**Cryptography codes**

**1.CAESAR CIPHER**

**#include <stdio.h>**

**#include <string.h>**

**#include <ctype.h>**

**void encrypt(char input[], int key) {**

**for (int i = 0; i < strlen(input); i++) {**

**if (isalpha(input[i])) {**

**char base = islower(input[i]) ? 'a' : 'A';**

**input[i] = (input[i] - base + key) % 26 + base;**

**}**

**}**

**}**

**void decrypt(char input[], int key) {**

**encrypt(input, 26 - key);**

**}**

**int main() {**

**int key;**

**char input[1000];**

**printf("Enter the value of key: ");**

**scanf("%d", &key);**

**printf("Enter input string: ");**

**scanf("%s", input);**

**// Encryption**

**printf("Encryption is: ");**

**encrypt(input, key);**

**printf("%s\n", input);**

**// Decryption**

**printf("Decryption is: ");**

**decrypt(input, key);**

**printf("%s\n", input);**

**return 0;**

**}**

**2.MONOALPHABETIC CIPHER**

**#include <stdio.h>**

**#include <string.h>**

**#include <ctype.h>**

**void generateKey(char key[]) {**

**// You can customize the key to your liking**

**strcpy(key, "ZYXWVUTSRQPONMLKJIHGFEDCBA");**

**}**

**void encrypt(char input[], char key[]) {**

**for (int i = 0; i < strlen(input); i++) {**

**if (isalpha(input[i])) {**

**char base = islower(input[i]) ? 'a' : 'A';**

**input[i] = key[input[i] - base];**

**}**

**}**

**}**

**void decrypt(char input[], char key[]) {**

**char reverseKey[26];**

**for (int i = 0; i < 26; i++) {**

**reverseKey[key[i] - 'A'] = 'A' + i;**

**}**

**for (int i = 0; i < strlen(input); i++) {**

**if (isalpha(input[i])) {**

**char base = islower(input[i]) ? 'a' : 'A';**

**input[i] = reverseKey[input[i] - base];**

**}**

**}**

**}**

**int main() {**

**char key[26];**

**char input[1000];**

**generateKey(key);**

**printf("Enter input string: ");**

**scanf(" %[^\n]s", input);**

**// Encryption**

**printf("Encryption is: ");**

**encrypt(input, key);**

**printf("%s\n", input);**

**// Decryption**

**printf("Decryption is: ");**

**decrypt(input, key);**

**printf("%s\n", input);**

**return 0;**

**}**

**3.PLAYFAIR CIPHER**

**4.POLYALPHABETIC SUBSTITUTION CIPHER**

**#include <stdio.h>**

**#include <conio.h>**

**#include <string.h>**

**int main() {**

**char pt[100] = {'\0'}, ct[100] = {'\0'}, key[100] = {'\0'}, rt[100] = {'\0'};**

**int i, j;**

**printf("\nEnter the plain text:\n");**

**fgets(pt, sizeof(pt), stdin);**

**pt[strcspn(pt, "\n")] = '\0'; // Remove the newline character**

**printf("\nEnter the key:\n");**

**fgets(key, sizeof(key), stdin);**

**key[strcspn(key, "\n")] = '\0'; // Remove the newline character**

**j = 0;**

**for (i = strlen(key); i < strlen(pt); i++) {**

**if (j == strlen(key)) {**

**j = 0;**

**}**

**key[i] = key[j];**

**j++;**

**}**

**printf("\nNew key is: %s\n", key);**

**for (i = 0; i < strlen(pt); i++) {**

**if (pt[i] == ' ') {**

**ct[i] = ' '; // Preserve space**

**} else {**

**ct[i] = (((pt[i] - 97) + (key[i] - 97)) % 26) + 97;**

**}**

**}**

**printf("\nCipher text is: %s\n", ct);**

**for (i = 0; i < strlen(ct); i++) {**

**if (ct[i] == ' ') {**

**rt[i] = ' '; // Preserve space**

**} else if (ct[i] < key[i]) {**

**rt[i] = 26 + ((ct[i] - 97) - (key[i] - 97)) + 97;**

**} else {**

**rt[i] = (((ct[i] - 97) - (key[i] - 97)) % 26) + 97;**

**}**

**}**

**printf("\nPlain text is: %s\n", rt);**

**getch();**

**return 0;**

**}**

**5.MONOALPHABETIC\_SENDER\_RECEIVER**

**#include <stdio.h>**

**#include <string.h>**

**#include <ctype.h>**

**void generateCipherKey(char keyword[], char cipherKey[]) {**

**int i, j, k;**

**char temp[26];**

**int keywordLength = strlen(keyword);**

**// Initialize the temporary array**

**for (i = 0, k = 0; i < 26; ++i) {**

**temp[i] = 'A' + i;**

**}**

**// Place the keyword in the cipher key**

**for (i = 0; i < keywordLength; ++i) {**

**cipherKey[i] = toupper(keyword[i]);**

**temp[cipherKey[i] - 'A'] = 0; // Mark the used letters as 0**

**}**

**// Fill the remaining letters in the cipher key**

**for (i = 0, j = keywordLength; i < 26; ++i) {**

**if (temp[i] != 0) {**

**cipherKey[j++] = temp[i];**

**}**

**}**

**cipherKey[j] = '\0'; // Null-terminate the cipher key**

**}**

**6.FREQUENCY ATTACK MONOALPHABETIC**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**// Function to calculate the frequency of each letter in the ciphertext**

