**Cipher Encryption:   
Enhancement of Protection of data by creating hybrid ciphers**

by

R.B.ADVAITHBARATH 18BCE1061

JITENDRA SAI 18BCE1329

DHEERAJ KANNAN 18BCE1134

A project report submitted to

**Prof. Gayathri Raj**

**SCHOOL OF COMPUTER SCIENCE ENGINEERING**

in partial fulfilment of the requirements for the course of

**CSE4003-** **Introduction to cyber security**

in

**B.Tech. COMPUTER SCIENCE AND ENGINEERING**

****

**VIT CHENNAI**

**Vandalur – Kelambakkam Road**

**Chennai – 600127**

**JUNE 2020**

**BONAFIDE CERTIFICATE**

Certified that this project report entitled “**Cipher Encryption: Enhancement of Protection of data by creating hybrid ciphers”** is a Bonafede work of **JITENDRA SAI** who carried out the Project work under my supervision and guidance for **CSE4003- Introduction to cyber security**

**Prof. Gayathri Raj**

Assistant Professor

School of Computer Science Engineering (SCSE),

VIT Chennai

Chennai – 600 127.

**BONAFIDE CERTIFICATE**

Certified that this project report entitled “**Cipher Encryption: Enhancement of Protection of data by creating hybrid ciphers”** is a Bonafide work of **R.B.Advaithbarath** who carried out the Project work under my supervision and guidance for **CSE4003- Introduction to cyber security**

**Prof. Gayathri Raj**

Assistant Professor

School of Computer Science Engineering (SCSE),

VIT Chennai

Chennai – 600 127.

**BONAFIDE CERTIFICATE**

Certified that this project report entitled “**Cipher Encryption: Enhancement of Protection of data by creating hybrid ciphers”** is a Bonafide work of **DHEERAJ KANNAN** who carried out the Project work under my supervision and guidance for **CSE4003- Introduction to cyber security**

**Prof. Gayathri Raj**

Assistant Professor

School of Computer Science Engineering (SCSE),

VIT Chennai

Chennai – 600 127.

**ACKNOWLEDGEMENT**

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Prof. Gayathri Raj,** Assistant Professor, School of Computer Science Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Jagadeesh Kannan**, Dean of the School of Computer Science Engineering, VIT Chennai, for extending the facilities of the School towards our project and for his unstinting support.

We express our thanks to our Head of the Department **Dr. Justus S** for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SERIAL NO.** | **NAME** | **PAGE NO.** |
|  | ACKNOWLEDGEMENT | 5 |
|  |  |  |
| 1 | OBJECTIVE | 7 |
|  |  |  |
| 2. | ABSTRACT | 7 |
|  |  |  |
| 3. | INTRODUCTION | 8 |
|  |  |  |
| 4. | LITERATURE SURVEY | 9 |
|  |  |  |
| 5 | ISSUE WITH OUR CURRENT MODEL | 12 |
|  |  |  |
| 6 | PROPOSED ARCHITECTURE | 12 |
|  |  |  |
| 7 | MODULE AND ITS DESCRIPTION | 13 |
|  |  |  |
| 8 | CODE AND OUTPUT | 19 |
|  |  |  |
| 9 | RESULT AND INFERENCE | 68 |
|  |  |  |
| 10 | CONCLUSION | 68 |
|  |  |  |
| 11 | REFERENCES | 69 |

**Objective :**

To explore the alternatives of the Caesar cipher which is the weakest encryption algorithm and compare with the same as well as other alternatives

To create a new hybrid cipher algorithm by utilizing or combining the existing cipher algorithm to enhance data for highly classified code or message like nuclear code, launch code, bank locker key,etc.

**Abstract :**

A great many apps and websites rely on user passwords and password verification software to facilitate access to valuable data. Besides learning how to create a secure password, consumers can’t do a lot to encrypt their passwords besides using a password manager, and a proficient password manager must use high-quality encryption to protect what is essentially a treasure trove of data. Nuclear code must also be kept secretly to avoid getting hacked. That why digital security is becoming increasingly important to protect us as we bank, as we shop, and as we communicate. And at the core of that security lies encryption.

One types of encryption is Caesar cipher. However it not the strongest encryption algorithm as we know. Its becoming weak and weak due to easy brute force algorithm. That why new better cipher algorithm is needed. Its can be developed by mixing one cipher algorithm to another. Or its can also be done by strengthening the key to unlock the code.

**Introduction:**

Messages without encryption are prone to hacking. That why simple types of encryption like Caesar cipher are used to keep it safe. But these cipher encryption are highly prone to brute force hacking due to its simple algorithm. Its brute force code are easily available and the encrypted message will be obtained within a second.

So alternatives symetric cipher like multiplicative, affine, autokey, Playfair, Vigenère and many others cipher encryption are used. But even if they do offer higher level of protection, they are prone to advanced level of hackers whom they use their own set of brute force algorithm which is only available to them to crack the message.

That why many confidential code or message does not uses these types of algorithm. So we came up with the solution of selecting these types of cipher and create a hybrid cipher to enhance more protection to encrypted data and making it difficult to hack. We are trying to implement in many different varieties to find out the best way to encrypt it using the hybrid cipher algorithm. We will also rank them which are the generated hybrid cipher is best and effective to implement. The implementation depends on not only encryption but also decryption in the best effective manner

**Literature survey :**

**Abdalbasit Mohammed Qadir and Nurhayat Varol et al [1**] demonstrated a review of some of the exploration that has been led in the field of cryptography just as of how the different algorithms utilized in cryptography for various security purposes work. Cryptography will keep on arising with IT and strategies as to securing individual, monetary, clinical, and online business information and giving a good degree of protection

**Omar G. Abood, Shawkat K. Guirguis et al [2]** discussed a few significant algorithms utilized for the encryption and decoding of information in all fields, to make a relative report for most significant algorithms as far as information security viability, key size, intricacy and time, and so on This examination zeroed in on various sorts of cryptography algorithms that are existing, as AES, DES, TDES, DSA, RSA, ECC, EEE and CR4… and so on.

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation.

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation.

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation.

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation.

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation.

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation

Singh et al. [25] presented a work that ran a comparison between the most popular encoding algorithms. According to the

work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The

results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of

the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the

old XOR with a new operation

**Singh et al [3]** presented a work that ran a comparison between the most popular encoding algorithms. According to the work, the most popular algorithms were AES, DES, 3DES and Blowfish in the terms of security and energy consumption. The results of the comparison contrasted with the some of the previous studies and showed that AES is better than the basic form of the Blowfish algorithms. However, to make BA stronger against every type of attack, extra keys could be added to substitute the old XOR with a new operation

**Seth et al [4]** compared three algorithms: DES, AES and RSA. They inferred that RSA requires the longest encoding time and higher memory than the other two algorithms; however, with minimal output byte in RSA algorithm. Meanwhile, they also found that DES utilizes minimum enciphering time while AES requires the smallest storage memory. Furthermore, encoding time in both AES algorithm and DES algorithm is almost the same.

**Marwah et al [5]** also compared three algorithms, namely: DES, 3DES and RSA. The result is that the privacy ensured by 3DES is better than that of DES and RSA. DES is economical in energy memory required as well as fast in encoding and decoding database time. DES is vulnerable though, in comparison to 3DES and RSA.

**Saini et al [6]** sums up that the superior algorithms are prominent for their popularity. An efficient cryptography achieves two parts of an equation, possibility and acceptability.

**Alam et al [7]** has proved that 3DES requires more energy and processes less input than those of DES, this is because of its triple time feature. However, RC2 proved to be quicker due to smaller sizes of the throughput if contrasted to Blowfish. Blowfish input value is bigger than 3DES, DES, CAST-128, IDEA and RC2. Blowfish consumes the least power. Eventually, it turns out that Blowfish is the best, in terms of time, throughput and power.

**Singh et al [8]** is a prime example of that as it compared between the different symmetric algorithms including the DES, 3DES, AES and the Blowfish algorithms. The work found that Blowfish was the best amongst the other methods despite their popularity in the field of encoding and decoding. Accordingly, it was found that the AES algorithm was not proficient enough in comparison to other algorithm, for it needs higher processing time.

**Apoorva et al [9]** concluded that blowfish in the best algorithm to be used in terms of security and time to process because it consumes little time in comparison to the rest.

**Cornwell et al [10]** found that the Blowfish algorithm had the ability to support security for a relatively long time without any suspicious violations of the code. According to the researcher, the Blowfish algorithm is superior in terms of security and efficiency. However, further research should be carried on in order to re-estimate the results discussed by the Cornwell research on Blowfish to provide more evidence on the results.

