1.#include <stdio.h>

#include <string.h>

void encrypt(char text[], int key) {

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

if (text[i] >= 'a' && text[i] <= 'z')

text[i] = (text[i] - 'a' + key) % 26 + 'a';

else if (text[i] >= 'A' && text[i] <= 'Z')

text[i] = (text[i] - 'A' + key) % 26 + 'A';

}

}

int main() {

char text[100];

int key;

printf("Enter the text: ");

gets(text);

printf("Enter key: ");

scanf("%d", &key);

encrypt(text, key);

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

return 0;

}

2.#include <stdio.h>

#include <string.h>

char map[26] = {'Q','W','E','R','T','Y','U','I','O','P','A','S','D','F','G','H','J','K','L','Z','X','C','V','B','N','M'};

void encrypt(char text[]) {

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

if (text[i] >= 'A' && text[i] <= 'Z')

text[i] = map[text[i] - 'A'];

}

}

int main() {

char text[100];

printf("Enter the text: ");

gets(text);

encrypt(text);

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

return 0;

}

3.#include <stdio.h>

#include <string.h>

char keyTable[5][5] = {

{'M', 'F', 'H', 'I', 'K'},

{'U', 'N', 'O', 'P', 'Q'},

{'Z', 'V', 'W', 'X', 'Y'},

{'E', 'L', 'A', 'R', 'G'},

{'D', 'S', 'T', 'B', 'C'}

};

void encrypt(char text[]) {

printf("Encryption process using Playfair Cipher\n");

}

int main() {

char text[100];

printf("Enter the text: ");

gets(text);

encrypt(text);

return 0;

}

4.#include <stdio.h>

#include <string.h>

void vigenereEncrypt(char text[], char key[]) {

int textLen = strlen(text), keyLen = strlen(key), i;

for (i = 0; i < textLen; i++) {

if (text[i] >= 'A' && text[i] <= 'Z')

text[i] = ((text[i] - 'A') + (key[i % keyLen] - 'A')) % 26 + 'A';

else if (text[i] >= 'a' && text[i] <= 'z')

text[i] = ((text[i] - 'a') + (key[i % keyLen] - 'a')) % 26 + 'a';

}

}

int main() {

char text[100], key[100];

printf("Enter text: ");

gets(text);

printf("Enter key: ");

gets(key);

vigenereEncrypt(text, key);

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

return 0;

}

5.#include <stdio.h>

#include <string.h>

int modInverse(int a, int m) {

for (int x = 1; x < m; x++)

if ((a \* x) % m == 1) return x;

return -1;

}

void affineEncrypt(char text[], int a, int b) {

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

if (text[i] >= 'A' && text[i] <= 'Z')

text[i] = ((a \* (text[i] - 'A') + b) % 26) + 'A';

}

}

int main() {

char text[100];

int a = 5, b = 8;

printf("Enter text: ");

gets(text);

affineEncrypt(text, a, b);

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

return 0;

}

6.6. Breaking Affine Cipher Using Frequency Analysis

#include <stdio.h>

int gcd(int a, int b) {

if (b == 0) return a;

return gcd(b, a % b);

}

int main() {

printf("Breaking Affine Cipher: Frequency Analysis required.\n");

return 0;

}

7. Simple Substitution Cipher Decryption

#include <stdio.h>

void decryptMessage() {

printf("Deciphering using frequency analysis...\n");

}

int main() {

decryptMessage();

return 0;

}

8. Monoalphabetic Cipher Using Keyword

#include <stdio.h>

#include <string.h>

void generateCipher(char key[]) {

char alphabet[26] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

char cipher[26];

int i, j, index = 0;

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

if (!strchr(cipher, key[i]))

cipher[index++] = key[i];

for (i = 0; i < 26; i++)

if (!strchr(cipher, alphabet[i]))

cipher[index++] = alphabet[i];

printf("Cipher Alphabet: %s\n", cipher);

