Cryptography Assignment - CS6530
Report

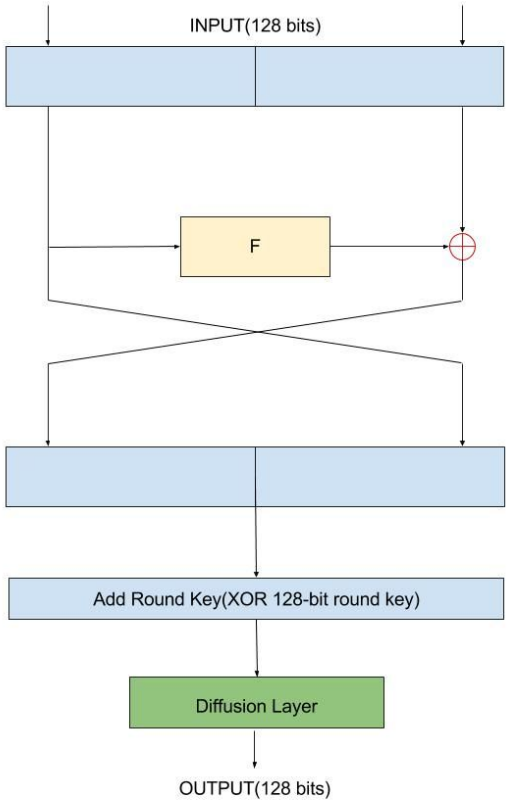
Team name: titanX

Team members:
Srinidhi Prabhu, CS14B028
Sai Pavan D, CS14B041

1) ZEN128

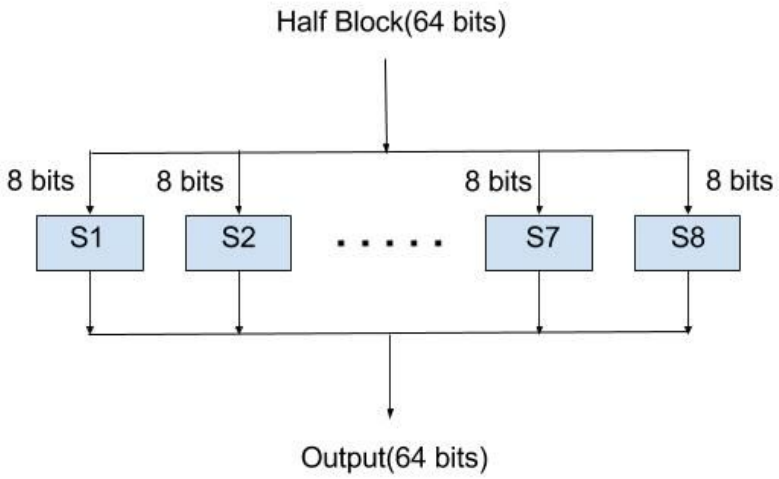
2) 20 rounds

3) The following diagram explains what is done in each round:



The diffusion layer is **not present** in the 19th and 20th round.

F is illustrated in the following diagram:



4) We followed the following steps to decide the number of rounds for our cipher.
    a) We first generated the S-Boxes as described in (5).
    b) We computed the maximum bias and the maximum propagation ratio for each S-Box. This was computed from the Linear Approximation Table(LAT) and Differential Distribution Table(DDT) of that S-Box.
    c) We then found the number of rounds required in the worst case for Linear Cryptanalysis(LC) and Differential Cryptanalysis(DC). It was found that it took 17 rounds for LC to reach a bias less than $1/2^{128}$ and 14 rounds for DC to reach a bias less than $1/2^{128}$.
    d) To ensure more security, we add 3 more rounds. This gives us a total of 20 rounds.

5)
    a) To choose the S-Box type, we decided to have a straight S-Box to avoid expansion or compression of input bits.
    b) To choose the S-Box size, we decided to have 8 x 8 S-Boxes because optimization in the space of 8-bit Boolean functions was fast(compared to 16-bit) and also gave a larger number of highly fit Boolean functions(compared to 4-bit).

