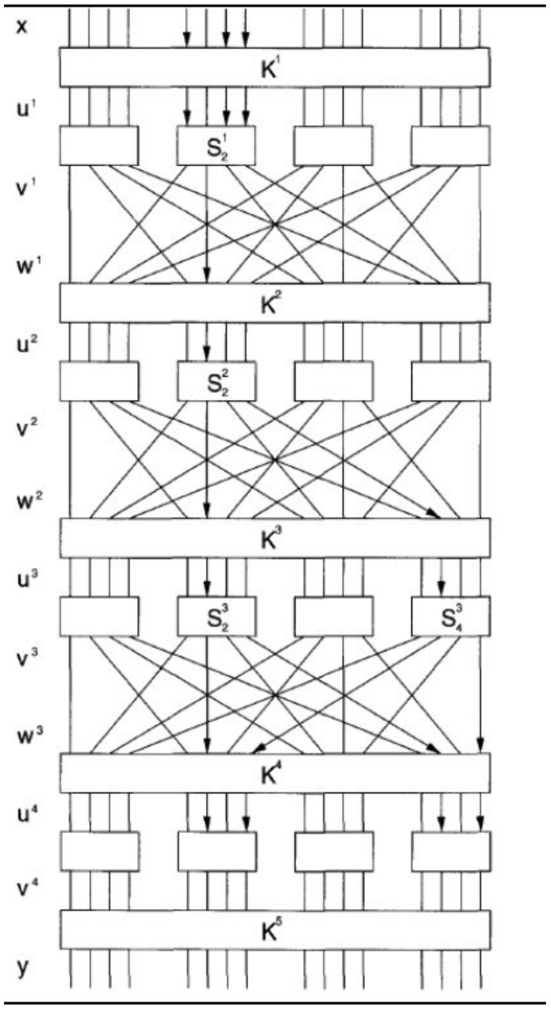
CS 6530 Assignment 1
Section 2

Team: CodeBreakers
CS14B050 Rachit
CS14B056 Vishwanath

References:
Linear cryptanalysis tutorial
Differential cryptanalysis tutorial
Matsui DES

Assumptions

In order to make the analysis easy we make the assumptions that the rounds undergo a key xor and there is no feistel structure involved i.e we have a simple xoring of the key then a sbox and a permutation layer, we make such a distinction so as to ease our analysis, this assumption would affect as the trails would be different, but it is enough to show that linear and differential cryptanalysis are more difficult than brute force. The assumption is this:-



Though in our cipher there are 4 block ciphers such that first two s boxes are different from the last two s boxes and the s box in odd layer is different from s box in even layer. And in our cipher we go through 10 rounds instead of 4 here. Also we have a total of 16 s boxes in one layer instead of the 4 here.
We mention a trail now.

12.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.

Ans. We wrote some trails and mention the bias in those trails, in order to do linear cryptanalysis we would need a large amount of plaintext ciphertext pairs. We then guess the key, and on basis of the guessed key we find the observed bias in our trail as mentioned we can then guess how good is the key that we guessed.
As our rounds go though we find a trail that has a higher bias, the higher the bias means we can say with a higher probability if a key can be used to check as being correct or not.
The best attack had a complexity of 1.65726383457e-77. This is far from brute force as 10^-70< 2^-210 and worser than 2^128 which is the brute force attack on the keys.
To do this we have implemented a greedy strategy, strategy is such that initialy and at each round we choose a sbox that would give us the best bias and then we continue with that trail.

Here we present the code

```
#Initially setting the values of the second s box here.
cols = 2 ** outputbits
maxvalue = 0
maxi = maxj = -1
for i in xrange(1,rows):
    for j in xrange(cols):
        x = lincrypt1[i][j]
        if x<0 :
            x = -x
        if (maxvalue < x):
            maxvalue = x
            maxi = i
            maxj = j

bias = 1
inround = np.zeros(shape=(128),dtype = np.int)
outround = np.zeros(shape=(128),dtype = np.int)
outzeros = np.zeros(shape=(128),dtype = np.int)
roundcarry = np.zeros(shape=(10,128),dtype = np.int)
roundno = 0
biascount = 0
```

```
count = 0
for i in xrange(8):
   inround[32+i] = c[maxi][i];
while roundno<10:
   for s in xrange(16):
         no = 0
         for i in xrange(8):
               no = 2*no + inround[8*s+i]
         if(no>0):
               maxvalue = 0
               maxj = -1
               for j in xrange(cols):
                     if(roundno%2==0):
                           if(s%2==0):
                                 x = lincrypt1[no][j]
                           else:
                                 x = lincrypt2[no][j]
                     else:
                           if(s%2==0):
                                 x = lincrypt3[no][j]
                           else:
                                 x = lincrypt4[no][j]
                     if(x<0):
                           x = -x
                     if(maxvalue<x):             #picking on a greedy strategy
                           maxvalue = x
                           maxj = j
               for i in xrange(8):
                     outround[8*s+i] = c[maxj][i];
               bias = bias*maxvalue
               count += 1
               while(bias>1):
                     bias = bias*(1.0)/256
                     biascount += 1
   for i in xrange(128):
         inround[p[i]] = outround[i];
         outround[i]= 0
   for i in xrange(128):
         roundcarry[roundno][i] = inround[i]
   roundno += 1
while(biascount<count):
   bias = bias*(1.0)/256
   biascount += 1
biascount = 0
while(biascount<count-1):
    bias = bias*(2.0)
    biascount += 1
print roundcarry
print bias
```

