МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

ОДЕССКОГО НАЦИОНАЛЬНОГО УНИВЕРСИТЕТА ИМ. И.И. МЕЧНИКОВА

**Отчет по финальному проекту по дисциплине «Алгоритмы на графах»**

студента 3 курса

специальности

«Компьютерная инженерия»

Когана Владислава

преподаватель:

Антоненко А.С.

Одесса, 2020

В рамках данной работы создано двойное шифрование на основе алгоритмов простой замены, а именно шифра Цезаря и квадрата Полибия. Создан алгоритм, позволяющий достичь устойчивости к частотному анализу. Создан алгоритм, позволяющий взломать шифр методом грубой силы путем поиска N-грамм.

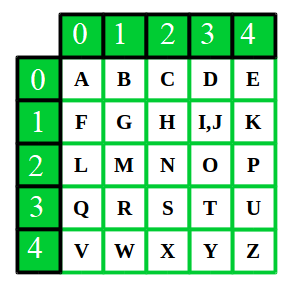
Однако последнее является следствием неотъемлемых качеств алгоритмов простой замены; все же следует отметить существенно большую устойчивость такого шифрования к взлому.

**Алгоритм**

Перед преобразованием в квадрат Полибия сообщение шифруется с помощью шифра Цезаря. При сдвиге на 1 A становится В, Z – A и так далее.

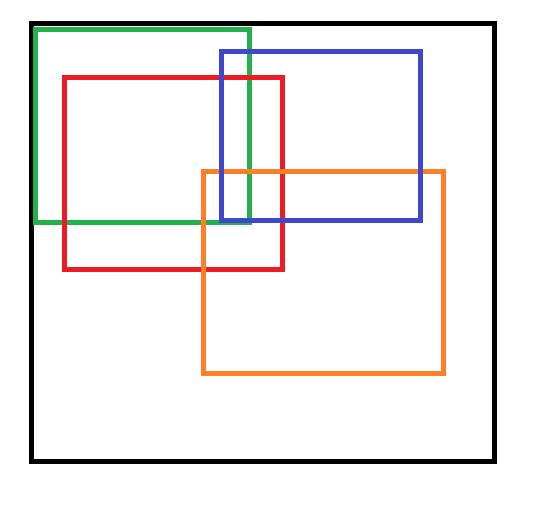
Квадрат Полибия основан на замене букв комбинацией из двух чисел. В рамках программы для этого будут реализованы два вектора с переменными типа int. Например A - (0,0), B - (0,1), H - (2,1) и так далее.

В латинском алфавите 26 букв, что очень близко к пяти в квадрате, поэтому при шифровании в квадрат Полибия зачастую преобразуют J в I. Тут же мы получаем одну из первых констант, фигурирующих в программе, polibius = 5.



Для того, чтобы это не было просто переписыванием чисел, была реализована матрица размером sizeCyph, обязательно превышающей 5. В рамках этой матрицы шифруемый квадрат будет сдвигаться по строкам и столбцам. Таким образом, при сдвиге на а=2 по строкам и b=3 по столбцам А будет уже не (0,0), а (2,3). Всего будет существовать ((sizeCyph-polibius)^2) вариантов комбинаций.

Таким образом, мы может зашифровать квадрат Полибия в синий квадрат, и для стороннего наблюдателя он на первый взгляд не будет отличаться от красного, оранжевого или зеленого.



Однако просто такой алгоритм был бы уязвим к частотному анализу, поэтому помимо значений из «настоящего» квадрата будет случайно создаваться «шум», записывающий ничего не значащие переменные, соответствующие случайным значениям из всей черной матрицы размером cyphSize, не входящей в синий квадрат (который мы шифруем). Таким образом, получатель при расшифровке, зная ключи доступа к синему квадрату, вообще не будет расшифровывать любые значения за его пределами. Для стороннего же наблюдателя, как указано раннее, подлинные значения неотличимы от фальшивых.

