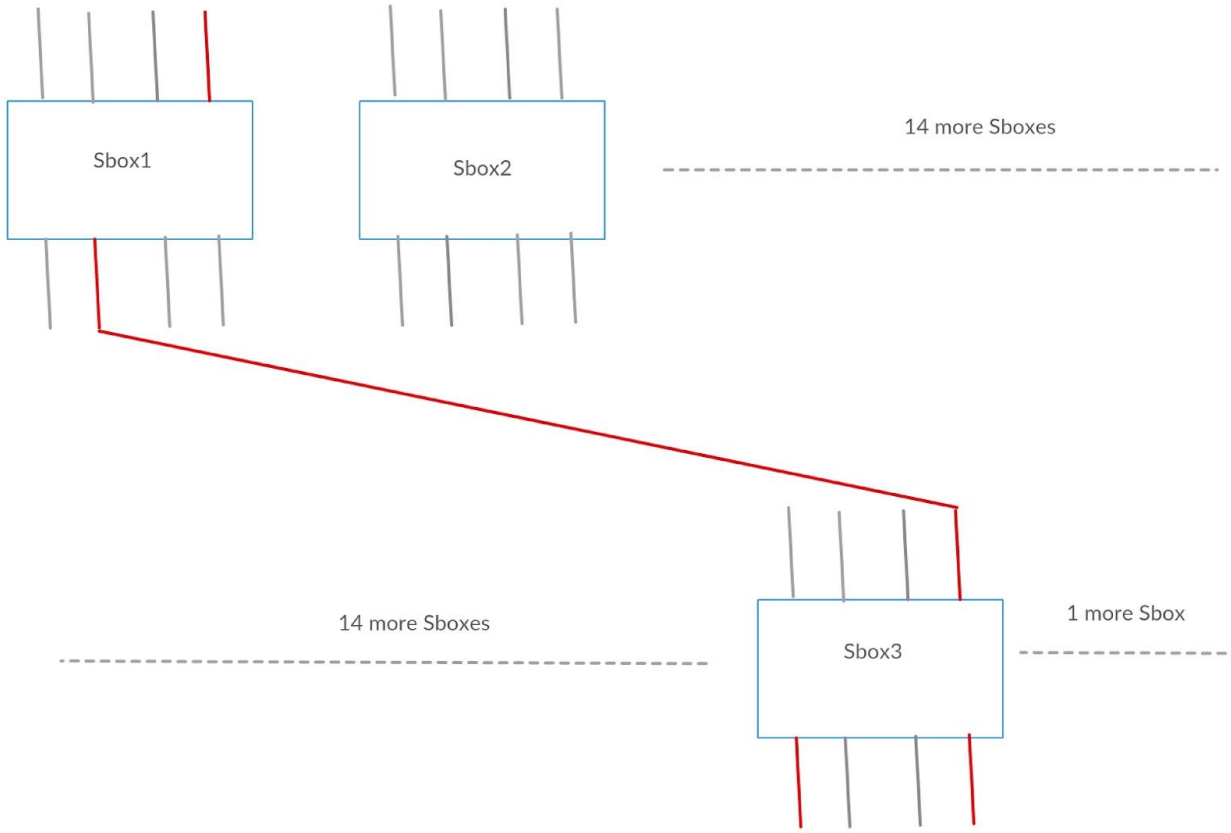
T1 = $U_3^1 \oplus V_2^1$ has bias ¼

T2 = $U_{56}^2 \oplus V_{53}^2 \oplus V_{56}^2$ has bias ¼

T1 ⊕ T2 has bias 2*¼*¼ =⅛

No of rounds =128*2*⅛ = 32

Show formally and with figures and code, the most deadly differential trail in your cipher. Ensure that an attacker would find differential cryptanalysis more difficult than brute force.



In S_1^1 , R_p (0001,0100) = ¼

In S_{15}^2 , R_p (0001,1001) = ¼

Assuming independence between the s-boxes in the trail, propagation ratio for the trail is the product of individual propagation ratios.

Propagation ratio for the trail is ¼*¼ = 1/16

No of rounds 128*2*1/16 = 16 rounds

Revisit all questions in Section 1. If you decide to make changes in any of the answers, mention them here and justify why you are making the changes.

