Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информационных технологий автоматизированных систем

Отчет по лабораторной работе №2

по курсу «ИТОКБ»

на тему: «Простейшие криптографические преобразования»

Вариант 4

|  |  |
| --- | --- |
| Выполнил магистрант группы 025941: | Колесников В.Г. |
| Проверил: | Боброва Т.С. |

МИНСК

2021

**Задание 1:** Реализовать в виде программы шифр (зашифрование и расшифрование) перестановки, использующий простые (прямоугольные) таблицы. Язык исходного текста — английский.

Код программы приведен в конце лабораторной работы.

Результаты работы программы зашифрования и расшифрования приведены на рисунках 1.1 и 1.2.

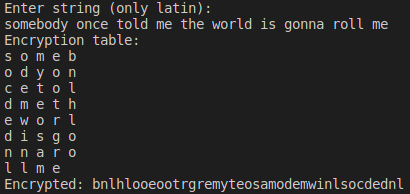


Рисунок 1.1 — Результат зашифрования текста шифром перестановки

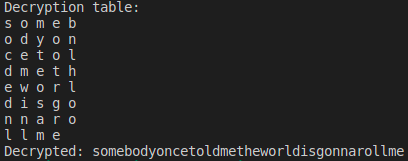


Рисунок 1.2 — Результат расшифрования текста, зашифрованного шифром перестановки

**Задание 2:** Реализовать в виде программы атаку полным перебором ключа, используя для оценки правильности выбора ключа визуальный метод или исходный текст для автоматического сравнения результата дешифрования.

Используемый шифр в качестве ключа задействует размер таблицы. Таким образом, чтобы узнать ключ, достаточно подобрать точное количество столбцов. При шифровании текста таким способом важно учитывать его размер, т. к. если количество символов больше, чем колонок в столбце, шифрование не удастся. Зная алгоритм шифрования, в данном случае расшифровка не составит труда при использовании вычислительных средств.

Зашифрованный текст с ключом в 100 колонок представлен на рисунке 2.1. Расшифрованный текст и таблица дешивровки представлены на рисунке 2.2.

