**NITTE MAHALINGA ADYANTHAYA MEMORIAL INSTITUTE OF TECHNOLOGY**

**An Autonomous College Affiliated to VTU Belgaum**

**Nitte -574110, Udupi District**



**PROJECT REPORT**

**ON**

**HOMOPHONIC SUBSTITUTION CIPHER**

**Submitted by**

**PRAJNA V NIVEDITHA G POOJARY**

**4NM17CS128 4NM17CS117**

**VI SEM,’A’ SEC VI SEM,’A’ SEC**

**Department of CSE Department of CSE**

**NMAMIT,Nitte NMAMIT, Nitte**

**SUBMITTED TO**

**Mr. Radhakrishna Dodmane**

**Associate Professor**

**Department of Computer Science and Engineering**

**NMAM Institute of Technology**

**NMAM INSTITUTE OF TECHNOLOGY**

**(An autonomous Institute affiliated to VTU, Belgaum)**

**Nitte-574110, Karkala, Udupi District**

**Department of Computer Science and Engineering**

**CERTIFICATE**

Certified that the project work carried out by Prajna V (4NM17CS128) and Niveditha G Poojary (4NM17CS117), bonafide students of NMAM Institute of Technology, Nitte in fulfilment for the subject cryptography and Network Security in Computer Science and Engineering during the academic year 2019 – 2020.

Signature of lecturer

Date:

# 

**ACKNOWLEDGEMENT**

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible because

“Success is the abstract of hard work and perseverance, but steadfast of all is encouraging guidance.”

So, We acknowledge all those whose guidance and encouragement served as a beacon light and crowned our efforts with success.

I would like to thank our principal **Prof. Niranjan N Chiplunkar** firstly, for providing us with this unique opportunity to do the project in the 6th semester of Computer Science and engineering.

I would like to thank our college administration for providing a conductive environment and also suitable facilities for this project. I would like to thank our HOD **Prof.Uday KumarReddy** for showing us the path and providing the inspiration required for taking the project to its completion. It is my great pleasure to thank our guide **Mr. Radhakrishna Dodmane** for his continuous encouragement, guidance and support throughout this project.

We thank all the staff members of the department of CSE for providing resources for the completion of the project.

Prajna V (4NM17CS128)

Niveditha G Poojary (4nm17cs117)

**ABSTRACT**

The homophonic substitution cipher is a substitution cipher in which single plain text letters can be replaced any of several different cipher text letters. They are generally much more difficult to break then standard substitution cipher.

We have made some modifications to the code in order to achieve better security in the same standard time.

In this project we show the difference between the outputs since in the original homophonic substitution cipher system can encrypt and decrypt only alphabets but in the modified substitution cipher system we can encrypt and decrypt any kind of file type like alphanumeric and also images.

**INDEX**

1. Introduction
2. Literature Survey and implementation
3. Modification on homophonic substitution cipher
4. Problem definition
5. Methodology
6. The complete code
7. Outputs
8. Advantages, Disadvantages and Applications
9. Results, Discussion
10. References

Assuming that each of the 26 plaintext letters

maps to at least one ciphertext symbol, the theoretical key space is of size

n

26:ð1Þ

Using the bound

**INTRODUCTION**

Cryptographic techniques increase the security of the network communication techniques to a great extent. The target of the hardware implementation of the algorithm mainly focuses on three major factors. The main focus is to optimize area, to minimize the time and cost.

Initially this approaches takes 8 character and gets all unique possible states .This disperses all these 27 ways. This allows to store the encrypted values in the mapping file which is approximately 350MB

**Implementation:**

This approach uses following keywords:

1. **Genmaps**: In this file the mapping to different files are generated.

The code for this is

def genmaps (\_):

print ('Generating mapping file...')

maps = [''.join (i) for i in itertools.product (list ('espinoza'), repeat=8)]

random.shuffle (maps)

states = [''.join (i) for i in itertools.product (['1', '0'], repeat=8)]

map\_dict = {}

for s in states:

s\_index = states. index(s)

map\_dict[s] = maps [65536\*s\_index:65536\*(s\_index+1)]

with open ('mapping.p', 'wb') as f:

pickle.dump (map\_dict, f)

1. **Encryption**: In this file the encrypted data will be stored.

The code for this is

def encryption (params):

input\_file, output\_file = params

print ('Encrypting file...')

with open ('mapping.p', 'rb') as f:

map\_dict = pickle. loads (f.read ())

ciphertext = ' '

with open (input\_file, 'rb') as f:

while True:

byte = f.read (1)

if not byte: break

b = bin (int.from\_bytes (byte, 'little')) [2:].zfill (8)

ciphertext += random. choice (map\_dict[b])

with open (output\_file, 'w') as f:

f.write (ciphertext)

1. **Decryption: In** this file decrypted text or the plain text is obtained. The code for this is

def decryption (params):

input\_file, output\_file = params

print ('Decrypting file...')

with open ('mapping.p', 'rb') as f:

map\_dict = pickle. loads (f.read ())

with open (input\_file) as f:

ciphertext = f.read ()

enc\_letters = [ciphertext[i: i+8] for i in range (0, len(ciphertext), 8)]

def search\_mapping(c):

for k, vals in map\_dict. Items ():

for v in vals:

if v == c:

return k

bin\_array = array ('B') #this part of code is to map the files

for c in enc\_letters:

bin\_array.append (int (b, 2))

f = open (output\_file, 'wb')

bin\_array.tofile (f)

f.close ()

1. **Timeit** :This code will generate the time required for encryption and decryption .The code for this is:

def timeit (func, params):

start\_time = time.t6ime ()

out = func(params)

print ('Completed {0} in {1:.2f} seconds.'.format (func.\_\_name\_\_, time.time () - start\_time))

return out

**MODIFICATION ON HOMOPHONIC SUBSTITUTION CIPHER**

• Can encrypt all alphanumeric values.