**void calculateFrequency(char \*ciphertext, int \*frequency) {**

**int i = 0;**

**while (ciphertext[i] != '\0') {**

**if (ciphertext[i] >= 'A' && ciphertext[i] <= 'Z') {**

**frequency[ciphertext[i] - 'A']++;**

**}**

**i++;**

**}**

**}**

**// Function to perform a basic letter frequency attack**

**void letterFrequencyAttack(char \*ciphertext, int topN) {**

**int frequency[26] = {0}; // Initialize frequency array for 26 letters**

**calculateFrequency(ciphertext, frequency);**

**// Create a copy of the frequency array to sort it later**

**int sortedFrequency[26];**

**for (int i = 0; i < 26; i++) {**

**sortedFrequency[i] = frequency[i];**

**}**

**// Sort the frequency array in descending order**

**for (int i = 0; i < 26 - 1; i++) {**

**for (int j = 0; j < 26 - i - 1; j++) {**

**if (sortedFrequency[j] < sortedFrequency[j + 1]) {**

**int temp = sortedFrequency[j];**

**sortedFrequency[j] = sortedFrequency[j + 1];**

**sortedFrequency[j + 1] = temp;**

**}**

**}**

**}**

**// Guess the most likely substitutions based on frequency analysis**

**printf("Top %d possible plaintexts:\n", topN);**

**for (int n = 0; n < topN; n++) {**

**printf("%d. ", n + 1);**

**for (int i = 0; i < 26; i++) {**

**if (frequency[i] == sortedFrequency[n]) {**

**printf("%c", 'A' + i);**

**}**

**}**

**printf(": ");**

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

**if (ciphertext[i] >= 'A' && ciphertext[i] <= 'Z') {**

**printf("%c", 'A' + (ciphertext[i] - 'A' + 26 - sortedFrequency[n]) % 26);**

**} else {**

**printf(" ");**

**}**

**}**

**printf("\n");**

**}**

**}**

**int main() {**

**char ciphertext[] = "CIPHER TEXT GOES HERE"; // Replace with your ciphertext**

**int topN = 10; // Number of top possible plaintexts to show**

**letterFrequencyAttack(ciphertext, topN);**

**return 0;**

**}**

**void encryptMonoalphabetic(char plaintext[], char cipherKey[], char ciphertext[]) {**

**int i;**

**for (i = 0; plaintext[i] != '\0'; ++i) {**

**if (isalpha(plaintext[i])) {**

**char original = toupper(plaintext[i]);**

**if (original >= 'A' && original <= 'Z') {**

**ciphertext[i] = cipherKey[original - 'A'];**

**} else {**

**ciphertext[i] = tolower(cipherKey[original - 'a']);**

**}**

**} else {**

**ciphertext[i] = plaintext[i]; // Non-alphabetic characters remain unchanged**

**}**

**}**

**ciphertext[i] = '\0'; // Null-terminate the ciphertext**

**}**

**int main() {**

**char keyword[26];**

**char cipherKey[26];**

**char plaintext[100];**

**char ciphertext[100];**

**// Input the keyword**

**printf("Enter the keyword: ");**

**scanf("%s", keyword);**

**// Check if the keyword is valid (contains only alphabetic characters)**

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

**if (!isalpha(keyword[i])) {**

**printf("Invalid keyword. Please enter only alphabetic characters.\n");**

**return 1;**

**}**

**}**

**// Input the cipher key**

**printf("Enter the cipher key: ");**

**scanf("%s", cipherKey);**

**// Check if the cipher key is valid (contains only alphabetic characters and has a length of 26)**

**if (strlen(cipherKey) != 26) {**

**printf("Invalid cipher key. Please enter a key with exactly 26 characters.\n");**

**return 1;**

**}**

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

