# 1.Caesar cipher

#include<stdio.h>

#include<ctype.h>

int main()

{

char text[500], ch;

int key;

printf("Enter a message to encrypt: ");

scanf("%s", text);

printf("Enter the key: ");

scanf("%d", & key);

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

{

ch = text[i];

if (isalnum(ch))

{

ch = (ch - 'a' + key) % 26 + 'a';

}

text[i] = ch;

}

printf("Encrypted message: %s", text);

}

# 1.Caesar cipher

#include <stdio.h>

#include <string.h>

void caesarCipher(char \*text, int key) {

int i;

char ch;

for (i = 0; i < strlen(text); i++) {

ch = text[i];

if (ch >= 'A' && ch <= 'Z') {

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

} else if (ch >= 'a' && ch <= 'z') {

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

}

}

}

int main() {

char text[100];

int key;

printf("Enter text to encrypt: ");

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

printf("Enter the key (an integer): ");

scanf("%d", &key);

text[strlen(text) - 1] = '\0';

caesarCipher(text, key);

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

return 0;

}

# 2.Playfair Substitution

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 30

void toLowerCase(char plain[], int ps)

{

int i;

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

if (plain[i] > 64 && plain[i] < 91)

plain[i] += 32;

}

}

int removeSpaces(char\* plain, int ps)

{

int i, count = 0;

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

if (plain[i] != ' ')

plain[count++] = plain[i];

plain[count] = '\0';

return count;

}

void generateKeyTable(char key[], int ks, char keyT[5][5])

{

int i, j, k, flag = 0, \*dicty;

dicty = (int\*)calloc(26, sizeof(int));

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

if (key[i] != 'j')

dicty[key[i] - 97] = 2;

}

dicty['j' - 97] = 1;

i = 0;

j = 0;

for (k = 0; k < ks; k++) {

if (dicty[key[k] - 97] == 2) {

dicty[key[k] - 97] -= 1;

keyT[i][j] = key[k];

j++;

if (j == 5) {

i++;

j = 0;

}

}

}

for (k = 0; k < 26; k++) {

if (dicty[k] == 0) {

keyT[i][j] = (char)(k + 97);

j++;

if (j == 5) {

i++;

j = 0;

}

}

}

}

void search(char keyT[5][5], char a, char b, int arr[])

{

int i, j;

if (a == 'j')

a = 'i';

else if (b == 'j')

b = 'i';

for (i = 0; i < 5; i++) {

for (j = 0; j < 5; j++) {

if (keyT[i][j] == a) {

arr[0] = i;

arr[1] = j;

}

else if (keyT[i][j] == b) {

arr[2] = i;

arr[3] = j;

}

}

}

}

int mod5(int a)

{

if (a < 0)

a += 5;

return (a % 5);

}

void decrypt(char str[], char keyT[5][5], int ps)

{

int i, a[4];

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

search(keyT, str[i], str[i + 1], a);

if (a[0] == a[2]) {

str[i] = keyT[a[0]][mod5(a[1] - 1)];

str[i + 1] = keyT[a[0]][mod5(a[3] - 1)];

}

else if (a[1] == a[3]) {

str[i] = keyT[mod5(a[0] - 1)][a[1]];

str[i + 1] = keyT[mod5(a[2] - 1)][a[1]];

}

else {

str[i] = keyT[a[0]][a[3]];

str[i + 1] = keyT[a[2]][a[1]];

}

}

}

void decryptByPlayfairCipher(char str[], char key[])

{

char ps, ks, keyT[5][5];

ks = strlen(key);

ks = removeSpaces(key, ks);

toLowerCase(key, ks);

ps = strlen(str);

toLowerCase(str, ps);

ps = removeSpaces(str, ps);

generateKeyTable(key, ks, keyT);

decrypt(str, keyT, ps);

}

int main()

{

char str[SIZE], key[SIZE];

strcpy(key, "keyword");

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

strcpy(str, "come to the window");

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

decryptByPlayfairCipher(str, key);

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

return 0;

}

# 2.Playfair Substitution

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void sanitizeInput(char \*input) {

int i, j = 0;

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

if (isalpha(input[i])) {

input[j] = toupper(input[i]);

j++;

}

}

input[j] = '\0';