**Issues with the current model**

The Caesar cipher text can be hacked with various possibilities. One of such possibility is Brute Force Technique, which involves trying every possible decryption key. This technique does not demand much effort and is relatively simple for a hacker. Symmetric can be easily hacked using modern hacking brute force method. This makes the usage of this algorithm more vulnerable of stealing sensitive data.

**Proposed Architecture:**

To combat the existing problem of brute force, cipher hybridization is done. This is done in many ways. First a cipher algorithm is taken, its is then mixed up with other different cipher algorithm to improve the strength of the algorithm. The other way is to improve and introduce the key unlocking system. New algorithm for acceptance of key can avoid direct brute force attack from hacker. Its can be either done by setting either the key requirement or change the acceptance algorithm.

**Symmetric Cipher used:**

Caesar cipher

Multiplicative Cipher

Affine Cipher

Vigenère Cipher

Autokey Cipher

Playfair Cipher

Hill Cipher

**Hybrid cipher created from existing ciphers**

Autokey🡪Vigneere

Affine 🡪Vigneere

Hill🡪Autokey

Hill🡪Affine

Affine🡪Autokey🡪Vigneere

**Modules and its description**

In this project we use symmetric cipher and its types as our basics. There are two types of symmetric ciphers

Mono-alphabetic Cipher – In mono-alphabetic ciphers, each symbol in plain-text is mapped to one cipher-text symbol. No matter how many times a symbol occurs in the plain-text, it will correspond to the same cipher-text symbol.

In poly-alphabetic ciphers, every symbol in plain-text is mapped to a different cipher-text symbol regardless of its occurrence. Every different occurrence of a symbol has different mapping to a cipher-text.

The modules are divided into the following:

**Caesar cipher:**

One of the earliest known example of substitution cipher. Said to have been used by Julius Caesar to communicate with his army Each character of a plaintext message is replaced by a character n position down in the alphabet. The cipher text can be hacked with various possibilities. One of such possibility is Brute Force Technique, which involves trying every possible decryption key. This technique does not demand much effort and is relatively simple for a hacker.

**Multiplicative cipher:**

While using Caesar cipher technique, encrypting and decrypting symbols involves converting the values into numbers with a simple basic procedure of addition or subtraction. If multiplication is used to convert to cipher text, it is called a wrap-around situation. The numbers will be used for multiplication procedure and the associated key is 7. The basic formula to be used in such a scenario to generate a multiplicative cipher is as follows –



**Affine cipher**

The Affine cipher is a type of monoalphabetic substitution cipher, wherein each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter. The formula used means that each letter encrypts to one other letter, and back again, meaning the cipher is essentially a standard substitution cipher with a rule governing which letter goes to which the whole process relies on working modulo. In the affine cipher, the letters of an alphabet of size m are first mapped to the integers in the range 0 … m-1.The ‘key’ for the Affine cipher consists of 2 numbers, we’ll call them a and b. The following discussion assumes the use of a 26 character alphabet (m = 26). a should be chosen to be relatively prime to m

**Vigenère cipher**

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets .The encryption of the original text is done using the Vigenère square or Vigenère table. The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers. At different points in the encryption process, the cipher uses a different alphabet from one of the rows.The alphabet used at each point depends on a repeating keyword.

**Auto key cipher**

An autokey cipher is a cipher that incorporates the message into the key. The key is generated from the message in some automated fashion, sometimes by selecting certain letters from the text or, more commonly, by adding a short primer key to the front of the message.

**Playfair cipher**

The Playfair cipher or Playfair square or Wheatstone–Playfair cipher is a manual symmetric encryption technique and was the first literal diagram substitution cipher. The scheme was invented in 1854 by Charles Wheatstone, but bears the name of Lord Playfair for promoting its use. The technique encrypts pairs of letters, instead of single letters as in the simple substitution cipher and rather more complex Vigenère cipher systems then in use. The Playfair is thus significantly harder to break since the frequency analysis used for simple substitution ciphers does not work with it. The frequency analysis of bigrams is possible, but considerably more difficult.

**Hill cipher**

The Hill Cipher uses an area of mathematics called Linear Algebra, and in particular requires the user to have an elementary understanding of matrices. It also make use of Modulo Arithmetic (like the Affine Cipher). Because of this, the cipher has a significantly more mathematical nature than some of the others. However, it is this nature that allows it to act easily on larger blocks of letters.

**HYBRID CIPHERS:**

**Combination 1:**

This hybridutilizes the Autokey first followed by Vigenère during encryption. During decryption reverse order is followed. Both the algorithm uses the same key.

**Combination 2:**

This hybridutilizes the Affine first followed by Vigenère during encryption. During decryption reverse order is followed. Both the algorithm uses different key. First affine numerical key is typed then followed by Vigenère key

**Combination 3:**

This hybridutilizes the Autokey first followed by Hill during encryption. During decryption reverse order is followed. Both the algorithm uses different key. Key is entered for autokey first followed by string with the length of 4 for hill

**Combination 4:**

This hybridutilizes the Affine first followed by Hill during encryption. During decryption reverse order is followed. Both the algorithm uses different key. Key is entered for affine first followed by string with the length of 4 for hill

**Combination 5:**

This hybridutilizes the Affine first followed by Autokey and then by Vigenère during encryption. During decryption reverse order is followed. The algorithm uses different key but Autokey and Vigenère follows the same key. Key is entered for affine first followed by Autokey and Vigenère

**KEY COMBINATION:**

**Key Combination 1:**

Affine->Vigenère Hybrid accept the common key with four numbers in which number and string are separated. Affine uses the number as key and Vigenère uses string as key. This key combination separates number from string irrespective of the location of the number with respect to string.

**Key Combination 2:**

Its is similar to the above key combination but its separates number from string only if the number is found at the particular location of the string.

**Key Combination 3:**

Its is similar to the above key combination but its reverses the number obtain and process the result.

**Key Combination 4:**

Affine cipher accept the common key with four numbers . The digits are then swapped and the new key is created. This key is uses for encryption/decryption.

**Key Combination 5:**

Autokey->Vigenère Hybrid accept the common key which is string. This key will first used in Autokey then for the Vigenère, a substring is taken from the main string from key, and used as a second key.

**Code and Output**

**Mono Cipher(Caesar and multiplicative):**

import re

def display():

print("Modern Cipher Solver\n")

print("\nEnter 'C' for Caesar Cipher \nEnter 'M' for Multiplicative Cipher\n ")

while True:

cc = (input("Please enter your choice: "))

if cc not in ('C', 'M'):

print("Not an appropriate choice!")

else:

break

if cc == 'M':

print("\nMultiplicative Cipher\n")

print("MENU:\nEnter 'E' for ENCRYPTION \nEnter 'D' for DECRYPTION \nEnter 'B' for BRUTE\_FORCE \nEnter 'X' for "

"Exit \n ")

choice()

else:

print("\nCaesar Cipher\n")

print("MENU:\nEnter 'E' for ENCRYPTION \nEnter 'D' for DECRYPTION \nEnter 'B' for BRUTE\_FORCE \nEnter 'X' for "

"Exit \n ")

choice2()

def choice():

while True:

ch = (input("Please enter your choice: "))

if ch not in ('E', 'D', 'B', 'X'):

print("Not an appropriate choice!")

else:

break

if ch == 'E':

print("\nWelcome to Encryption\n")

encryption()

elif ch == 'D':

print("\nWelcome to Decryption\n")

decryption()

elif ch == 'B':

print("\nWelcome to Brute\_Force\n")

brute\_force()

elif ch == 'X':

exit("\nHmm, The program is now terminated! \n")

else:

choice()

print()

def choice2():

while True:

ch = (input("Please enter your choice: "))

if ch not in ('E', 'D', 'B', 'X'):

print("Not an appropriate choice.")

else:

break

if ch == 'E':

print("\nWelcome to Encryption\n")

encryption2()

elif ch == 'D':

print("\nWelcome to Decryption\n")

decryption2()

elif ch == 'B':

print("\nWelcome to Brute\_Force\n")

brute\_force2()

elif ch == 'X':

exit("\nHmm, The program is now terminated! \n")

else:

choice2()

print()

def plain\_text():

while True:

plt = input("Enter plain text: ").strip()

if plt.isalpha() or bool(re.search(r"\s", plt)):

return plt.upper()

else:

print("Sorry, I didn't understand that.")

def cipher\_text():

while True:

clt = input("Enter cipher text: ").strip()

if clt.isalpha() or bool(re.search(r"\s", clt)):

return clt.upper()

else:

print("Sorry, I didn't understand that.")