We have decided to change Q no 2 and 4 base on the above. The number rounds needed for our cipher is 32 to ensure that linear cryptanalysis is more difficult than brute force.

SECTION 3

Have you thought about efficient implementations? If so, demonstrate and document tricks in software and cipher design choices that have makes your cipher efficient to implement on the target platform. (you can either choose to optimize to take minimum memory or minimum operations)

Instead of encoding the text characters in ASCII we could use Huffman or Shannon coding to significantly reduce the file size.

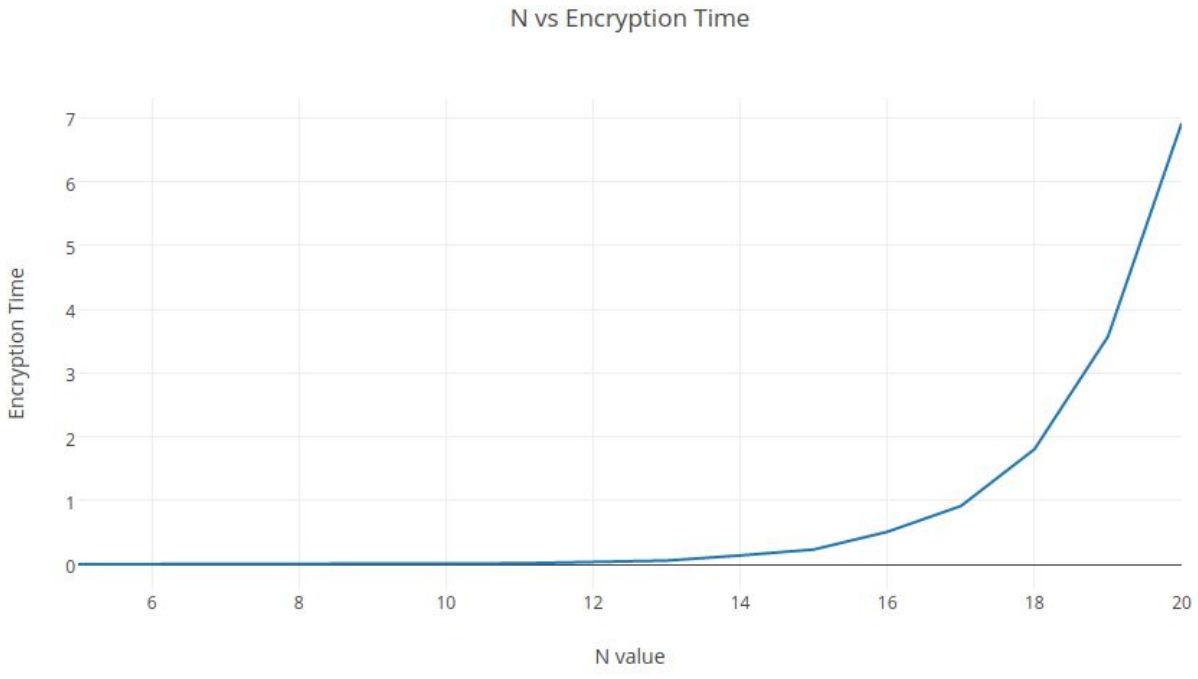
Revisit all questions in Section 1 and Section 2. If you decide to make changes in any of the answers, mention them here and justify why you are making the changes.

We decided to increase no. of rounds to 33 from 32 for implementation purposes

For files of sizes 2^n , where n=5 to 20, plot the file size vs the encryption time on the target platform (Intel 64-bit x86).

N value	Encryption Time(in sec)
5	0.00064
6	0.00088
7	0.00120
8	0.00237
9	0.00436
10	0.00760

11	0.01540
12	0.03717
13	0.05831
14	0.13893
15	0.23217
16	0.50975
17	0.91395
18	1.80475
19	3.57342
20	6.91765



Demonstrate the working of your cipher. Encrypt file alice.txt and save the result in enc_alice.txt. Decrypt enc_alice.txt and save the decrypted result in dec_alice.txt.

Refer README included in folder. The required files are included in the submitted Codes folder.