# **Practical – 1**

**AIM: A. Write a C program that contains a string (char pointer) with a value 'Hello. The program should XOR, AND and OR each character in this string with 0 and displays the result.**

* **Code:**

#include <iostream>

#include <string>

using namespace std;

int main() {

string str = "Hello World";

int var\_xor = 0, var\_and = 0, var\_or = 0;

cout << "XOR, AND, OR operations with 0" << endl;

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

cout << str[i] << " (" << int(str[i]) << ") ";

cout << "XOR " << var\_xor << ": " << (str[i] ^ var\_xor) << ", ";

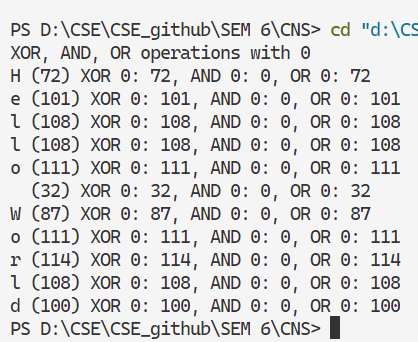
cout << "AND " << var\_and << ": " << (str[i] & var\_and) << ", ";

cout << "OR " << var\_or << ": " << (str[i] | var\_or) << endl;

}

return 0;

}

* **Output:**

**B. Write a C program that contains a string (char pointer) with a value 'Hello World’. The program should XOR, AND and OR each character in this string with 127 and displays the result.**

* **Code:**

#include <iostream>

#include <string>

using namespace std;

int main() {

string str = "Hello World";

int var\_xor = 127, var\_and = 127, var\_or = 127;

cout << "XOR, AND, OR operations with 127" << endl;

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

cout << str[i] << " (" << int(str[i]) << ") ";

cout << "XOR " << var\_xor << ": " << (str[i] ^ var\_xor) << ", ";

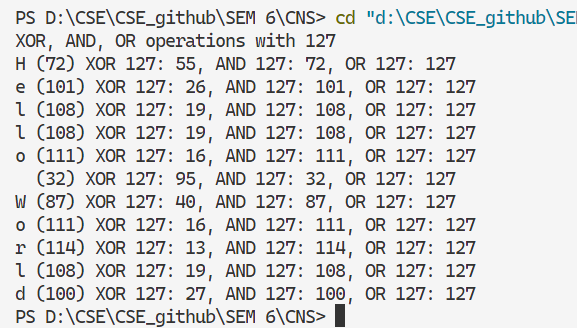
cout << "AND " << var\_and << ": " << (str[i] & var\_and) << ", ";

cout << "OR " << var\_or << ": " << (str[i] | var\_or) << endl;

}

return0**;**

**}**

* **Output:**

**C. Write a C program that contains a string (char pointer) with a value 'Hello World’. The program should bitwise OR, left shift and right shift each character in this string and displays the result.**

* **Code:**

#include <iostream>

#include <string>

using namespace std;

int main() {

string str = "Hello World";

int var\_or = 127;

cout << "Bitwise OR, Left Shift, and Right Shift operations" << endl;

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

cout << str[i] << " (" << int(str[i]) << ") ";

cout << "OR " << var\_or << ": " << (str[i] | var\_or) << ", ";

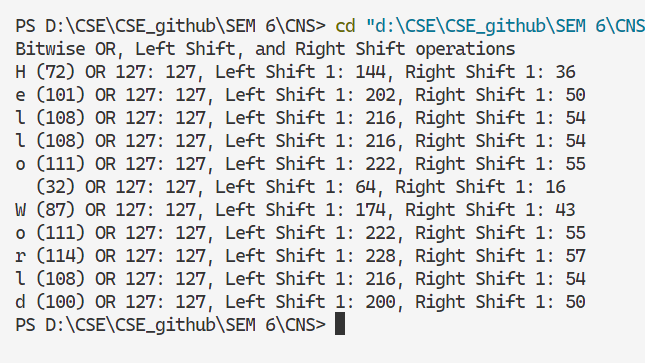
cout << "Left Shift 1: " << (str[i] << 1) << ", ";

cout << "Right Shift 1: " << (str[i] >> 1) << endl;

}

return 0;

}

* **Output:**

# **Practical – 2**

**AIM: To implement Caesar Cipher Encryption - Decryption**

* **Code:**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**string encryption(string str, int key) {**

**string encrypted\_str = "";**

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

**char c = str.at(i);**

**if(c == ' ') {**

**} else {**

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

**if (islower(c)) {**

**c = (c - 'a' + key) % 26 + 'a';**

**} else if (isupper(c)) {**

**c = (c - 'A' + key) % 26 + 'A';**

**}**

**} else {**

**int temp = (int) c;**

**temp += key;**

**c = (char) temp;**

**}**

**}**

**encrypted\_str += c;**

**}**

**return encrypted\_str;**

**}**

**string description(string encrypted\_str, int key) {**

**string str = "";**

**for (int i = 0; i < encrypted\_str.length(); i++) {**

**char c = encrypted\_str.at(i);**

**if(c == ' ') {**

**} else {**

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

**if (islower(c)) {**

**c = (c - 'a' - key + 26) % 26 + 'a';**

**} else if (isupper(c)) {**

**c = (c - 'A' - key + 26) % 26 + 'A';**

**}**

**} else {**

**int temp = (int) c;**

**temp -= key;**

**c = (char) temp;**

**}**

**}**

**str += c;**

**}**

**return str;**

**}**

**int main() {**

**string str = "hello world";**

**unsigned int key = 5;**

**cout << "Enter any text" << endl;**

**getline(cin, str);**

**cout << "Enter key" << endl;**

**cin >> key;**

**string encrypted\_str = encryption(str, key);**

**string description\_str = description(encrypted\_str, key);**

**cout << "text: " << str << endl;**

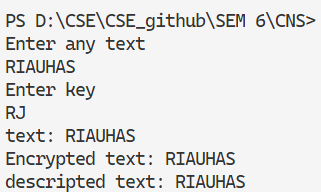
**cout << "Encrypted text: " << encrypted\_str << endl;**

**cout << "descripted text: " << description\_str << endl;**

**return 0;**

**}**

* **Output:**

****

# **Practical – 3**

**AIM: To implement Mono-alphabetic Cipher Encryption – Decryption.**

* **Code:**

#include <iostream>

#include <string>

using namespace std;

string encrypt\_text(string plaintext, string key) {

string normal = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

string ciphertext = "";

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

char ch = plaintext[i];

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

int index = ch - 'A';

ciphertext += key[index];

}

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

int index = ch - 'a';

ciphertext += tolower(key[index]);

}

else {

ciphertext += ch;

}

}

return ciphertext;