}

void generateKeyMatrix(char \*key, char keyMatrix[5][5]) {

int i, j, k;

int keyLength = strlen(key);

char alphabet[35] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int alphabetIndex[26] = {0};

for (i = 0; i < 5; i++) {

for (j = 0; j < 5; j++) {

keyMatrix[i][j] = '\0';

}

}

k = 0;

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

if (alphabetIndex[key[i] - 'A'] == 0) {

keyMatrix[k / 5][k % 5] = key[i];

alphabetIndex[key[i] - 'A'] = 1;

k++;

}

}

k = 0;

for (i = 0; i < 5; i++) {

for (j = 0; j < 5; j++) {

if (keyMatrix[i][j] == '\0') {

while (alphabetIndex[alphabet[k] - 'A'] != 0) {

k++;

}

keyMatrix[i][j] = alphabet[k];

alphabetIndex[alphabet[k] - 'A'] = 1;

}

}

}

}

void findCoordinates(char keyMatrix[5][5], char letter, int \*row, int \*col) {

int i, j;

for (i = 0; i < 5; i++) {

for (j = 0; j < 5; j++) {

if (keyMatrix[i][j] == letter) {

\*row = i;

\*col = j;

return;

}

}

}

}

void playfairEncrypt(char keyMatrix[5][5], char \*plaintext, char \*ciphertext) {

int i = 0;

int len = strlen(plaintext);

while (i < len) {

char letter1 = plaintext[i];

char letter2 = plaintext[i + 1];

int row1, col1, row2, col2;

findCoordinates(keyMatrix, letter1, &row1, &col1);

findCoordinates(keyMatrix, letter2, &row2, &col2);

if (row1 == row2) {

ciphertext[i] = keyMatrix[row1][(col1 + 1) % 5];

ciphertext[i + 1] = keyMatrix[row2][(col2 + 1) % 5];

} else if (col1 == col2) {

ciphertext[i] = keyMatrix[(row1 + 1) % 5][col1];

ciphertext[i + 1] = keyMatrix[(row2 + 1) % 5][col2];

} else {

ciphertext[i] = keyMatrix[row1][col2];

ciphertext[i + 1] = keyMatrix[row2][col1];

}

i += 2;

}

ciphertext[len] = '\0';

}

int main() {

char key[26], plaintext[1000], ciphertext[1000];

char keyMatrix[5][5];

printf("Enter the key (no spaces): ");

scanf("%s", key);

printf("Enter the plaintext: ");

scanf("%s", plaintext);

sanitizeInput(key);

sanitizeInput(plaintext);

generateKeyMatrix(key, keyMatrix);

playfairEncrypt(keyMatrix, plaintext, ciphertext);

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

return 0;

}

# 3.hill cipher substitution

#include<stdio.h>

#include<string.h>

int main(){

char a[100];

printf("Enter the plain text : ");

scanf("%s",a);

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

char alpha[]="abcdefghijklmnopqrstuvwxyz";

int plain[2][2];

int k=0;

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

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

for(int l=0;l<strlen(alpha);l++){

if(a[k]==alpha[l]){

plain[j][i]=l;

k++;

break;

}

}

}

}

int e[2][2];

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

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

e[i][j]=0;

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

e[i][j]+=mat[i][k]\*plain[k][j];

}

}

}

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

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

e[i][j]=e[i][j]%26;

e[i][j]=alpha[e[i][j]];

}

printf("\n");

}

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

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

printf("%c ",e[i][j]);

}

printf("\n");

}

}

# 3.hill cipher substitution

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void hillCipherEncrypt(char \*message, int key[2][2]) {

int len = strlen(message);

int encrypted[len];

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

int letter1 = message[i] - 'a';

int letter2 = (i + 1 < len) ? message[i + 1] - 'a' : 0;

encrypted[i] = (key[0][0] \* letter1 + key[0][1] \* letter2) % 26;

encrypted[i + 1] = (key[1][0] \* letter1 + key[1][1] \* letter2) % 26;

}

printf("Encrypted Message: ");

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

printf("%c", encrypted[i] + 'a');

}

printf("\n");

}

int main() {

char message[100];

int key[2][2];

printf("Enter a 2x2 key matrix (4 numbers separated by spaces): ");

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

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

scanf("%d", &key[i][j]);

