Sinhgad College of Engineering Department of Computer Engineering



Name of Student : Abhishek Turukmane

Roll.No : 405C032 PRN Number: 71707754D

Class : BE DIV: 3

Name of Laboratory : Laboratory practical(LPIII)

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Name of Experiment** | **Start Date** | **Completion Date** |
| 1 | To implement Linear Regression to find the equation of the best fit line for  given data. |  |  |
| 2 | To implement Decision Tree Classifier. |  |  |
| 3 | To implement k-NN algorithm for classifying the points on given graph. |  |  |
| 4 | To implement K-means Algorithm for clustering. |  |  |
| 5 | To implement a Simplified Data Encryption Standard (S-DES) algorithm. |  |  |
| 6 | To implement a Simplified Advanced Encryption Standard (S-AES)  algorithm. |  |  |
| 7 | To implement a Diffie-Hellman Key Exchange algorithm. |  |  |
| 8 | To implement a RSA algorithm. |  |  |
| 9 | Mini Project on Machine Learning |  |  |
| 10 | Mini Project on Information and Cyber Security |  |  |

# Assignment No. – 1

Code of the program –

#import numpy as np import pandas as py

import matplotlib.pyplot as plt

print(“Name: Abhishek Turukmane”) def estimate\_coef(x, y):

n = np.size(x) # number of observations/points

m\_x, m\_y = np.mean(x), np.mean(y) # mean of x and y vector

# calculating cross-deviation and deviation about x SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

# calculating regression coefficients b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x return(b\_0, b\_1)

def plot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot plt.scatter(x, y, color = "m",

marker = "o", s = 30) # predicted response vector

y\_pred = b[0] + b[1]\*x

# plotting the regression line plt.plot(x, y\_pred, color = "g") # putting labels plt.xlabel('x')

plt.ylabel('y')

plt.show() # function to show plot

def main():

data = py.read\_csv("linear.csv") #reading table data from csv file

x = data['x']

y = data['y']

b = estimate\_coef(x,y) # estimating coefficients print("Estimated coefficients:\nb\_0 = {} \

\nb\_1 = {}".format(b[0], b[1])) plot\_regression\_line(x,y,b) # plotting regression line

if name == " main ": main()

Input for the code – **#linear.csv** x,y

10,95

9,80

2,10

15,50

10,45

16,98

11,38

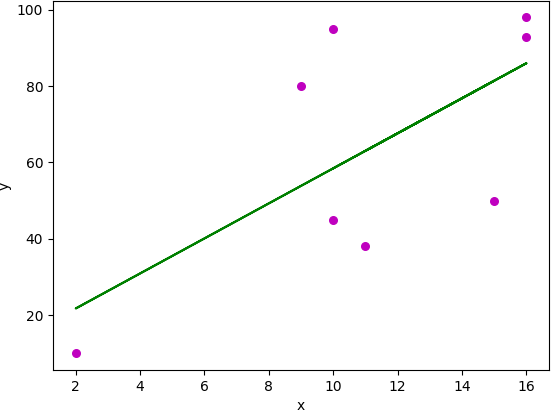
16,93

Output –

(base) C:\Users\Abhishek \Downloads\ML\A1>python linear.py Name: Abhishek Turukmane

Estimated coefficients: b\_0 = 12.584627964

b\_1 = 4.58789860998



# Assignment No – 2

Code of the program –

**dtree.py:-**

#Abhishek Turukmane import numpy as np import pandas as pd

from sklearn.preprocessing import LabelEncoder from sklearn.tree import DecisionTreeClassifier from sklearn.externals.six import StringIO import pydotplus as pdd

from IPython.display import Image

from sklearn.tree import export\_graphviz

print(“Name: Abhishek Turukmane”)

data=pd.read\_csv("sales.csv") data

data.describe()

print(data['Buys'].value\_counts()) le=LabelEncoder();

#data=data.apply(le.fit\_transform)

x=data.iloc[:,:-1] #-1 means don't take last column x=x.apply(le.fit\_transform)

#Store labels in Y y=data.iloc[:,-1]

classifier=DecisionTreeClassifier(criterion='entropy') classifier.fit(x,y)

#Predict value for the given Expression

#[Age < 21, Income = Low,Gender = Female, Marital Status = Married]

test\_x=np.array([1,1,0,0])

pred\_y=classifier.predict([test\_x]) print("Predicted class for input [Age < 21, Income =

Low,Gender = Female, Marital Status = Married]\n", test\_x," is ",pred\_y[0])