<u>Output</u>

These are the inputs before each round are

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1
  1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
  0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
  0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0
  0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
  0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0 1 1 0 1 0 0 1 0 1 0 0 0
  0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0
  0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0
  0 1 0 0 1 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 0
  0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1]
 [1 1 1 0 0 1 0 1 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1 1 0
  0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
  0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 1 0 0 0 1
  0 1 0 0 0 0 0 1 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1 1 1 0 1 1 0 0
  1 0 0 0 1 0 0 1 1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0
  0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0]
 [1 1 0 1 0 0 1 0 0 1 0 0 0 1 1 0 0 1 1 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1
  0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0
  1 0 1 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 0 0 1 0 1 0 1
  1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [1 1 0 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1
  1 1 0 0 1 1 0 0 1 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 0 1 0 0 0 1 0 1 1 0 0
  0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0
  0 0 0 0 0 1 0 1 1 0 1 1 1 0 1 1 0]]

[[  1.09375000e-01]
 [  2.44140625e-04]
 [  2.98023224e-05]
 [  8.14907253e-09]
 [  9.99200722e-14]
 [  1.30104261e-14]
 [  1.21759292e-15]
 [  7.61109925e-13]
 [  2.54009677e-17]
 [  3.37288564e-17]]
```
Also these are the bias created in each round, since the bias created in each round is small eventually, our decisions for 10 rounds holds good.
The code is present in trails.py and requires some csv files to be run.

13.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.
Ans. We wrote some trails and mention the probabilities in those trails, in order to do differential cryptanalysis we would need a large amount of plaintext ciphertext pairs. We then guess the key, and on basis of the guessed key we find the observed properties in our trail as mentioned we can then guess how good is the key that we guessed.
As our rounds go though we find a trail that has a higher probability for the change in our input bits, the higher the probability means we can say with a higher probability if a key can be used to check as being correct or not.
The best attack had a complexity of 9.72346137166e-63. This is far from brute force as 10^-63< 2^-189 and worser than 2^128 which is the brute force attack on the keys.
To do this we have implemented a greedy strategy, strategy is such that at each round we choose a sbox that would give us the best probability and then we continue with that trail.

Here we observed that initial settings of some random input were better than choosing the initial layer greedily. Hence we went through a lot of initial configurations and we report the best attack that we could find.

```
while roundno<10:
   for s in xrange(16):                                //divides into 16 s boxes
         no = 0
         for i in xrange(8):                           //for each s box see if an input is present
                no = 2*no + inround[8*s+i]
         if(no>0):
                maxvalue = 0                           //value with max prob
                maxj = -1
                for j in xrange(cols):                 //our algo uses multiple s boxes and hence
its more difficult to find a trail
                        if(roundno%2==0):
                                if(s%2==0):
                                        x = diffcrypt1[no][j]
                                else:
                                        x = diffcrypt2[no][j]
                        else:
                                if(s%2==0):
                                        x = diffcrypt3[no][j]
                                else:
                                        x = diffcrypt4[no][j]

                        if(maxvalue<x):
                                maxvalue = x
                                maxj = j
                for i in xrange(8):
                        outround[8*s+i] = c[maxj][i];          //outround right now has unpermuted
values
                bias = bias*maxvalue
                count += 1
                while(bias>1):
                        bias = bias*(1.0)/256
                        biascount += 1
   for i in xrange(128):
         inround[p[i]] = outround[i];                          //permute based on a permutation p
         outround[i]= 0
   print inround                                               //permutation that goes into a next
round
   roundno += 1
while(biascount<count):                                        //just dividing to get a probability
   bias = bias*(1.0)/256
   biascount += 1
```

We mention the output at the end of each round for our best trail

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
  0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0
  0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0
  0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]]
```

This is the product of the highest probability at each round i.e propagation ratio in each round

```
[[  7.81250000e-03]
 [  7.81250000e-03]
 [  7.81250000e-03]
 [  6.10351562e-05]
 [  1.56250000e-02]
 [  4.76837158e-07]
 [  4.76837158e-07]
 [  2.91038305e-11]
 [  4.54747351e-13]
 [  7.10542736e-15]]
```

14.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.

Ans. We changed our permutation layer as there was a bug in the permutation layer mentioned last time, some inputs were present more than twice and hence it wasn't a permutation.

P = [46, 12, 21, 67, 10, 48, 1, 23, 22, 4, 45, 75, 2, 28, 9, 47, 30, 24, 5, 115, 26, 32, 49, 7, 6, 52, 29, 27, 50, 8, 25, 95, 14, 44, 33, 99, 42, 16, 53, 55, 54, 36, 13, 107, 34, 56, 41, 15, 62, 60, 37, 19, 58, 64, 17, 39, 38, 20, 61, 123, 18, 40, 57, 127, 110, 76, 85, 3, 74, 112, 65, 87, 86, 68, 109, 11, 66, 96, 73, 111, 94, 92, 69, 51, 90, 88, 113, 71, 70, 116, 93, 63, 82, 72, 89, 31, 78, 108, 117, 35, 106, 80, 97, 122, 118, 100, 77, 43, 98, 120, 105, 79, 126, 124, 101, 83, 119, 128, 59, 103, 102, 84, 125, 91, 114, 104, 121, 81]

The bug was at value 15 and 93. Now it is fixed.

References:

[Linear cryptanalysis tutorial]
[Differential cryptanalysis tutorial]
[Matsui DES]