8/29/2024

Cryptology – b keerthana

Amruthesh

241059041

M.E - Cyber Security

MSIS, MAHE

1. **Write a Python program to reverse the content of the string.**

**Do not use built in functions.**

* def reverse\_string(s):

reversed\_str = ""

for i in range(len(s) - 1, -1, -1):

reversed\_str += s[i]

return reversed\_str

input\_str = "Hello World"

print(reverse\_string(input\_str))

1. **Create a program that performs basic string compression using the counts of repeated characters. For example, the string “aabcccccaaa” would become “a2b1c5a3”.**

* def compress\_string(s):

compressed\_str = ""

count = 1

for i in range(1, len(s)):

if s[i] == s[i - 1]:

count += 1

else:

compressed\_str += s[i - 1] + str(count)

count = 1

compressed\_str += s[-1] + str(count)

return compressed\_str

input\_str = "aabcccccaaa"

print(compress\_string(input\_str))

1. **Get the Caesar cipher from the user Decrypt the cipher**

* def caesar\_decrypt(cipher\_text, shift):

decrypted\_text = ""

for char in cipher\_text:

if char.isalpha():

shift\_base = 65 if char.isupper() else 97

decrypted\_text += chr((ord(char) - shift\_base - shift) % 26 + shift\_base)

else:

decrypted\_text += char

return decrypted\_text

cipher\_text = "Khoor Zruog"

shift = 3

print(caesar\_decrypt(cipher\_text, shift))

1. **Get the cipher encrypted using shift cipher. Identify the key used to encrypt using brute force. i.e, all the values in the key space**

* def brute\_force\_caesar(cipher\_text):

for shift in range(26):

print(f"Shift {shift}: {caesar\_decrypt(cipher\_text, shift)}")

cipher\_text = "Khoor Zruog"

brute\_force\_caesar(cipher\_text)

1. **Find the k value, Provided cipher text and plain text**

* def find\_caesar\_cipher\_key(plaintext, ciphertext):

key = (ord(ciphertext[0]) - ord(plaintext[0])) % 26

return key

plaintext = "HELLO"

ciphertext = "KHOOR"

key = find\_caesar\_cipher\_key(plaintext, ciphertext)

print("Caesar Cipher Key:", key)

1. **Encrypt and decrypt the string using Atbash cipher**

* def atbash\_cipher(text):

def transform(char):

if char.isalpha():

return chr(155 - ord(char)) if char.isupper() else chr(219 - ord(char))

return char

return ''.join(transform(char) for char in text)

plaintext = "HELLO"

ciphertext = atbash\_cipher(plaintext)

print("Atbash Encrypted Text:", ciphertext)

decrypted\_text = atbash\_cipher(ciphertext)

print("Atbash Decrypted Text:", decrypted\_text)

1. **Encrypt and decrypt using Affine cipher add validation**

* def affine\_encrypt(text, a, b):

def encrypt\_char(char):

if char.isalpha():

base = 65 if char.isupper() else 97

return chr((a \* (ord(char) - base) + b) % 26 + base)

return char

return ''.join(encrypt\_char(char) for char in text)

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

def mod\_inverse(x, mod):

for i in range(1, mod):

if (x \* i) % mod == 1:

return i

return None

inv\_a = mod\_inverse(a, 26)

if inv\_a is None:

raise ValueError("Invalid 'a' value; no modular inverse exists.")

def decrypt\_char(char):

if char.isalpha():

base = 65 if char.isupper() else 97

return chr((inv\_a \* (ord(char) - base - b)) % 26 + base)

return char

return ''.join(decrypt\_char(char) for char in ciphertext)

plaintext = "HELLO"

a = 5

b = 8

if a % 2 == 0 or a % 13 == 0:

raise ValueError("Invalid 'a' value. 'a' must be coprime with 26.")

ciphertext = affine\_encrypt(plaintext, a, b)

print("Affine Encrypted Text:", ciphertext)

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

print("Affine Decrypted Text:", decrypted\_text)