6) We used a genetic algorithm to find good S-Box mappings. Here, good means in terms of the following criteria:
    a) **Balancedness**: A score out of 100 is given to a Boolean function. The score is closer to 100 if it is more balanced.
    b) **Non-linearity**: A score out of 100 is given to a Boolean function. The score is closer to 100 if it has more non-linearity. We know that the maximum

non-linearity is for a Bent function, so we assigned a non-linearity of 100 to Bent functions and scaled the non-linearity of every Boolean function appropriately.

c) **SAC score**: A score out of 100 is given to a Boolean function. The score is closer to 100, if it is as close to a balanced function when XORed with all vectors with Hamming Weight(HW) of 1. The average over all vectors is taken and the score is assigned.

d) We use the above criteria and optimize in the space of all Boolean functions with 8 input variables. The fitness is defined as a linear combination of the above scores. This gives us one bit of output. We take the 8 most-fit individuals from the final population(and it is observed that all these functions are different and have almost same fitness) to get one S-Box mapping.

e) We also tried simulated annealing and obtained similar results.

f) We ran the genetic algorithm 16 times and obtained 16 possible S-Box mappings. Then, we generated the LAT and DDT for these S-Boxes and chose 8 of them with the least maximum bias and propagation ratio.

7) As mentioned in (6), we optimized Boolean functions with respect to balancedness, non-linearity and SAC. A script “Scores/SBOX_score.py” is included to calculate the average values for every S-Box.

8) The LAT for each S-Box is included in the “LinearTables” folder. A script “LAT_DDT/LAT_generator.py” is also included to compute the LAT for an S-Box.

9) The DDT for each S-Box is included in the “DifferentialTables” folder. A script “LAT_DDT/DDT generator.py” is also included to compute the DDT for an S-Box.

10) The diffusion layer is a linear layer and should help in effecting the changes to all bits in the output. Both input and output are 16 bytes long. The following algorithm is used to compute the mappings:
Divide the input of 128 bits into 16 bytes. The bytes are assumed to be numbered from 0 to 15. The bits in each byte are numbered from 0 to 7.

```
for each byte:
  for each bit:
    i := current byte number
    j := current bit number
                Map bit j of byte i to bit (8*i+j)/16
                          of byte (8*i+j)%16
```

11) When one bit of plaintext is changed, by construction of the diffusion layer, all 8 S-boxes in the next round will be affected. Since all S-Boxes have a SAC score in the range 94 to 97, we can expect that after the second round 64 of the 128 bits(that pass through the

S-Boxes) will be affected. After the third round, the other 64 bits will be affected. So, it will take 3 rounds to obtain complete diffusion.

12) To find the most deadly linear trail, we find a linear trail bottom-up, i.e. from the ciphertext to the plain text. We take the case where the attacker wants to guess only 1 bit of the key.
If the attacker wants to guess more than 1 bit, we can show that the bias of any trail will be less than the bias of a trail in which he guesses exactly 1 bit. This is because the bias of every Sbox is less than 0.16 and guessing 2 or more bits will affect at least as many Sboxes as guessing a single bit.
To find the trail, we first find the bits affected in one round and go backwards along the Fiestel round and the diffusion layer(if it is present). This helps us find the bits affected in the previous round. We find the bits affected in the previous round in such a way that the least number of Sboxes are affected in the previous round. This is done over all possible bit locations(from 1 to 128).
To find the bias, we use the piling up lemma to find the bias over all the affected SBoxes. For each SBox, we take the maximum bias over all linear combinations of input and output.
Using this method, we find that it takes 17 rounds to reach a bias less than $1/2^{128}$. To ensure a better security, we add 3 more rounds to have 20 rounds in the cipher.
The program to find the deadliest trails for each bit, and the most deadly trail is generated in “LC_DC/main_LC.py”. The supporting functions are present in “LC_DC/LC.py”.