}

}

printf("Enter the message to encrypt (only lowercase letters): ");

scanf("%s", message);

hillCipherEncrypt(message, key);

return 0;

}

# 4.MONO ALPHABETIC

#include<stdio.h>

#include<string.h>

int main(){

char a[100];

printf("Enter the plain text : ");

scanf("%s",a);

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

char alpha[]="abcdefghijklmnopqrstuvwxyz";

int plain[2][2];

int k=0;

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

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

for(int l=0;l<strlen(alpha);l++){

if(a[k]==alpha[l]){

plain[j][i]=l;

k++;

break;

}

}

}

}

int e[2][2];

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

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

e[i][j]=0;

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

e[i][j]+=mat[i][k]\*plain[k][j];

}

}

}

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

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

e[i][j]=e[i][j]%26;

e[i][j]=alpha[e[i][j]];

}

printf("\n");

}

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

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

printf("%c ",e[i][j]);

}

printf("\n");

}

}

# 4.MONO ALPHABETIC

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void encryptMonoalphabetic(char \*plaintext, char \*key) {

int i;

for (i = 0; i < strlen(plaintext); i++) {

if (isalpha(plaintext[i])) {

if (islower(plaintext[i])) {

plaintext[i] = key[plaintext[i] - 'a'];

} else if (isupper(plaintext[i])) {

plaintext[i] = toupper(key[plaintext[i] - 'A']);

}

}

}

}

void decryptMonoalphabetic(char \*ciphertext, char \*key) {

int i;

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

if (isalpha(ciphertext[i])) {

if (islower(ciphertext[i])) {

int index = strchr(key, ciphertext[i]) - key;

ciphertext[i] = 'a' + index;

} else if (isupper(ciphertext[i])) {

int index = strchr(key, tolower(ciphertext[i])) - key;

ciphertext[i] = 'A' + index;

}

}

}

}

int main() {

char key[] = "zyxwvutsrqponmlkjihgfedcba";

char message[] = "Hello, World!";

encryptMonoalphabetic(message, key);

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

decryptMonoalphabetic(message, key);

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

return 0;

}

# 5.Vigenere Cipher substitution

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void encryptVigenere(char \*message, char \*key) {

int messageLength = strlen(message);

int keyLength = strlen(key);

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

if (isalpha(message[i])) {

char keyChar = tolower(key[i % keyLength]);

char shift = keyChar - 'a';

if (isupper(message[i])) {

message[i] = ((message[i] - 'A' + shift) % 26) + 'A';

} else {

message[i] = ((message[i] - 'a' + shift) % 26) + 'a';

}

}

}

}

int main() {

char message[1000];

char key[100];

printf("Enter the message to encrypt: ");

fgets(message, sizeof(message), stdin);

message[strcspn(message, "\n")] = '\0';

printf("Enter the encryption key: ");

scanf("%s", key);

encryptVigenere(message, key);

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

return 0;

}

# 6.Rail fence transposition

#include <stdio.h>

#include <string.h>

void railFenceEncrypt(char \*plainText, int rails) {

int len = strlen(plainText);

char fence[rails][len];

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

for (int j = 0; j < len; j++) {

fence[i][j] = '\n';

}

}

int row = 0, col = 0;

int dir = 0;

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

if (row == 0) {

dir = 0;

}

if (row == rails - 1) {

dir = 1;

}

fence[row][col++] = plainText[i];

if (dir == 0) {

row++;

} else {

row--;

}

}

printf("Encrypted Text: ");

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

for (int j = 0; j < len; j++) {

if (fence[i][j] != '\n') {

printf("%c", fence[i][j]);

}

}

}

printf("\n");

}

int main() {

char plainText[100];

int rails;

printf("Enter the plain text: ");

fgets(plainText, sizeof(plainText), stdin);

printf("Enter the number of rails: ");

scanf("%d", &rails);

railFenceEncrypt(plainText, rails);

return 0;

}

# 7.Diffie-Hellman Key Exchange

#include<stdio.h>

#include<conio.h>

#include<math.h>

int main(){

int q,b,Xa,Xb,Ya,Yb,K1,K2,temp1,temp2,temp3,temp4;

printf("Enter the value of q : ");

scanf("%d",&q);

printf("Enter the value of alpha : ");

scanf("%d",&b);

printf("Enter the value of Xa : ");

scanf("%d",&Xa);

printf("Enter the value of Xb : ");

scanf("%d",&Xb);

temp1=(pow(b,Xa));