• The data to be transferred will be more secured

• n bit is replaced by 2n bits

**PROBLEM DEFINITION**

The aim of the project is to enhance the security of homophonic substitution cipher .This method takes 8 random, non-repeating letters and gathers all the permutations with repetitions that is Cartesian product , of those letters it generates 8^8 or 16,777,216 unique strings. These permutations are then shuffled, and dispersed equally 27 ways including spaces. This gives each letter 6,21,378 unique representations .

When a message is encrypted one of these 6,21,378 strings belonging to each letter are selected at random to represent each character. During decryption strings are searched for and remapped to their original letters.

**METHODOLOGY**

The key size is calculated by:

(n26 )26!-26n-26

Using the bound (nk) < nk/k! the above equation becomes

(n26)26!-26n-26 < n2626n-26= (n/26)26 \*26n -- (\*)

If n value 52 cipher text symbols , then the bound in the equation (\*)

Is slightly less than 2271 as another example , for n value 104 we can that

The upper bound in equation (\*) yields 2541 .We can obtain that value by the following method

(mkk) >= k-1/2[(mm(k-1)+1) / ((m-1)(m-1)(k-1)]

**THE COMPLETE CODE FOR MODIFIED HOMOPHONIC SUBSTITUTION CIPHER**

#! /usr/bin/env python

import argparse

import binascii

import itertools

import pickle

import random

import sys

import time

from array import \*

def genmaps (\_):

print ('Generating mapping file...')

maps = [''.join (i) for i in itertools.product (list('espinoza'), repeat=8)]

random.shuffle (maps)

states = [''.join (i) for i in itertools.product (['1', '0'], repeat=8)]

map\_dict = {}

for s in states:

s\_index = states. index(s)

map\_dict[s] = maps[65536\*s\_index:65536\*(s\_index+1)]

with open ('mapping.p', 'wb') as f:

pickle.dump (map\_dict, f)

def encryption (params):

input\_file, output\_file = params

print ('Encrypting file...')

with open ('mapping.p', 'rb') as f:

map\_dict = pickle.loads (f.read ())

ciphertext = ''

with open (input\_file, 'rb') as f:

while True:

byte = f.read (1)

if not byte:

break

b = bin (int.from\_bytes (byte, 'little'))[2:].zfill(8)

ciphertext += random.choice(map\_dict[b])

with open (output\_file, 'w') as f:

f.write (ciphertext)

def decryption (params):

input\_file, output\_file = params

print ('Decrypting file...')

with open ('mapping.p', 'rb') as f:

map\_dict = pickle.loads (f.read ())

with open (input\_file) as f:

ciphertext = f.read ()

enc\_letters = [ciphertext [i: i+8] for i in range (0, len(ciphertext), 8)]

def search\_mapping(c):

for k, vals in map\_dict. items ():

for v in vals:

if v == c:

return k

bin\_ array = array ('B')

for c in enc\_letters:

b = search\_mapping(c)

bin\_array.append (int (b, 2))

f = open (output\_file, 'wb')

bin\_array.tofile (f)

f.close ()

def timeit (func, params):

start\_time = time.time ()

out = func(params)

print ('Completed {0} in {1:.2f} seconds.’format (func.\_\_name\_\_, time.time () - start\_time))

return out

def get\_parser ():

parser = argparse.ArgumentParser (description='rich man\'s homophonic substitution cipher')

parser.add\_argument ('-g', '--genmaps',

help='generate mapping file', action='store\_true')

parser.add\_argument('-e', '--encrypt',

help='encrypt a file', action='store\_true')

parser.add\_argument('-d', '--decrypt',

help='decrypt a file', action='store\_true')

parser.add\_argument('-i', '--input',

help='input file', type=str)

parser.add\_argument ('-o', '--output',

help='output file', type=str)

return parser

def main ():

parser = get\_parser ()

args = vars (parser.parse\_args ())

if not (args ['genmaps'] or args ['encrypt'] or args ['decrypt']):

parser.print\_help ()

return

if args ['genmaps']:

timeit (genmaps, params=None)

return

if not args['input']:

print ('Input file not supplied.')

return

if not args['output']:

print ('Output file not supplied.')

return

if args['encrypt']:

timeit (encryption, params= [args ['input'], args ['output']])

return

if args ['decrypt']:

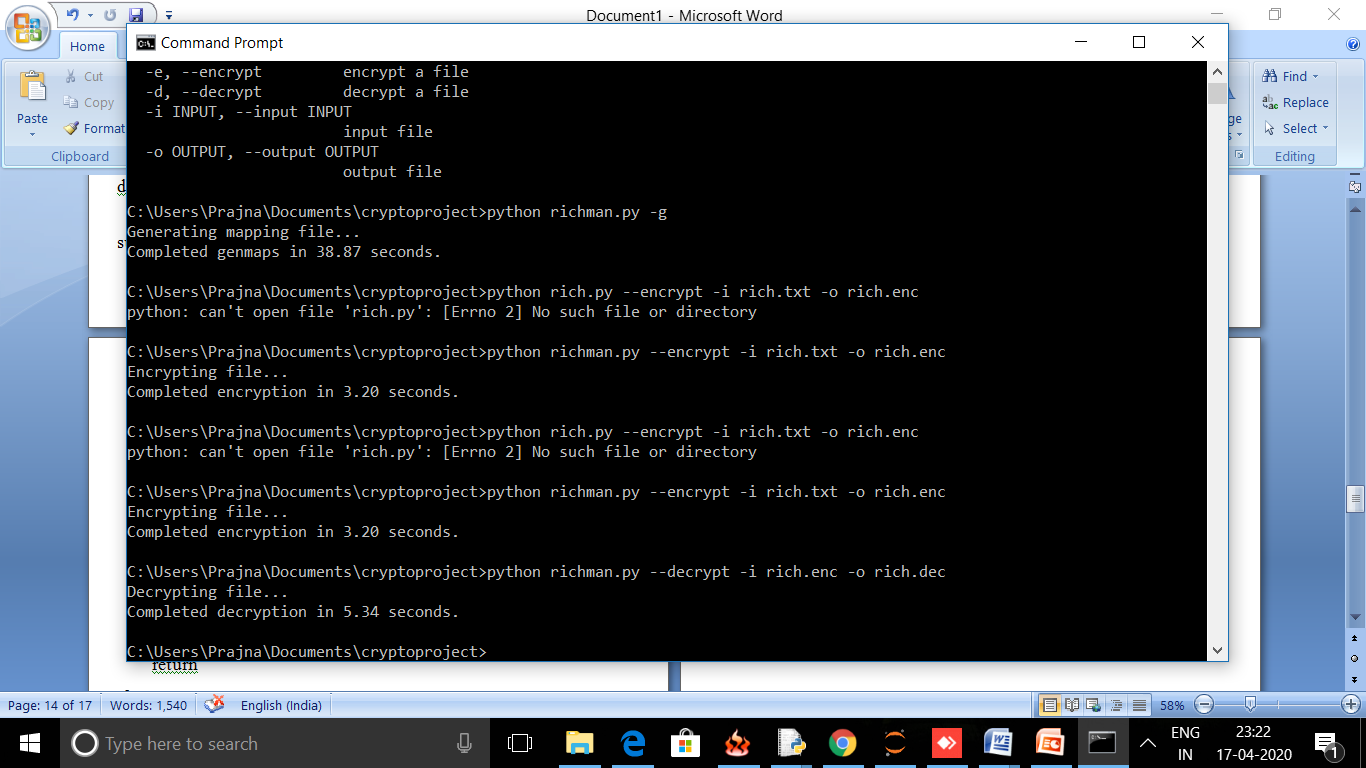
timeit (decryption, params= [args ['input'], args ['output']])

return

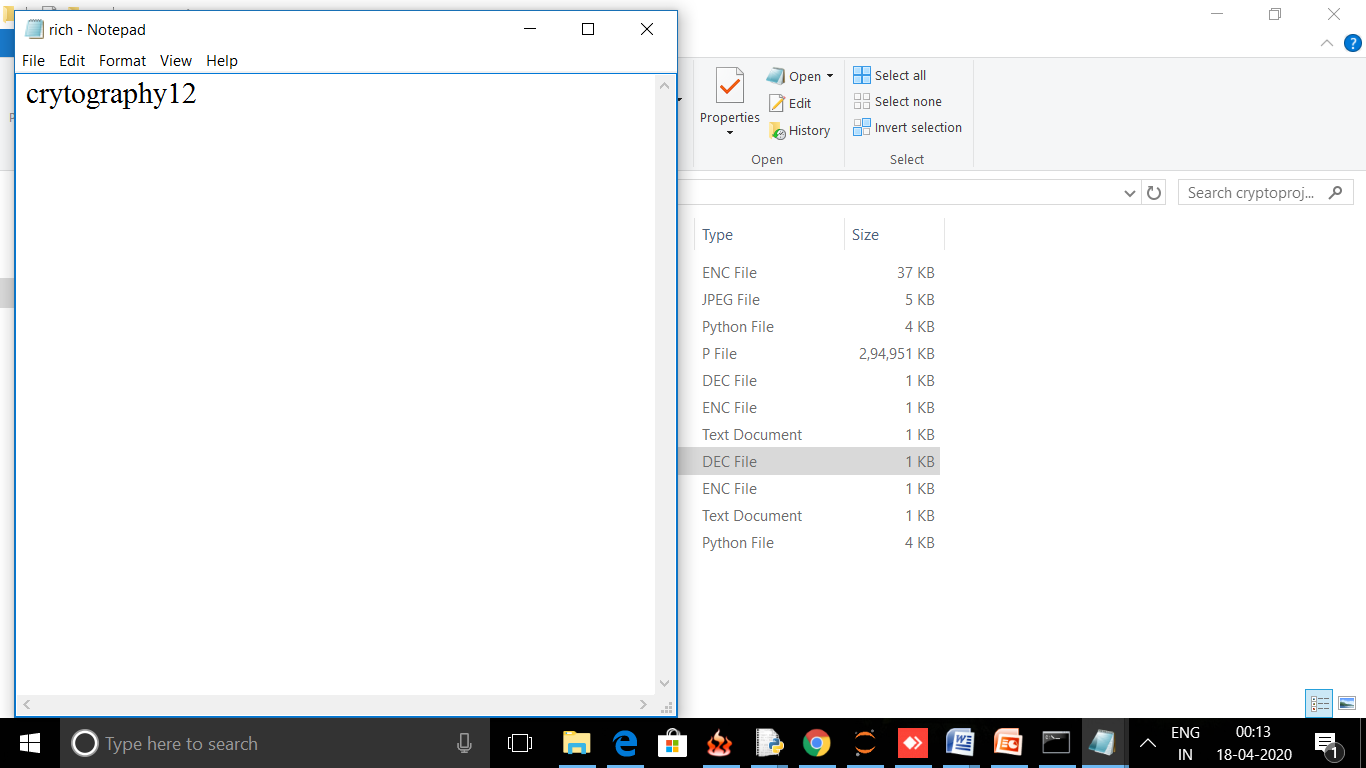
if \_\_name\_\_ == '\_\_main\_\_':