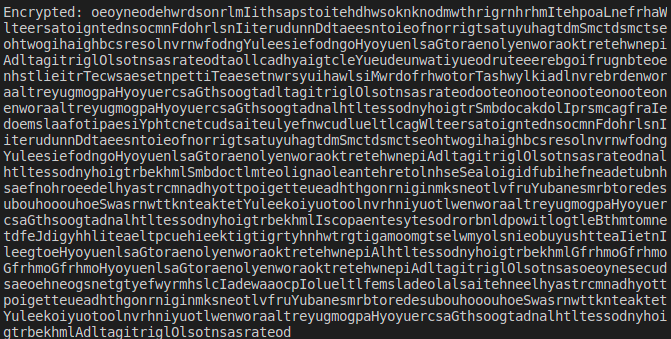


Рисунок 2.1 — Зашифрованный текст

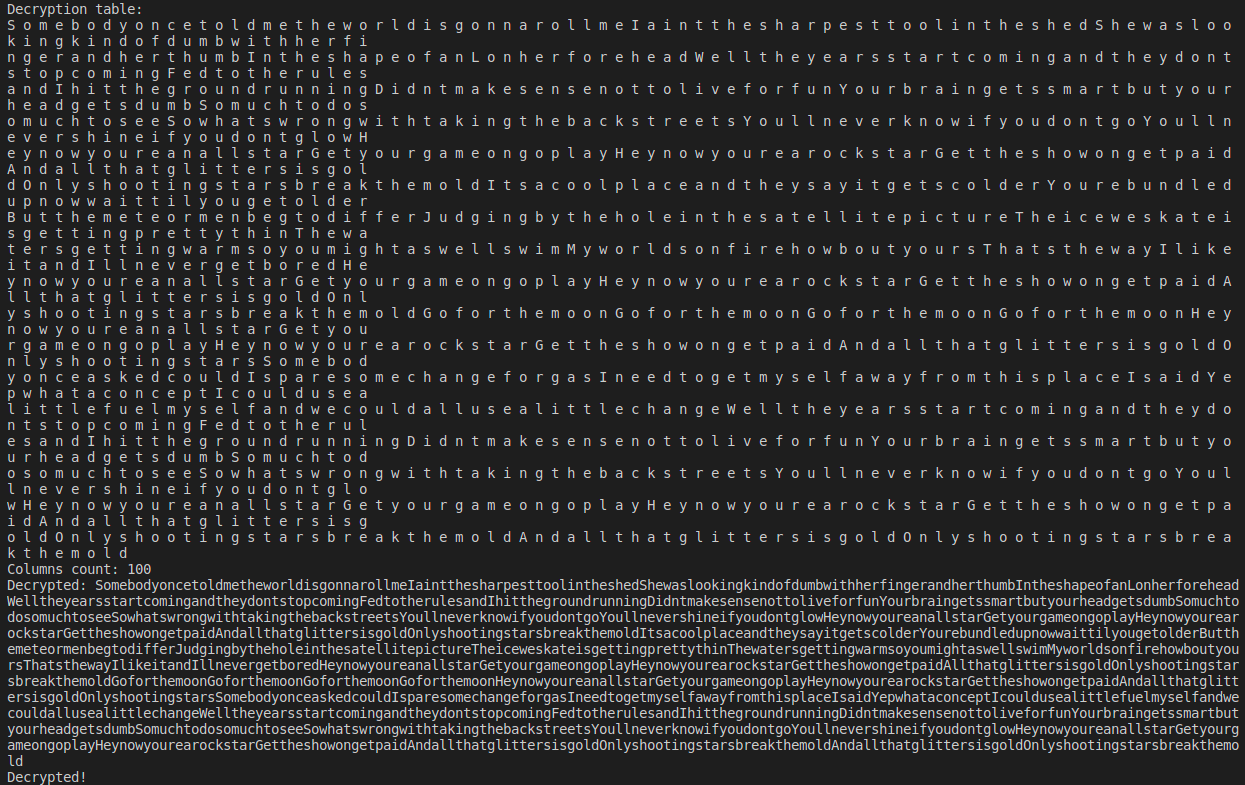


Рисунок 2.2 — Таблица дешифровки и расшифрованный текст

**Задание 3:** Оценить криптографическую стойкость реализованного шифра.

Реализованный алгоритм шифрования обладает не слишком большой стойкостью, т. к. криптоаналитик при знании алгоритма может запросто подобрать необходимый размер таблицы дешифровки. Даже при незначительных изменениях в способе шифрования алгоритм все еще остается легковзламываемым.

**Задание 4:** Предложить варианты усложнения шифра. Предложенные варианты оформить в виде алгоритма.

Один из способов усложнения шифра — зашифровать зашифрованный текст еще раз другим размером таблицы. Кроме того, можно комбинировать данный шифр с каким-либо еще шифром из данной лабораторной работы. При необходимости дополнительного усложнения можно зашифровать шифр с помощью букв, равноотстоящих от каждой следующей буквы на определенное растояние в одной из таблиц кодировки.

Пример шифрования и дешифрования первым способом представлен на рисунках 4.1 и 4.2.

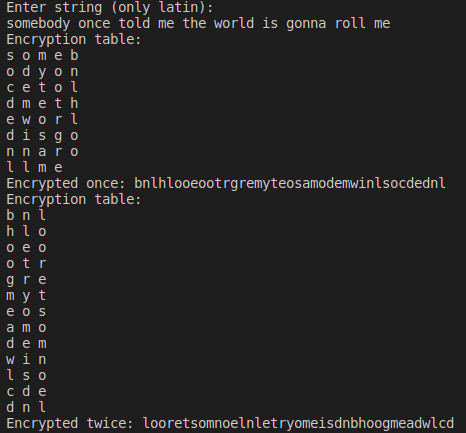


Рисунок 4.1 — Двойное шифрование с таблицами на 5 и на 3 колонки

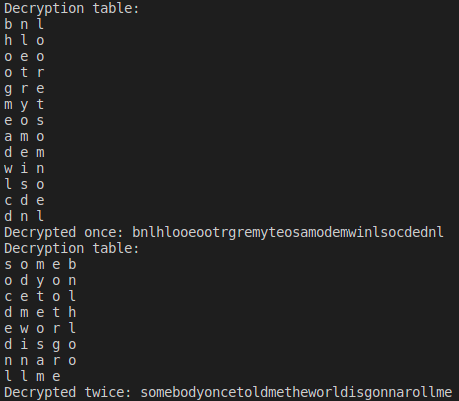


Рисунок 4.2 — Дешифрование шифра с таблицами на 5 и на 3 колонки

**Код программы:**

#include <iostream>

#include <string>

#include <cmath>

#include <limits.h>

using namespace std;

char \*\*createEncryptionTable(const int rowsCount, const int colsCount, string strToEncryptClean)

{

char \*\*encryptionTable = new char \*[rowsCount];

for (int row = 0; row < rowsCount; row++)