def key():

while True:

in\_key = input("Enter the key: ")

if in\_key.isdecimal():

return in\_key

else:

print("Invalid key entered!")

def encryption2():

ct = ""

pt = plain\_text()

k = int(key())

try:

k = k % 26

for i in pt.split():

for j in i:

total = (((ord(j) - 65) + k) % 26)

ct = ct + (chr(total + 65))

ct = ct + " "

print("Cipher Text is: ", ct.lower())

except:

print("Something went wrong!")

encryption2()

display()

def decryption2():

plt = ""

ct = cipher\_text()

k = int(key())

try:

k = k % 26

# if(k >= 1 and k <= 26):

if True:

for i in ct.split():

for j in i:

total = (((ord(j) - 65) - k) % 26)

plt = plt + (chr(total + 65))

plt = plt + " "

print("Plain Text is: ", plt.upper())

except:

print("Something went wrong!")

decryption2()

display()

def brute\_force2():

k = 0

pt = plain\_text()

ct = cipher\_text()

try:

while chr(ord(pt[0]) + k) != ct[0]:

k = k + 1

print("Key is:\t", k)

except:

print("Invalid text entered")

display()

def encryption():

ct = ""

pt = plain\_text()

k = int(key())

try:

for i in pt.split():

for j in i:

total = (((ord(j) - 65) \* k) % 26)

ct = ct + (chr(total + 65))

ct = ct + " "

print("Cipher Text is: ", ct.lower())

except:

print("Something went wrong")

encryption()

display()

def inverse():

a = int(key())

a = a % 26

for x in range(1, 26):

if (a \* x) % 26 == 1:

return x

return 1

def decryption():

plt = ""

ct = cipher\_text()

k = inverse()

try:

# if 1 <= k <= 26:

if True:

for i in ct.split():

for j in i:

total = (((ord(j) - 65) \* k) % 26)

plt = plt + (chr(total + 65))

plt = plt + " "

print("Plain Text is: ", plt.upper())

except:

print("Something went wrong")

decryption()

display()

def brute\_force():

ct = cipher\_text()

pt = plain\_text()

try:

if True:

for a in range(1, 26):

k = for\_brute(a)

plt = ""

for i in ct.split():

for j in i:

total = (((ord(j) - 65) \* k) % 26)

plt = plt + (chr(total + 65))

plt = plt + " "

if plt.lower() == (pt.lower() + " "):

print("Key is: ", a)

except:

print("Something went wrong")

brute\_force()

display()

def for\_brute(a):

a = a % 26

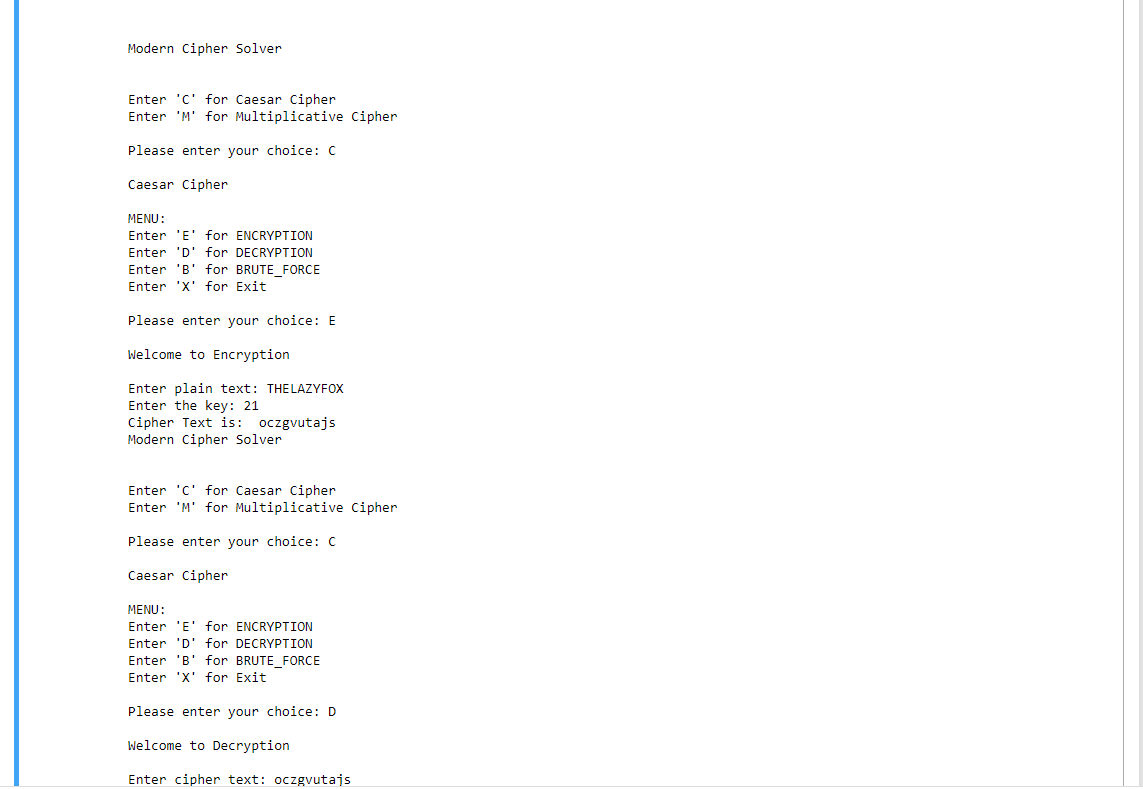
for x in range(1, 26):

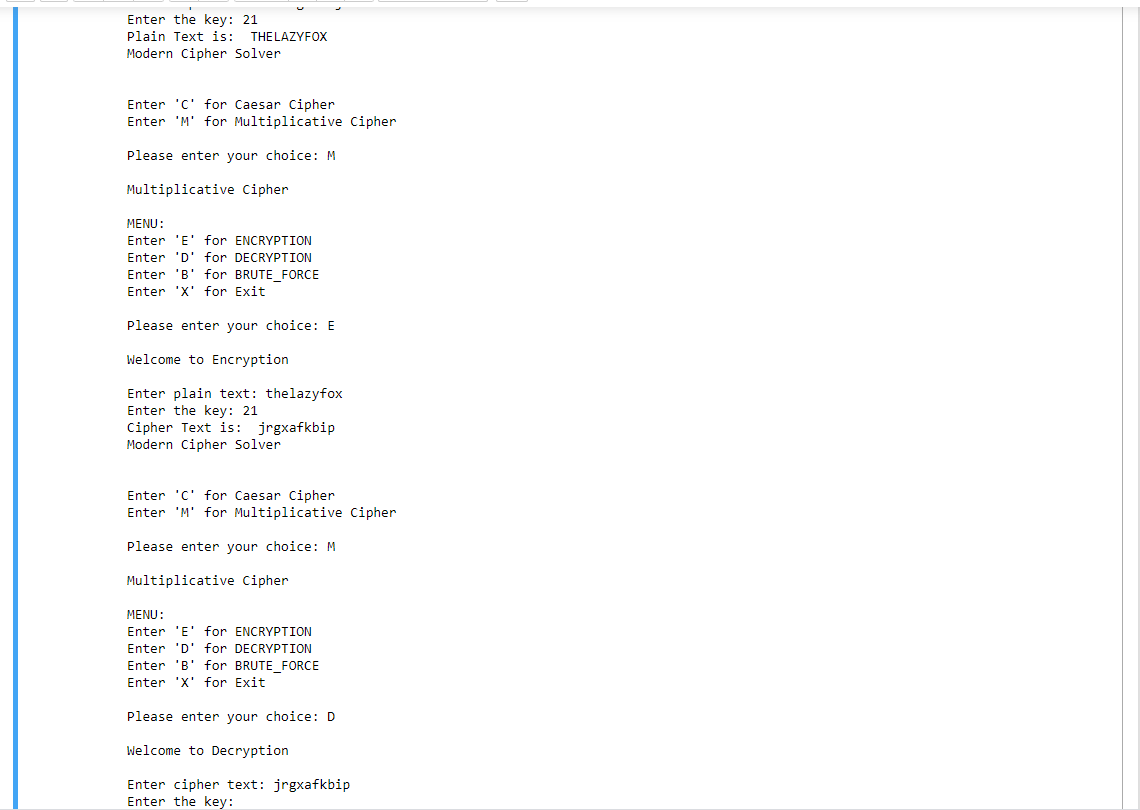
if (a \* x) % 26 == 1:

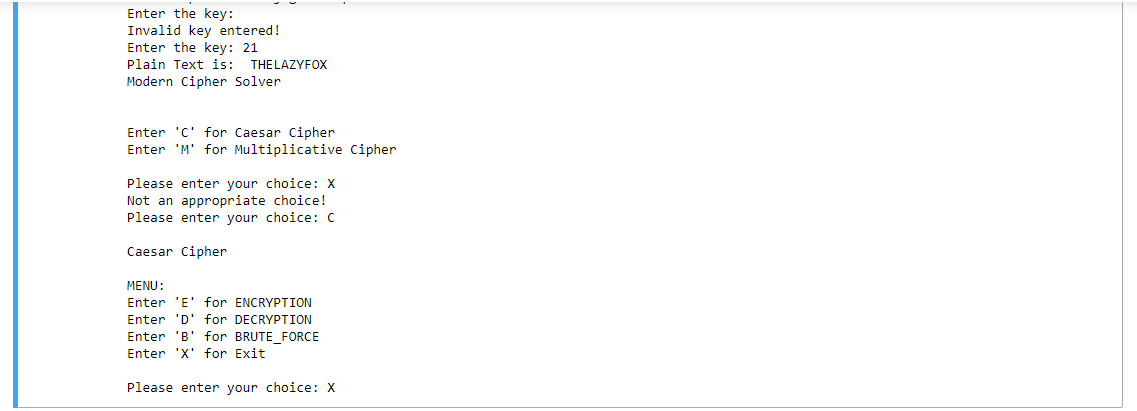
return x

return 1

display()





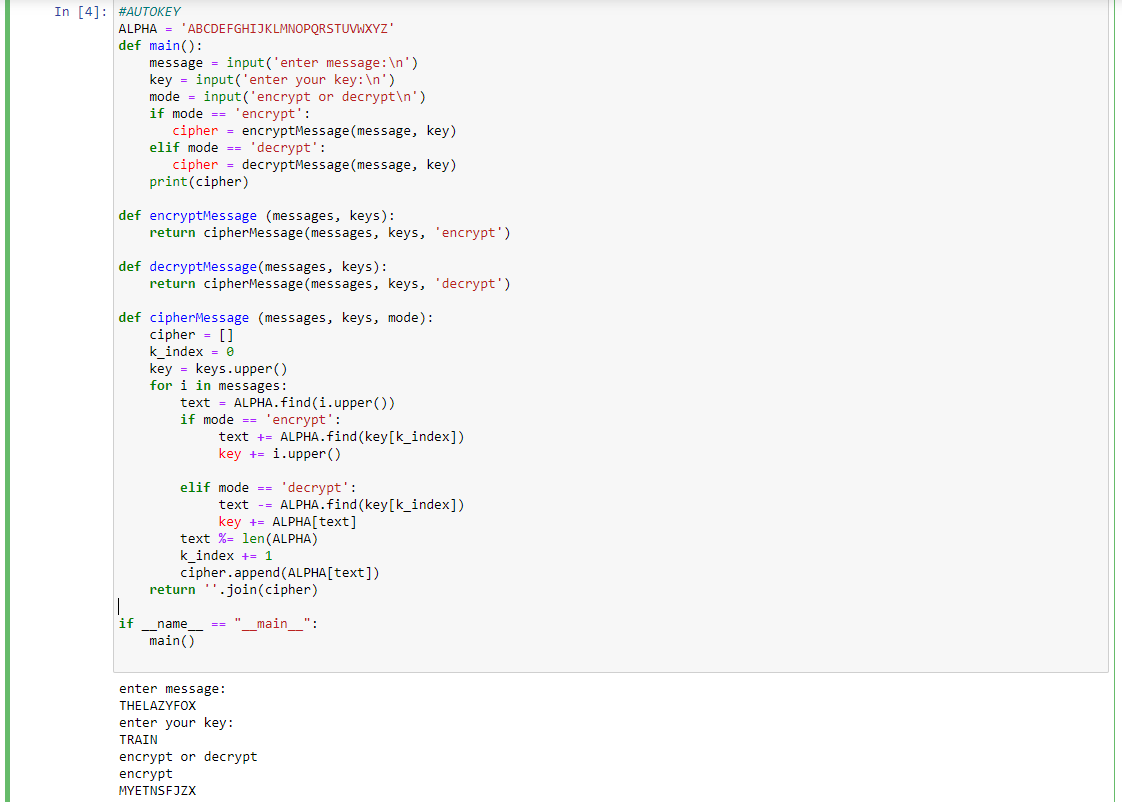


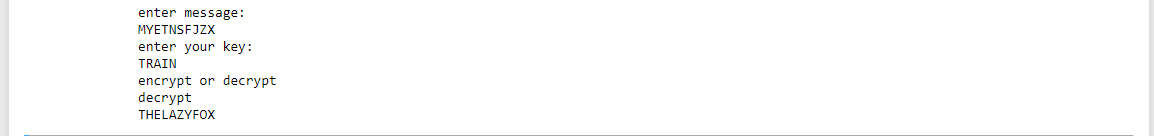
**Affine Cipher:**





**Autokey Cipher:**



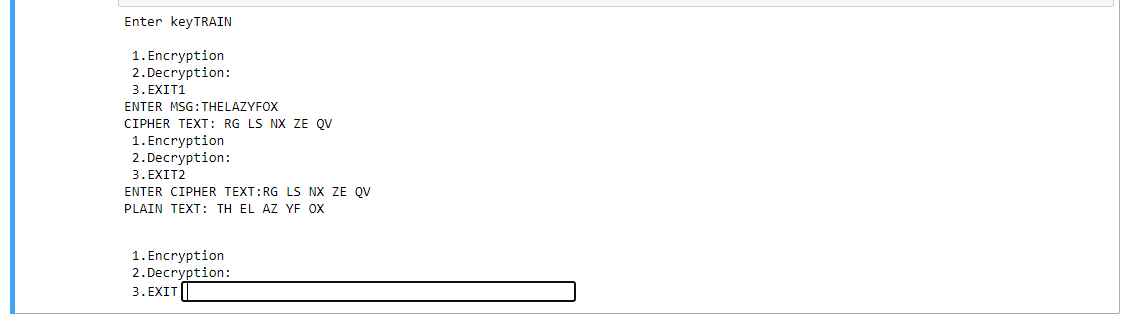


**Vigenère Cipher:**

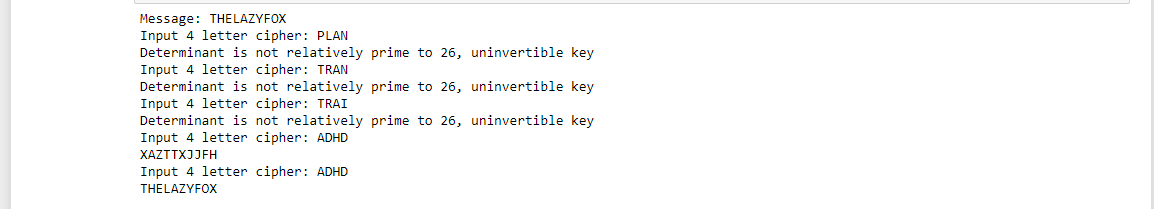




**Playfair cipher:**



**Hill cipher:**



**HYBRID CIPHER:**

**Combination(1) Autokey > Vigneere**

#Autokey --> Vigneere

ALPHA = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def main():

message = input('enter message:\n')

key = input('enter your key:\n')

mode = input('encrypt or decrypt\n')

if mode == 'encrypt':

cipher = encryptMessage(message, key)

print(cipher)

key1= generateKey(cipher, key)

cipher\_text = cipherText(cipher,key1)

print(cipher\_text)

elif mode == 'decrypt':

key1= generateKey(message, key)

cipher = originalText(message, key1)

print(cipher)

cipher\_text = decryptMessage(cipher, key)

print(cipher\_text)

def generateKey(string, key):

key = list(key)

if len(string) == len(key):

return(key)

else:

for i in range(len(string) -

len(key)):

key.append(key[i % len(key)])

return("" . join(key))

def cipherText(string, key):

cipher\_text = []

for i in range(len(string)):

x = (ord(string[i]) +

ord(key[i])) % 26

x += ord('A')

cipher\_text.append(chr(x))

return("" . join(cipher\_text))

def originalText(text, key):

orig\_text = []

for i in range(len(text)):

x = (ord(text[i]) -

ord(key[i]) + 26) % 26

x += ord('A')

orig\_text.append(chr(x))

return("" . join(orig\_text))

def encryptMessage (messages, keys):

return cipherMessage(messages, keys, 'encrypt')

def decryptMessage(messages, keys):

return cipherMessage(messages, keys, 'decrypt')

def cipherMessage (messages, keys, mode):

cipher = []

k\_index = 0

key = keys.upper()

for i in messages:

text = ALPHA.find(i.upper())

if mode == 'encrypt':

text += ALPHA.find(key[k\_index])

key += i.upper()

elif mode == 'decrypt':

text -= ALPHA.find(key[k\_index])

key += ALPHA[text]

text %= len(ALPHA)