#method to generate graph

dot\_dat=export\_graphviz(classifier,out\_file=None,feature\_names

=x.columns,class\_names=["No","Yes"]) graph=pdd.graph\_from\_dot\_data(dot\_dat) graph.write\_png("tree.png")

Image(graph.create\_jpg())

Input for the code –

**#sales.csv** ID,Age,Income,Gender,MaritialStatus,Buys 1,<21,High,Male,Single,No 2,<21,High,Male,Married,No

3,21-35,High,Male,Single,Yes 4,>35,Medium,Male,Single,Yes

5,>35,Low,Female,Single,Yes

6,>35,Low,Female,Married,No 7,21-35,Low,Female,Married,Yes 8,<21,Medium,Male,Single,No 9,<21,Low,Female,Married,Yes 10,>35,Medium,Female,Single,Yes 11,<21,Medium,Female,Married,Yes

12,21-35,Medium,Male,Married,Yes 13,21-35,High,Female,Single,Yes

14,>35,Medium,Male,Married,No

Output –

(base) C:\Users\Saurabh\Downloads\ML\A2>python dtree.py Name: Abhishek Turukmane

Yes 9

No 5

Name: Buys, dtype: int64

Predicted class for input [Age < 21, Income = Low,Gender = Female, Marital Status = Married]

[1 1 0 0] is Yes

Assignment No – 3

Code of the program –

**knn.py:-**

#Abhishek Turukmane import numpy as np import pandas as pd import math

print(“Name: Abhishek Turukmane”) x=np.array([[2,4],[4,2],[4,4],[4,6],[6,2],[6,4]])

y=np.array([0,0,1,0,1,0])

# 0=negative 1=positive class

def eucledian\_distance(x1,y1,x2,y2):

return math.sqrt((x1-x2)\*\*2+(y1-y2)\*\*2)

def chooseK(arr):

print("Size of array :",arr.shape[0]) k=round(math.sqrt(arr.shape[0])) if(k%2==0):

k=k+1;

#k should be odd so that classfication can be done properly(No chance of 50%-50% classification)

print("Choosen value of K : ",k) return k;

chooseK(x)

def classifyPoint(x,y,point,k): inputSize=x.shape[0];

distance=[]; #for string eucledian distance for i in range(inputSize):

distance.append(eucledian\_distance(point[0],point[1],x[i][0],x [i][1]));

mergedList=list(zip(distance,y));

mergedList.sort(); #sort according to increasing distance

freq0=0; #Freq of group 0 (negative) freq1=0; #Freq of group 1 (positive)

for i in range(int(k)): #Iterate for k neighbours if(mergedList[i][1]==0):

freq0=freq0+1;

elif (mergedList[i][1]==1): freq1=freq1+1;

if(freq0>freq1): return 0;

else:

return 1;

def main():

print("Input X coordinate"); x\_co=int(input()) print("Enter Y coordinate") y\_co=int(input())

pointt=(x\_co,y\_co) print(pointt) k=chooseK(x); label="--"

if(classifyPoint(x=x,y=y,point=pointt,k=k)==0): label="Negative";

else:

label="Positive";

print("Point {} belongs to {} class".format(pointt,label)) print (classifyPoint(x=x,y=y,point=pointt,k=k))

main()

Output –

(base) C:\Users\Saurabh\Downloads\ML\A3>python knn.py Name: Abhishek Turukmane

('Size of array :', 6) ('Choosen value of K : ', 3.0) Input X coordinate

6

Enter Y coordinate 6

(6, 6)

('Size of array :', 6) ('Choosen value of K : ', 3.0)

Point (6, 6) belongs to Negative class

0

# Assignment No – 4

Code of the program – **#kmeans.py: -** #Abhishek Turukmane import numpy as np

import matplotlib.pyplot as plt import math

print(“Name: Abhishek Turukmane”)

x = np.array([0.1,0.15,0.08,0.16,0.2,0.25,0.24,0.3])

y = np.array([0.6,0.71,0.9,0.85,0.3,0.5,0.1,0.2])

plt.plot(x,y,"o") plt.show()

def eucledian\_distance(x1,y1,x2,y2):

return math.sqrt((x1-x2)\*\*2+(y1-y2)\*\*2)

def manhattan\_distance(x1,y1,x2,y2):

return math.fabs(x1-x2)+math.fabs(y1-y2)

def returnCluster(m1,m2,x\_co,y\_co):

#if we use manhattan\_distance then clusters are classified more correctly..