main ()

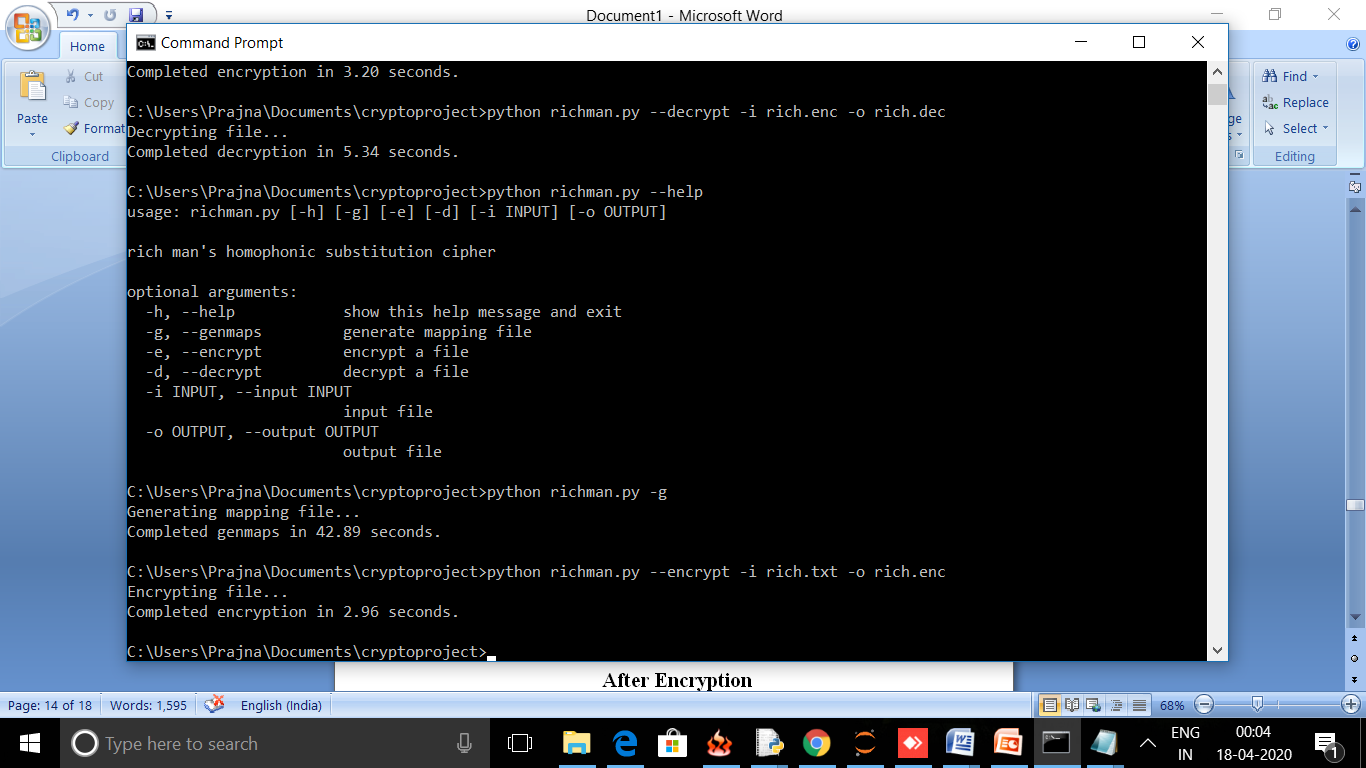
**OUTPUT**

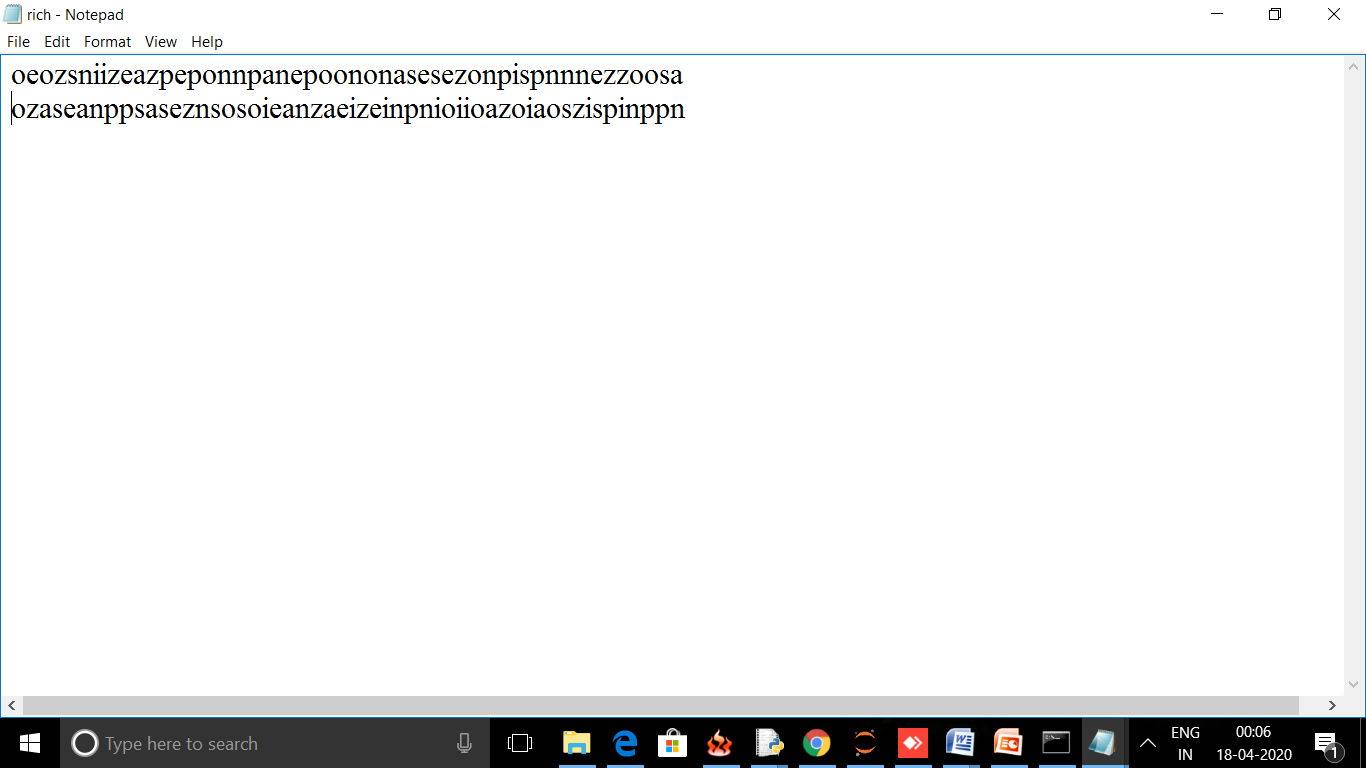
****

**To generate maps**

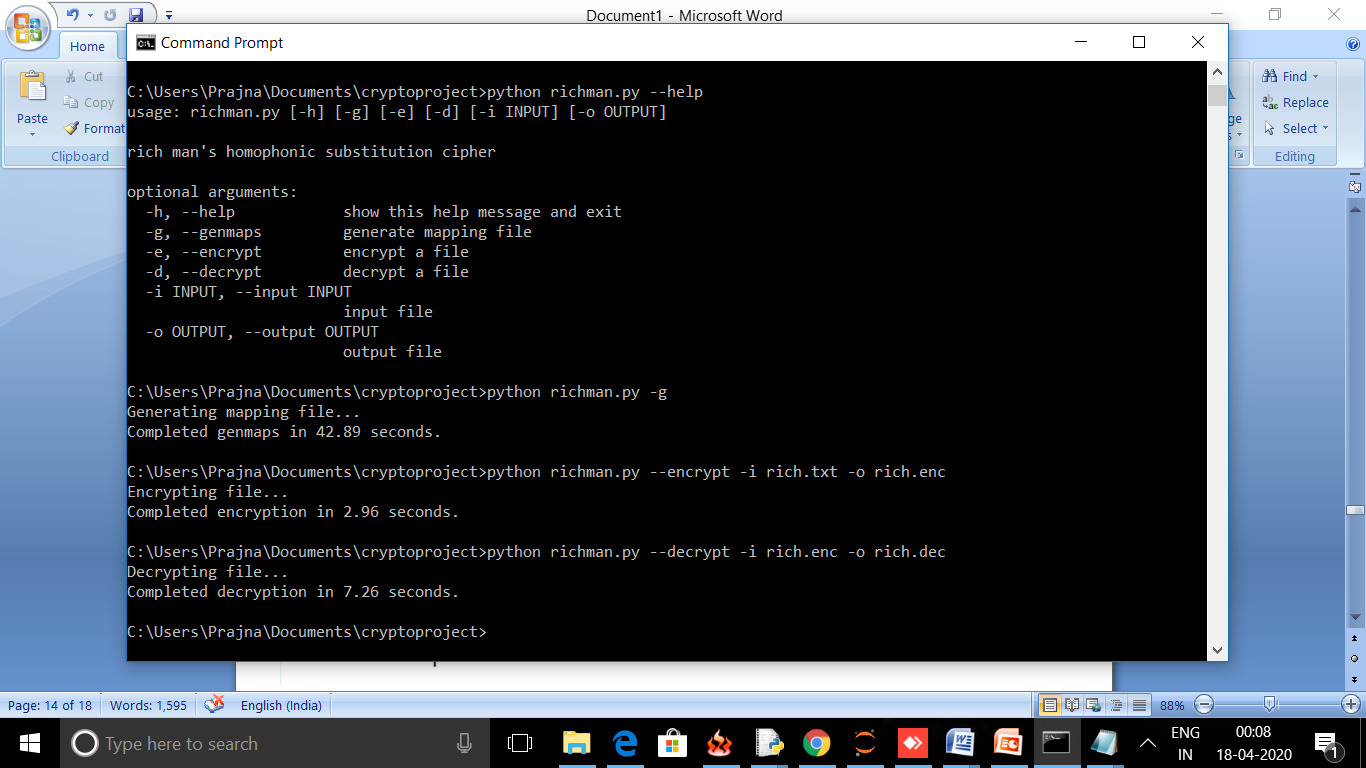
****

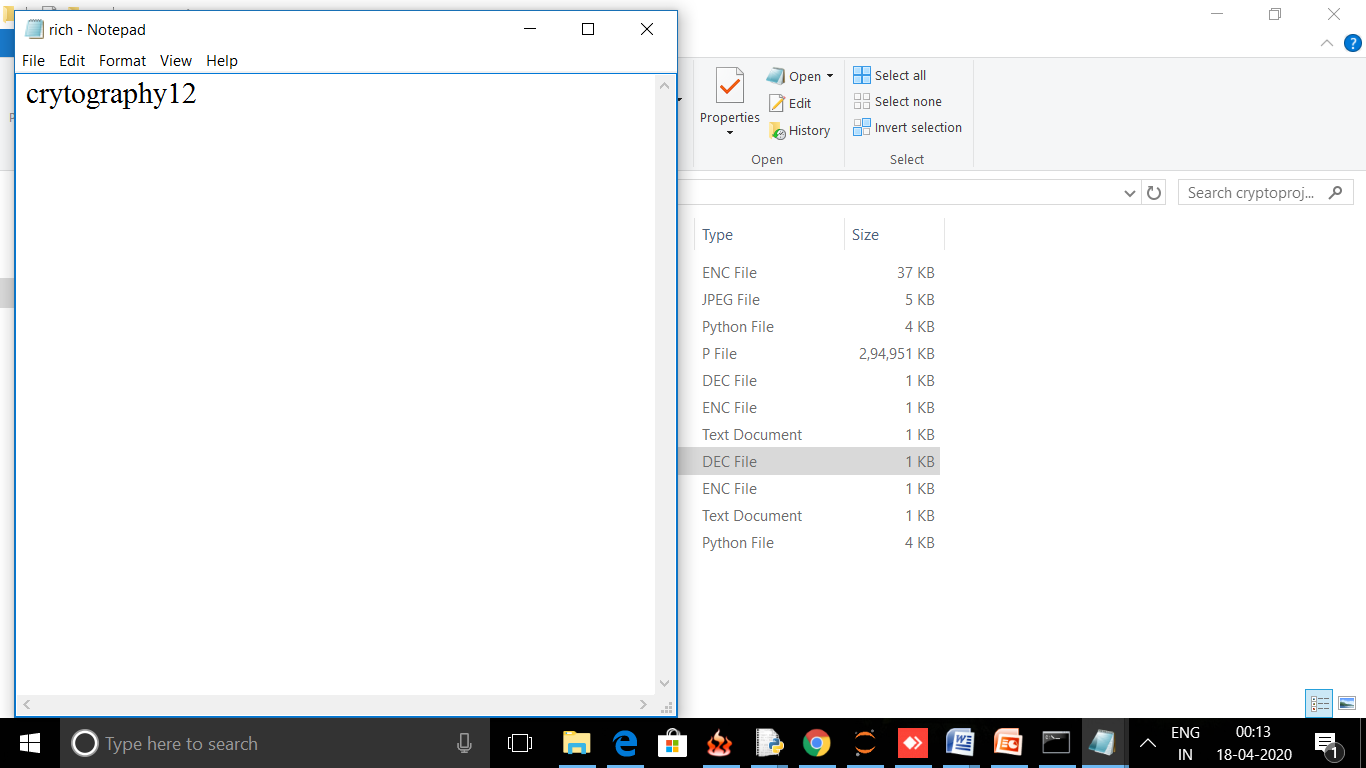
**Plaintext**





**After Encryption**

****

****

**After Decryption**

**ADVANTAGES, DISADVANTAGES AND APPLICATIONS**

**Advantages**

* Provides high level of security.
* The values are non repeating

**DISADVANTAGES**

* The length of cipher text is large.
* Can encrypt and decrypt the plaintext of short length

**APPLICATIONS**

* Used to very safely share relatively short, important, nerdy messages.

plaintext letter, so the decryption is unique. As mentioned above, such a mapping

tends to flatten the frequency statistics in the resulting ciphertext which makes attacks

based on frequency analysis more difficult.