13) To find the most deadly differential trail, we follow a procedure similar to the linear trail. We find the least propagation ratio over all output differences for each Sbox in the final round. Then, we find the bits affected in the previous round similar to the linear trails.
To find the propagation ratio, we multiply the maximum propagation ratio of the affected SBoxes in each round.
Using this method, we find that it takes 13 rounds to reach a propagation ratio less than $1/2^{128}$.
The program to find the deadliest trails for each output difference, and the most deadliest trail is present in “LC_DC/main_DC.py”. The supporting functions are present in “LC_DC/LC.py”.

14) The following changes were made:
    a) Question 2 - Number of rounds were changed to 20. This is because our earlier calculation of the bias did not use the piling up lemma, due to which we got a fewer number of rounds.
    b) Question 3 - We changed the round key addition to be after the Fiestel step. This ensured that we have 128-bit keys in every round instead of just 64 bits.
    c) Diffusion layer was changed. This is because the previous diffusion layer had a lot of identity mappings(diffusion(x) = x, where x is the position of the bit from 1 to

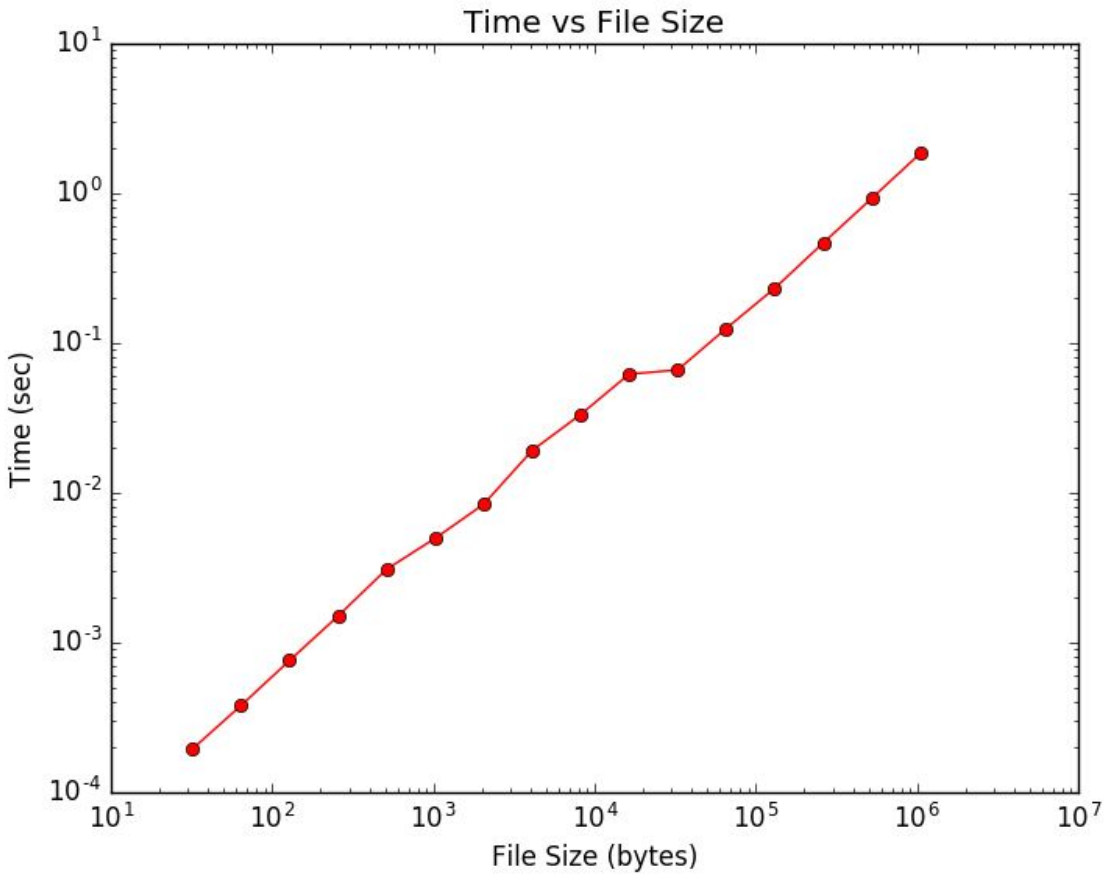
128). So, it required approximately 70 rounds to reach a bias less than $1/2^{128}$. The new diffusion layer requires only 17 rounds.

d) The diffusion layer is not present in the last 2 rounds. This is because the presence of a diffusion layer makes it easier to guess right half of the keys in the last round. To prevent this, we would have to increase the number of rounds, making the encryptions slower.

