Cryptography programss

1. Write a C program for Caesar cipher involves replacing each letter of the alphabet with the letter standing k places further down the alphabet, for k in the range 1 through 25.

#include <stdio.h>

int main() {

char c;

int k;

printf("Enter shift (1-25): ");

scanf("%d", &k);

getchar();

printf("Enter text: ");

while ((c = getchar()) != '\n') {

if (c >= 'a' && c <= 'z')

c = (c - 'a' + k) % 26 + 'a';

else if (c >= 'A' && c <= 'Z')

c = (c - 'A' + k) % 26 + 'A';

putchar(c);

}

    return 0;

}

2. Write a C program for monoalphabetic substitution cipher maps a plaintext alphabet to a ciphertext alphabet, so that each letter of the plaintext alphabet maps to a single unique letter of the ciphertext alphabet.

#include <stdio.h>

int main() {

char key[27] = "QWERTYUIOPASDFGHJKLZXCVBNM";

char text[100], c;

printf("Enter plaintext: ");

gets(text);

for (int i = 0; text[i] != '\0'; i++)

{

c = text[i];

if (c >= 'A' && c <= 'Z')

text[i] = key[c - 'A'];

else if (c >= 'a' && c <= 'z')

text[i] = key[c - 'a'] + 32;

}

printf("Ciphertext: %s\n", text);

return 0;

}

3. Write a C program for Playfair algorithm is based on the use of a 5 X 5 matrix of letters constructed using a keyword. Plaintext is encrypted two letters at a time using this matrix.

def create\_matrix(keyword):

matrix = []

seen = set()

for char in keyword.upper().replace("J", "I"):

if char not in seen and char.isalpha():

seen.add(char)

matrix.append(char)

for char in "ABCDEFGHIKLMNOPQRSTUVWXYZ":

if char not in seen:

seen.add(char)

matrix.append(char)

return [matrix[i:i + 5] for i in range(0, 25, 5)]

def encrypt(plaintext, keyword):

matrix = create\_matrix(keyword)

plaintext = plaintext.upper().replace("J", "I").replace(" ", "")

pairs = [plaintext[i:i + 2] for i in range(0, len(plaintext), 2)]

if len(pairs[-1]) == 1:

pairs[-1] += 'X'

encrypted = ""

for a, b in pairs:

row\_a, col\_a = divmod(matrix.index(a), 5)

row\_b, col\_b = divmod(matrix.index(b), 5)

if row\_a == row\_b:

encrypted += matrix[row\_a \* 5 + (col\_a + 1) % 5]

encrypted += matrix[row\_b \* 5 + (col\_b + 1) % 5]

elif col\_a == col\_b:

encrypted += matrix[((row\_a + 1) % 5) \* 5 + col\_a]

encrypted += matrix[((row\_b + 1) % 5) \* 5 + col\_b]

else:

encrypted += matrix[row\_a \* 5 + col\_b]

encrypted += matrix[row\_b \* 5 + col\_a]

return encrypted

# Example usage

keyword = "KEYWORD"

plaintext = "HELLO"

print(encrypt(plaintext, keyword))

6. Write a C program for ciphertext has been generated with an affine cipher. The most frequent letter of the ciphertext is “B,” and the second most frequent letter of the ciphertext is “U.”Break this code.

#include <stdio.h>

int mod\_inverse(int a) {

for (int i = 1; i < 26; i++)

if ((a \* i) % 26 == 1) return i;

return -1;

}

char affine\_decrypt(char c, int a, int b) {

int a\_inv = mod\_inverse(a);

return (a\_inv \* ((c - 'A' - b + 26) % 26)) % 26 + 'A';

}

int main() {

char ciphertext[] = "BUBB"; // Example cipher text

int a = 7, b = 1;

printf("Decrypted: ");

for (int i = 0; ciphertext[i]; i++)

printf("%c", affine\_decrypt(ciphertext[i], a, b));

printf("\n");

return 0;

}

7. Write a C program for the following ciphertext was generated using a simple substitution algorithm. 53‡‡†305))6\*;4826)4‡.)4‡);806\*;48†8¶60))85;;]8\*;:‡\*8†83 (88)5\*†;46(;88\*96\*?;8)\*‡(;485);5\*†2:\*‡(;4956\*2(5\*—4)8¶8\* ;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†528806\*81 (‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?; Decrypt this message. 1. As you know, the most frequently occurring letter in English is e. Therefore, the first or second (or perhaps third?) most common character in the message is likely to stand for e. Also, e is often seen in pairs (e.g., meet, fleet, speed, seen, been, agree, etc.). Try to find a character in the ciphertext that decodes to e. 2. The most common word in English is “the.” Use this fact to guess the characters that stand for t and h. 3. Decipher the rest of the message by deducing additional words.