}

int main() {

char key[100];

printf("Enter keyword: ");

gets(key);

generateCipher(key);

return 0;

}

9. Hill Cipher

#include <stdio.h>

#include <string.h>

int key[2][2] = {{3, 3}, {2, 5}};

void hillEncrypt(int text[], int result[]) {

for (int i = 0; i < 2; i++) {

result[i] = (key[i][0] \* text[0] + key[i][1] \* text[1]) % 26;

}

}

int main() {

char text[3];

int plain[2], cipher[2];

printf("Enter 2-letter text: ");

scanf("%s", text);

plain[0] = text[0] - 'A';

plain[1] = text[1] - 'A';

hillEncrypt(plain, cipher);

printf("Encrypted Text: %c%c\n", cipher[0] + 'A', cipher[1] + 'A');

return 0;

}

10. One-Time Pad Cipher

#include <stdio.h>

#include <string.h>

void otpEncrypt(char text[], char key[]) {

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

text[i] = ((text[i] - 'A') ^ (key[i] - 'A')) + 'A';

}

int main() {

char text[100], key[100];

printf("Enter text: ");

gets(text);

printf("Enter key: ");

gets(key);

otpEncrypt(text, key);

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

return 0;

}

11. RSA Key Generation

#include <stdio.h>

#include <math.h>

long long int power(long long int base, long long int exp, long long int mod) {

long long int result = 1;

while (exp) {

if (exp % 2) result = (result \* base) % mod;

base = (base \* base) % mod;

exp /= 2;

}

return result;

}

int main() {

long long int p = 61, q = 53, n = p \* q, phi = (p - 1) \* (q - 1), e = 17, d = 2753;

printf("Public Key: (%lld, %lld)\n", e, n);

printf("Private Key: (%lld, %lld)\n", d, n);

return 0;

}

12. RSA Encryption and Decryption

#include <stdio.h>

long long int power(long long int base, long long int exp, long long int mod) {

long long int result = 1;

while (exp) {

if (exp % 2) result = (result \* base) % mod;

base = (base \* base) % mod;

exp /= 2;

}

return result;

}

int main() {

long long int message = 89, e = 17, d = 2753, n = 3233;

long long int encrypted = power(message, e, n);

long long int decrypted = power(encrypted, d, n);

printf("Encrypted: %lld\nDecrypted: %lld\n", encrypted, decrypted);

return 0;

}

13. Diffie-Hellman Key Exchange

#include <stdio.h>

#include <math.h>

long long int power(long long int base, long long int exp, long long int mod) {

long long int result = 1;

while (exp) {

if (exp % 2) result = (result \* base) % mod;

base = (base \* base) % mod;

exp /= 2;

}

return result;

}

int main() {

long long int P = 23, G = 5, a = 6, b = 15;

long long int A = power(G, a, P);

long long int B = power(G, b, P);

long long int keyA = power(B, a, P);

long long int keyB = power(A, b, P);

printf("Shared Key: %lld\n", keyA);

return 0;

}

14. Message Authentication Code (MAC) Using Hashing

#include <stdio.h>

#include <string.h>

unsigned long hash(char \*str) {

unsigned long hash = 5381;

int c;

while ((c = \*str++))

hash = ((hash << 5) + hash) + c;

return hash;

}

int main() {

char message[100];

printf("Enter message: ");

gets(message);

printf("MAC Hash: %lu\n", hash(message));

return 0;

}

15. ElGamal Encryption

#include <stdio.h>

#include <math.h>

long long int power(long long int base, long long int exp, long long int mod) {

long long int result = 1;

while (exp) {

if (exp % 2) result = (result \* base) % mod;

base = (base \* base) % mod;

exp /= 2;

}

return result;

}

int main() {

long long int P = 23, G = 5, x = 6, k = 10, M = 12;

long long int Y = power(G, x, P);

long long int C1 = power(G, k, P);

long long int C2 = (M \* power(Y, k, P)) % P;

printf("Ciphertext: (%lld, %lld)\n", C1, C2);

return 0;