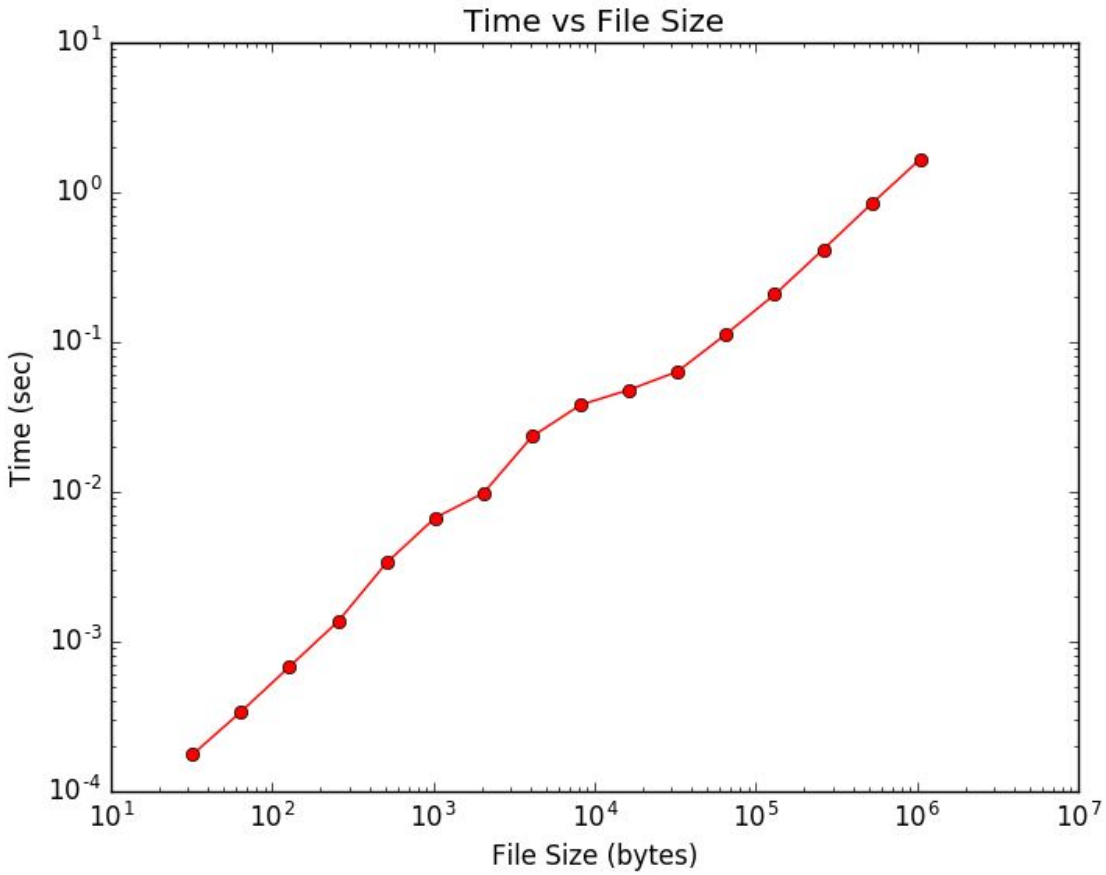
15) We are mostly using bitwise operations. The usage of arithmetic operations has been minimised. We are trying to optimise wrt CPU cycles. Our algorithm takes a little over 2 KB of runtime memory. We tried using unions for faster access but it did not give any improvement in the encryption time.

16) No changes have been made.

17) Plot for Time vs File Size during encryption



Plot for Time vs File Size during decryption



18) Instructions for generating plaintext files, encrypting them and decrypting them are given in “Implementation/readme.txt”.