#include <stdio.h>

#include <string.h>

void analyze\_frequency(char \*ciphertext, int freq[256]) {

for (int i = 0; ciphertext[i]; i++)

freq[(unsigned char)ciphertext[i]]++;

}

void decrypt(char \*ciphertext, char \*mapping) {

for (int i = 0; ciphertext[i]; i++)

printf("%c", mapping[(unsigned char)ciphertext[i]]);

}

int main() {

char ciphertext[] = "53‡‡†305))6\*;4826)4‡.)4‡);806\*;...";

int freq[256] = {0};

analyze\_frequency(ciphertext, freq);

printf("Most Frequent Characters:\n");

for (int i = 0; i < 256; i++)

if (freq[i] > 0) printf("%c: %d\n", i, freq[i]);

char mapping[256] = {0};

mapping['‡'] = 'E';

mapping['8'] = 'T';

mapping['4'] = 'H';

mapping[')'] = 'O'; // Example mappings

printf("Decrypted Text:\n");

decrypt(ciphertext, mapping);

printf("\n");

return 0;

}

DIGITAL SIGNATURE

p = 10

g = 19

x = 16

y = (g \*\* x) % p

msg = 14

k = 5

r = (g \*\* k) % p

s = (msg - x \* r) % (p - 1)

v1 = (g \*\* msg) % p

v2 = (y \*\* r \* r \*\* s) % p

print("Message:", msg)

print("Signature: (r =", r, ", s =", s, ")")

print("Verification:", "Valid" if v1 == v2 else "Invalid")

DIFFIE HELMANN

# Small Diffie-Hellman demo in Python

p = 23

g = 5

a = 6

b = 15

A = (g \*\* a) % p

B = (g \*\* b) % p

secret\_A = (B \*\* a) % p

secret\_B = (A \*\* b) % p

print("Alice's Secret:", secret\_A)

print("Bob's Secret:  ", secret\_B)

RSA ALGORITHM

# Small RSA demo in Python

p, q = 3, 11

n = p \* q # 33

phi = (p-1)\*(q-1) # 20

e, d = 3, 7 # public and private keys

msg = 5 # message to encrypt

cipher = (msg \*\* e) % n

decrypted = (cipher \*\* d) % n

print("Message:", msg)

print("Encrypted:", cipher)

print("Decrypted:", decrypted)

Write a C program for ECB mode, if there is an error in a block of the transmitted ciphertext, only the corresponding plaintext block is affected. However, in the CBC mode, this error propagates. For example, an error in the transmitted C1 obviously corrupts P1 and P2. a. Are any blocks beyond P2 affected? b. Suppose that there is a bit error in the source version of P1. Through how many ciphertext blocks is this error propagated? What is the effect at the receiver?

def xor(a, b):

return bytes([x ^ y for x, y in zip(a, b)])

# Simple "encryption": XOR with key

def encrypt\_block(block, key):

return xor(block, key)

# ECB Mode

def ecb\_encrypt(data, key):

return b''.join([encrypt\_block(data[i:i+2], key) for i in range(0, len(data), 2)])

def ecb\_decrypt(cipher, key):

return b''.join([encrypt\_block(cipher[i:i+2], key) for i in range(0, len(cipher), 2)])

# CBC Mode

def cbc\_encrypt(data, key, iv):

out = []

prev = iv

for i in range(0, len(data), 2):

block = xor(data[i:i+2], prev)

enc = encrypt\_block(block, key)

out.append(enc)

prev = enc

return b''.join(out)

def cbc\_decrypt(cipher, key, iv):

out = []

prev = iv

for i in range(0, len(cipher), 2):

dec = encrypt\_block(cipher[i:i+2], key)

out.append(xor(dec, prev))

prev = cipher[i:i+2]

return b''.join(out)

# ==== Demo ====

key = b'ab'

iv = b'cd'

msg = b'HELLO!' # 6 bytes = 3 blocks of 2 bytes

# ECB

ecb = ecb\_encrypt(msg, key)

ecb\_error = bytearray(ecb)

ecb\_error[2] ^= 1 # Simulate error in 2nd block

print("ECB Decrypted with Error:", ecb\_decrypt(bytes(ecb\_error), key))

# CBC

cbc = cbc\_encrypt(msg, key, iv)

cbc\_error = bytearray(cbc)

cbc\_error[0] ^= 1 # Error in first ciphertext block

print("CBC Decrypted with Error:", cbc\_decrypt(bytes(cbc\_error), key, iv))

SHA

import random

lanes = [0] \* 25 # 25 lanes = 1600 bits (each lane = 64 bits)

rate\_lanes = 1024 // 64 # 16 lanes

capacity\_lanes = lanes[16:] # 9 lanes (capacity)

block\_count = 0

# Initialize rate lanes with non-zero values

for i in range(rate\_lanes):

lanes[i] = random.getrandbits(64)

# Loop until all capacity lanes become non-zero

while any(x == 0 for x in lanes[16:]):

block\_count += 1

# New message block: random 16 lanes

new\_block = [random.getrandbits(64) for \_ in range(16)]

# XOR message block into rate lanes

for i in range(16):

lanes[i] ^= new\_block[i]

# Mix into capacity lanes (just randomly flip some bits)

for i in range(16, 25):

lanes[i] ^= random.getrandbits(64)

print("Blocks needed until all capacity lanes are non-zero:", block\_count)

ONE PAD TIME

import random

def encrypt(plaintext, key):

return ''.join(chr((ord(c) - 65 + k) % 26 + 65) for c, k in zip(plaintext, key))

def decrypt(ciphertext, key):

return ''.join(chr((ord(c) - 65 - k) % 26 + 65) for c, k in zip(ciphertext, key))

plaintext = "HELLO"

key = [random.randint(0, 26) for \_ in plaintext]

cipher = encrypt(plaintext, key)

original = decrypt(cipher, key)

print("Plaintext :", plaintext)

print("Key :", key)

print("Encrypted :", cipher)

print("Decrypted :", original)