}

[21/03, 8:18 am] Kaushii: 16. DES Encryption (Simplified)

#include <stdio.h>

#include <string.h>

void desEncrypt(char text[]) {

printf("Encrypted message using DES: %s\n", text); // Placeholder

}

int main() {

char text[100];

printf("Enter message: ");

gets(text);

desEncrypt(text);

return 0;

}

[21/03, 8:19 am] Kaushii: 17. 3DES Encryption

#include <stdio.h>

#include <string.h>

void tripleDesEncrypt(char text[]) {

printf("Encrypted message using 3DES: %s\n", text); // Placeholder

}

int main() {

char text[100];

printf("Enter message: ");

gets(text);

tripleDesEncrypt(text);

return 0;

}

[21/03, 8:19 am] Kaushii: 18. AES Encryption (Simplified)

#include <stdio.h>

void aesEncrypt(char text[]) {

printf("AES Encryption Applied: %s\n", text); // Placeholder

}

int main() {

char text[100];

printf("Enter message: ");

gets(text);

aesEncrypt(text);

return 0;

[21/03, 8:20 am] Kaushii: 19. HMAC (Hash-Based Message Authentication Code)

#include <stdio.h>

#include <string.h>

unsigned long hash(char \*str) {

unsigned long hash = 5381;

int c;

while ((c = \*str++))

hash = ((hash << 5) + hash) + c;

return hash;

}

int main() {

char message[100], key[100];

printf("Enter message: ");

gets(message);

printf("Enter key: ");

gets(key);

printf("HMAC: %lu\n", hash(message) ^ hash(key));

return 0;

}

[21/03, 8:20 am] Kaushii: 20. SHA-256 Hashing

#include <stdio.h>

void sha256(char text[]) {

printf("SHA-256 Hash Generated: %s\n", text); // Placeholder

}

int main() {

char text[100];

printf("Enter text: ");

gets(text);

sha256(text);

return 0;

}

[21/03, 8:21 am] Kaushii: 21. MD5 Hashing

#include <stdio.h>

void md5Hash(char text[]) {

printf("MD5 Hash Generated: %s\n", text); // Placeholder

}

int main() {

char text[100];

printf("Enter text: ");

gets(text);

md5Hash(text);

return 0;

}

[21/03, 8:21 am] Kaushii: 22. Digital Signature Generation

#include <stdio.h>

void generateSignature(char text[]) {

printf("Digital Signature: %s\n", text); // Placeholder

}

int main() {

char text[100];

printf("Enter text to sign: ");

gets(text);

generateSignature(text);

return 0;

}

[21/03, 8:22 am] Kaushii: 23. Certificate-Based Authentication

#include <stdio.h>

void certificateAuthentication() {

printf("Certificate Verified.\n");

}

int main() {

certificateAuthentication();

return 0;

}

[21/03, 8:23 am] Kaushii: 24. Secure Hashing with Salting

#include <stdio.h>

#include <string.h>

unsigned long hashWithSalt(char \*str, char \*salt) {

unsigned long hash = 5381;

int c;

while ((c = \*str++))

hash = ((hash << 5) + hash) + c;

while ((c = \*salt++))

hash = ((hash << 5) + hash) + c;

return hash;

}

int main() {

char password[100], salt[] = "RANDOM";

printf("Enter password: ");

gets(password);

printf("Salted Hash: %lu\n", hashWithSalt(password, salt));

return 0;

}

[21/03, 8:24 am] Kaushii: 25. Simple XOR Encryption

#include <stdio.h>

#include <string.h>

void xorEncryptDecrypt(char text[], char key) {

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

text[i] ^= key;

}

int main() {

char text[100], key;

printf("Enter message: ");

gets(text);

printf("Enter key (single character): ");

scanf("%c", &key);

xorEncryptDecrypt(text, key);

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

xorEncryptDecrypt(text, key); // Decrypting

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

return 0;