An example of a homophonic substitution cipher is given in Table 1. For the key

in Table 1, any of the symbols R,3,or9can be substituted for plaintext E, and either

Yor 6can be substituted for plaintext L. Using this key, plaintext HELLO can be

encrypted as U96YB. In this case, a cryptanalyst has no indication that ciphertext

6and Yboth represent the same plaintext letter. In contrast, if we encrypt HELLO

using a simple substitution, the ciphertext would reveal that the 3rd and 4th plaintext

letters are the same.

Suppose that a given homophonic substitution cipher has ndistinct ciphertext

symbols, and the plaintext is English. Assuming that each of the 26 plaintext letters

maps to at least one ciphertext symbol, the theoretical key space is of size

n

26

plaintext letter, so the decryption is unique. As mentioned above, such a mapping

tends to flatten the frequency statistics in the resulting ciphertext which makes attacks

based on frequency analysis more difficult.

An example of a homophonic substitution cipher is given in Table 1. For the key

in Table 1, any of the symbols R,3,or9can be substituted for plaintext E, and either

Yor 6can be substituted for plaintext L. Using this key, plaintext HELLO can be

encrypted as U96YB. In this case, a cryptanalyst has no indication that ciphertext

6and Yboth represent the same plaintext letter. In contrast, if we encrypt HELLO

using a simple substitution, the ciphertext would reveal that the 3rd and 4th plaintext

letters are the same.

Suppose that a given homophonic substitution cipher has ndistinct ciphertext

symbols, and the plaintext is English. Assuming that each of the 26 plaintext letters

maps to at least one ciphertext symbol, the theoretical key space is of size

n

2

**RESULTS**

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Plaintext | Ciphertext  (normal) | Ciphertext  (modified) |
| Data1 | 3.37 | 3.5 | 3.1 |
| Data2 | 1 | 1.5 | 2.98 |
| Data3 | 2.75 | 2.75 | 2.88 |
| Data4 | 2.52 | 2.58 | 2.83 |
| Data5 | 2.24 | 2.23 | 2.78 |
| Data6 | 3 | 2.94 | 2.92 |

**Entrophy values:**

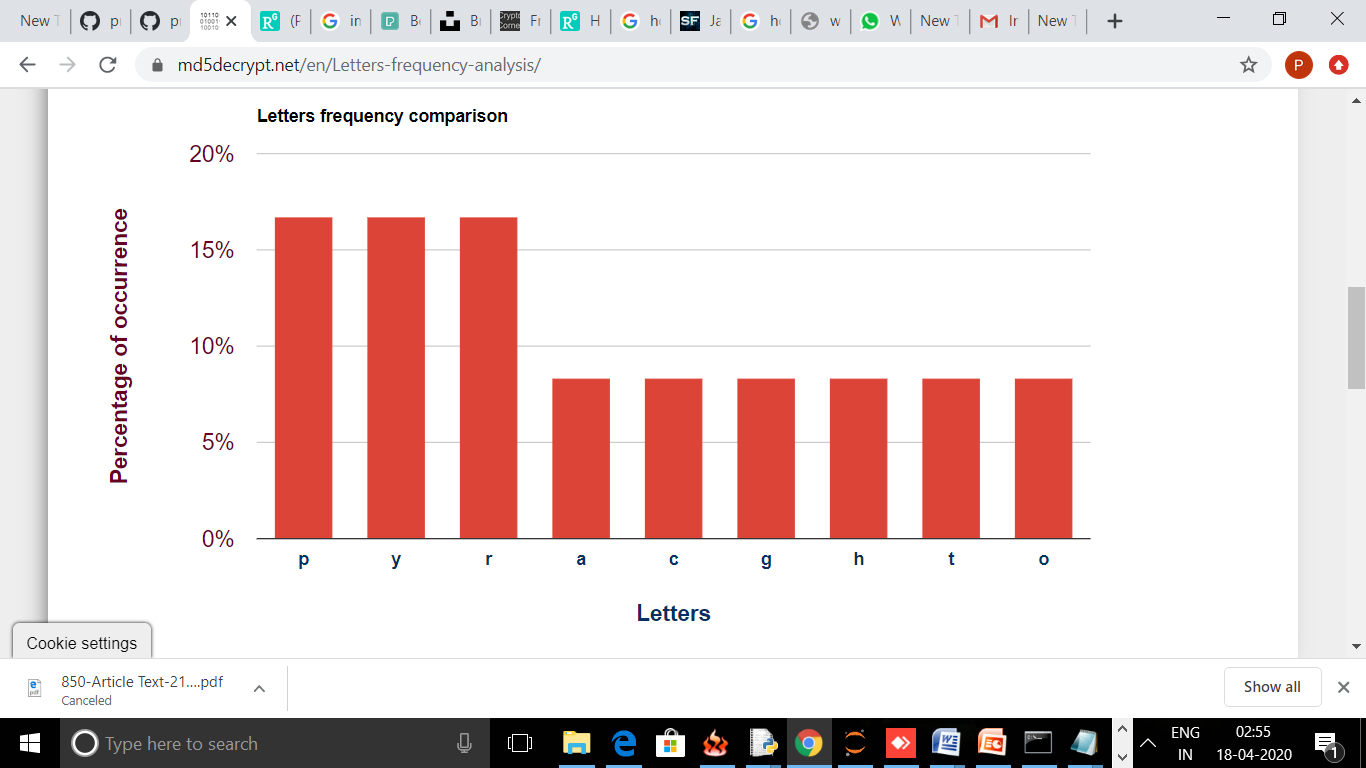
Entrophy s the measure of unpredictability of information contained in the message.Higher the entrophy in message makes it difficult for the cryptanalyst to get the plaintext back.