}

string decrypt\_cipher(string ciphertext, string key) {

string normal = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

string plaintext = "";

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

char ch = ciphertext[i];

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

int index = key.find(ch);

plaintext += normal[index];

}

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

int index = key.find(toupper(ch));

plaintext += tolower(normal[index]);

}

else {

plaintext += ch;

}

}

return plaintext;

}

int main() {

string key = "QWERTYUIOPLKJHGFDSAZXCVBNM";

string message;

cout << "Enter the message: ";

getline(cin, message);

string encryptedText = encrypt\_text(message, key);

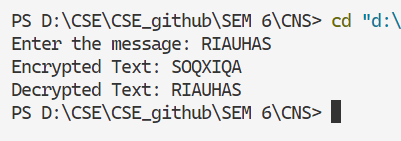
string decryptedText = decrypt\_cipher(encryptedText, key);

cout << "Encrypted Text: " << encryptedText << endl;

cout << "Decrypted Text: " << decryptedText << endl;

return 0;

}

* **Output:**

# **Practical – 4**

**AIM: To implement Hill Cipher Encryption**

* Code:

#include <iostream>

#include <cstring>

using namespace std;

const int SIZE = 3;

void multiplyMatrix(int key[SIZE][SIZE], int text[SIZE], int result[SIZE]) {

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

result[i] = 0;

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

result[i] += key[i][j] \* text[j];

}

result[i] = (result[i] % 26 + 26) % 26;

}

}

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;

}

void findInverseMatrix(int key[SIZE][SIZE], int inverseKey[SIZE][SIZE]) {

int determinant = key[0][0] \* (key[1][1] \* key[2][2] - key[1][2] \* key[2][1]) -

key[0][1] \* (key[1][0] \* key[2][2] - key[1][2] \* key[2][0]) +

key[0][2] \* (key[1][0] \* key[2][1] - key[1][1] \* key[2][0]);

determinant = (determinant % 26 + 26) % 26;

int determinantInverse = modInverse(determinant, 26);

if (determinantInverse == -1) {

cout << "Key matrix is not invertible.\n";

return;

}

int adjoint[SIZE][SIZE] = {

{(key[1][1] \* key[2][2] - key[1][2] \* key[2][1]), -(key[0][1] \* key[2][2] - key[0][2] \* key[2][1]), (key[0][1] \* key[1][2] - key[0][2] \* key[1][1])},

{-(key[1][0] \* key[2][2] - key[1][2] \* key[2][0]), (key[0][0] \* key[2][2] - key[0][2] \* key[2][0]), -(key[0][0] \* key[1][2] - key[0][2] \* key[1][0])},

{(key[1][0] \* key[2][1] - key[1][1] \* key[2][0]), -(key[0][0] \* key[2][1] - key[0][1] \* key[2][0]), (key[0][0] \* key[1][1] - key[0][1] \* key[1][0])}

};

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

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

inverseKey[i][j] = (adjoint[i][j] \* determinantInverse) % 26;

if (inverseKey[i][j] < 0) {

inverseKey[i][j] += 26;

}

}

}

}

void encrypt\_text(char plaintext[], char ciphertext[]) {

int key[SIZE][SIZE] = {{6, 24, 1}, {13, 16, 10}, {20, 17, 15}};

int textBlock[SIZE], encryptedBlock[SIZE];

int len = strlen(plaintext);

while (len % SIZE != 0) {

plaintext[len] = 'X';

len++;

plaintext[len] = '\0';

}

cout << "Encrypted Text: ";

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

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

textBlock[j] = plaintext[i + j] - 'A';

}

multiplyMatrix(key, textBlock, encryptedBlock);

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

ciphertext[i + j] = (char)(encryptedBlock[j] + 'A');

cout << ciphertext[i + j];

}

}

ciphertext[len] = '\0'; // Null terminate the ciphertext

cout << endl;

}

void decrypt\_cipher(char ciphertext[]) {

int key[SIZE][SIZE] = {{6, 24, 1}, {13, 16, 10}, {20, 17, 15}};

int inverseKey[SIZE][SIZE];

findInverseMatrix(key, inverseKey);

int textBlock[SIZE], decryptedBlock[SIZE];

int len = strlen(ciphertext);

cout << "Decrypted Text: ";

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

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

textBlock[j] = ciphertext[i + j] - 'A';

}

multiplyMatrix(inverseKey, textBlock, decryptedBlock);

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

cout << (char)(decryptedBlock[j] + 'A');

}

}

cout << endl;

}

int main() {

char plaintext[100], ciphertext[100];

cout << "Enter plaintext (uppercase letters only): ";

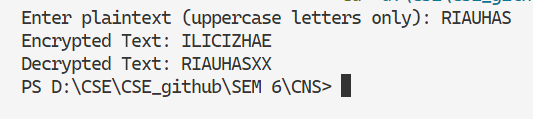
cin >> plaintext;

encrypt\_text(plaintext, ciphertext);

decrypt\_cipher(ciphertext);

return 0;

}

* ****Output:

# **Practical – 5**

**AIM: To implement Poly-alphabetic Cipher (Vigener Cipher) Technique**

* Code:

#include <iostream>

#include <cstring>

using namespace std;

void generateKey(const char\* text, const char\* key, char\* newKey) {

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

for (int i = 0, j = 0; i < textLen; i++, j++) {

if (j == keyLen) {

j = 0;

}

newKey[i] = key[j];

}

newKey[textLen] = '\0';

}

void encrypt\_text(const char\* text, const char\* key, char\* encryptedText) {

int len = strlen(text);

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

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

}

encryptedText[len] = '\0';

}

void decrypt\_cipher(const char\* encryptedText, const char\* key, char\* decryptedText) {

int len = strlen(encryptedText);

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

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

}

decryptedText[len] = '\0';

}

int main() {

char text[100], key[100], newKey[100], encryptedText[100], decryptedText[100];

cout << "Enter text (uppercase letters only): ";

cin >> text;

cout << "Enter key (uppercase letters only): ";

cin >> key;

generateKey(text, key, newKey);

encrypt\_text(text, newKey, encryptedText);

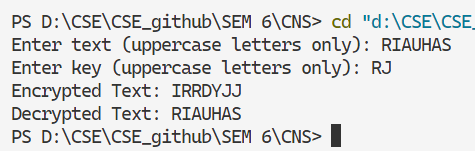
decrypt\_cipher(encryptedText, newKey, decryptedText);

cout << "Encrypted Text: " << encryptedText << endl;

cout << "Decrypted Text: " << decryptedText << endl;

return 0;

}

* ****Output:

# **Practical – 6**

**AIM: To implement Play-Fair Cipher Technique.**

* **Code:**

#include <iostream>

#include <string>

using namespace std;

const int SIZE = 5;

void generateKeyMatrix(const string& key, char keyMatrix[SIZE][SIZE]) {

bool used[26] = {false};

string refinedKey = "";

for (char ch : key) {

if (ch >= 'a' && ch <= 'z') ch -= 32;

if (ch == 'J') ch = 'I';

if (ch >= 'A' && ch <= 'Z' && !used[ch - 'A']) {

refinedKey += ch;

used[ch - 'A'] = true;

}

}

for (char ch = 'A'; ch <= 'Z'; ++ch) {

if (ch == 'J') continue;

if (!used[ch - 'A']) {

refinedKey += ch;

used[ch - 'A'] = true;

}

}

int index = 0;

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

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

keyMatrix[i][j] = refinedKey[index++];

}

}

}

void findPosition(char keyMatrix[SIZE][SIZE], char ch, int &row, int &col) {

if (ch == 'J') ch = 'I';

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

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

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

row = i;

col = j;

return;

}

}

}

}

string prepareText(const string& text) {

string result = "";

for (char ch : text) {

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

if (ch >= 'a' && ch <= 'z') ch -= 32;

result += ch;

}

}

string processed = "";

for (size\_t i = 0; i < result.length(); ++i) {

processed += result[i];

if (i + 1 < result.length() && result[i] == result[i + 1]) {

processed += 'X';

}

}

if (processed.length() % 2 != 0) {

processed += 'X';

}

return processed;

}

string encrypt(const string& text, char keyMatrix[SIZE][SIZE]) {

string preparedText = prepareText(text);

string encryptedText = "";

for (size\_t i = 0; i < preparedText.length(); i += 2) {

char first = preparedText[i];

char second = preparedText[i + 1];

int row1, col1, row2, col2;

findPosition(keyMatrix, first, row1, col1);

findPosition(keyMatrix, second, row2, col2);

if (row1 == row2) {

encryptedText += keyMatrix[row1][(col1 + 1) % SIZE];

encryptedText += keyMatrix[row2][(col2 + 1) % SIZE];

} else if (col1 == col2) {

encryptedText += keyMatrix[(row1 + 1) % SIZE][col1];

encryptedText += keyMatrix[(row2 + 1) % SIZE][col2];

} else {

encryptedText += keyMatrix[row1][col2];

encryptedText += keyMatrix[row2][col1];

}

}

return encryptedText;

}

string decrypt(const string& text, char keyMatrix[SIZE][SIZE]) {

string decryptedText = "";

for (size\_t i = 0; i < text.length(); i += 2) {

char first = text[i];

char second = text[i + 1];

int row1, col1, row2, col2;

findPosition(keyMatrix, first, row1, col1);

findPosition(keyMatrix, second, row2, col2);

if (row1 == row2) {

decryptedText += keyMatrix[row1][(col1 - 1 + SIZE) % SIZE];

decryptedText += keyMatrix[row2][(col2 - 1 + SIZE) % SIZE];

} else if (col1 == col2) {

decryptedText += keyMatrix[(row1 - 1 + SIZE) % SIZE][col1];

decryptedText += keyMatrix[(row2 - 1 + SIZE) % SIZE][col2];

} else {

decryptedText += keyMatrix[row1][col2];

decryptedText += keyMatrix[row2][col1];

}

}

return decryptedText;

}

int main() {

string key, text;

char keyMatrix[SIZE][SIZE];

cout << "Enter key: ";

cin >> key;

generateKeyMatrix(key, keyMatrix);

cout << "Enter text to encrypt: ";

cin >> text;

string encryptedText = encrypt(text, keyMatrix);

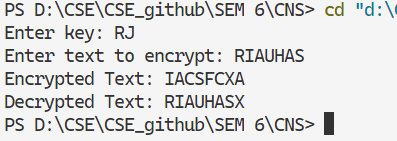
cout << "Encrypted Text: " << encryptedText << endl;

string decryptedText = decrypt(encryptedText, keyMatrix);

cout << "Decrypted Text: " << decryptedText << endl;

return 0;

}

* **Output:**

# **Practical – 7**

**AIM: Write a program to implement Rail-Fence, Simple columnar Encryption Technique.**

* **Code:**

#include <iostream>

#include <cstring>

using namespace std;

string railFenceEncrypt(string text, int key) {

char rail[key][text.length()];

memset(rail, ' ', sizeof(rail));

int row = 0, direction = 1;

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

rail[row][i] = text[i];

row += direction;

if (row == key - 1 || row == 0) direction \*= -1;

}

string encryptedText = "";

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

for (int j = 0; j < text.length(); j++) {

if (rail[i][j] != ' ') encryptedText += rail[i][j];

}

}

return encryptedText;

}

string railFenceDecrypt(string cipher, int key) {

char rail[key][cipher.length()];

memset(rail, ' ', sizeof(rail));

int row = 0, direction = 1;

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

rail[row][i] = '\*';

row += direction;

if (row == key - 1 || row == 0) direction \*= -1;

}

int index = 0;

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

for (int j = 0; j < cipher.length(); j++) {

if (rail[i][j] == '\*' && index < cipher.length()) {

rail[i][j] = cipher[index++];

}

}

}

string decryptedText = "";

row = 0, direction = 1;

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

decryptedText += rail[row][i];

row += direction;

if (row == key - 1 || row == 0) direction \*= -1;

}

return decryptedText;

}

void sortKey(string key, int keyOrder[]) {

int len = key.length();

char tempKey[len];

for (int i = 0; i < len; i++) tempKey[i] = key[i];

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

int minIdx = i;

for (int j = i + 1; j < len; j++) {

if (tempKey[j] < tempKey[minIdx]) {

minIdx = j;

}

}

swap(tempKey[i], tempKey[minIdx]);

swap(keyOrder[i], keyOrder[minIdx]);

}

}

string columnarEncrypt(string text, string key) {

int keyLen = key.length();

int textLen = text.length();

int numRows = (textLen + keyLen - 1) / keyLen;

char grid[numRows][keyLen];

int index = 0;

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

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

grid[i][j] = (index < textLen) ? text[index++] : 'X';

}

}

int keyOrder[keyLen];

for (int i = 0; i < keyLen; i++) keyOrder[i] = i;

sortKey(key, keyOrder);

string encryptedText = "";

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

int col = keyOrder[i];

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

encryptedText += grid[j][col];

}

}

return encryptedText;

}

string columnarDecrypt(string cipher, string key) {

int keyLen = key.length();

int cipherLen = cipher.length();

int numRows = (cipherLen + keyLen - 1) / keyLen;

char grid[numRows][keyLen];

int keyOrder[keyLen];

for (int i = 0; i < keyLen; i++) keyOrder[i] = i;

sortKey(key, keyOrder);

int index = 0;

