|  |  |
| --- | --- |
| Roll. No. A016 | Name: Varun Khadayate |
| Class B.Tech CsBs | Batch: 1 |
| Date of Experiment: 23-07-2022 | Subject: Cryptology |

# Aim

To implement Ceaser Cipher.

# Theory

The earliest known, and the simplest, use of a substitution cipher was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet. For example,

plain: meet me after the toga partycipher: PHHW PH DIWHU WKH WRJD SDUWB

Note that the alphabet is wrapped around, so that the letter following Z is A.We can define the transformation by listing all possibilities, as follows:

plain: a b c d e f g h i j k l m n o p q r s t u v w x y z cipher: D E F g H I J K L M N O P q R S T U V W x y z a B C

Let us assign a numerical equivalent to each letter:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | d | e | f | g | h | i | j | k | l | m |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | o | p | q | r | s | t | u | v | w | x | y | z |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

Then the algorithm can be expressed as follows. For each plaintext letter *p*, substi-tute the ciphertext letter *C*:2

*C* = E(3, *p*) = (*p* + 3) mod 26

A shift may be of any amount, so that the general Caesar algorithm is

*C* = E(*k*, *p*) = (*p* + *k*) mod 26 **(2.1)**

where *k* takes on a value in the range 1 to 25. The decryption algorithm is simply

*p* = D(*k*, *C*) = (*C* - *k*) mod 26 **(2.2)**

If it is known that a given ciphertext is a Caesar cipher, then a brute-force cryptanalysis is easily performed: simply try all the 25 possible keys. Figure 2.3 shows the results of applying this strategy to the example ciphertext. In this case, the plaintext leaps out as occupying the third line.

Three important characteristics of this problem enabled us to use a brute- force cryptanalysis:

1. The encryption and decryption algorithms are known.
2. There are only 25 keys to try.
3. The language of the plaintext is known and easily recognizable.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | PHHW | PH | DIWHU | WKH | WRJD | SDUWB |
| KEY |  |  |  |  |  |  |
| 1 | oggv | og | chvgt | vjg | vqic | rctva |
| 2 | nffu | nf | bgufs | uif | uphb | qbsuz |
| 3 | meet | me | after | the | toga | party |
| 4 | ldds | ld | zesdq | sgd | snfz | ozqsx |
| 5 | kccr | kc | ydrcp | rfc | rmey | nyprw |
| 6 | jbbq | jb | xcqbo | qeb | qldx | mxoqv |
| 7 | iaap | ia | wbpan | pda | pkcw | lwnpu |
| 8 | hzzo | hz | vaozm | ocz | ojbv | kvmot |
| 9 | gyyn | gy | uznyl | nby | niau | julns |
| 10 | fxxm | fx | tymxk | max | mhzt | itkmr |
| 11 | ewwl | ew | sxlwj | lzw | lgys | hsjlq |
| 12 | dvvk | dv | rwkvi | kyv | kfxr | grikp |
| 13 | cuuj | cu | qvjuh | jxu | jewq | fqhjo |
| 14 | btti | bt | puitg | iwt | idvp | epgin |
| 15 | assh | as | othsf | hvs | hcuo | dofhm |
| 16 | zrrg | zr | nsgre | gur | gbtn | cnegl |
| 17 | yqqf | yq | mrfqd | ftq | fasm | bmdfk |
| 18 | xppe | xp | lqepc | esp | ezrl | alcej |
| 19 | wood | wo | kpdob | dro | dyqk | zkbdi |
| 20 | vnnc | vn | jocna | cqn | cxpj | yjach |
| 21 | ummb | um | inbmz | bpm | bwoi | xizbg |
| 22 | tlla | tl | hmaly | aol | avnh | whyaf |
| 23 | skkz | sk | glzkx | znk | zumg | vgxze |
| 24 | rjjy | rj | fkyjw | ymj | ytlf | ufwyd |
| 25 | qiix | qi | ejxiv | xli | xske | tevxc |

In most networking situations, we can assume that the algorithms are known. What generally makes brute-force cryptanalysis impractical is the use of an algo- rithm that employs a large number of keys. For example, the triple DES algorithm, examined in Chapter 6, makes use of a 168-bit key, giving a key space of 2168 or greater than 3.7 \* 1050 possible keys.