distance1=manhattan\_distance(m1[0],m1[1],x\_co,y\_co)

distance2=manhattan\_distance(m2[0],m2[1],x\_co,y\_co)

if(distance1<distance2): return 1;

else:

return 2;

#initial centroids for cluster1 nd cluster 2 m1=[0.1,0.6]

m2=[0.3,0.2]

#difference and iteration is for controlling iteration difference = math.inf

threshold=0.02 iteration=0;

while difference>threshold: #use any one condition #iteration one is easy

print("Iteration ",iteration, " : m1=",m1, " m2=",m2) cluster1=[];

cluster2=[];

#step1 assign all points to nearest cluster for i in range(0,np.size(x)):

clusterNumber=returnCluster(m1,m2,x[i],y[i]) point=[x[i],y[i]]

if clusterNumber==1:

cluster1.append(point); else:

cluster2.append(point)

print("cluster 1", cluster1,"\nCLuster 2: ", cluster2)

#step 2: Calculating new centriod for cluster1 m1\_old=m1;

m1=[]

m1=np.mean(cluster1, axis=0) #axis=0 means columnwise

#calculating centroid for cluster2 m2\_old=m2;

m2=[];

m2=np.mean(cluster2,axis=0) print("m1 = ",m1," m2=",m2)

#adjusting diffrences of adjustment between m1 nd m1\_old xAvg=0.0;

yAvg=0.0;

xAvg=math.fabs(m1[0]-m1\_old[0])+math.fabs(m2[0]-m2\_old[0]) xAvg=xAvg/2;

yAvg=math.fabs(m1[1]-m1\_old[1])+math.fabs(m2[1]-m2\_old[1]) yAvg=yAvg/2;

if(xAvg>yAvg): difference=xAvg;

else:

difference=yAvg;

print("Difference : ", difference) iteration+=1;

print("") #final Output

print("Cluster 1 centroid : m1 = ",m1) print("CLuster 1 points: ", cluster1) print("Cluster 2 centroid : m2 = ",m2) print("CLuster 2 points: ", cluster2)

clust1=np.array(cluster1) clust2=np.array(cluster2)

#cluster 1 points plt.plot(clust1[:,0],clust1[:,1],"o")

#cluster2 points

plt.plot(clust2[:,0], clust2[:,1],"\*")

#centroids plt.plot([m1[0],m2[0]],[m1[1],m2[1]],"^")

plt.show() #same code

plt.scatter(clust1[:,0],clust1[:,1])

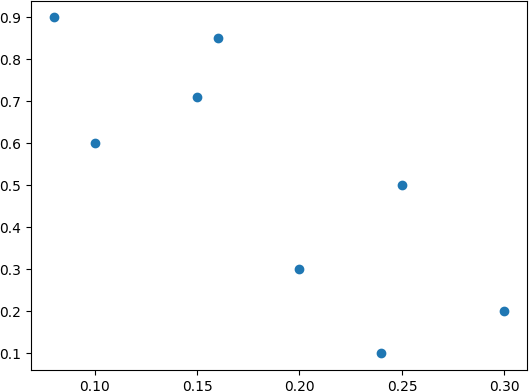
plt.scatter(clust2[:,0],clust2[:,1]) plt.scatter([m1[0],m2[0]],[m1[1],m2[1]],marker="\*")

plt.show()

Output –

(base) C:\Users\Saurabh\Downloads\ML\A4>python kmean.py Name: Abhishek Turukmane

Iteration 0 : m1= [0.1, 0.6] m2= [0.3, 0.2]



cluster 1 [[0.1, 0.6], [0.15, 0.71], [0.08, 0.9], [0.16,

0.85], [0.25, 0.5]]

CLuster 2: [[0.2, 0.3], [0.24, 0.1], [0.3, 0.2]]

m1 = [0.148 0.712] m2= [0.24666667 0.2 ]

Difference : 0.05600000000000001

Iteration 1 : m1= [0.148 0.712] m2= [0.24666667 0.2 ]

cluster 1 [[0.1, 0.6], [0.15, 0.71], [0.08, 0.9], [0.16,

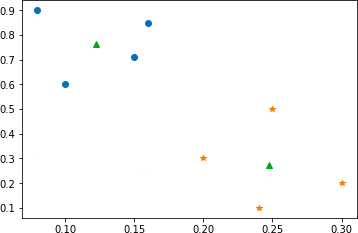
0.85]]

CLuster 2: [[0.2, 0.3], [0.25, 0.5], [0.24, 0.1], [0.3, 0.2]]

m1 = [0.1225 0.765 ] m2= [0.2475 0.275 ]

