🔐 1. Caesar Cipher (C++)

#include <iostream>

using namespace std;

string caesarEncrypt(string text, int key) {

string result = "";

for (char c : text) {

if (isalpha(c))

result += char((tolower(c) - 'a' + key + 26) % 26 + 'a');

else

result += c;

}

return result;

}

string caesarDecrypt(string cipher, int key) {

return caesarEncrypt(cipher, -key);

}

int main() {

string text = "hello";

int key = 3;

string encrypted = caesarEncrypt(text, key);

string decrypted = caesarDecrypt(encrypted, key);

cout << "Encrypted: " << encrypted << "\nDecrypted: " << decrypted << endl;

return 0;

}

🔠 2. Monoalphabetic Cipher (C++)

#include <iostream>

#include <map>

#include <algorithm>

using namespace std;

string monoEncrypt(string text, map<char, char>& keyMap) {

string result = "";

for (char c : text)

result += keyMap[tolower(c)];

return result;

}

string monoDecrypt(string text, map<char, char>& keyMap) {

map<char, char> reverseMap;

for (auto& p : keyMap)

reverseMap[p.second] = p.first;

string result = "";

for (char c : text)

result += reverseMap[c];

return result;

}

int main() {

map<char, char> keyMap;

string plain = "abcdefghijklmnopqrstuvwxyz";

string key = "phqgiumeaylnofdxjkrcvstzwb"; // example key

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

keyMap[plain[i]] = key[i];

string text = "hello";

string cipher = monoEncrypt(text, keyMap);

string decrypted = monoDecrypt(cipher, keyMap);

cout << "Encrypted: " << cipher << "\nDecrypted: " << decrypted << endl;

return 0;

}

**🔲 3. Playfair Cipher (C++)**

#include <iostream>

#include <vector>

using namespace std;

string prepareText(string text) {

string res = "";

for (char c : text) {

if (isalpha(c)) {

c = tolower(c);

if (c == 'j') c = 'i';

res += c;

}

}

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

if (i + 1 == res.length())

res += 'x';

else if (res[i] == res[i + 1])

res.insert(i + 1, "x");

}

return res;

}

void createMatrix(string key, char matrix[5][5]) {

string fullKey = "";

bool used[26] = {false};

for (char c : key) {

c = (c == 'j') ? 'i' : c;

if (!used[c - 'a']) {

fullKey += c;

used[c - 'a'] = true;

}

}

for (char c = 'a'; c <= 'z'; ++c) {

if (c == 'j') continue;

if (!used[c - 'a']) {

fullKey += c;

used[c - 'a'] = true;

}

}

for (int i = 0; i < 25; ++i)

matrix[i / 5][i % 5] = fullKey[i];

}

void findPos(char m[5][5], char c, int &row, int &col) {

if (c == 'j') c = 'i';

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

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

if (m[i][j] == c) {

row = i;

col = j;

return;

}

}

string playfairEncrypt(string text, string key) {

text = prepareText(text);

char m[5][5];

createMatrix(key, m);

string res = "";

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

int r1, c1, r2, c2;

findPos(m, text[i], r1, c1);

findPos(m, text[i + 1], r2, c2);

if (r1 == r2)

res += m[r1][(c1 + 1) % 5], res += m[r2][(c2 + 1) % 5];

else if (c1 == c2)

res += m[(r1 + 1) % 5][c1], res += m[(r2 + 1) % 5][c2];

else

res += m[r1][c2], res += m[r2][c1];

}

return res;

}

string playfairDecrypt(string text, string key) {

char m[5][5];

createMatrix(key, m);

string res = "";

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

int r1, c1, r2, c2;

findPos(m, text[i], r1, c1);

findPos(m, text[i + 1], r2, c2);

if (r1 == r2)

res += m[r1][(c1 + 4) % 5], res += m[r2][(c2 + 4) % 5];

else if (c1 == c2)

res += m[(r1 + 4) % 5][c1], res += m[(r2 + 4) % 5][c2];

else

res += m[r1][c2], res += m[r2][c1];