The graph depicts the entrophy value

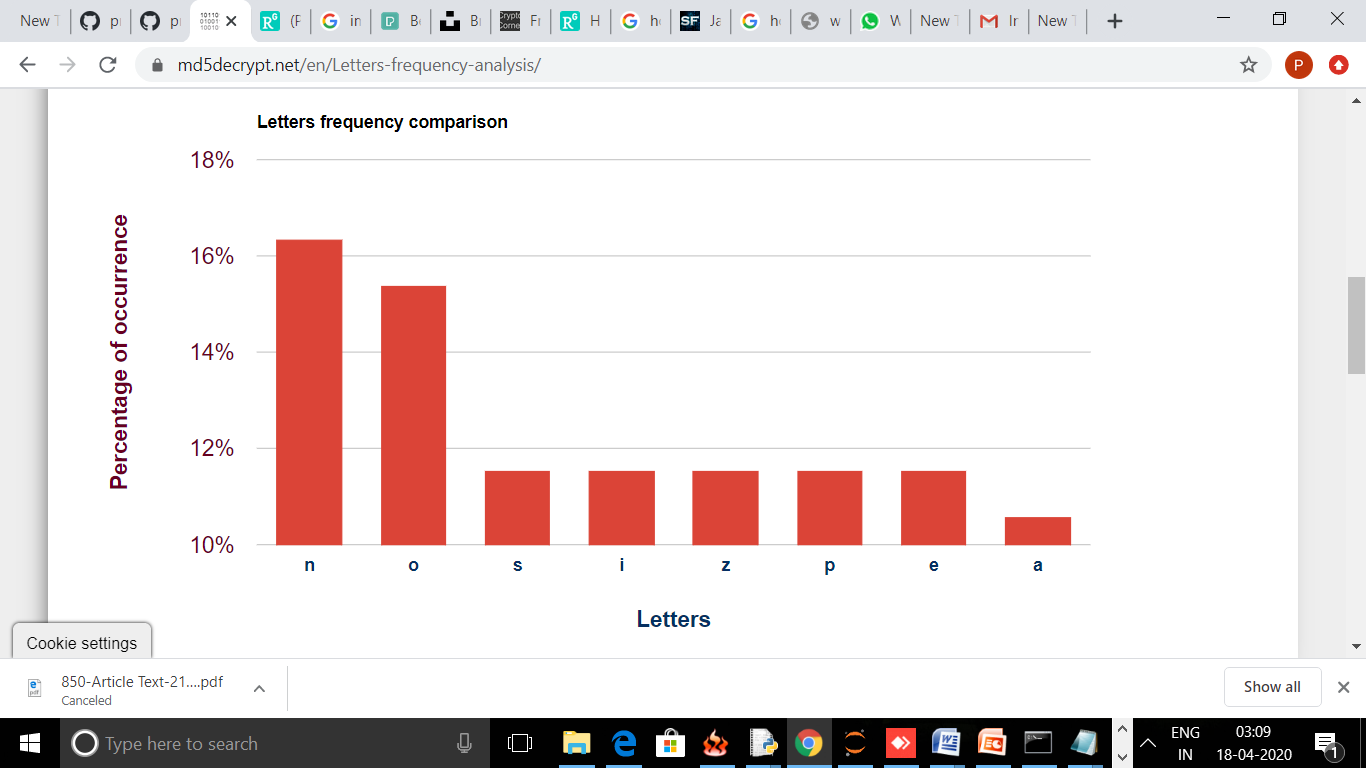
**GRAPH FOR ENTROPHY:**

The given graph depicts the percentage of occurrence of each character. The given algorithm is expected to ensure that the percentage of occurrence of characters in the plain text and cipher text should be different.

**HISTOGRAM FOR PLAIN TEXT**



**HISTOGRAM FOR CIPHERTEXT**



**REFERENCES:**

Austin Jackson.Richman’s homophonic substitution cipher .DOI: <https://github.com/vesche/pmhsc>