Difference : 0.06400000000000002

Iteration 2 : m1= [0.1225 0.765 ] m2= [0.2475 0.275 ]



cluster 1 [[0.1, 0.6], [0.15, 0.71], [0.08, 0.9], [0.16,

0.85]]

CLuster 2: [[0.2, 0.3], [0.25, 0.5], [0.24, 0.1], [0.3, 0.2]]

m1 = [0.1225 0.765 ] m2= [0.2475 0.275 ]

Difference : 0.0

|  |  |  |
| --- | --- | --- |
| Cluster 1 centroid | : m1 = | [0.1225 0.765 ] |
| CLuster 1 points:  [0.16, 0.85]] | [[0.1, | 0.6], [0.15, 0.71], [0.08, 0.9], |
| Cluster 2 centroid | : m2 = | [0.2475 0.275 ] |
| CLuster 2 points:  [0.3, 0.2]] | [[0.2, | 0.3], [0.25, 0.5], [0.24, 0.1], |

# Assignment No – 5

#Initial permut made on the key CP\_1 = [57, 49, 41, 33, 25, 17, 9,

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Code of the program –  """  @author: Abhishek Turukmane """  #Initial permut matrix for the datas | | | | | | | |
| PI = [58, | 50, | 42, | 34, | 26, | 18, | 10, | 2, |
| 60, | 52, | 44, | 36, | 28, | 20, | 12, | 4, |
| 62, | 54, | 46, | 38, | 30, | 22, | 14, | 6, |
| 64, | 56, | 48, | 40, | 32, | 24, | 16, | 8, |
| 57, | 49, | 41, | 33, | 25, | 17, | 9, 1, | |
| 59, | 51, | 43, | 35, | 27, | 19, | 11, | 3, |
| 61, | 53, | 45, | 37, | 29, | 21, | 13, | 5, |
| 63, | 55, | 47, | 39, | 31, | 23, | 15, | 7] |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1, 58, 50, | | 42, | 34, | 26, | 18, |
| 10, | 2, 59, | 51, | 43, | 35, | 27, |
| 19, | 11, 3, | 60, | 52, | 44, | 36, |
| 63, | 55, 47, 39, 31, 23, 15, | | | | |
| 7, 62, 54, | | 46, | 38, | 30, | 22, |
| 14, 6, 61, | | 53, | 45, | 37, | 29, |
| 21, 13, 5, | | 28, | 20, | 12, | 4] |

#Permut applied on shifted key to get Ki+1 CP\_2 = [14, 17, 11, 24, 1, 5, 3, 28,

15, 6, 21, 10, 23, 19, 12, 4,

26, 8, 16, 7, 27, 20, 13, 2,

#Expand matrix to get a 48bits matrix of datas to apply the xor with Ki

E = [32, 1, 2, 3, 4, 5,

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 41, | 52, | 31, | 37, | 47, | 55, | 30, | 40, |
| 51, | 45, | 33, | 48, | 44, | 49, | 39, | 56, |
| 34, | 53, | 46, | 42, | 50, | 36, | 29, | 32] |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4, | 5, 6, 7, 8, 9, | | | | | | |
| 8, | 9, 10, 11, 12, 13, | | | | | | |
| 12, | 13, | 14, | 15, | 16, | 17, | | |
| 16, | 17, | 18, | 19, | 20, | 21, | | |
| 20, | 21, | 22, | 23, | 24, | 25, | | |
| 24, | 25, | 26, | 27, | 28, | 29, | | |
| 28, | 29, | 30, | 31, | 32, | 1] | | |
| #SBOX S\_BOX = [  [[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],  [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],  [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],  [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13],  ], | | | | | | | |
| [[15, 1, | 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, | | | | | 0, | 5, 10], |
| [3, 13, | 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, | | | | | 9, | 11, 5], |
| [0, 14, | 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, | | | | | 3, | 2, 15], |
| [13, 8, | 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, | | | | | 5, | 14, 9], |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ],  [[10, | 0, | 9, | 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, | | | | | | | | 8], |
| [13, | 7, | 0, | 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, | | | | | | | | 1], |
| [13, | 6, | 4, | 9, 8, 15, | | | 3, | 0, | 11, 1, | 2, 12, 5, 10, | 14, | 7], |
| [1, | 10, | 13, | 0, | 6, | 9, | 8, | 7, | 4, 15, | 14, 3, 11, 5, | 2, 12], | |
| ], |  |  |  |  |  |  |  |  |  |  | |
| [[7, | 13, | 14, | 3, | 0, | 6, | 9, | 10, | 1, 2, | 8, 5, 11, 12, | 4, 15], | |
| [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],  [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],  [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14],  ],  [[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9], | | | | | | | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| [14, 11, 2, 12, 4, 7, | 13, 1, 5, | 0, 15, | 10, 3, 9, 8, 6], |
| [4, 2, 1, 11, 10, 13, | 7, 8, 15, | 9, 12, | 5, 6, 3, 0, 14], |
| [11, 8, 12, 7, 1, 14,  ], | 2, 13, 6, | 15, 0, | 9, 10, 4, 5, 3], |
| [[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],  [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],  [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],  [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13],  ],  [[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],  [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],  [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],  [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12],  ], | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [[13, 2, | | 8, 4, 6, 15, 11, | 1, | 10, 9, 3, | 14, | 5, 0, 12, | | 7], |
| [1, | 15, | 13, 8, 10, 3, 7, | 4, | 12, 5, 6, | 11, | 0, 14, 9, | | 2], |
| [7, | 11, | 4, 1, 9, 12, 14, | 2, | 0, 6, 10, | 13, | 15, 3, 5, | | 8], |
| [2, | 1, 14, 7, 4, 10, 8, 13, | | | 15, 12, 9, 0, | | 3, | 5, 6, 11], | |