}

return res;

}

int main() {

string key = "monarchy";

string text = "balloon";

string enc = playfairEncrypt(text, key);

string dec = playfairDecrypt(enc, key);

cout << "Encrypted: " << enc << "\nDecrypted: " << dec << endl;

return 0;

}

**🔡 4. Polyalphabetic (Vigenère) Cipher (C++)**

#include <iostream>

using namespace std;

string generateKey(string text, string key) {

int x = text.size();

for (int i = key.size(); i < x; ++i)

key += key[i % key.size()];

return key;

}

string vigenereEncrypt(string text, string key) {

string cipher = "";

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

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

return cipher;

}

string vigenereDecrypt(string cipher, string key) {

string text = "";

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

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

return text;

}

int main() {

string text = "attackatdawn";

string keyword = "lemon";

string key = generateKey(text, keyword);

string cipher = vigenereEncrypt(text, key);

string plain = vigenereDecrypt(cipher, key);

cout << "Encrypted: " << cipher << "\nDecrypted: " << plain << endl;

return 0;

}

**🧮 5. Hill Cipher (2x2 Matrix, C++)**

#include <iostream>

#include <vector>

using namespace std;

vector<vector<int>> keyMatrix = {{3, 3}, {2, 5}}; // Example key

vector<int> multiplyMatrix(vector<vector<int>> matrix, vector<int> vec) {

vector<int> res(2);

res[0] = (matrix[0][0]\*vec[0] + matrix[0][1]\*vec[1]) % 26;

res[1] = (matrix[1][0]\*vec[0] + matrix[1][1]\*vec[1]) % 26;

return res;

}

vector<vector<int>> inverseMatrix() {

vector<vector<int>> inv(2, vector<int>(2));

int det = (keyMatrix[0][0]\*keyMatrix[1][1] - keyMatrix[0][1]\*keyMatrix[1][0]) % 26;

if (det < 0) det += 26;

int invDet = -1;

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

if ((det \* i) % 26 == 1) { invDet = i; break; }

inv[0][0] = keyMatrix[1][1] \* invDet % 26;

inv[0][1] = -keyMatrix[0][1] \* invDet % 26;

inv[1][0] = -keyMatrix[1][0] \* invDet % 26;

inv[1][1] = keyMatrix[0][0] \* invDet % 26;

for (auto& row : inv)

for (auto& x : row)

x = (x + 26) % 26;

return inv;

}

string hillEncrypt(string text) {

if (text.length() % 2 != 0) text += 'x';

string res = "";

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

vector<int> vec = {text[i] - 'a', text[i+1] - 'a'};

vector<int> enc = multiplyMatrix(keyMatrix, vec);

res += char(enc[0] + 'a');

res += char(enc[1] + 'a');

}

return res;

}

string hillDecrypt(string text) {

vector<vector<int>> inv = inverseMatrix();

string res = "";

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

vector<int> vec = {text[i] - 'a', text[i+1] - 'a'};

vector<int> dec = multiplyMatrix(inv, vec);

res += char(dec[0] + 'a');

res += char(dec[1] + 'a');

}

return res;

}

int main() {

string text = "help";

string cipher = hillEncrypt(text);

string plain = hillDecrypt(cipher);

cout << "Encrypted: " << cipher << "\nDecrypted: " << plain << endl;

return 0;

}

**🚂 6(a). Rail Fence Cipher (C++)**

#include <iostream>

#include <vector>

using namespace std;

string railFenceEncrypt(string text, int rails) {

vector<string> rail(rails);

int dir = 1, row = 0;

for (char c : text) {

rail[row] += c;

row += dir;

if (row == 0 || row == rails - 1) dir \*= -1;

}

string result = "";

for (string r : rail) result += r;

return result;

}

string railFenceDecrypt(string cipher, int rails) {

vector<string> rail(rails, string(cipher.size(), '.'));

int dir = 1, row = 0, index = 0;