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

int col = keyOrder[i];

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

grid[j][col] = cipher[index++];

}

}

string decryptedText = "";

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

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

decryptedText += grid[i][j];

}

}

return decryptedText;

}

int main() {

string text, key;

int railKey;

cout << "Enter text to encrypt: ";

cin >> text;

cout << "Enter Rail-Fence key: ";

cin >> railKey;

string railCipher = railFenceEncrypt(text, railKey);

cout << "Rail-Fence Encrypted Text: " << railCipher << endl;

string railDecrypted = railFenceDecrypt(railCipher, railKey);

cout << "Rail-Fence Decrypted Text: " << railDecrypted << endl;

cout << "Enter Columnar Transposition key (word): ";

cin >> key;

string columnCipher = columnarEncrypt(text, key);

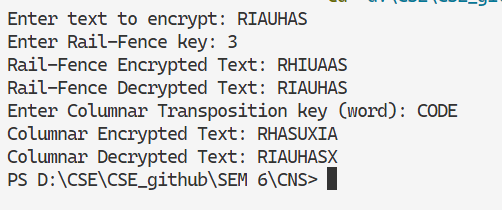
cout << "Columnar Encrypted Text: " << columnCipher << endl;

string columnDecrypted = columnarDecrypt(columnCipher, key);

cout << "Columnar Decrypted Text: " << columnDecrypted << endl;

return 0;

}

* **Output:**

# **Practical – 8**

**AIM: To implement the S-DES algorithm for data encryption.**

* **Code:**

#include <iostream>

#include <string>

using namespace std;

int P10[] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};

int P8[] = {6, 3, 7, 4, 8, 5, 10, 9};

int IP[] = {2, 6, 3, 1, 4, 8, 5, 7};

int IP\_inv[] = {4, 1, 3, 5, 7, 2, 8, 6};

int EP[] = {4, 1, 2, 3, 2, 3, 4, 1};

int P4[] = {2, 4, 3, 1};

int S0[4][4] = {

{1, 0, 3, 2}, {3, 2, 1, 0}, {0, 2, 1, 3}, {3, 1, 3, 2}};

int S1[4][4] = {

{0, 1, 2, 3}, {2, 0, 1, 3}, {3, 0, 1, 0}, {2, 1, 0, 3}};

string permute(string input, int\* table, int size) {

string output = "";

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

output += input[table[i] - 1];

}

return output;

}

string leftShift(string key, int shifts) {

return key.substr(shifts) + key.substr(0, shifts);

}

string generateKey(string key, bool first) {

key = permute(key, P10, 10);

key = leftShift(key.substr(0, 5), first ? 1 : 2) + leftShift(key.substr(5, 5), first ? 1 : 2);

return permute(key, P8, 8);

}

string xorOperation(string a, string b) {

string result = "";

for (size\_t i = 0; i < a.length(); i++) {

result += (a[i] == b[i]) ? '0' : '1';

}

return result;

}

string sBox(string input, int S[4][4]) {

int row = (input[0] - '0') \* 2 + (input[3] - '0');

int col = (input[1] - '0') \* 2 + (input[2] - '0');

int val = S[row][col];

return string(1, '0' + (val / 2)) + string(1, '0' + (val % 2));

}

string fk(string input, string key) {

string left = input.substr(0, 4);

string right = input.substr(4, 4);

string expandedRight = permute(right, EP, 8);

string xored = xorOperation(expandedRight, key);

string sboxOut = sBox(xored.substr(0, 4), S0) + sBox(xored.substr(4, 4), S1);

string permuted = permute(sboxOut, P4, 4);

return xorOperation(left, permuted) + right;

}

string swapHalves(string input) {

return input.substr(4, 4) + input.substr(0, 4);

}

string encrypt(string plaintext, string key) {

string K1 = generateKey(key, true);

string K2 = generateKey(key, false);

string permutedText = permute(plaintext, IP, 8);

string firstRound = fk(permutedText, K1);

string swapped = swapHalves(firstRound);

string secondRound = fk(swapped, K2);

return permute(secondRound, IP\_inv, 8);

}

string decrypt(string ciphertext, string key) {

string K1 = generateKey(key, true);

string K2 = generateKey(key, false);

string permutedText = permute(ciphertext, IP, 8);

string firstRound = fk(permutedText, K2);

string swapped = swapHalves(firstRound);

string secondRound = fk(swapped, K1);

return permute(secondRound, IP\_inv, 8);

}

int main() {

string key, plaintext;

cout << "Enter 10-bit key: ";

cin >> key;

cout << "Enter 8-bit plaintext: ";

cin >> plaintext;

string ciphertext = encrypt(plaintext, key);

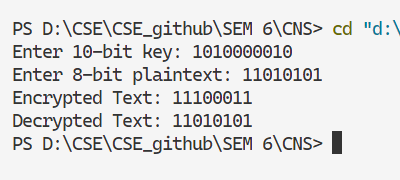
cout << "Encrypted Text: " << ciphertext << endl;

string decryptedText = decrypt(ciphertext, key);

cout << "Decrypted Text: " << decryptedText << endl;

return 0;

}

* **Output:**

# **Practical – 9**

**AIM: Write a program to implement RSA asymmetric (public key and private key) -Encryption**

* **Code:**

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

long long modExp(long long base, long long exp, long long mod) {

long long result = 1;

base = base % mod;

while (exp > 0) {

if (exp % 2 == 1) {

result = (result \* base) % mod;

}

exp = exp >> 1;

base = (base \* base) % mod;

}

return result;

}

long long simpleHash(const string& message, long long n) {

long long hash = 0;

for (char c : message) {

hash = (hash \* 31 + c) % n;

}

return hash;

}

long long gcdExtended(long long a, long long b, long long\* x, long long\* y) {

if (a == 0) {

\*x = 0;

\*y = 1;

return b;

}

long long x1, y1;

long long gcd = gcdExtended(b % a, a, &x1, &y1);

\*x = y1 - (b / a) \* x1;

\*y = x1;

return gcd;

}

long long modInverse(long long e, long long phi) {

long long x, y;

long long g = gcdExtended(e, phi, &x, &y);

if (g != 1) {

cerr << "Modular inverse doesn't exist";

exit(EXIT\_FAILURE);

}

return (x % phi + phi) % phi;

}

void generateKeys(long long& e, long long& d, long long& n) {

long long p = 1009;

long long q = 1013;

n = p \* q;

long long phi = (p - 1) \* (q - 1);

e = 65537;

d = modInverse(e, phi);

}

long long signMessage(long long hash, long long d, long long n) {

return modExp(hash, d, n);

}

bool verifySignature(long long hash, long long signature, long long e, long long n) {

long long decryptedHash = modExp(signature, e, n);

return hash == decryptedHash;