**Код**

В рамках программы создан ряд функций: ToSquare, formSquare, showSquare, caesarEncryption, caesarDecryption, polibiusEncryption, polibiusDecryption, findNgram и bruteforce. Кроме того, объявлены глобальные переменные sizeCyph и polibius.

#include <algorithm>

#include <iostream>

#include <vector>

#include <string>

const auto sizeCyph (10);

const auto polibius(5);

using namespace std;

string toSquare(string word); //Сокращение количество знаков сообщения до 25

void formSquare(bool sq[][sizeCyph], const int a, const int b); //Формирование таблицы подлинных символов

void showSquare(bool sq[][sizeCyph]); //Распечатка таблицы

string caesarEncryption(const string word, const int a);

string caesarDecryption(string word, const int key);

void polibiusEncryption(const bool sq[][sizeCyph], const string word, vector <int>& v1, vector <int>& v2, const int a, const int b); //Шифрование сообщения

string polibiusDecryption(const bool sq[][sizeCyph], const vector <int> v1, const vector <int> v2, const int a, const int b);

int findNgram(string word); //Поиcк N-грамм

void bruteforce(bool sq[][sizeCyph], const vector <int> v1, const vector <int> v2); //Алгоритм взлома

int main()

{

string word = "ABCDHIKMTHEWORDISSHEYZ";

auto a = 2; // any number 0 <= n <= polibius

cout << "A = "<< a << " THIS IS 1ST POLIBIUS KEY" << endl;

auto b = 3; // any number 0 <= n <= polibius

cout << "B = " << b << " THIS IS 2ND POLIBIUS KEY"<< endl;

auto c = 5; //any number 0 <= n <= 26

cout << "C = " << c << " THIS IS CAESAR KEY" << endl;

word = toSquare(word);

cout << word << endl;

word = caesarEncryption(word, c);

cout << "ENCRYPTED TO CAESAR" << endl << word << endl;

bool sq[sizeCyph][sizeCyph];

formSquare(sq, a, b);

cout << "THIS IS POLIBIUS SQUARE, ALL THAN ISN'T 1 IS NOISE" << endl;

showSquare(sq);

vector <int> v1 = {};

vector <int> v2 = {};

polibiusEncryption(sq, word, v1, v2, a, b);

cout << "ENCRYPTED TO CAESAR & POLIBIUS " << endl;

for (auto el : v1)

cout << el << " ";

cout << endl;

for (auto el : v2)

cout << el << " ";;

cout << endl<< "DECRYPTED FROM POLIBIUS" << endl;

cout << polibiusDecryption(sq, v1, v2, a, b) << endl;

cout << "DECRYPTED FROM POLIBIUS & CAESAR" << endl;

cout << caesarDecryption(polibiusDecryption(sq, v1, v2, a, b), c) << endl;

cout << endl;

bruteforce(sq, v1, v2);

return 0;

}

void formSquare(bool sq[][sizeCyph], const int a, const int b)

{

for (auto i = 0; i < sizeCyph; i++) {

for (auto j = 0; j < sizeCyph; j++) {

sq[i][j] = false;

}

}

for (auto i = a; i < a + polibius; i++) {

for (auto j = b; j < b + polibius; j++) {

sq[i][j] = true;

}

}

}

void showSquare(bool sq[][sizeCyph])

{

for (auto i = 0; i < sizeCyph; i++) {

for (auto j = 0; j < sizeCyph; j++) {

cout << sq[i][j] << " ";

}

cout << endl;

}

}

string toSquare(string word)

{ //Сокращение количество знаков сообщения до 25

string alp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

string al = "JI";

for (auto i = 0; i < word.size(); i++) {

if (word[i] == al[0])

word[i] = al[1];

}

return word;

}

string caesarEncryption(string word, const int key)

{

string alp = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

string result = "";

for (auto i = 0; i < word.size(); i++)

for (auto j = 0; j < alp.size(); j++)

if (word[i] == alp[j])

result+=alp[(j+key)%alp.size()];

return result;