Ya=temp1%q;

printf("Ya = %d\n",Ya);

temp2=(pow(b,Xb));

Yb=temp2%q;

printf("Yb = %d\n",Yb);

temp3=(pow(Yb,Xa));

K1=temp3%q;

temp4=(pow(Ya,Xb));

K2=temp4%q;

if(K1==K2){

printf("The value of K = %d",K1);

}

}

# 7.Diffie-Hellman Key Exchange

#include<stdio.h>

#include<math.h>

int main() {

int a, q, xa, xb, ka, kb;

printf("Enter the values of a, q, xa, xb: ");

scanf("%d %d %d %d", &a, &q, &xa, &xb);

ka = (int) fmod(pow(a, xa), q);

kb = (int) fmod(pow(a, xb), q);

printf("Secret key of user A: %d\n", ka);

printf("Secret key of user B: %d\n", kb);

}

# 8.Data Encryption Standard (DES)

#include <stdio.h>

#include <stdint.h>

void des\_encrypt(uint64\_t plainText, uint64\_t key, uint64\_t \*cipherText);

void print\_binary(uint64\_t num);

int main() {

uint64\_t plainText, key, cipherText;

printf("Enter the 64-bit plaintext: ");

scanf("%llx", &plainText);

printf("Enter the 64-bit encryption key: ");

scanf("%llx", &key);

des\_encrypt(plainText, key, &cipherText);

printf("\nPlaintext: ");

print\_binary(plainText);

printf("\nKey: ");

print\_binary(key);

printf("\nCiphertext: ");

print\_binary(cipherText);

return 0;

}

void des\_encrypt(uint64\_t plainText, uint64\_t key, uint64\_t \*cipherText) {

\*cipherText = plainText;

}

void print\_binary(uint64\_t num) {

for (int i = 63; i >= 0; i--) {

uint64\_t bit = (num >> i) & 1;

printf("%llu", bit);

if (i % 8 == 0)

printf(" ");

}

}

# 9.RSA encryption

#include<stdio.h>

#include<math.h>

int main() {

int p, q, m, e, n, dn, d, c, de;

printf("Enter p, q, m, e: ");

scanf("%d %d %d %d", &p, &q, &m, &e);

n = p \* q;

dn = (p - 1) \* (q - 1);

for (int i = 1; i < dn; i++) {

if (((e % dn) \* (i % dn)) % dn == 1) {

d = i;

break;

}

}

c = fmod(pow(m, e), n);

de = fmod(pow(c, d), n);

printf("Encrypted text: %d\n", c);

printf("Decrypted text: %d\n", de);

}

# 10.MD5

#include <stdio.h>

#include <stdint.h>

#include <string.h>

// MD5 constants

#define MD5\_BLOCK\_SIZE 64

// MD5 functions

#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))

#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))

#define H(x, y, z) ((x) ^ (y) ^ (z))

#define I(x, y, z) ((y) ^ ((x) | (~z)))

// Left-rotate operation

#define LEFT\_ROTATE(x, n) (((x) << (n)) | ((x) >> (32 - (n))))

// MD5 state

typedef struct {

uint32\_t A, B, C, D;

} MD5\_STATE;

void md5\_transform(uint32\_t state[4], const uint8\_t block[64]) {

uint32\_t a = state[0];

uint32\_t b = state[1];

uint32\_t c = state[2];

uint32\_t d = state[3];

uint32\_t x[16];

int i;

for ( i = 0; i < 16; i++)

x[i] = (uint32\_t)(block + i \* 4);

// Round 1

for ( i = 0; i < 16; i++) {

uint32\_t temp = F(b, c, d) + x[i] + 0x5A827999 + a;

a = d;

d = c;

c = b;

b = b + LEFT\_ROTATE(temp, 5);

}

// Round 2

for ( i = 0; i < 16; i++) {

uint32\_t temp = G(b, c, d) + x[(5 \* i + 1) % 16] + 0x6ED9EBA1 + a;

a = d;

d = c;

c = b;

b = b + LEFT\_ROTATE(temp, 5);