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

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

row += dir;

if (row == 0 || row == rails - 1) dir \*= -1;

}

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

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

if (rail[i][j] == '\*')

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

string result = "";

dir = 1; row = 0;

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

result += rail[row][i];

row += dir;

if (row == 0 || row == rails - 1) dir \*= -1;

}

return result;

}

int main() {

string text = "meetmeafterparty";

int rails = 3;

string cipher = railFenceEncrypt(text, rails);

string plain = railFenceDecrypt(cipher, rails);

cout << "Encrypted: " << cipher << "\nDecrypted: " << plain << endl;

return 0;

}

**🧱 6(b). Columnar Transposition Cipher (C++)**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

string columnarEncrypt(string text, string key) {

int cols = key.length();

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

text += string(rows \* cols - text.length(), 'x');

vector<vector<char>> matrix(rows, vector<char>(cols));

int k = 0;

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

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

matrix[i][j] = text[k++];

vector<pair<char, int>> order;

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

order.push\_back({key[i], i});

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

string cipher = "";

for (auto [ch, idx] : order)

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

cipher += matrix[i][idx];

return cipher;

}

string columnarDecrypt(string cipher, string key) {

int cols = key.length();

int rows = cipher.length() / cols;

vector<vector<char>> matrix(rows, vector<char>(cols));

vector<pair<char, int>> order;

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

order.push\_back({key[i], i});

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

int k = 0;

for (auto [ch, idx] : order)

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

matrix[i][idx] = cipher[k++];

string plain = "";

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

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

plain += matrix[i][j];

return plain;

}

int main() {

string text = "attackpostponed";

string key = "4312";

string cipher = columnarEncrypt(text, key);

string plain = columnarDecrypt(cipher, key);

cout << "Encrypted: " << cipher << "\nDecrypted: " << plain << endl;

return 0;

}

**🔐 7. Simplified DES (Conceptual C++ - Simplified Example)**

#include <iostream>

using namespace std;

string XOR(string a, string b) {

string res = "";

for (int i = 0; i < a.size(); ++i)

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

return res;

}

string sdesEncrypt(string plaintext, string key) {

string step1 = XOR(plaintext, key);

reverse(step1.begin(), step1.end()); // simplified substitute

return step1;

}

string sdesDecrypt(string cipher, string key) {

reverse(cipher.begin(), cipher.end());

return XOR(cipher, key);

}

int main() {

string plain = "10110011";

string key = "11001010";

string cipher = sdesEncrypt(plain, key);

string dec = sdesDecrypt(cipher, key);

cout << "Encrypted: " << cipher << "\nDecrypted: " << dec << endl;

return 0;

}

**🔑 8. Diffie-Hellman Key Exchange (C++)**

#include <iostream>

#include <cmath>

using namespace std;

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

int result = 1;

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

result = (result \* base) % mod;

return result;

}

int main() {

int p = 23, g = 5;

int a = 6, b = 15;

int A = power(g, a, p);

int B = power(g, b, p);

int keyA = power(B, a, p);

int keyB = power(A, b, p);

cout << "Shared Key A: " << keyA << "\nShared Key B: " << keyB << endl;

return 0;

}

**🔒 9. RSA Encryption-Decryption (C++)**

#include <iostream>

using namespace std;

int gcd(int a, int b) {

return b == 0 ? a : gcd(b, a % b);

}

int modInverse(int e, int phi) {

for (int d = 1; d < phi; ++d)

if ((e \* d) % phi == 1)

return d;

return -1;

}

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

int result = 1;

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

result = (result \* base) % mod;

return result;

}

int main() {

int p = 3, q = 11;

int n = p \* q;

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

int e = 3;

int d = modInverse(e, phi);

int msg = 7;

int cipher = power(msg, e, n);

int decrypted = power(cipher, d, n);

cout << "Encrypted: " << cipher << "\nDecrypted: " << decrypted << endl;

return 0;

}