**if (!isalpha(cipherKey[i])) {**

**printf("Invalid cipher key. Please enter only alphabetic characters.\n");**

**return 1;**

**}**

**}**

**// Encrypt the plaintext**

**generateCipherKey(keyword, cipherKey);**

**printf("Enter the plaintext: ");**

**scanf(" %[^\n]s", plaintext);**

**encryptMonoalphabetic(plaintext, cipherKey, ciphertext);**

**// Display the results**

**printf("\nGenerated Cipher Key: %s\n", cipherKey);**

**printf("Encrypted Ciphertext: %s\n", ciphertext);**

**return 0;**

**}**

**8.ADDITIVE CIPHER FREQUENCY ATTACK**

**#include <stdio.h>**

**#include <string.h>**

**#include <ctype.h>**

**#define MAX\_LENGTH 100**

**void encrypt(char plaintext[], int key) {**

**int length = strlen(plaintext);**

**for (int i = 0; i < length; i++) {**

**if (isalpha(plaintext[i])) {**

**char base = islower(plaintext[i]) ? 'a' : 'A';**

**plaintext[i] = (plaintext[i] - base + key) % 26 + base;**

**}**

**}**

**}**

**void decrypt(char ciphertext[], int key) {**

**int length = strlen(ciphertext);**

**for (int i = 0; i < length; i++) {**

**if (isalpha(ciphertext[i])) {**

**char base = islower(ciphertext[i]) ? 'a' : 'A';**

**ciphertext[i] = (ciphertext[i] - base - key + 26) % 26 + base;**

**}**

**}**

**}**

**int main() {**

**char text[MAX\_LENGTH];**

**int key;**

**printf("Enter the text: ");**

**fgets(text, sizeof(text), stdin);**

**printf("Enter the key (0-25): ");**

**scanf("%d", &key);**

**encrypt(text, key);**

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

**decrypt(text, key);**

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

**return 0;**

**}**

**9.AFFINE CAESAR CIPHER**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <ctype.h>**

**#define ALPHABET\_SIZE 26**

**// Function to calculate the modulo inverse of a number a with respect to m**

**int modInverse(int a, int m) {**

**a = a % m;**

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

**if ((a \* x) % m == 1) {**

**return x;**

**}**

**}**

**return -1; // Inverse doesn't exist**

**}**

**// Function to encrypt a plaintext character using the affine Caesar cipher**

**char encryptCharacter(int a, int b, char p) {**

**if (isalpha(p)) {**

**int base = islower(p) ? 'a' : 'A';**

**return (a \* (p - base) + b) % ALPHABET\_SIZE + base;**

**}**

**return p;**

**}**

**// Function to decrypt a ciphertext character using the affine Caesar cipher**

**char decryptCharacter(int a, int b, char c) {**

**if (isalpha(c)) {**

**int base = islower(c) ? 'a' : 'A';**

**int modInvA = modInverse(a, ALPHABET\_SIZE);**

**if (modInvA == -1) {**

**printf("Inverse doesn't exist. Decryption is not possible.