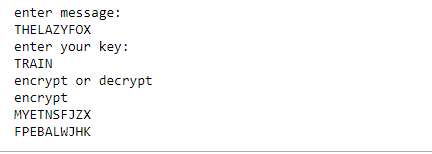
k\_index += 1

cipher.append(ALPHA[text])

return ''.join(cipher)

if \_\_name\_\_ == "\_\_main\_\_":

main()





**Combination(2) Affine > Vigneere**

#Affine --> Vigneere

def generateKey(string, key):

key = list(key)

if len(string) == len(key):

return(key)

else:

for i in range(len(string) -

len(key)):

key.append(key[i % len(key)])

return("" . join(key))

def cipherText(string, key):

cipher\_text = []

for i in range(len(string)):

x = (ord(string[i]) +

ord(key[i])) % 26

x += ord('A')

cipher\_text.append(chr(x))

return("" . join(cipher\_text))

def originalText(text, key):

orig\_text = []

for i in range(len(text)):

x = (ord(text[i]) -

ord(key[i]) + 26) % 26

x += ord('A')

orig\_text.append(chr(x))

return("" . join(orig\_text))

def egcd(a, b):

x,y, u,v = 0,1, 1,0

while a != 0:

q, r = b//a, b%a

m, n = x-u\*q, y-v\*q

b,a, x,y, u,v = a,r, u,v, m,n

gcd = b

return gcd, x, y

def modinv(a, m):

gcd, x, y = egcd(a, m)

if gcd != 1:

return None

else:

return x % m

def affine\_encrypt(text, key):

'''

C = (a\*P + b) % 26

'''

return ''.join([ chr((( key[0]\*(ord(t) - ord('A')) + key[1] ) % 26)

+ ord('A')) for t in text.upper().replace(' ', '') ])

def affine\_decrypt(cipher, key):

'''

P = (a^-1 \* (C - b)) % 26

'''

return ''.join([ chr((( modinv(key[0], 26)\*(ord(c) - ord('A') - key[1]))

% 26) + ord('A')) for c in cipher ])

def main():

string = input("Enter String: ")

key1= int(input("Enter key 1: "))

key2= int(input("Enter key 2: "))

key = [key1, key2]

keyword = input("Enter Keyword: ")

choice = input("Encrypt/Decrypt: ")

key3 = generateKey(string, keyword)

if choice == "Encrypt":

text=affine\_encrypt(string, key)

cipher\_text = cipherText(text,key3)

print("Ciphertext :", cipher\_text)

elif choice == "Decrypt":

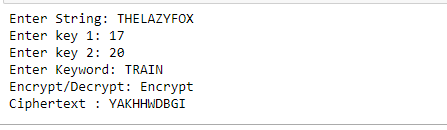
text= originalText(string, key3)

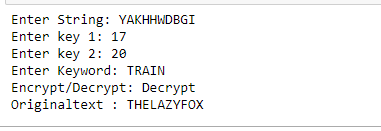
original=affine\_decrypt(text, key)

print("Originaltext :", original)

if \_\_name\_\_ == "\_\_main\_\_":

main()





**Combination(3) Autokey > Hill**

#Autokey --> Hill

import numpy as np

ALPHA = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def encryptMessage (messages, keys):

return cipherMessage(messages, keys, 'encrypt')

def decryptMessage(messages, keys):

return cipherMessage(messages, keys, 'decrypt')

def cipherMessage (messages, keys, mode):

cipher = []

k\_index = 0

key = keys.upper()

for i in messages:

text = ALPHA.find(i.upper())

if mode == 'encrypt':

text += ALPHA.find(key[k\_index])

key += i.upper()

elif mode == 'decrypt':

text -= ALPHA.find(key[k\_index])

key += ALPHA[text]

text %= len(ALPHA)

k\_index += 1

cipher.append(ALPHA[text])

return ''.join(cipher)

def encrypt(msg):

msg = msg.replace(" ", "")

C = make\_key()

len\_check = len(msg) % 2 == 0

if not len\_check:

msg += "0"

P = create\_matrix\_of\_integers\_from\_string(msg)

msg\_len = int(len(msg) / 2)

encrypted\_msg = ""

for i in range(msg\_len):

# Dot product

row\_0 = P[0][i] \* C[0][0] + P[1][i] \* C[0][1]

integer = int(row\_0 % 26 + 65)

encrypted\_msg += chr(integer)

row\_1 = P[0][i] \* C[1][0] + P[1][i] \* C[1][1]

integer = int(row\_1 % 26 + 65)

encrypted\_msg += chr(integer)

return encrypted\_msg

def decrypt(encrypted\_msg):

C = make\_key()

determinant = C[0][0] \* C[1][1] - C[0][1] \* C[1][0]

determinant = determinant % 26

multiplicative\_inverse = find\_multiplicative\_inverse(determinant)

C\_inverse = C

C\_inverse[0][0], C\_inverse[1][1] = C\_inverse[1, 1], C\_inverse[0, 0]

C[0][1] \*= -1

C[1][0] \*= -1

for row in range(2):

for column in range(2):

C\_inverse[row][column] \*= multiplicative\_inverse

C\_inverse[row][column] = C\_inverse[row][column] % 26

P = create\_matrix\_of\_integers\_from\_string(encrypted\_msg)

msg\_len = int(len(encrypted\_msg) / 2)

decrypted\_msg = ""

for i in range(msg\_len):

column\_0 = P[0][i] \* C\_inverse[0][0] + P[1][i] \* C\_inverse[0][1]

integer = int(column\_0 % 26 + 65)

decrypted\_msg += chr(integer)

column\_1 = P[0][i] \* C\_inverse[1][0] + P[1][i] \* C\_inverse[1][1]

integer = int(column\_1 % 26 + 65)

decrypted\_msg += chr(integer)

if decrypted\_msg[-1] == "0":

decrypted\_msg = decrypted\_msg[:-1]

return decrypted\_msg

def find\_multiplicative\_inverse(determinant):

multiplicative\_inverse = -1

for i in range(26):

inverse = determinant \* i

if inverse % 26 == 1:

multiplicative\_inverse = i

break

return multiplicative\_inverse

def make\_key():

determinant = 0

C = None

while True:

cipher = input("Input 4 letter cipher: ")

C = create\_matrix\_of\_integers\_from\_string(cipher)

determinant = C[0][0] \* C[1][1] - C[0][1] \* C[1][0]

determinant = determinant % 26

inverse\_element = find\_multiplicative\_inverse(determinant)

if inverse\_element == -1:

print("Determinant is not relatively prime to 26, uninvertible key")

elif np.amax(C) > 26 and np.amin(C) < 0:

print("Only a-z characters are accepted")

print(np.amax(C), np.amin(C))

else:

break

return C

def create\_matrix\_of\_integers\_from\_string(string):

integers = [chr\_to\_int(c) for c in string]

length = len(integers)

M = np.zeros((2, int(length / 2)), dtype=np.int32)

iterator = 0

for column in range(int(length / 2)):

for row in range(2):

M[row][column] = integers[iterator]

iterator += 1

return M

def chr\_to\_int(char):

char = char.upper()

integer = ord(char) - 65

return integer

def main():

msg = input("Message: ")

key1= input('enter your key:\n')

cipher = encryptMessage(msg, key1)

encrypted\_msg = encrypt(cipher)

print(encrypted\_msg)

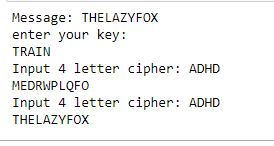
decrypted\_msg = decrypt(encrypted\_msg)

cipher = decryptMessage(decrypted\_msg, key1)

print(cipher)

if \_\_name\_\_ == "\_\_main\_\_":

main()



**Combination(4) Affine > Hill**

#Affine --> Hill

import numpy as np

def egcd(a, b):

x,y, u,v = 0,1, 1,0

while a != 0:

q, r = b//a, b%a

m, n = x-u\*q, y-v\*q

b,a, x,y, u,v = a,r, u,v, m,n

gcd = b

return gcd, x, y

def modinv(a, m):

gcd, x, y = egcd(a, m)

if gcd != 1:

return None

else:

return x % m

def affine\_encrypt(text, key):

'''

C = (a\*P + b) % 26

'''

return ''.join([ chr((( key[0]\*(ord(t) - ord('A')) + key[1] ) % 26)