]

]

#Permut made after each SBox substitution for each round P = [16, 7, 20, 21, 29, 12, 28, 17,

1, 15, 23, 26, 5, 18, 31, 10,

2, 8, 24, 14, 32, 27, 3, 9,

19, 13, 30, 6, 22, 11, 4, 25]

#Final permut for datas after the 16 rounds

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PI\_1 = [40, | 8, | 48, | 16, | 56, | 24, | 64, | 32, |
| 39, | 7, | 47, | 15, | 55, | 23, | 63, | 31, |
| 38, | 6, | 46, | 14, | 54, | 22, | 62, | 30, |
| 37, | 5, | 45, | 13, | 53, | 21, | 61, | 29, |
| 36, | 4, | 44, | 12, | 52, | 20, | 60, | 28, |
| 35, | 3, | 43, | 11, | 51, | 19, | 59, | 27, |
| 34, | 2, | 42, | 10, | 50, | 18, | 58, | 26, |
| 33, | 1, | 41, | 9, 49, 17, 57, 25] | | | | |

#Matrix that determine the shift for each round of keys SHIFT = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

def string\_to\_bit\_array(text):#Convert a string into a list of bits array = list()

for char in text:

binval = binvalue(char, 8)#Get the char value on one byte array.extend([int(x) for x in list(binval)]) #Add the bits

to the final list return array

def bit\_array\_to\_string(array): #Recreate the string from the bit array

res = ''.join([chr(int(y,2)) for y in [''.join([str(x) for x in

\_bytes]) for \_bytes in nsplit(array,8)]]) return res

def binvalue(val, bitsize): #Return the binary value as a string of the given size

binval = bin(val)[2:] if isinstance(val, int) else bin(ord(val))[2:]

if len(binval) > bitsize:

raise "binary value larger than the expected size" while len(binval) < bitsize:

binval = "0"+binval #Add as many 0 as needed to get the wanted size

return binval

def nsplit(s, n):#Split a list into sublists of size "n" return [s[k:k+n] for k in range(0, len(s), n)]

ENCRYPT=1 DECRYPT=0

class des():

def init (self): self.password = None self.text = None self.keys = list()

def run(self, key, text, action=ENCRYPT, padding=False): if len(key) < 8:

raise "Key Should be 8 bytes long" elif len(key) > 8:

key = key[:8] #If key size is above 8bytes, cut to be

8bytes long

self.password = key self.text = text

if padding and action==ENCRYPT: self.addPadding()

elif len(self.text) % 8 != 0:#If not padding specified data size must be multiple of 8 bytes

raise "Data size should be multiple of 8"

self.generatekeys() #Generate all the keys

text\_blocks = nsplit(self.text, 8) #Split the text in blocks of 8 bytes so 64 bits

result = list()

for block in text\_blocks:#Loop over all the blocks of data block = string\_to\_bit\_array(block)#Convert the block in

bit array permutation

(48bits)