\n");**

**exit(1);**

**}**

**return (modInvA \* (c - base - b + ALPHABET\_SIZE)) % ALPHABET\_SIZE + base;**

**}**

**return c;**

**}**

**// Function to perform the affine Caesar cipher encryption**

**void encryptAffineCaesar(int a, int b, const char \*plaintext, char \*ciphertext) {**

**int i;**

**for (i = 0; plaintext[i] != '\0'; i++) {**

**ciphertext[i] = encryptCharacter(a, b, plaintext[i]);**

**}**

**ciphertext[i] = '\0'; // Terminate the string**

**}**

**// Function to perform the affine Caesar cipher decryption**

**void decryptAffineCaesar(int a, int b, const char \*ciphertext, char \*plaintext) {**

**int i;**

**for (i = 0; ciphertext[i] != '\0'; i++) {**

**plaintext[i] = decryptCharacter(a, b, ciphertext[i]);**

**}**

**plaintext[i] = '\0'; // Terminate the string**

**}**

**int main() {**

**int a, b;**

**printf("Enter the value of 'a' (should be coprime with %d): ", ALPHABET\_SIZE);**

**scanf("%d", &a);**

**printf("Enter the value of 'b': ");**

**scanf("%d", &b);**

**char plaintext[100];**

**char ciphertext[100];**

**char decryptedText[100];**

**printf("Enter the plaintext: ");**

**scanf(" %[^\n]", plaintext);**

**encryptAffineCaesar(a, b, plaintext, ciphertext);**

**printf("Encrypted: %s\n", ciphertext);**

**decryptAffineCaesar(a, b, ciphertext, decryptedText);**

**printf("Decrypted: %s\n", decryptedText);**

**return 0;**

**}**

**10.VIGNERE CIPHER**

**#include <stdio.h>**

**#include <conio.h>**

**#include <ctype.h>**

**#include <string.h>**

**#include <stdlib.h>**

**void encipher();**

**void decipher();**

**int main() {**

**int choice;**

**while (1) {**

**printf("\n1. Encrypt Text");**

**printf("\t2. Decrypt Text");**

**printf("\t3. Exit");**

**printf("\n\nEnter Your Choice : ");**

**scanf("%d", &choice);**

**if (choice == 3) {**

**printf("Exiting the program.\n");**

**exit(0);**

**} else if (choice == 1)**

**encipher();**

**else if (choice == 2)**

**decipher();**

**else**

**printf("Please Enter Valid Option.");**

**}**

**}**

**void encipher() {**

**char input[100], key[10];**

**printf("\n\nEnter Plain Text:\n");**

**getchar(); // Clear newline character from the buffer**

**fgets(input, sizeof(input), stdin);**

**printf("\nEnter Key Value: ");**

**scanf("%s", key);**

**printf("\nResultant Cipher Text: ");**

**for (int i = 0, j = 0; i < strlen(input); i++) {**

**if (isalpha(input[i])) {**

**int keyIndex = j % strlen(key);**

**char currentKey = toupper(key[keyIndex]) - 'A';**

**char currentChar = toupper(input[i]) - 'A';**

**char encryptedChar = ((currentChar + currentKey) % 26) + 'A';**

**printf("%c", encryptedChar);**

**if (isalpha(input[i]))**

**j++;**

**} else {**

**printf("%c", input[i]);**

**}**

**}**

**}**

**void decipher() {**

**char input[100], key[10];**

**printf("\n\nEnter Cipher Text:\n");**

**getchar(); // Clear newline character from the buffer**

**fgets(input, sizeof(input), stdin);**

**printf("\n\nEnter the key value: ");**

**scanf("%s", key);**

**for (int i = 0, j = 0; i < strlen(input); i++) {**

**if (isalpha(input[i])) {**

**int keyIndex = j % strlen(key);**

**char currentKey = toupper(key[keyIndex]) - 'A';**

**char currentChar = toupper(input[i]) - 'A';**

**char decryptedChar = ((currentChar - currentKey + 26) % 26) + 'A';**

**printf("%c", decryptedChar);**

**if (isalpha(input[i]))**

**j++;**

**} else {**

**printf("%c", input[i]);**

**}**

**}**

**}**

**11.CBC MAC**

**#include <stdio.h>**

**#include <string.h>**

**// Example key (16 bytes)**

**unsigned char key[16] = {**

**0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,**

**0xab, 0xf7, 0x97, 0x46, 0x19, 0xa4, 0x62, 0x55**

**};**

**// XOR operation for two blocks of data**

**void xorBlocks(unsigned char \*dest, const unsigned char \*src1, const unsigned char \*src2, int blockSize) {**

**for (int i = 0; i < blockSize; i++) {**

**dest[i] = src1[i] ^ src2[i];**

**}**

**}**

**// AES encryption (dummy function)**

**void aesEncrypt(unsigned char \*output, const unsigned char \*input, const unsigned char \*key, int blockSize) {**

**// This is where actual AES encryption should be performed.**

**// For simplicity, this function is a dummy and does nothing.**

**memcpy(output, input, blockSize);**

**}**

**// CBC-MAC calculation**

**void cbcMac(unsigned char \*mac, const unsigned char \*message, int blockSize) {**

**unsigned char iv[blockSize]; // Initialization vector**

**unsigned char tmp[blockSize]; // Temporary storage for XOR result**

**memset(iv, 0, blockSize); // Initialize IV to zeros**

**// Encrypt the message using CBC mode with a dummy encryption function (AES)**

**for (int i = 0; i < blockSize; i++) {**

**tmp[i] = message[i] ^ iv[i];**

**}**

**aesEncrypt(mac, tmp, key, blockSize);**

**}**

**int main() {**

**int blockSize = 16; // Block size in bytes**

**unsigned char X[blockSize] = {**

**0x54, 0x68, 0x69, 0x73, 0x20, 0x69, 0x73, 0x20,**

**0x61, 0x20, 0x74, 0x65, 0x73, 0x74, 0x20, 0x58**

**}; // One-block message**

**unsigned char T[blockSize];**

**cbcMac(T, X, blockSize); // Calculate CBC-MAC for X**

**unsigned char X\_Xor\_T[blockSize \* 2];**

**xorBlocks(X\_Xor\_T, X, T, blockSize); // X ? T**

**unsigned char T\_2[blockSize];**

**cbcMac(T\_2, X\_Xor\_T, blockSize \* 2); // Calculate CBC-MAC for X || (X ? T)**

**printf("CBC-MAC for X: ");**

**for (int i = 0; i < blockSize; i++) {**

**printf("%02x ", T[i]);**

**}**

**printf("\n");**

**printf("CBC-MAC for X || (X ? T): ");**

**for (int i = 0; i < blockSize; i++) {**

**printf("%02x ", T\_2[i]);**

**}**

**printf("\n");**

**return 0;**

**}**

**12.CMAC**

**#include <stdio.h>**

**#include <stdint.h>**

**// Function to perform left shift on a byte array**

**void leftShift(uint8\_t \*input, int len) {**

**uint8\_t carry = 0;**

**for (int i = 0; i < len; i++) {**

**uint8\_t nextCarry = input[i] >> 7;**

**input[i] = (input[i] << 1) | carry;**

**carry = nextCarry;**

**}**

**}**

**// Function to generate subkey and print it**

**void generateAndPrintSubkey(uint8\_t \*zeroBlock, const uint8\_t constRb, const char \*subkeyName, int blockSize) {**

**leftShift(zeroBlock, blockSize);**

**// If MSB of zeroBlock is 1, XOR with constRb**

**if (zeroBlock[0] & 0x80) {**

**zeroBlock[blockSize - 1] ^= constRb;**

**}**