+ ord('A')) for t in text.upper().replace(' ', '') ])

def affine\_decrypt(cipher, key):

'''

P = (a^-1 \* (C - b)) % 26

'''

return ''.join([ chr((( modinv(key[0], 26)\*(ord(c) - ord('A') - key[1]))

% 26) + ord('A')) for c in cipher ])

def encrypt(msg):

msg = msg.replace(" ", "")

C = make\_key()

len\_check = len(msg) % 2 == 0

if not len\_check:

msg += "0"

P = create\_matrix\_of\_integers\_from\_string(msg)

msg\_len = int(len(msg) / 2)

encrypted\_msg = ""

for i in range(msg\_len):

row\_0 = P[0][i] \* C[0][0] + P[1][i] \* C[0][1]

integer = int(row\_0 % 26 + 65)

encrypted\_msg += chr(integer)

row\_1 = P[0][i] \* C[1][0] + P[1][i] \* C[1][1]

integer = int(row\_1 % 26 + 65)

encrypted\_msg += chr(integer)

return encrypted\_msg

def decrypt(encrypted\_msg):

C = make\_key()

determinant = C[0][0] \* C[1][1] - C[0][1] \* C[1][0]

determinant = determinant % 26

multiplicative\_inverse = find\_multiplicative\_inverse(determinant)

C\_inverse = C

C\_inverse[0][0], C\_inverse[1][1] = C\_inverse[1, 1], C\_inverse[0, 0]

C[0][1] \*= -1

C[1][0] \*= -1

for row in range(2):

for column in range(2):

C\_inverse[row][column] \*= multiplicative\_inverse

C\_inverse[row][column] = C\_inverse[row][column] % 26

P = create\_matrix\_of\_integers\_from\_string(encrypted\_msg)

msg\_len = int(len(encrypted\_msg) / 2)

decrypted\_msg = ""

for i in range(msg\_len):

column\_0 = P[0][i] \* C\_inverse[0][0] + P[1][i] \* C\_inverse[0][1]

integer = int(column\_0 % 26 + 65)

decrypted\_msg += chr(integer)

column\_1 = P[0][i] \* C\_inverse[1][0] + P[1][i] \* C\_inverse[1][1]

integer = int(column\_1 % 26 + 65)

decrypted\_msg += chr(integer)

if decrypted\_msg[-1] == "0":

decrypted\_msg = decrypted\_msg[:-1]

return decrypted\_msg

def find\_multiplicative\_inverse(determinant):

multiplicative\_inverse = -1

for i in range(26):

inverse = determinant \* i

if inverse % 26 == 1:

multiplicative\_inverse = i

break

return multiplicative\_inverse

def make\_key():

determinant = 0

C = None

while True:

cipher = input("Input 4 letter cipher: ")

C = create\_matrix\_of\_integers\_from\_string(cipher)

determinant = C[0][0] \* C[1][1] - C[0][1] \* C[1][0]

determinant = determinant % 26

inverse\_element = find\_multiplicative\_inverse(determinant)

if inverse\_element == -1:

print("Determinant is not relatively prime to 26, uninvertible key")

elif np.amax(C) > 26 and np.amin(C) < 0:

print("Only a-z characters are accepted")

print(np.amax(C), np.amin(C))

else:

break

return C

def create\_matrix\_of\_integers\_from\_string(string):

integers = [chr\_to\_int(c) for c in string]

length = len(integers)

M = np.zeros((2, int(length / 2)), dtype=np.int32)

iterator = 0

for column in range(int(length / 2)):

for row in range(2):

M[row][column] = integers[iterator]

iterator += 1

return M

def chr\_to\_int(char):

char = char.upper()

integer = ord(char) - 65

return integer

def main():

msg = input("Message: ")

key1= int(input("Enter key 1: "))

key2= int(input("Enter key 2: "))

key = [key1, key2]

text = affine\_encrypt(msg, key)

encrypted\_msg = encrypt(text)

print(encrypted\_msg)

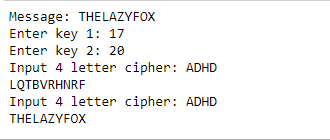
text = decrypt(encrypted\_msg)

decrypted\_msg = affine\_decrypt(text, key)

print(decrypted\_msg)

if \_\_name\_\_ == "\_\_main\_\_":

main()



**Combination(5) Affine > Hill**

#Affine --> Autokey --> Vigneere

ALPHA = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def main():

message = input('enter message:\n')

key1= int(input("Enter key 1: "))

key2= int(input("Enter key 2: "))

key = [key1, key2]

key3 = input('enter your key:\n')

mode = input('encrypt or decrypt\n')

if mode == 'encrypt':

message = affine\_encrypt(message, key)

cipher = encryptMessage(message, key3)

key4= generateKey(cipher, key3)

cipher\_text = cipherText(cipher,key4)

print(cipher\_text)

elif mode == 'decrypt':

key4= generateKey(message, key3)

cipher = originalText(message, key4)

cipher\_text = decryptMessage(cipher, key3)

cipher\_text = affine\_decrypt(cipher\_text, key)

print(cipher\_text)

def generateKey(string, key):

key = list(key)

if len(string) == len(key):

return(key)

else:

for i in range(len(string) -

len(key)):

key.append(key[i % len(key)])

return("" . join(key))

def cipherText(string, key):

cipher\_text = []

for i in range(len(string)):

x = (ord(string[i]) +

ord(key[i])) % 26

x += ord('A')

cipher\_text.append(chr(x))

return("" . join(cipher\_text))

def originalText(text, key):

orig\_text = []

for i in range(len(text)):

x = (ord(text[i]) -

ord(key[i]) + 26) % 26

x += ord('A')

orig\_text.append(chr(x))

return("" . join(orig\_text))

def encryptMessage (messages, keys):

return cipherMessage(messages, keys, 'encrypt')

def decryptMessage(messages, keys):

return cipherMessage(messages, keys, 'decrypt')

def cipherMessage (messages, keys, mode):

cipher = []

k\_index = 0

key = keys.upper()

for i in messages:

text = ALPHA.find(i.upper())

if mode == 'encrypt':

text += ALPHA.find(key[k\_index])

key += i.upper()

elif mode == 'decrypt':

text -= ALPHA.find(key[k\_index])

key += ALPHA[text]

text %= len(ALPHA)

k\_index += 1

cipher.append(ALPHA[text])

return ''.join(cipher)

def egcd(a, b):

x,y, u,v = 0,1, 1,0

while a != 0:

q, r = b//a, b%a

m, n = x-u\*q, y-v\*q

b,a, x,y, u,v = a,r, u,v, m,n

gcd = b

return gcd, x, y

def modinv(a, m):

gcd, x, y = egcd(a, m)

if gcd != 1:

return None

else:

return x % m

def affine\_encrypt(text, key):

'''

C = (a\*P + b) % 26

'''

return ''.join([ chr((( key[0]\*(ord(t) - ord('A')) + key[1] ) % 26)

+ ord('A')) for t in text.upper().replace(' ', '') ])

def affine\_decrypt(cipher, key):

'''

P = (a^-1 \* (C - b)) % 26

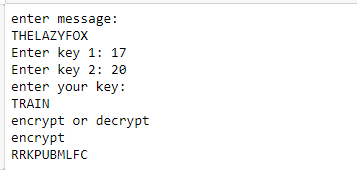
'''

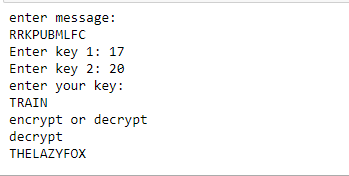
return ''.join([ chr((( modinv(key[0], 26)\*(ord(c) - ord('A') - key[1]))

% 26) + ord('A')) for c in cipher ])

if \_\_name\_\_ == "\_\_main\_\_":

main()





**KEY COMBINATION**

**Splitting of text and num dynamically**

#Affine -- > Vigneere

#Splitting of text and num dynamically

def generateKey(string, key):

key = list(key)

if len(string) == len(key):

return(key)

else:

for i in range(len(string) -

len(key)):

key.append(key[i % len(key)])

return("" . join(key))

def cipherText(string, key):

cipher\_text = []

for i in range(len(string)):

x = (ord(string[i]) +

ord(key[i])) % 26

x += ord('A')

cipher\_text.append(chr(x))

return("" . join(cipher\_text))

def originalText(text, key):

orig\_text = []

for i in range(len(text)):

x = (ord(text[i]) -

ord(key[i]) + 26) % 26

x += ord('A')

orig\_text.append(chr(x))

return("" . join(orig\_text))

def egcd(a, b):

x,y, u,v = 0,1, 1,0

while a != 0:

q, r = b//a, b%a

m, n = x-u\*q, y-v\*q

b,a, x,y, u,v = a,r, u,v, m,n

gcd = b

return gcd, x, y

def modinv(a, m):

gcd, x, y = egcd(a, m)

if gcd != 1:

return None

else:

return x % m

def affine\_encrypt(text, key):