}

int main() {

string message;

cout << "Enter message: ";

getline(cin, message);

long long e, d, n;

generateKeys(e, d, n);

long long hashValue = simpleHash(message, n);

long long signature = signMessage(hashValue, d, n);

cout << "Original Hash: " << hashValue << endl;

cout << "Signature: " << signature << endl;

if (verifySignature(hashValue, signature, e, n)) {

cout << "Signature verified successfully!" << endl;

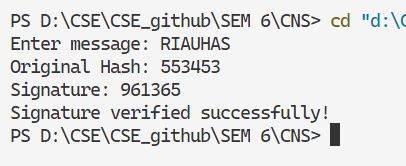
} else {

cout << "Signature verification failed!" << endl;

}

return 0;

}

* **Output:**

# **Practical – 10**

**AIM: Write a program to generate digital signature using Hash code.**

* **Code**

#include <iostream>

#include <string>

using namespace std;

int generateHash(string message) {

int hash = 0;

for (char ch : message) {

hash += (int)ch;

}

return hash % 1009;

}

int signHash(int hash, int privateKey) {

return (hash \* privateKey) % 1009;

}

int main() {

string message;

int privateKey = 17;

cout << "Enter the message to sign: ";

getline(cin, message);

int hash = generateHash(message);

int digitalSignature = signHash(hash, privateKey);

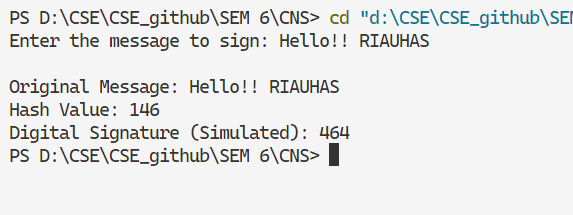
cout << "\nOriginal Message: " << message << endl;

cout << "Hash Value: " << hash << endl;

cout << "Digital Signature (Simulated): " << digitalSignature << endl;

return 0;

}

* **Output**

# **Practical – 11**

**AIM: Case Study on Kerberos.**

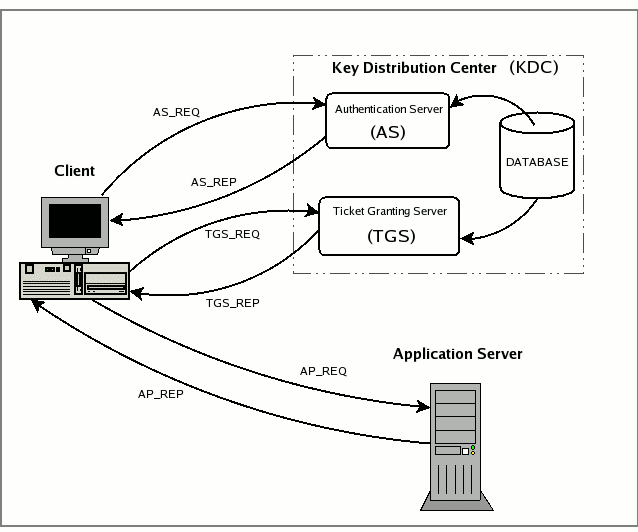
1. **Introduction**

Kerberos is a **network authentication protocol** designed to provide strong authentication for client-server applications using secret-key cryptography. Developed at **MIT** as part of **Project Athena**, it allows entities communicating over a non-secure network to prove their identity securely.

Kerberos is widely used in enterprise environments, including **Microsoft Windows**, where it's the default authentication protocol in Active Directory.

1. **Why Kerberos?**

Traditional systems used passwords for authentication, which are vulnerable to interception and replay attacks. Kerberos addresses this with **tickets** and **time-stamped authentication**, avoiding the direct transmission of passwords.

1. **Kerberos Architecture**

Kerberos relies on a centralized **Key Distribution Center (KDC)**, which is split into two main parts:

* **Authentication Server (AS)**
* **Ticket Granting Server (TGS)**

**Components:**

* **Client/User**: The person or process requesting access.
* **KDC**: The trusted third party responsible for issuing authentication and service tickets.
* **AS**: Verifies the user’s credentials and provides a TGT (Ticket Granting Ticket).
* **TGS**: Issues a service ticket using the TGT.
* **Service Server**: The final destination that the user wants to access (e.g., file server, web app).

1. **How Kerberos Works (Step-by-step)**

**Step 1: Authentication Request**

The client sends a request to the **Authentication Server**.

**Step 2: Ticket Granting Ticket (TGT)**

The AS verifies the credentials and sends back a **TGT**, encrypted using the user’s password-derived key.

**Step 3: Requesting Access**

The client uses the TGT to request access to a particular service from the **TGS**.

**Step 4: Service Ticket**

TGS validates the TGT and sends a **service ticket**, which the client can present to the **Service Server**.

**Step 5: Access Granted**

The client presents the ticket to the server, and if valid, access is granted.

**5. Real-World Use Case**

Kerberos is the default authentication method in:

* **Microsoft Active Directory (Windows)**
* **Hadoop clusters** for securing resource manager and data nodes
* **SSH in enterprise Linux environments**

**6. Advantages of Kerberos**

1. **Strong security** with mutual authentication
2. **No passwords** transmitted over the network
3. **Time-based tickets** prevent replay attacks
4. **Scalable** for large networks
5. **Widely supported** across OS and services

**7. Disadvantages of Kerberos**

* **Single point of failure** – If KDC is down, no one can authenticate
* **Requires synchronized time** between clients and servers
* **Key management complexity** for large setups
* **Initial setup** can be complicated for beginners

**8. Conclusion**

Kerberos is a battle-tested and powerful authentication protocol that is especially effective in distributed systems. With its robust ticketing system and mutual authentication, it has become a critical piece of the security puzzle in many enterprise and academic systems. Understanding Kerberos helps in grasping how real-world secure communication works behind the scenes.

