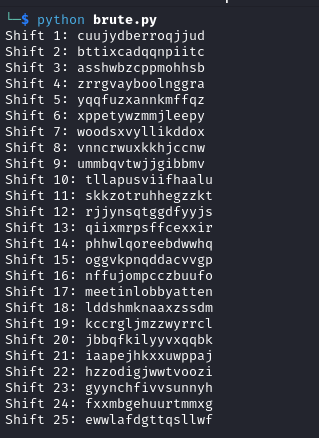
1. Write a Python Program to perform brute force attack on the cipher text “dvvkzecfssprkkve"





2. Write a Python program to use brute force attack to decipher the message.

Assume Affine cipher was used and "ab" is encrypted as "GL". Find the value of keys.

XPALASXYFGFUKPXUSOGEUTKCDGFXANMGNVS

#!/usr/bin/python

import string

from sympy import mod\_inverse

def affine\_decrypt(ciphertext, a, b):

alphabet = string.ascii\_uppercase

m = len(alphabet)

a\_inv = mod\_inverse(a, m) # Find modular inverse of a modulo m

plaintext = ""

for char in ciphertext:

if char in alphabet:

y = alphabet.index(char)

x = (a\_inv \* (y - b)) % m

plaintext += alphabet[x]

else:

plaintext += char

return plaintext

def brute\_force\_affine(ciphertext, known\_plaintext, known\_ciphertext):

alphabet = string.ascii\_uppercase

m = len(alphabet)

known\_plaintext = known\_plaintext.upper()

known\_ciphertext = known\_ciphertext.upper()

# Create a dictionary of all possible (a, b) pairs

for a in range(1, m):

if gcd(a, m) != 1:

continue # 'a' must be coprime with m (i.e., 1 < a < 26)

for b in range(m):

# Encrypt known plaintext using (a, b) to check if it matches the known ciphertext

test\_ciphertext = ""

for char in known\_plaintext:

if char in alphabet:

x = alphabet.index(char)

y = (a \* x + b) % m

test\_ciphertext += alphabet[y]

else:

test\_ciphertext += char

if test\_ciphertext == known\_ciphertext:

# Found the correct (a, b) pair

print(f"Found keys: a = {a}, b = {b}")

# Decrypt the ciphertext using found keys

decrypted\_message = affine\_decrypt(ciphertext, a, b)

print(f"Decrypted message: {decrypted\_message}")

def gcd(a, b):

while b:

a, b = b, a % b

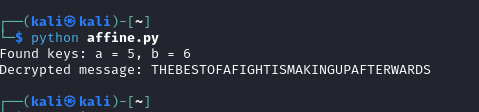
return a

ciphertext = "XPALASXYFGFUKPXUSOGEUTKCDGFXANMGNVS"

known\_plaintext = "ab"

known\_ciphertext = "GL"

brute\_force\_affine(ciphertext, known\_plaintext, known\_ciphertext)



1. Encrypt the plain text using Rail Fence cipher

def encrypt\_rail\_fence\_cipher(plain\_text, num\_rails):

rail\_matrix = [['\n' for \_ in range(len(plain\_text))]

for \_ in range(num\_rails)]

# Variables to determine the direction of zigzag

dir\_down = False

row, col = 0, 0

# Fill the rail matrix in a zigzag pattern

for char in plain\_text:

# Place the character in the current row

rail\_matrix[row][col] = char

col += 1

# Change direction if we hit the top or bottom

if row == 0 or row == num\_rails - 1:

dir\_down = not dir\_down

row += 1 if dir\_down else -1

cipher\_text = []

for i in range(num\_rails):

for j in range(len(plain\_text)):

if rail\_matrix[i][j] != '\n':

cipher\_text.append(rail\_matrix[i][j])

return "".join(cipher\_text)

plain\_text = "HELLO WORLD"

num\_rails = 3

cipher\_text = encrypt\_rail\_fence\_cipher(plain\_text, num\_rails)

print("Cipher Text:", cipher\_text)



4. Decrypt the cipher using Rail Fence

AAIUJ SIHBE KTEAO TEADE SNUTF EAEMR TAHSA

RHROA YHNFO AITTE EHCBO FVCAT RNMNS NUTFE

RASHL WFHLN HIUJS IHTKS OEHNI FISAE FNTIG

RMRSO LSTIS OKIEH PEOE

def decrypt\_rail\_fence\_cipher(cipher\_text, num\_rails):

# Create an empty matrix to mark the positions in the zigzag pattern

rail\_matrix = [['\n' for \_ in range(len(cipher\_text))]

for \_ in range(num\_rails)]

dir\_down = None

row, col = 0, 0

for i in range(len(cipher\_text)):

if row == 0:

dir\_down = True

if row == num\_rails - 1:

dir\_down = False

# Place a marker at the current position

rail\_matrix[row][col] = '\*'

col += 1

row += 1 if dir\_down else -1

index = 0

for i in range(num\_rails):

for j in range(len(cipher\_text)):

if rail\_matrix[i][j] == '\*' and index < len(cipher\_text):

rail\_matrix[i][j] = cipher\_text[index]

index += 1

result = []

row, col = 0, 0

for i in range(len(cipher\_text)):

if row == 0:

dir\_down = True

if row == num\_rails - 1:

dir\_down = False

if rail\_matrix[row][col] != '\n':

result.append(rail\_matrix[row][col])

col += 1

# Move to the next row in the current direction

row += 1 if dir\_down else -1

return "".join(result)

cipher\_text = (

"AAIUJSIHBEKTEAOTEADESNUTFEAEMRTAHSARHROA"

"YHNFOAITTEEHCBOFVCATRNMNSNUTFERASHLWFHLN"

"HIUJSIHTKSOEHNIFISAEFNTIGRMRSOLSTISOKIEHPEOE"

)

num\_rails = 4

plain\_text = decrypt\_rail\_fence\_cipher(cipher\_text.replace(" ", ""), num\_rails)

print("Plain Text:", plain\_text)