'''

C = (a\*P + b) % 26

'''

return ''.join([ chr((( key[0]\*(ord(t) - ord('A')) + key[1] ) % 26)

+ ord('A')) for t in text.upper().replace(' ', '') ])

def affine\_decrypt(cipher, key):

'''

P = (a^-1 \* (C - b)) % 26

'''

return ''.join([ chr((( modinv(key[0], 26)\*(ord(c) - ord('A') - key[1]))

% 26) + ord('A')) for c in cipher ])

def main():

string = input("Enter String: ")

strings = input("Enter Key: ")

alpha = ""

num = ""

special = ""

for i in range(len(strings)):

if (strings[i].isdigit()):

num = num+ strings[i]

elif((strings[i] >= 'A' and strings[i] <= 'Z') or

(strings[i] >= 'a' and strings[i] <= 'z')):

alpha += strings[i]

else:

special += strings[i]

keyword=alpha

K=2

res = [int(num[idx : idx + K]) for idx in range(0, len(num), K)]

key = res

choice = input("Encrypt/Decrypt: ")

key3 = generateKey(string, keyword)

if choice == "Encrypt":

text=affine\_encrypt(string, key)

cipher\_text = cipherText(text,key3)

print("Ciphertext :", cipher\_text)

elif choice == "Decrypt":

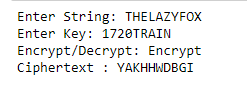
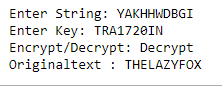
text= originalText(string, key3)

original=affine\_decrypt(text, key)

print("Originaltext :", original)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Splitting of text and num selectively**

Condition: THE SECOND , THIRD ,FIFTH AND SIXTH POS OF KEY MUST BE A NUMBER

#Affine -- > Vigneere

#Splitting of text and num selectively

def generateKey(string, key):

key = list(key)

if len(string) == len(key):

return(key)

else:

for i in range(len(string) -

len(key)):

key.append(key[i % len(key)])

return("" . join(key))

def cipherText(string, key):

cipher\_text = []

for i in range(len(string)):

x = (ord(string[i]) +

ord(key[i])) % 26

x += ord('A')

cipher\_text.append(chr(x))

return("" . join(cipher\_text))

def originalText(text, key):

orig\_text = []

for i in range(len(text)):

x = (ord(text[i]) -

ord(key[i]) + 26) % 26

x += ord('A')

orig\_text.append(chr(x))

return("" . join(orig\_text))

def egcd(a, b):

x,y, u,v = 0,1, 1,0

while a != 0:

q, r = b//a, b%a

m, n = x-u\*q, y-v\*q

b,a, x,y, u,v = a,r, u,v, m,n

gcd = b

return gcd, x, y

def modinv(a, m):

gcd, x, y = egcd(a, m)

if gcd != 1:

return None

else:

return x % m

def affine\_encrypt(text, key):

'''

C = (a\*P + b) % 26

'''

return ''.join([ chr((( key[0]\*(ord(t) - ord('A')) + key[1] ) % 26)

+ ord('A')) for t in text.upper().replace(' ', '') ])

def affine\_decrypt(cipher, key):

'''

P = (a^-1 \* (C - b)) % 26

'''

return ''.join([ chr((( modinv(key[0], 26)\*(ord(c) - ord('A') - key[1]))

% 26) + ord('A')) for c in cipher ])

def main():

string = input("Enter String: ")

strings = input("Enter Key: ")

alpha = ""

num = ""

a=2

b=3

c=5

d=6

special = ""

for i in range(len(strings)):

if ( (i > 1 and i != 4 and i < 7 )and (strings[i].isdigit()) ): #example AD17I20NE

num = num+ strings[i]

elif((strings[i] >= 'A' and strings[i] <= 'Z') or

(strings[i] >= 'a' and strings[i] <= 'z')):

alpha += strings[i]

else:

special += strings[i]

keyword=alpha

K=2

res = [int(num[idx : idx + K]) for idx in range(0, len(num), K)]

key = res

choice = input("Encrypt/Decrypt: ")

key3 = generateKey(string, keyword)

if choice == "Encrypt":

text=affine\_encrypt(string, key)

cipher\_text = cipherText(text,key3)

print("Ciphertext :", cipher\_text)

elif choice == "Decrypt":

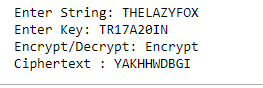
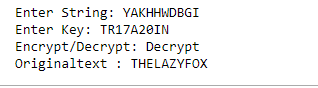
text= originalText(string, key3)

original=affine\_decrypt(text, key)

print("Originaltext :", original)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Splitting of text and num selectively(Includes rearranging of number)**

Condition: Same condition as previous but in reverse

#Affine -- > Vigneere

#Splitting of text and num selectively(Includes rearranging of number)

def generateKey(string, key):

key = list(key)

if len(string) == len(key):

return(key)

else:

for i in range(len(string) -

len(key)):

key.append(key[i % len(key)])

return("" . join(key))

def cipherText(string, key):

cipher\_text = []

for i in range(len(string)):

x = (ord(string[i]) +

ord(key[i])) % 26

x += ord('A')

cipher\_text.append(chr(x))

return("" . join(cipher\_text))

def originalText(text, key):

orig\_text = []

for i in range(len(text)):

x = (ord(text[i]) -

ord(key[i]) + 26) % 26

x += ord('A')

orig\_text.append(chr(x))

return("" . join(orig\_text))

def egcd(a, b):

x,y, u,v = 0,1, 1,0

while a != 0:

q, r = b//a, b%a

m, n = x-u\*q, y-v\*q

b,a, x,y, u,v = a,r, u,v, m,n

gcd = b

return gcd, x, y

def modinv(a, m):

gcd, x, y = egcd(a, m)

if gcd != 1:

return None

else:

return x % m

def affine\_encrypt(text, key):

'''

C = (a\*P + b) % 26

'''

return ''.join([ chr((( key[0]\*(ord(t) - ord('A')) + key[1] ) % 26)

+ ord('A')) for t in text.upper().replace(' ', '') ])

def affine\_decrypt(cipher, key):

'''

P = (a^-1 \* (C - b)) % 26

'''

return ''.join([ chr((( modinv(key[0], 26)\*(ord(c) - ord('A') - key[1]))

% 26) + ord('A')) for c in cipher ])

def main():

string = input("Enter String: ")

strings = input("Enter Key: ")

alpha = ""

num = ""

special = ""

for i in range(len(strings)):

if ( (i > 1 and i != 4 and i < 7 )and (strings[i].isdigit()) ):

num = num+ strings[i]

elif((strings[i] >= 'A' and strings[i] <= 'Z') or

(strings[i] >= 'a' and strings[i] <= 'z')):

alpha += strings[i]

else:

special += strings[i]

n=int(num)

keyword=alpha

c = 0

temp = n

x = []

no = 0

while temp != 0:

temp = temp//10

c = c+1

for i in range(c):

x.append(n%10)

n = n//10

x.sort()

for i in x:

no = (no\*10)+i

number=str(no)

K=2

res = [int(number[idx : idx + K]) for idx in range(0, len(number), K)]

key = res

choice = input("Encrypt/Decrypt: ")

key3 = generateKey(string, keyword)

if choice == "Encrypt":

text=affine\_encrypt(string, key)

cipher\_text = cipherText(text,key3)

print("Ciphertext :", cipher\_text)

elif choice == "Decrypt":

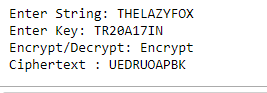
text= originalText(string, key3)

original=affine\_decrypt(text, key)

print("Originaltext :", original)

if \_\_name\_\_ == "\_\_main\_\_":

main()



**rearranging of number and splitting it**

CONDITION: 1st and 3rd digit will be swapped

#Affine

#rearranging of number and splitting it

def egcd(a, b):

x,y, u,v = 0,1, 1,0

while a != 0:

q, r = b//a, b%a

m, n = x-u\*q, y-v\*q

b,a, x,y, u,v = a,r, u,v, m,n

gcd = b

return gcd, x, y

def modinv(a, m):

gcd, x, y = egcd(a, m)

if gcd != 1:

return None

else:

return x % m

def affine\_encrypt(text, key):

'''