}

string caesarDecryption(string word, const int key)

{

string alp = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

string result = "";

for (auto i = 0; i < word.size(); i++)

for (auto j = 0; j < alp.size(); j++)

if (word[i] == alp[j])

result += alp[(alp.size() + (j - key)) % alp.size()];

return result;

}

void polibiusEncryption(const bool sq[][sizeCyph], const string word, vector <int>& v1, vector <int>& v2, const int a, const int b)

{

string alp = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

vector <int> wordnum = {};

for (auto i = 0; i < word.size(); i++)

for (auto j = 0; j < alp.size(); j++)

if (word[i] == alp[j])

wordnum.push\_back(j); //Преобразуем исходную строку в вектор чисел по номерам букв\*/

auto c = 0; //Счетчик добавленных символов

while (c < wordnum.size()) {

auto tmp0 = rand() % sizeCyph;

auto tmp1 = rand() % sizeCyph;

if (sq[tmp0][tmp1]) { //Кодируем подлинные знаки в квадрат Полибия

v1.push\_back((wordnum[c] / polibius) + a); //Строка

v2.push\_back((wordnum[c] % polibius) + b); //Столбец

c++;

}

else { //Иначе шум

v1.push\_back(tmp0); //Строка

v2.push\_back(tmp1); //Столбец

}

}

}

string polibiusDecryption(const bool sq[][sizeCyph], const vector <int> v1, const vector <int> v2, const int a, const int b)

{

string alpModified = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

string result = "";

for (int i = 0; i < v1.size(); i++) {

if (sq[v1[i]][v2[i]]) {

//cout << (v1[i] - a) \* polibius + (v2[i] - b) << endl;

result += alpModified[(v1[i] - a) \* polibius + (v2[i] - b)];

}

}

return result;

}

int findNgram(string word)

{

int result = 0;

string ngrams[] = { "THE", "HIS", "ITI", "HER", "SHE", "YOU", "THI", "WOR", "TES" };

for (auto i : ngrams) {

auto tmp = word.find(i);

if (tmp<word.size()) { //В противном случае выведется максимальное значение

result++;

}

}

return result;

}

void bruteforce(bool sq[][sizeCyph], const vector <int> v1, const vector <int> v2)

{

vector <int> result = {};

int c = 0;

for (auto key0=0;key0<(sizeCyph-polibius); key0++) //Не выйти за рамки

for (auto key1=0; key1 <(sizeCyph - polibius); key1++) //Не выйти за рамки

for (auto key2=0; key2 < (pow(polibius,2)); key2++) {

formSquare(sq, key0, key1);

//cout << "KEY0 " << key0 << " KEY1 " << key1 << endl;

string pd = polibiusDecryption(sq, v1, v2, key0, key1);

string cd = caesarDecryption(pd, key2);

if (findNgram(cd) > c) {

c = findNgram(cd);

result = { key0,key1,key2 };

}

}

cout<< endl;

if (c) {

cout << "BRUTEFORCED, KEY0, KEY1 AND KEY 2 IS " << endl;

for (auto el : result)

cout << el << endl;