# **Practical – 12**

**AIM: Case Study on Kerberos.**

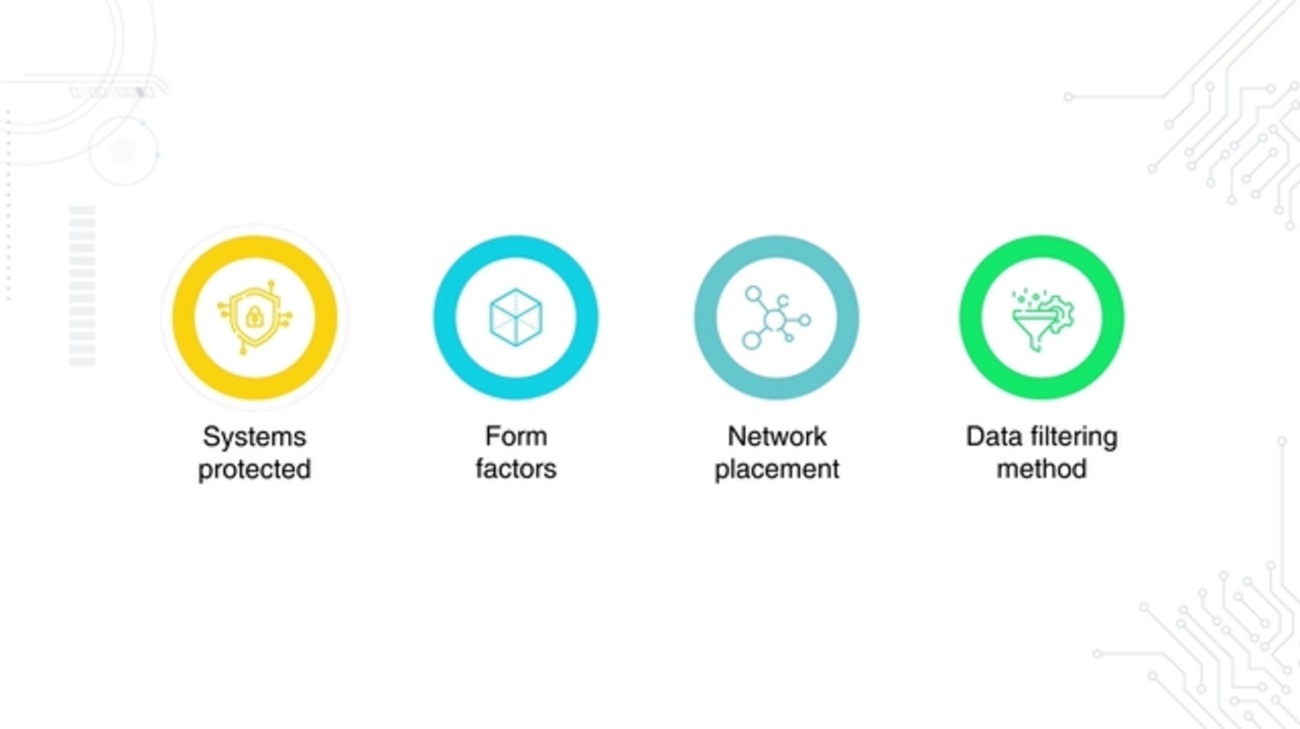
1. **Introduction**

A **firewall** is a **network security system** that monitors and controls incoming and outgoing network traffic based on predetermined security rules. It acts as a barrier between a trusted internal network and untrusted external networks, like the Internet.

Firewalls are essential for both personal computers and enterprise networks to protect against unauthorized access, malware, and other cyber threats.

1. **Why Use a Firewall?**

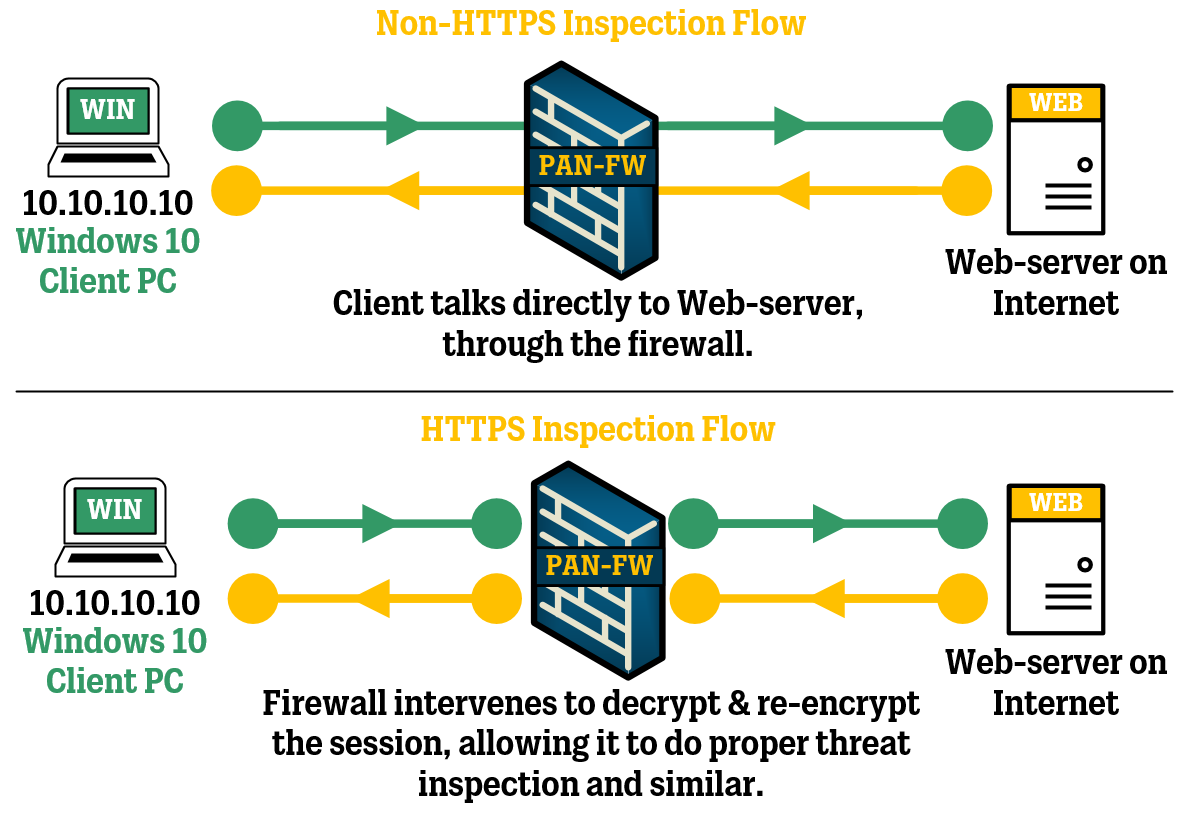
* Prevent unauthorized access
* Block malicious traffic and attacks
* Filter content and data leakage
* Reduce attack surface
* Control how internal users access external resources

1. **Types of Firewalls**
   1. **Packet Filtering Firewall**

* Works at **Network Layer (Layer 3)**
* Inspects source/destination IP, port, and protocol
* **Fast** but limited in context (can’t detect complex attacks)
  1. **Stateful Inspection Firewall**
* Keeps track of the **state of active connections**
* Makes decisions based on both **header information and connection state**
* More secure than simple packet filters
  1. **Application Layer Firewall (Proxy Firewall)**
* Operates at the **Application Layer (Layer 7)**
* Can filter HTTP, FTP, DNS traffic, etc.
* Can inspect **application payloads**
  1. **Next-Generation Firewall (NGFW)**