Ki

block = self.permut(block,PI)#Apply the initial

g, d = nsplit(block, 32) #g(LEFT), d(RIGHT) tmp = None

for i in range(16): #Do the 16 rounds

d\_e = self.expand(d, E) #Expand d to match Ki size

if action == ENCRYPT:

tmp = self.xor(self.keys[i], d\_e)#If encrypt use

else:

tmp = self.xor(self.keys[15-i], d\_e)#If decrypt

start by the last key

tmp = self.substitute(tmp) #Method that will apply

the SBOXes

tmp = self.permut(tmp, P) tmp = self.xor(g, tmp)

g = d d = tmp

result += self.permut(d+g, PI\_1) #Do the last permut and append the result to result

final\_res = bit\_array\_to\_string(result) if padding and action==DECRYPT:

return self.removePadding(final\_res) #Remove the padding if decrypt and padding is true

else:

return final\_res #Return the final string of data ciphered/deciphered

bits

def substitute(self, d\_e):#Substitute bytes using SBOX subblocks = nsplit(d\_e, 6)#Split bit array into sublist of 6

result = list()

for i in range(len(subblocks)): #For all the sublists block = subblocks[i]

row = int(str(block[0])+str(block[5]),2)#Get the row

with the first and last bit

column = int(''.join([str(x) for x in block[1:][:- 1]]),2) #Column is the 2,3,4,5th bits

val = S\_BOX[i][row][column] #Take the value in the SBOX appropriated for the round (i)

bin = binvalue(val, 4)#Convert the value to binary

result += [int(x) for x in bin]#And append it to the resulting list

return result

def permut(self, block, table):#Permut the given block using the given table (so generic method)

return [block[x-1] for x in table]

def expand(self, block, table):#Do the exact same thing than permut but for more clarity has been renamed

return [block[x-1] for x in table]

def xor(self, t1, t2):#Apply a xor and return the resulting list return [x^y for x,y in zip(t1,t2)]

def generatekeys(self):#Algorithm that generates all the keys self.keys = []

key = string\_to\_bit\_array(self.password)

key = self.permut(key, CP\_1) #Apply the initial permut on

the key

g, d = nsplit(key, 28) #Split it in to (g->LEFT),(d->RIGHT) for i in range(16):#Apply the 16 rounds

g, d = self.shift(g, d, SHIFT[i]) #Apply the shift

associated with the round (not always 1)

tmp = g + d #Merge them self.keys.append(self.permut(tmp, CP\_2)) #Apply the

permut to get the Ki

def shift(self, g, d, n): #Shift a list of the given value return g[n:] + g[:n], d[n:] + d[:n]

def addPadding(self):#Add padding to the datas using PKCS5 spec. pad\_len = 8 - (len(self.text) % 8)

self.text += pad\_len \* chr(pad\_len)

def removePadding(self, data):#Remove the padding of the plain text (it assume there is padding)

pad\_len = ord(data[-1]) return data[:-pad\_len]

def encrypt(self, key, text, padding=False): return self.run(key, text, ENCRYPT, padding)

def decrypt(self, key, text, padding=False): return self.run(key, text, DECRYPT, padding)

if name == ' main ':

print("Name: Abhishek Turukmane") key = "secret\_key"

text= "Hello world" d = des()

r = d.encrypt(key,text,padding=True) r2 = d.decrypt(key,r,padding=True) print("Ciphered: %r" % r)

print("Deciphered: ", r2

Output –

(base) C:\Users\Saurabh\Downloads\A1>python des.py Name: Abhishek Turukmane

Ciphered: 'ßåýåÚ\x9f\\\x9d\x89r\x9c\x16Û\x0fá\x8b'

Deciphered: Hello world.

# Assignment No – 6

Code of the program –

"""

@author: Abhishek Turukmane """

import sys

# S-Box

sBox = [0x9, 0x4, 0xa, 0xb, 0xd, 0x1, 0x8, 0x5, 0x6, 0x2, 0x0, 0x3, 0xc, 0xe, 0xf, 0x7]

# Inverse S-Box

sBoxI = [0xa, 0x5, 0x9, 0xb, 0x1, 0x7, 0x8, 0xf, 0x6, 0x0, 0x2, 0x3, 0xc, 0x4, 0xd, 0xe]

# Round keys: K0 = w0 + w1; K1 = w2 + w3; K2 = w4 + w5 w = [None] \* 6

def mult(p1, p2):

"""Multiply two polynomials in GF(2^4)/x^4 + x + 1""" p = 0

while p2:

if p2 & 0b1:

p ^= p1

p1 <<= 1

if p1 & 0b10000: p1 ^= 0b11

p2 >>= 1

return p & 0b1111

def intToVec(n):

"""Convert a 2-byte integer into a 4-element vector""" return [n >> 12, (n >> 4) & 0xf, (n >> 8) & 0xf, n & 0xf]

def vecToInt(m):