C = (a\*P + b) % 26

'''

return ''.join([ chr((( key[0]\*(ord(t) - ord('A')) + key[1] ) % 26)

+ ord('A')) for t in text.upper().replace(' ', '') ])

def affine\_decrypt(cipher, key):

'''

P = (a^-1 \* (C - b)) % 26

'''

return ''.join([ chr((( modinv(key[0], 26)\*(ord(c) - ord('A') - key[1]))

% 26) + ord('A')) for c in cipher ])

def main():

text = input("Enter String: ")

n= int(input("Enter 4 Digit key : "))

c = 0

temp = n

x = []

no = 0

while temp != 0:

temp = temp//10

c = c+1

for i in range(c):

x.append(n%10)

n = n//10

x.sort()

for i in x:

no = (no\*10)+i

number=str(no)

K=2

res = [int(number[idx : idx + K]) for idx in range(0, len(number), K)]

key = res

choice = input("Encrypt/Decrypt: ")

if choice == "Encrypt":

affine\_encrypted\_text = affine\_encrypt(text, key)

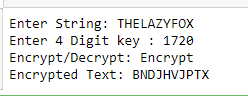
print('Encrypted Text: {}'.format( affine\_encrypted\_text ))

elif choice == "Decrypt":

print('Decrypted Text: {}'.format( affine\_decrypt(text, key) ))

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

main()



**Taking a section of string from key and adding it as second key**

CONDITION: TAKING A 5 LETTER SUBSTRING FROM MAIN STRING AS SECOND STRING BY TAKING THE STRING FROM STARTING POSITION 1

Eg: TRAINING: KEY1 RAINI: KEY2

#Autokey --> Vigneere

#Taking a section of string from key and adding it as second key

ALPHA = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def main():

message = input('enter message:\n')

key = input('enter your key:\n')

ke=key[1:5]

mode = input('encrypt or decrypt\n')

if mode == 'encrypt':

cipher = encryptMessage(message, key)

key1= generateKey(cipher, ke)

cipher\_text = cipherText(cipher,key1)

print(cipher\_text)

elif mode == 'decrypt':

key1= generateKey(message, ke)

cipher = originalText(message, key1)

cipher\_text = decryptMessage(cipher, key)

print(cipher\_text)

def generateKey(string, key):

key = list(key)

if len(string) == len(key):

return(key)

else:

for i in range(len(string) -

len(key)):

key.append(key[i % len(key)])

return("" . join(key))

def cipherText(string, key):

cipher\_text = []

for i in range(len(string)):

x = (ord(string[i]) +

ord(key[i])) % 26

x += ord('A')

cipher\_text.append(chr(x))

return("" . join(cipher\_text))

def originalText(text, key):

orig\_text = []

for i in range(len(text)):

x = (ord(text[i]) -

ord(key[i]) + 26) % 26

x += ord('A')

orig\_text.append(chr(x))

return("" . join(orig\_text))

def encryptMessage (messages, keys):

return cipherMessage(messages, keys, 'encrypt')

def decryptMessage(messages, keys):

return cipherMessage(messages, keys, 'decrypt')

def cipherMessage (messages, keys, mode):

cipher = []

k\_index = 0

key = keys.upper()

for i in messages:

text = ALPHA.find(i.upper())

if mode == 'encrypt':

text += ALPHA.find(key[k\_index])

key += i.upper()

elif mode == 'decrypt':

text -= ALPHA.find(key[k\_index])

key += ALPHA[text]

text %= len(ALPHA)

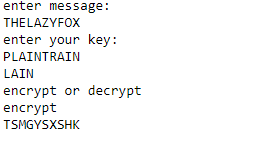
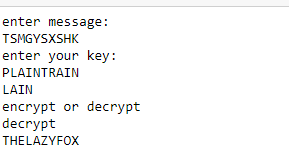
k\_index += 1

cipher.append(ALPHA[text])

return ''.join(cipher)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Result and inference**

While taking the comparison of symmetric ciphers, we can rank these cipher algorithms in the given table below(1-best,7-worst)

|  |  |
| --- | --- |
| *Ciphers* | *Rank* |
| Caesar Cipher | 7 |
| Multiplicative Cipher | 6 |
| Affine Cipher | 5 |
| Autokey Cipher | 2 |
| Vigenère Cipher | 3 |
| Playfair Cipher | 1 |
| Hill Cipher | 4 |

It’s given that Playfair is the strongest and it doesn’t need to be used for any hybridization for cipher. Rest of the algorithm have open key vulnerabilities and result in easy prediction of the algorithm by hacker. So, hybridization of the remaining algorithm is needed. Caesar cipher is the weakest and it can be eliminated. Also same goes for multiplicative cipher. With the rest after hybridization we can compare the result and rank in the given table below

|  |  |
| --- | --- |
| *Ciphers* | *Rank* |
| Autokey -> Vigenère | 4 |
| Affine -> Vigenère | 5 |
| Autokey -> Hill | 1 |
| Affine -> Hill | 3 |
| Affine -> Autokey -> Vigenère | 2 |

Autokey - > Hill is the strongest since both the key input are string. Hybridization allows the key to become more predictable and easily detect the algorithm. So key will be strengthened. Now we will compare the result.

|  |  |
| --- | --- |
| Key combination | *Rank* |
| Splitting of text and num dynamically (Affine -> Vigenère) | 5 |
| Splitting of text and num selectively (Affine -> Vigenère) | 2 |
| Splitting of text and num selectively and reversing it (Affine -> Vigenère) | 1 |
| Rearranging of number and splitting it (Affine) | 4 |
| Taking a section of string from key and adding it as second key  (Autokey -> Vigenère) | 3 |

Most of the key combination are strong except for **“ Splitting of text and num dynamically (Affine -> Vigenère) “** because the keys are predictable while typing in different format. **“Splitting of text and num selectively and reversing it (Affine -> Vigenère)”** is the most reliable among the five.

**Conclusion:**

From the above result Playfair cipher is the strongest that doesn’t require hybridization and not prone to hackers. Autokey-> Hill is the strongest hybrid cipher and can minimize the risk of prone to hackers. Affine->Vigenère can be improvised stronger by using the key combination and acceptance algorithm of Splitting of text and num selectively and reversing it . So therefore all algorithm are hybridized and risk of brute force can be minimized

**References:**

5- Singh, G.: A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security. International Journal of Computer Applications,

Vol. 67, No. 19, (2013).

6- Singh, M. G., Singla, M. A., & Sandha, M. K.: Cryptography Algorithm Comparison for Security Enhancement in Wireless Intrusion Detection System.

International Journal of Multidisciplinary Research, Vol. 1, No. 4, 143-151, (2011).

7- William, S.: Cryptography and Network Security: principles and practices. Pearson Education India (2006).

8- Chowdhury, Z. J., Pishva, D., & Nishantha, G. G. D.: AES and Confidentiality from the Inside Out. In Advanced Communication Technology

(ICACT), 2010 The 12th International Conference on IEEE, Vol. 2, 1587-1591, (February 2010).

9- Mandal, A. K., Parakash, C., & Tiwari, A.: Performance Evaluation of Cryptographic Algorithms: DES and AES. In Electrical, Electronics and

Computer Science (SCEECS), 2012 IEEE Students' Conference on, 1-5, (March 2012).

10- Adhie, R. P., Hutama, Y., Ahmar, A. S., & Setiawan, M. I.: Implementation Cryptography Data Encryption Standard (DES) and Triple Data

Encryption Standard (3DES) Method in Communication System Based Near Field Communication (NFC). In Journal of Physics: Conference Series,

Vol. 954, No. 1, (January 2018).

11- Koko, S. O. A. F. M., & Babiker, A.: Comparison of Various Encryption Algorithms and Techniques for improving secured data Communication. IOSR

Journal of Computer Engineering (IOSR-JCE), Vol. 17, No. 1, 62-69, (2015).

12- Kumar, A., Jakhar, D. S., & Makkar, M. S.: Comparative Analysis between DES and RSA Algorithm’s. International Journal of Advanced Research in

Computer Science and Software Engineering, Vol. 2, No. 7, 386-391, (2012).

13- Zhou, X., & Tang, X.: Research and Implementation of RSA Algorithm for Encryption and Decryption. In Strategic Technology (IFOST), 2011 6th

International Forum on IEEE, Vol. 2, 1118-1121, (August 2011).

14- Somani, U., Lakhani, K., & Mundra, M.: Implementing Digital Signature with RSA Encryption Algorithm to Enhance the Data Security of Cloud in

Cloud Computing. In Parallel Distributed and Grid Computing (PDGC), 2010 1st International Conference on IEEE, 211-216, (October 2010).