Combines traditional firewall features with:

* **Deep packet inspection**
* **Intrusion prevention**
* **Antivirus**
* **Content filtering**
* **User identity tracking**

1. **How Firewalls Work**

* All traffic enters and exits through the firewall
* The firewall applies **predefined rules** to each packet or request
* Based on rules, traffic is either **allowed, blocked, or flagged**
* Can be **hardware**, **software**, or a **hybrid**

1. **Real-World Use Case: Corporate Firewall**

**Scenario**: A mid-sized company wants to protect its internal network from the internet.

**Solution**:

* Deploy a **Stateful + Application Layer Firewall**
* Allow ports 80 (HTTP) and 443 (HTTPS)
* Block social media, torrent, and gaming traffic during office hours
* Monitor outgoing traffic to detect data exfiltration

1. **Advantages of Firewalls**

* Provides **network perimeter security**
* Filters **unwanted traffic**
* Can **log, monitor**, and **alert** on suspicious activity
* Enforces **access control policies**
* Helps maintain **compliance** with data protection laws

1. **Disadvantages of Firewalls**

* Can’t protect against **internal threats**
* Not effective if misconfigured
* May cause **latency or bottlenecks**
* Some advanced attacks can bypass them (e.g., phishing, social engineering)

1. **Firewall Best Practices**

* Regularly **update and patch** firmware
* Implement **least privilege policies**
* **Monitor logs** and set alerts
* Conduct **regular audits** and **pen tests**
* Use **cloud-based firewalls** for remote environments

**9. Conclusion**

Firewalls are the **first line of defense** in network security. From personal use to enterprise-grade systems, firewalls help filter traffic, enforce policies, and prevent breaches. While they’re not a silver bullet, when combined with other security layers like IDS/IPS and antivirus, they play a critical role in building a secure network architecture.

# **Practical – 13**

**AIM: Study of MD5 hash function and implement the hash code using MD5.**

* **What is MD5?**

The **MD5 (Message Digest 5)** algorithm is a widely-used **cryptographic hash function** that produces a **128-bit (16-byte)** hash value. It was developed by **Ronald Rivest** in 1991.

* It takes an input (or message) and returns a fixed-length hash value.
* Commonly used for **integrity checking**, **digital signatures**, and **password hashing** (though not recommended for passwords anymore due to vulnerabilities).
* **Code**

#include <iostream>

#include <cstring>

#include <iomanip>

#include <sstream>

typedef unsigned int uint32;

class MD5 {

public:

MD5() { reset(); }

std::string digest(const std::string& str) {

reset();

update((const unsigned char\*)str.c\_str(), str.length());

finalize();

return toHex();

}

private:

uint32 a, b, c, d;

uint32 msgLenLow, msgLenHigh;

unsigned char buffer[64];

uint32 block[16];

bool finalized;

void reset() {

finalized = false;

msgLenLow = msgLenHigh = 0;

a = 0x67452301;

b = 0xefcdab89;

c = 0x98badcfe;

d = 0x10325476;

}

static uint32 F(uint32 x, uint32 y, uint32 z) { return (x & y) | (~x & z); }

static uint32 G(uint32 x, uint32 y, uint32 z) { return (x & z) | (y & ~z); }

static uint32 H(uint32 x, uint32 y, uint32 z) { return x ^ y ^ z; }

static uint32 I(uint32 x, uint32 y, uint32 z) { return y ^ (x | ~z); }

static uint32 rotateLeft(uint32 x, int n) { return (x << n) | (x >> (32 - n)); }

void step(uint32& w, uint32 x, uint32 y, uint32 z, uint32 data, uint32 s, uint32 ac, uint32 (\*func)(uint32, uint32, uint32)) {

w = w + func(x, y, z) + data + ac;

w = rotateLeft(w, s) + x;

}

void transform(const unsigned char block[64]) {

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

this->block[i] = ((uint32)block[i \* 4]) | ((uint32)block[i \* 4 + 1] << 8) |

((uint32)block[i \* 4 + 2] << 16) | ((uint32)block[i \* 4 + 3] << 24);

uint32 A = a, B = b, C = c, D = d;

step(A, B, C, D, this->block[0], 7, 0xd76aa478, F);

step(D, A, B, C, this->block[1], 12, 0xe8c7b756, F);

step(C, D, A, B, this->block[2], 17, 0x242070db, F);

step(B, C, D, A, this->block[3], 22, 0xc1bdceee, F);

step(A, B, C, D, this->block[4], 7, 0xf57c0faf, F);

step(D, A, B, C, this->block[5], 12, 0x4787c62a, F);

step(C, D, A, B, this->block[6], 17, 0xa8304613, F);

step(B, C, D, A, this->block[7], 22, 0xfd469501, F);

step(A, B, C, D, this->block[8], 7, 0x698098d8, F);

step(D, A, B, C, this->block[9], 12, 0x8b44f7af, F);

step(C, D, A, B, this->block[10], 17, 0xffff5bb1, F);

step(B, C, D, A, this->block[11], 22, 0x895cd7be, F);

step(A, B, C, D, this->block[12], 7, 0x6b901122, F);

step(D, A, B, C, this->block[13], 12, 0xfd987193, F);

step(C, D, A, B, this->block[14], 17, 0xa679438e, F);

step(B, C, D, A, this->block[15], 22, 0x49b40821, F);

a += A; b += B; c += C; d += D;

}

void update(const unsigned char\* input, size\_t length) {

size\_t index = (msgLenLow >> 3) & 0x3F;

if ((msgLenLow += (uint32)(length << 3)) < (length << 3))

msgLenHigh++;

msgLenHigh += (uint32)(length >> 29);

size\_t partLen = 64 - index;

size\_t i = 0;

if (length >= partLen) {

memcpy(&buffer[index], input, partLen);

transform(buffer);

for (i = partLen; i + 63 < length; i += 64)

transform(&input[i]);

index = 0;

}

memcpy(&buffer[index], &input[i], length - i);

}

void finalize() {

static unsigned char PADDING[64] = { 0x80 };

if (finalized) return;

unsigned char bits[8];

for (int i = 0; i < 4; ++i) {

bits[i] = (unsigned char)(msgLenLow >> (i \* 8));

bits[i + 4] = (unsigned char)(msgLenHigh >> (i \* 8));

}

size\_t index = (msgLenLow >> 3) & 0x3f;

size\_t padLen = (index < 56) ? (56 - index) : (120 - index);

update(PADDING, padLen);

update(bits, 8);

finalized = true;