{

encryptionTable[row] = new char[colsCount];

for (int col = 0; col < colsCount; col++)

{

encryptionTable[row][col] = strToEncryptClean[row \* colsCount + col];

}

}

cout << "Encryption table:" << endl;

for (int row = 0; row < rowsCount; row++)

{

for (int col = 0; col < colsCount; col++)

{

cout << encryptionTable[row][col] << " ";

}

cout << endl;

}

return encryptionTable;

}

string encrypt(const int colsCount, string strToEncrypt)

{

const int rowsCount = ceil((double)strToEncrypt.length() / colsCount);

const int traillingSpacesCount = rowsCount \* colsCount - strToEncrypt.length();

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

{

strToEncrypt += ' ';

}

char \*\*encryptionTable = createEncryptionTable(rowsCount, colsCount, strToEncrypt);

string encrypted = "";

for (int col = colsCount - 1; col >= 0; col--)

{

for (int row = 0; row < rowsCount; row++)

{

if (encryptionTable[row][col] != ' ')

{

encrypted += encryptionTable[row][col];

}

}

}

return encrypted;

}

char \*\*createDecryptionTable(const int rowsCount, const int colsCount, string strToDecrypt)

{

const int strLength = strToDecrypt.length();

const int tableSize = rowsCount \* colsCount;

int bigColsCount = 0;

int bigRowsCount = rowsCount;

if (tableSize > strLength)

{

bigColsCount = strLength + colsCount - tableSize;

bigRowsCount--;

}

char \*\*decryptionTable = new char \*[rowsCount];

for (int row = 0; row < rowsCount; row++)

{

decryptionTable[row] = new char[colsCount];

}

int i = 0;

for (int col = colsCount - 1; col >= 0; col--)

{

for (int row = 0; row < rowsCount; row++)

{

if (bigColsCount > 0 && col >= bigColsCount && row == bigRowsCount)

{

break;

}

else

{

decryptionTable[row][col] = strToDecrypt[i];

i++;

}

}

}

cout << "Decryption table:" << endl;

for (int row = 0; row < rowsCount; row++)

{

for (int col = 0; col < colsCount; col++)

{

cout << decryptionTable[row][col] << " ";

}

cout << endl;

}

return decryptionTable;

}

string decrypt(const int colsCount, string strToDecrypt)

{

const int rowsCount = ceil((double)strToDecrypt.length() / colsCount);

char \*\*decryptionTable = createDecryptionTable(rowsCount, colsCount, strToDecrypt);

string decrypted = "";

for (int row = 0; row < rowsCount; row++)

{

for (int col = 0; col < colsCount; col++)

{

if (decryptionTable[row][col] != ' ' && decryptionTable[row][col] != '\0')

{

decrypted += decryptionTable[row][col];

}

}

}

return decrypted;

}

const void bruteForce(const string sourceStr, const int colsCount)

{

const string encrypted = encrypt(colsCount, sourceStr);

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

for (int i = 2; i < colsCount + 10; i++)

{

string decrypted = decrypt(i, encrypted);

cout << "Columns count: " << i << endl;

cout << "Decrypted: " << decrypted << endl;

if (sourceStr.compare(decrypted) == 0)

{

cout << "Decrypted!" << endl;

return;

}

}

return;

}

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

{

setlocale(LC\_ALL, "");

srand(time(NULL));

const int colsCount = 100;

string strToEncrypt;

cout << "Enter string (only latin):" << endl;

getline(cin, strToEncrypt);

string strToEncryptClean = strToEncrypt;

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

{

if (strToEncryptClean[i] == ' ')

{

strToEncryptClean.erase(i, 1);

i--;

}

}

// - 1

const string encrypted = encrypt(colsCount, strToEncryptClean);

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

const string decrypted = decrypt(colsCount, encrypted);

cout << "Decrypted: " << decrypted << endl;

// - 2

bruteForce(strToEncryptClean, colsCount);

// - 3

const int colsCount1 = 5;

const int colsCount2 = 3;

const string encryptedOnce = encrypt(colsCount1, strToEncryptClean);

cout << "Encrypted once: " << encryptedOnce << endl;

const string encryptedTwice = encrypt(colsCount2, encryptedOnce);

cout << "Encrypted twice: " << encryptedTwice << endl;

const string decryptedOnce = decrypt(colsCount2, encryptedTwice);

cout << "Decrypted once: " << decryptedOnce << endl;

const string decryptedTwice = decrypt(colsCount1, decryptedOnce);

cout << "Decrypted twice: " << decryptedTwice << endl;

return 0;

}