The third characteristic is also significant. If the language of the plaintext

is unknown, then plaintext output may not be recognizable. Furthermore, the input may be abbreviated or compressed in some fashion, again making recogni-tion difficult. For example, Figure 2.4 shows a portion of a text file compressedusing an algorithm called ZIP. If this file is then encrypted with a simple sub-stitution cipher (expanded to include more than just 26 alphabetic characters), then the plaintext may not be recognized when it is uncovered in the brute-force cryptanalysis.

# Code

def Encryption(plaintext, key\_val): ciphertext = ''

for i in plaintext: if i.isupper():

temp = 65 + ((ord(i) - 65 + key) % 26) ciphertext = ciphertext + chr(temp)

elif i.islower():

temp = 97 + ((ord(i) - 97 + key) % 26) ciphertext = ciphertext + chr(temp)

else:

ciphertext = ciphertext + i

return ciphertext

def Decryption(ciphertext, key\_val): plaintext = ''

for i in ciphertext: if i.isupper():

if ((ord(i) - 65 - key) < 0):

temp = 65 + ((ord(i) - 65 - key + 26) % 26) else:

temp = 65 + ((ord(i) - 65 - key) % 26) plaintext = plaintext + chr(temp)

elif i.islower():

if ((ord(i) - 97 - key) < 0):

temp = 97 + ((ord(i) - 97 - key + 26) % 26) else:

temp = 97 + ((ord(i) - 97 - key) % 26) plaintext = plaintext + chr(temp)

else:

plaintext = plaintext + i return plaintext

while True:

print('Welcome to Ceaser Cipher Encryption and Decryption Program Made by Varun Khadayate..\n [\*] Press 1 for Encryption \n [\*] Press 0 for Decryption

\n [\*] Press 01 to exit.. ')

print('Tip ---> Encryption/Decryption with shift value of your choice ! ') choice = input('Insert Here : ')

if choice.isdigit(): if choice == '1':

sen = input('Insert the plaintext : ')

key = int(input('Insert shift value(Only integer values) : ')) print(50 \* '-')

print(f'Your ciphertext ---> {Encryption(sen, key)}')

print(50 \* '-')

con = input('Shall we continue ? [Any Key/no]') if con == 'no':

print('Exiting..') break

else:

pass

elif choice == '0':

csen = input('Insert the ciphertext : ')

key = int(input('Insert shift value(Only integer values) : ')) print(50 \* '-')

print(f'Your plaintext ---> {Decryption(csen, key)}') print(50 \* '-')

con = input('Do you want to continue ? [Any Key/no]') if con == 'no':

print('Exiting..') break

else:

pass

elif choice == '01': print('Exiting..')

print('Thank You for using the system') break

else:

print('Exception error .. \n' 'Please insert 0 or 1 ')

# Output

Welcome to Ceaser Cipher Encryption and Decryption Program Made by Varun Khadayate.. [\*] Press 1 for Encryption

[\*] Press 0 for Decryption [\*] Press 01 to exit..

Tip ---> Encryption/Decryption with shift value of your choice ! Insert Here : 1

Insert the plaintext : Varun Khadayate Insert shift value(Only integer values) : 10

Your ciphertext ---> Fkbex Urknkikdo Shall we continue ? [Any Key/no]

Welcome to Ceaser Cipher Encryption and Decryption Program Made by Varun Khadayate.. [\*] Press 1 for Encryption

[\*] Press 0 for Decryption [\*] Press 01 to exit..

Tip ---> Encryption/Decryption with shift value of your choice ! Insert Here : 0

Insert the ciphertext : Fkbex Urknkikdo!!! Insert shift value(Only integer values) : 10

Your plaintext ---> Varun Khadayate!!!

Do you want to continue ? [Any Key/no]no Exiting..

## Conclusion

Hence, we were able to perform Caesar Cipher.