}

std::string toHex() const {

std::ostringstream os;

uint32 vals[4] = { a, b, c, d };

for (int i = 0; i < 4; ++i)

for (int j = 0; j < 4; ++j)

os << std::hex << std::setw(2) << std::setfill('0') << ((vals[i] >> (j \* 8)) & 0xff);

return os.str();

}

};

int main() {

MD5 md5;

std::string input;

std::cout << "Enter a message: ";

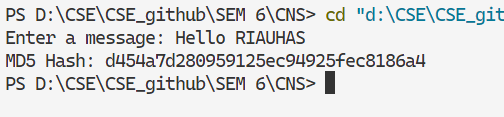
std::getline(std::cin, input);

std::string hash = md5.digest(input);

std::cout << "MD5 Hash: " << hash << std::endl;

return 0;

}

* **Output**

# **Practical – 14**

**AIM: Study of SHA-1 hash function and implement the hash code using SHA-1.**

**SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function that:**

* Produces a **160-bit hash value** (40 hexadecimal characters)
* Is **deterministic**: same input gives same output
* Was **designed by the NSA**, published by NIST in 1995
* Takes any input and compresses it into a **fixed-length 160-bit hash**
* Is now considered **broken for secure cryptography** due to collision vulnerabilities, but still useful for understanding hash mechanics

**Steps in SHA-1:**

1. **Preprocessing**:
   * Message is padded to make its length a multiple of 512 bits.
   * Original message length is added in last 64 bits.
2. **Divide into 512-bit blocks**
3. **Initialize five 32-bit variables** (A, B, C, D, E)
4. **For each block**:
   * Expand the 16 words into 80
   * Run 80 rounds of hashing using functions & bitwise logic
   * Update A, B, C, D, E
5. **Output**: Final 160-bit hash (5 words concatenated)

* **Code**

#include <iostream>

#include <sstream>

#include <iomanip>

#include <cstring>

#include <vector>

typedef unsigned int uint32;

class SHA1 {

public:

SHA1() { reset(); }

std::string digest(const std::string &message) {

reset();

update((const unsigned char\*)message.c\_str(), message.length());

finalize();

return toHex();

}

private:

uint32 h0, h1, h2, h3, h4;

std::vector<unsigned char> buffer;

uint64\_t messageLength;

void reset() {

h0 = 0x67452301;

h1 = 0xEFCDAB89;

h2 = 0x98BADCFE;

h3 = 0x10325476;

h4 = 0xC3D2E1F0;

buffer.clear();

messageLength = 0;

}

static uint32 rotateLeft(uint32 value, uint32 bits) {

return (value << bits) | (value >> (32 - bits));

}

void processBlock(const unsigned char block[64]) {

uint32 w[80];

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

w[i] = (block[i \* 4] << 24) |

(block[i \* 4 + 1] << 16) |

(block[i \* 4 + 2] << 8) |

(block[i \* 4 + 3]);

for (int i = 16; i < 80; ++i)

w[i] = rotateLeft(w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i - 16], 1);

uint32 a = h0, b = h1, c = h2, d = h3, e = h4;

for (int i = 0; i < 80; ++i) {

uint32 f, k;

if (i < 20) {

f = (b & c) | (~b & d);

k = 0x5A827999;

} else if (i < 40) {

f = b ^ c ^ d;

k = 0x6ED9EBA1;

} else if (i < 60) {

f = (b & c) | (b & d) | (c & d);

k = 0x8F1BBCDC;

} else {

f = b ^ c ^ d;

k = 0xCA62C1D6;

}

uint32 temp = rotateLeft(a, 5) + f + e + k + w[i];

e = d;

d = c;

c = rotateLeft(b, 30);

b = a;

a = temp;

}

h0 += a;

h1 += b;

h2 += c;

h3 += d;

h4 += e;

}

void update(const unsigned char \*data, size\_t length) {

messageLength += length \* 8;

buffer.insert(buffer.end(), data, data + length);

while (buffer.size() >= 64) {

processBlock(&buffer[0]);

buffer.erase(buffer.begin(), buffer.begin() + 64);

}

}

void finalize() {

buffer.push\_back(0x80);

while ((buffer.size() + 8) % 64 != 0)

buffer.push\_back(0x00);

for (int i = 7; i >= 0; --i)

buffer.push\_back((messageLength >> (i \* 8)) & 0xFF);

for (size\_t i = 0; i < buffer.size(); i += 64)

processBlock(&buffer[i]);

}

std::string toHex() const {

std::ostringstream result;

uint32 words[5] = { h0, h1, h2, h3, h4 };

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

result << std::hex << std::setw(8) << std::setfill('0') << words[i];

return result.str();

}

};

int main() {

SHA1 sha1;

std::string input;

std::cout << "Enter a message: ";

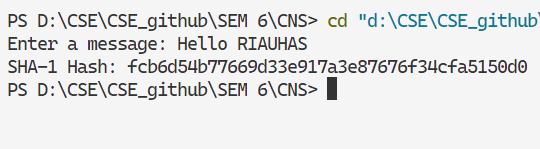
std::getline(std::cin, input);

std::string hash = sha1.digest(input);

std::cout << "SHA-1 Hash: " << hash << std::endl;

return 0;

}

* Output

# **Practical – 15**

**AIM: Write a program to implement transposition Encryption Technique**

* **Code**

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

vector<int> getOrder(string key) {

vector<pair<char, int>> keyMap;

for (int i = 0; i < key.length(); ++i)

keyMap.emplace\_back(key[i], i);

sort(keyMap.begin(), keyMap.end());

vector<int> order(key.length());

for (int i = 0; i < key.length(); ++i)

order[keyMap[i].second] = i;

return order;

}

string encrypt(string message, string key) {

int cols = key.length();

vector<int> order = getOrder(key);

int rows = (message.length() + cols - 1) / cols;

vector<vector<char>> grid(rows, vector<char>(cols, 'X'));

int k = 0;

for (int i = 0; i < rows && k < message.length(); ++i)

for (int j = 0; j < cols && k < message.length(); ++j)

grid[i][j] = message[k++];

string ciphertext = "";

for (int o = 0; o < cols; ++o) {

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

if (order[j] == o) {

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

ciphertext += grid[i][j];

break;

}

}

}

return ciphertext;

}

int main() {

string message, key;

cout << "Enter the plaintext message (no spaces): ";

cin >> message;

cout << "Enter the key (e.g., word or numbers): ";

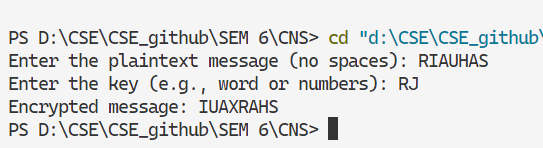
cin >> key;

string encrypted = encrypt(message, key);

cout << "Encrypted message: " << encrypted << endl;

return 0;

}

* **Output**