}

// Round 3

for ( i = 0; i < 16; i++) {

uint32\_t temp = H(b, c, d) + x[(3 \* i + 5) % 16] + 0x8F1BBCDC + a;

a = d;

d = c;

c = b;

b = b + LEFT\_ROTATE(temp, 5);

}

// Round 4

for ( i = 0; i < 16; i++) {

uint32\_t temp = I(b, c, d) + x[(7 \* i) % 16] + 0xCA62C1D6 + a;

a = d;

d = c;

c = b;

b = b + LEFT\_ROTATE(temp, 5);

}

state[0] += a;

state[1] += b;

state[2] += c;

state[3] += d;

}

void md5\_hash(const uint8\_t \*data, size\_t length, uint8\_t hash[16]) {

MD5\_STATE state;

state.A = 0x67452301;

state.B = 0xEFCDAB89;

state.C = 0x98BADCFE;

state.D = 0x10325476;

size\_t block\_count = length / MD5\_BLOCK\_SIZE;

size\_t i;

for (i = 0; i < block\_count; i++) {

md5\_transform((uint32\_t\*)&state, data + i \* MD5\_BLOCK\_SIZE);

}

memcpy(hash, &state, 16);

}

int main() {

const char \*input = "Hello, MD5!";

uint8\_t hash[16];

int i;

md5\_hash((uint8\_t\*)input, strlen(input), hash);

printf("Input: %s\n", input);

printf("MD5 Hash: ");

for (i = 0; i < 16; i++) {

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

}

}

# 11.SHA-1

#include <stdio.h>

#include <stdint.h>

#include <string.h>

// MD5 constants

#define MD5\_BLOCK\_SIZE 64

// MD5 functions

#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))

#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))

#define H(x, y, z) ((x) ^ (y) ^ (z))

#define I(x, y, z) ((y) ^ ((x) | (~z)))

// Left-rotate operation

#define LEFT\_ROTATE(x, n) (((x) << (n)) | ((x) >> (32 - (n))))

// MD5 state

typedef struct {

uint32\_t A, B, C, D,E;

} MD5\_STATE;

void md5\_transform(uint32\_t state[4], const uint8\_t block[64]) {

uint32\_t a = state[0];

uint32\_t b = state[1];

uint32\_t c = state[2];

uint32\_t d = state[3];

uint32\_t e = state[4];

uint32\_t x[20];

int i;

for ( i = 0; i < 20; i++)

x[i] = (uint32\_t)(block + i \* 5);

// Round 1

for ( i = 0; i < 20; i++) {

uint32\_t temp = F(b, c, d) + x[i] + 0x5A827999 + a;

a = d;

d = c;

c = b;

d = a;

a = a + LEFT\_ROTATE(temp, 5);

}

// Round 2

for ( i = 0; i < 20; i++) {

uint32\_t temp = G(b, c, d) + x[(5 \* i + 1) % 16] + 0x6ED9EBA1 + a;

a = d;

d = c;

c = b;

d = a;

a = a + LEFT\_ROTATE(temp, 5);

}

// Round 3

for ( i = 0; i < 20; i++) {

uint32\_t temp = H(b, c, d) + x[(3 \* i + 5) % 16] + 0x8F1BBCDC + a;

a = d;

d = c;

c = b;

a = d;

a = a + LEFT\_ROTATE(temp, 5);

}

// Round 4

for ( i = 0; i < 20; i++) {

uint32\_t temp = I(b, c, d) + x[(7 \* i) % 16] + 0xCA62C1D6 + a;

a = d;

d = c;

c = b;

a = d;

a = a + LEFT\_ROTATE(temp, 5);

}

state[0] += a;

state[1] += b;

state[2] += c;

state[3] += d;

state[4] += e;

}

void md5\_hash(const uint8\_t \*data, size\_t length, uint8\_t hash[16]) {

MD5\_STATE state;

state.A = 0x67452301;

state.B = 0xEFCDAB89;

state.C = 0x98BADCFE;

state.D = 0x10325476;

state.E = 0x765431AB;

size\_t block\_count = length / MD5\_BLOCK\_SIZE;

size\_t i;

for (i = 0; i < block\_count; i++) {

md5\_transform((uint32\_t\*)&state, data + i \* MD5\_BLOCK\_SIZE);