**// Print the subkey**

**printf("%s Subkey: ", subkeyName);**

**for (int i = 0; i < blockSize; i++) {**

**printf("%02x ", zeroBlock[i]);**

**}**

**printf("\n");**

**}**

**int main() {**

**int blockSize;**

**printf("Enter the block size: ");**

**scanf("%d", &blockSize);**

**uint8\_t zeroBlock[blockSize];**

**const uint8\_t constRb = 0x87; // Change the constant as needed**

**printf("Block cipher operation on zeroBlock\n");**

**// Generate and print the first subkey**

**generateAndPrintSubkey(zeroBlock, constRb, "First", blockSize);**

**// Generate and print the second subkey**

**generateAndPrintSubkey(zeroBlock, constRb, "Second", blockSize);**

**return 0;**

**}**

**12.DSA AND RSA**

**#include <stdio.h>**

**#include <openssl/dsa.h>**

**#include <openssl/rsa.h>**

**#include <openssl/pem.h>**

**#include <openssl/err.h>**

**void handle\_openssl\_error() {**

**ERR\_print\_errors\_fp(stderr);**

**exit(EXIT\_FAILURE);**

**}**

**int main() {**

**// Generate DSA key pair**

**DSA \*dsa = DSA\_new();**

**if (!DSA\_generate\_parameters\_ex(dsa, 1024, NULL, 0, NULL, NULL, NULL)) {**

**handle\_openssl\_error();**

**}**

**if (!DSA\_generate\_key(dsa)) {**

**handle\_openssl\_error();**

**}**

**// Generate RSA key pair**

**RSA \*rsa = RSA\_generate\_key(1024, RSA\_F4, NULL, NULL);**

**if (!rsa) {**

**handle\_openssl\_error();**

**}**

**// Message to be signed**

**unsigned char message[] = "Hello, world!";**

**size\_t message\_len = sizeof(message) - 1;**

**// Sign the message using DSA**

**unsigned char dsa\_signature1[DSA\_size(dsa)];**

**unsigned char dsa\_signature2[DSA\_size(dsa)];**

**unsigned int dsa\_signature\_len;**

**if (!DSA\_sign(0, message, message\_len, dsa\_signature1, &dsa\_signature\_len, dsa)) {**

**handle\_openssl\_error();**

**}**

**if (!DSA\_sign(0, message, message\_len, dsa\_signature2, &dsa\_signature\_len, dsa)) {**

**handle\_openssl\_error();**

**}**

**// Sign the message using RSA**

**unsigned char rsa\_signature1[RSA\_size(rsa)];**

**unsigned char rsa\_signature2[RSA\_size(rsa)];**

**unsigned int rsa\_signature\_len;**

**if (!RSA\_sign(NID\_sha1, message, message\_len, rsa\_signature1, &rsa\_signature\_len, rsa)) {**

**handle\_openssl\_error();**

**}**

**if (!RSA\_sign(NID\_sha1, message, message\_len, rsa\_signature2, &rsa\_signature\_len, rsa)) {**

**handle\_openssl\_error();**

**}**

**// Compare DSA signatures**

**int dsa\_signature\_diff = memcmp(dsa\_signature1, dsa\_signature2, dsa\_signature\_len);**

**if (dsa\_signature\_diff == 0) {**

**printf("DSA signatures are the same.\n");**

**} else {**

**printf("DSA signatures are different.\n");**

**}**

**// Compare RSA signatures**

**int rsa\_signature\_diff = memcmp(rsa\_signature1, rsa\_signature2, rsa\_signature\_len);**

**if (rsa\_signature\_diff == 0) {**

**printf("RSA signatures are the same.\n");**

**} else {**

**printf("RSA signatures are different.