"""Convert a 4-element vector into 2-byte integer""" return (m[0] << 12) + (m[2] << 8) + (m[1] << 4) + m[3]

def addKey(s1, s2):

"""Add two keys in GF(2^4)"""

return [i ^ j for i, j in zip(s1, s2)]

def sub4NibList(sbox, s):

"""Nibble substitution function""" return [sbox[e] for e in s]

def shiftRow(s): """ShiftRow function"""

return [s[0], s[1], s[3], s[2]]

def keyExp(key):

"""Generate the three round keys""" def sub2Nib(b):

"""Swap each nibble and substitute it using sBox""" return sBox[b >> 4] + (sBox[b & 0x0f] << 4)

Rcon1, Rcon2 = 0b10000000, 0b00110000

w[0] = (key & 0xff00) >> 8 w[1] = key & 0x00ff

w[2] = w[0] ^ Rcon1 ^ sub2Nib(w[1]) w[3] = w[2] ^ w[1]

w[4] = w[2] ^ Rcon2 ^ sub2Nib(w[3]) w[5] = w[4] ^ w[3]

def encrypt(ptext):

"""Encrypt plaintext block""" def mixCol(s):

return [s[0] ^ mult(4, s[2]), s[1] ^ mult(4, s[3]),

s[2] ^ mult(4, s[0]), s[3] ^ mult(4, s[1])]

state = intToVec(((w[0] << 8) + w[1]) ^ ptext) state = mixCol(shiftRow(sub4NibList(sBox, state))) state = addKey(intToVec((w[2] << 8) + w[3]), state) state = shiftRow(sub4NibList(sBox, state))

return vecToInt(addKey(intToVec((w[4] << 8) + w[5]), state))

def decrypt(ctext):

"""Decrypt ciphertext block""" def iMixCol(s):

return [mult(9, s[0]) ^ mult(2, s[2]), mult(9, s[1]) ^ mult(2, s[3]),

mult(9, s[2]) ^ mult(2, s[0]), mult(9, s[3]) ^

mult(2, s[1])]

state = intToVec(((w[4] << 8) + w[5]) ^ ctext) state = sub4NibList(sBoxI, shiftRow(state))

state = iMixCol(addKey(intToVec((w[2] << 8) + w[3]), state))

state = sub4NibList(sBoxI, shiftRow(state))

return vecToInt(addKey(intToVec((w[0] << 8) + w[1]), state))

if name == ' main ':

# Test vectors from "Simplified AES" (Steven Gordon) # [(http://hw.siit.net/files/001283.pdf)](http://hw.siit.net/files/001283.pdf)) print("Name: Abhishek Turukmane")

plaintext = 0b1101011100101000 key = 0b0100101011110101

ciphertext = 0b0010010011101100 keyExp(key)

try:

assert encrypt(plaintext) == ciphertext except AssertionError:

print("Encryption error") print(encrypt(plaintext), ciphertext) sys.exit(1)

try:

assert decrypt(ciphertext) == plaintext except AssertionError:

print("Decryption error") print(decrypt(ciphertext), plaintext) sys.exit(1)

print("Test ok!")

sys.exit()

Output –

With correct values of plaintext & ciphertext –

(base) C:\Users\Saurabh\Downloads\A2>python saes.py Name: Abhishek Turukmane

Test ok!

With incorrect values of plaintext & ciphertext –

(base) C:\Users\Saurabh\Downloads\A2>python saes.py Name: Abhishek Turukmane

Encryption error

34029 9452

# Assignment No – 7

Code of the program – Server Side Code:

"""

@author: Abhishek Turukmane """

import socket

print("Name: Abhishek Turukmane\n") def cal(g,a,p):

res = pow(g,a,p) return(res)

serv = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) serv.bind(('127.0.0.1', 8080))

serv.listen(5) while True:

conn, (ip, port) = serv.accept() conn.send("I am server")

str1 = conn.recv(100) str2 = conn.recv(100)

print("First Prime Number : "+str1+","+"Second Prime Number : "+str2)

a = input("Enter a number specific for User 1 :") print("Calculating the value of A ")

A = cal(int(str2),a,int(str1)) print("value of A :" + str(A))

conn.send(str(A))

B = conn.recv(100) print("Recieved value of B :" + B)

print("Calculating the value of Key K2 ") K2 = cal(int(B),a,int(str1)) print("value of Key K2 is :" + str(K2)) conn.close()

print('client is disconnected')