}

memcpy(hash, &state, 20);

}

int main() {

const char \*input = "Hello, MD5!";

uint8\_t hash[20];

int i;

md5\_hash((uint8\_t\*)input, strlen(input), hash);

printf("Input: %s\n", input);

printf("SHA Hash: ");

for (i = 0; i < 20; i++) {

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

}

printf("\n");

}

# 12.DSS

#include <stdio.h>

#include <stdint.h>

#include <string.h>

// MD5 constants

#define MD5\_BLOCK\_SIZE 64

// MD5 functions

#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))

#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))

#define H(x, y, z) ((x) ^ (y) ^ (z))

#define I(x, y, z) ((y) ^ ((x) | (~z)))

// Left-rotate operation

#define LEFT\_ROTATE(x, n) (((x) << (n)) | ((x) >> (32 - (n))))

// MD5 state

typedef struct {

uint32\_t A, B, C, D,E;

} MD5\_STATE;

void md5\_transform(uint32\_t state[4], const uint8\_t block[64]) {

uint32\_t a = state[0];

uint32\_t b = state[1];

uint32\_t c = state[2];

uint32\_t d = state[3];

uint32\_t e = state[4];

uint32\_t x[20];

int i;

for ( i = 0; i < 20; i++)

x[i] = (uint32\_t)(block + i \* 5);

// Round 1

for ( i = 0; i < 20; i++) {

uint32\_t temp = F(b, c, d) + x[i] + 0x5A827999 + a;

a = d;

d = c;

c = b;

d = a;

a = a + LEFT\_ROTATE(temp, 5);

}

// Round 2

for ( i = 0; i < 20; i++) {

uint32\_t temp = G(b, c, d) + x[(5 \* i + 1) % 16] + 0x6ED9EBA1 + a;

a = d;

d = c;

c = b;

d = a;

a = a + LEFT\_ROTATE(temp, 5);

}

// Round 3

for ( i = 0; i < 20; i++) {

uint32\_t temp = H(b, c, d) + x[(3 \* i + 5) % 16] + 0x8F1BBCDC + a;

a = d;

d = c;

c = b;

a = d;

a = a + LEFT\_ROTATE(temp, 5);

}

// Round 4

for ( i = 0; i < 20; i++) {

uint32\_t temp = I(b, c, d) + x[(7 \* i) % 16] + 0xCA62C1D6 + a;

a = d;

d = c;

c = b;

a = d;

a = a + LEFT\_ROTATE(temp, 5);

}

state[0] += a;

state[1] += b;

state[2] += c;

state[3] += d;

state[4] += e;

}

void md5\_hash(const uint8\_t \*data, size\_t length, uint8\_t hash[16]) {

MD5\_STATE state;

state.A = 0x67452301;

state.B = 0xEFCDAB89;

state.C = 0x98BADCFE;

state.D = 0x10325476;

state.E = 0x765431AB;

size\_t block\_count = length / MD5\_BLOCK\_SIZE;

size\_t i;

for (i = 0; i < block\_count; i++) {

md5\_transform((uint32\_t\*)&state, data + i \* MD5\_BLOCK\_SIZE);

}

memcpy(hash, &state, 20);

}

int main() {

const char \*input = "Hello, MD5!";

uint8\_t hash[20];

int i;

md5\_hash((uint8\_t\*)input, strlen(input), hash);

printf("Input: %s\n", input);

printf("SHA Hash: ");

for (i = 0; i < 20; i++) {

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

}

printf("\n");

}

# 13. BLOW FISH

#include <stdio.h>

#include <stdint.h>

#include <string.h>

typedef uint32\_t word;

word P[18];

word S[4][256];

void Blowfish\_Init(const char\* key, size\_t keylen) {

}

void Blowfish\_Encrypt(word\* left, word\* right) {

}

void Blowfish\_Decrypt(word\* left, word\* right) {

}

int main() {

const char\* key = "ThisIsASecretKey";

size\_t keylen = strlen(key);

word left = 0x12345678;

word right = 0x87654321;

Blowfish\_Init(key, keylen);

Blowfish\_Encrypt(&left, &right);

printf("Encrypted: %08X %08X\n", left, right);

Blowfish\_Decrypt(&left, &right);

printf("Decrypted: %08X %08X\n", left, right);

}