\n");**

**}**

**// Clean up**

**DSA\_free(dsa);**

**RSA\_free(rsa);**

**return 0;**

**}**

**13.ECB**

**#include <stdlib.h>**

**#include <stdio.h>**

**#include <string.h>**

**#include "aes.h"**

**#define min(a,b) (((a)<(b))?(a):(b))**

**unsigned char \*aes128\_func(unsigned char \*in, unsigned char \*aes\_key, char type, unsigned int \*len)**

**{**

**unsigned char ptext[KEY\_128]={0},ctext[KEY\_128]={0},\*out=NULL;**

**unsigned char \*temp\_in=NULL,\*temp\_out=NULL;**

**aes\_ctx\_t \*ctx = NULL;**

**if(!in || !len || !(aes\_key && (strlen(aes\_key) < 17))) return NULL;**

**init\_aes();**

**if(!(ctx = aes\_alloc\_ctx(aes\_key, KEY\_128))) {**

**return NULL;**

**}**

**memset(ctx, 0, sizeof(\*ctx));**

**\*len = \*len + ( (\*len % 16) ? (16 - (\*len % 16)): 0);**

**if(!(out =(char \*)malloc(sizeof(\*len+1))))**

**{**

**aes\_free\_ctx(ctx);**

**return NULL;**

**}**

**memset(out,0,\*len+1);**

**temp\_in=in;**

**temp\_out=out;**

**do{**

**memset(ptext, 0, sizeof(ptext));**

**memset(ctext, 0, sizeof(ctext));**

**if(type==0){**

**memcpy(ptext, temp\_in, min(sizeof(ptext),strlen(temp\_in)));**

**aes\_encrypt(ctx, ptext, ctext);**

**}else{**

**memcpy(ptext, temp\_in, sizeof(ptext));**

**aes\_decrypt(ctx, ptext, ctext);**

**}**

**memcpy(temp\_out,ctext,KEY\_128);**

**}while(((temp\_in += KEY\_128) < in+\*len) && ((temp\_out += KEY\_128) < out+\*len));**

**if(ctx) aes\_free\_ctx(ctx);**

**if(type==1) \*len=strlen(out);**

**printf ("\nout aes %s\n",out);**

**return out;**

**}**

**unsigned char \*process\_post\_data(unsigned char \*in, unsigned char \*aes\_key, char type, unsigned int \*ret\_len)**

**{**

**unsigned char \*step1\_buf=NULL, \*step2\_buf=NULL;**

**unsigned int len=0;**

**printf ("at process\_post\_data \nin=%s\t aes\_key=%s\t type=%s\t ret\_len=%u\n", in, aes\_key, type, ret\_len);**

**if(!in || !aes\_key) return NULL;**

**len = strlen(in);**

**if(type == 0)**

**{**

**if(!(step1\_buf = aes128\_func(in, aes\_key, type, &len)))**

**return NULL;**

**printf ("\naes enc data is \"%s\" \n",step1\_buf);**

**}**

**}**

**int main(int argc, char \*argv[])**

**{**

**unsigned int ret\_len;**

**char \*data=NULL, \*processed\_data=NULL, \*post\_data=NULL;**

**data="{\"RequestUniqueID\":\"987654321\",\"MethodName\":\"DstGenerateSessionID\"}";**

**printf ("data is \t %s\n ",data);**

**if(!(processed\_data = process\_post\_data(data,"1848519011345614",0,&ret\_len)))**

**{**

**printf ("sorry something worng try again\n");**

**}**

**}**

**14.SIMPLE SUBSTITUTION ALGORITHM**

**#include <stdio.h>**

**#include <string.h>**

**char substitution\_key[128];**

**void init\_substitution\_key() {**

**}**

**void decrypt\_substitution(const char \*ciphertext) {**

**for (int i = 0; i < strlen(ciphertext); i++) {**

**if (ciphertext[i] >= '0' && ciphertext[i] <= '9') {**

**putchar(substitution\_key[ciphertext[i]]);**

**} else {**

**putchar(ciphertext[i]);**

**}**

**}**

**putchar('\n');**

**}**

**int main() {**

**init\_substitution\_key();**

**const char \*ciphertext = "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\_substitution(ciphertext);**

**return 0;**

**}**