Client Side Code:

"""

@author: Abhishek Turukmane """

import socket

print("Name: Abhishek Turukmane\n") def cal(g,a,p):

res = pow(g,a,p) return(res)

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) client.connect(('127.0.0.1', 8080)) print(client.recv(100))

p = input("Enter first prime Number(P):") g = input("Enter second prime number (G):") client.send(str(p))

client.send(str(g))

b = input("Enter a number specific for User 2 :") print("Calculating the value of B ")

B = cal(g,b,p)

print("value of B :" + str(B)) client.send(str(B))

A = client.recv(100)

print("Recieved value of A :" + A) print("Calculating the value of Key K1 ") K1 = cal(int(A),b,p)

print("value of Key K1 is :" + str(K1))

client.close()

Output – Server Output:

(base) C:\Users\Saurabh\Downloads\A2>python dhcli.py Name: Abhishek Turukmane

I am server

Enter first prime Number(P):13 Enter second prime number (G):7

Enter a number specific for User 2 :3 Calculating the value of B

value of B :5

Recieved value of A :11 Calculating the value of Key K1 value of Key K1 is :5

Client Output:

(base) C:\Users\Saurabh\Downloads\A3>python dhsocket.py Name: Abhishek Turukmane

First Prime Number : 13,Second Prime Number : 7 Enter a number specific for User 1 :5 Calculating the value of A

value of A :11 Received value of B :5

Calculating the value of Key K2 value of Key K2 is :5

client is disconnected

# Assignment No – 8

Code of the program –

"""

@author: Abhishek Turukmane """

print("Name: Abhishek Turukmane") try:

input = raw\_input except NameError:

pass try:

chr = unichr except NameError:

pass

p=int(input('Enter prime p: ')) q=int(input('Enter prime q: '))

print("Choosen primes:\np=" + str(p) + ", q=" + str(q) + "\n") n=p\*q

print("n = p \* q = " + str(n) + "\n") phi=(p-1)\*(q-1)

print("Euler's function (totient) [phi(n)]: " + str(phi) + "\n")

def gcd(a, b):

while b != 0:

c = a % b a = b

b = c return a

def modinv(a, m):

for x in range(1, m):

if (a \* x) % m == 1: return x

return None def coprimes(a):

l = []

for x in range(2, a):

if gcd(a, x) == 1 and modinv(x,phi) != None and x < (p-1) and x < (q-1):

l.append(x) for x in l:

if x == modinv(x,phi): l.remove(x)

return l

def encrypt(p,k, plaintext):

#Unpack the key into it's components key, n = p,k

#Convert each letter in the plaintext to numbers based on the character using a^b mod m

cipher = [(ord(char) \*\* key) % n for char in plaintext] #Return the array of bytes

return cipher

def decrypt(p,k, ciphertext):

#Unpack the key into its components key, n = p,k

#Generate the plaintext based on the ciphertext and key using a^b mod m

plain = [chr((char \*\* key) % n) for char in ciphertext] #Return the array of bytes as a string

return ''.join(plain)

print("Choose an e from a below coprimes array:\n") print(str(coprimes(phi)) + "\n")

e=int(input()) d=modinv(e,phi)

print("\nYour public key is a pair of numbers (e=" + str(e) + ", n=" + str(n) + ").\n")

print("Your private key is a pair of numbers (d=" + str(d) + ", n=" + str(n) + ").\n")

plaintext=input("Enter plaintext : ")

print ("\n\nEncrypting message with public key (", d,",",n ,")

. . .")

print ("\nYour ciphertext is:") emsg = encrypt(d,n,plaintext)

print ("\t\t",''.join(map(lambda x: str(x), emsg)))

print ("\n\nDecrypting message with public key (", e,",",n ,")

. . .")

print ("\nYour message is:") print ("\t\t",decrypt(e,n, emsg))

Output –

(base) C:\Users\Saurabh\Downloads\A4>python rsa.py Name: Abhishek Turukmane

Enter prime p: 17 Enter prime q: 11 Chosen primes: p=17, q=11

n = p \* q = 187

Euler's function (totient) [phi(n)]: 160

Choose an e from a below coprimes array:

[3, 7, 9]

9

Your public key is a pair of numbers (e=9, n=187).

Your private key is a pair of numbers (d=89, n=187).

Enter plaintext : hello, how are you?

Encrypting message with public key ( 89 , 187 ) . . .

Your ciphertext is: 53501811811111433253111170325124503212111115173

Decrypting message with public key (9, 187) . . .

Your message is:

hello, how are you?