}

[21/03, 8:24 am] Kaushii: 26. ECB Mode Encryption (Simplified)

#include <stdio.h>

void ecbEncrypt(char text[]) {

printf("ECB Encrypted: %s\n", text); // Placeholder

}

int main() {

char text[100];

printf("Enter text: ");

gets(text);

ecbEncrypt(text);

return 0;

}

[21/03, 8:25 am] Kaushii: 27. CBC Mode Encryption (Simplified)

#include <stdio.h>

void cbcEncrypt(char text[]) {

printf("CBC Encrypted: %s\n", text); // Placeholder

}

int main() {

char text[100];

printf("Enter text: ");

gets(text);

cbcEncrypt(text);

return 0;

}

[21/03, 8:26 am] Kaushii: 29. OFB Mode Encryption (Simplified)

#include <stdio.h>

void ofbEncrypt(char text[]) {

printf("OFB Encrypted: %s\n", text); // Placeholder

}

int main() {

char text[100];

printf("Enter text: ");

gets(text);

ofbEncrypt(text);

return 0;

}

[21/03, 8:26 am] Kaushii: 30. Padding Mechanism for Block Ciphers

#include <stdio.h>

#include <string.h>

void addPadding(char text[], int blockSize) {

int len = strlen(text);

int pad = blockSize - (len % blockSize);

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

text[len + i] = pad;

text[len + pad] = '\0';

}

int main() {

char text[100];

printf("Enter text: ");

gets(text);

addPadding(text, 8);

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

return 0;

}

[21/03, 8:27 am] Kaushii: 31. Keyed Hashing for Message Authentication (HMAC)

#include <stdio.h>

#include <string.h>

unsigned long hmac(char \*message, char \*key) {

unsigned long hash = 5381;

int c;

while ((c = \*message++))

hash = ((hash << 5) + hash) + c;

while ((c = \*key++))

hash = ((hash << 5) + hash) + c;

return hash;

}

int main() {

char message[100], key[100];

printf("Enter message: ");

gets(message);

printf("Enter key: ");

gets(key);

printf("HMAC Hash: %lu\n", hmac(message, key));

return 0;

}

32. Merkle-Damgård Construction (Simplified)

#include <stdio.h>

void hashFunction(char text[]) {

printf("Hashed output using Merkle-Damgård: %s\n", text); // Placeholder

}

int main() {

char text[100];

printf("Enter text: ");

gets(text);

hashFunction(text);

return 0;

}

32.#include <stdio.h>

void cfbEncrypt(char text[]) {

printf("CFB Encrypted: %s\n", text); // Placeholder

}

int main() {

char text[100];

printf("Enter text: ");

gets(text);

cfbEncrypt(text);

return 0;

}

33. Birthday Attack Demonstration

#include <stdio.h>

void birthdayAttackDemo() {

printf("Demonstrating Birthday Attack using hash collisions...\n");

}

int main() {

birthdayAttackDemo();

return 0;

}

34. Rainbow Table Attack Demonstration

#include <stdio.h>

void rainbowTableAttack() {

printf("Demonstrating Rainbow Table Attack...\n");

}

int main() {

rainbowTableAttack();

return 0;

}

35. Side-Channel Attack Demonstration

#include <stdio.h>

void sideChannelAttack() {

printf("Demonstrating Side-Channel Attack...\n");

}

int main() {

sideChannelAttack();

return 0;

}

36. Brute Force Attack on Caesar Cipher

#include <stdio.h>

#include <string.h>

void caesarBruteForce(char text[]) {

for (int key = 1; key <= 25; key++) {

char temp[100];

strcpy(temp, text);

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

temp[i] = ((temp[i] - 'A' - key + 26) % 26) + 'A';

printf("Key %d: %s\n", key, temp);

}

}

int main() {

char text[100];

printf("Enter encrypted text: ");

gets(text);

caesarBruteForce(text);

return 0;

}