}

else {

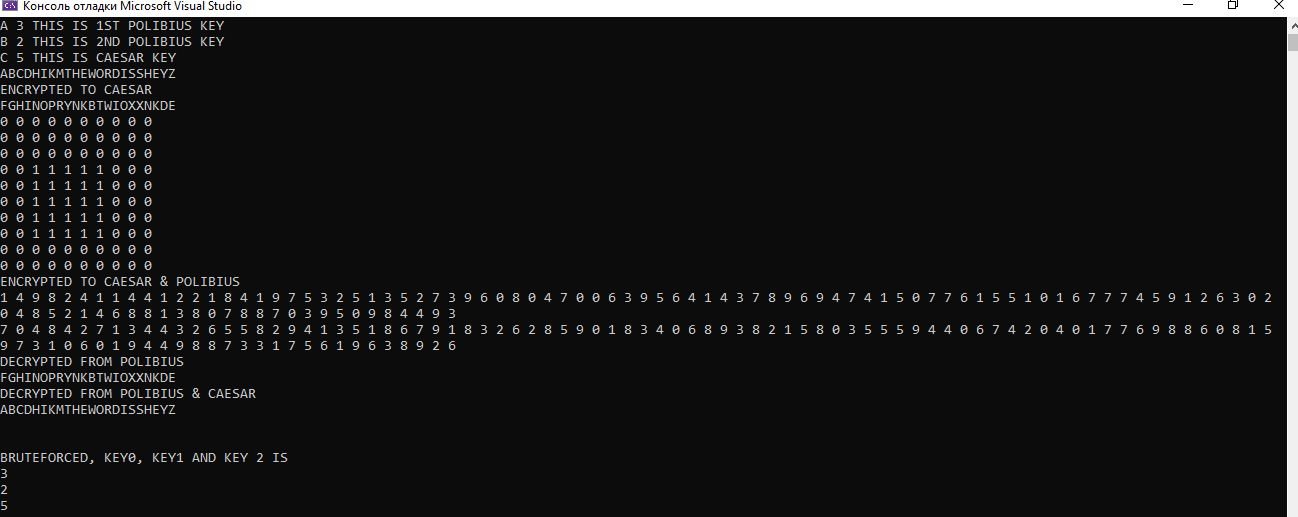
cout<< "BRUTEFORCE FAILED"<<endl;

}

}

;

Результат выполнения программы



Рассмотрим сами функции.

void formSquare(bool sq[][sizeCyph], const int a, const int b)

{

for (auto i = 0; i < sizeCyph; i++) {

for (auto j = 0; j < sizeCyph; j++) {

sq[i][j] = false;

}

}

for (auto i = a; i < a + polibius; i++) {

for (auto j = b; j < b + polibius; j++) {

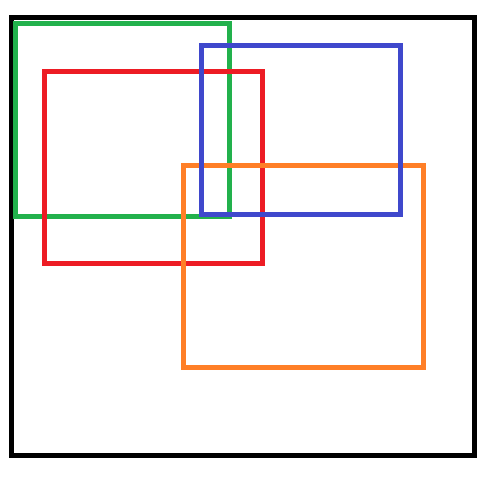
sq[i][j] = true;

}

}

}

Формирует таблицу истинных значений по ключам a и b. В рамках ранее показанной иллюстрации это означает, что в случае если мы шифруем в «синий» квадрат все значения в рамках «синего» квадрата отмечены единицами, а вне него – нулями.



void showSquare(bool sq[][sizeCyph])

{

for (auto i = 0; i < sizeCyph; i++) {

for (auto j = 0; j < sizeCyph; j++) {

cout << sq[i][j] << " ";

}

cout << endl;

}

}

Печатает таблицу истинности.

string toSquare(string word)

{ //Сокращение количество знаков сообщения до 25

string alp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

string al = "JI";

for (auto i = 0; i < word.size(); i++) {

if (word[i] == al[0])

word[i] = al[1];

}

return word;

}

Как было указано ранее, для квадрата Полибия нужно 25 символов, поэтому J преобразуется в I. Возвращает урезанную таким образом строку.

string caesarEncryption(string word, const int key)

{

string alp = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

string result = "";

for (auto i = 0; i < word.size(); i++)

for (auto j = 0; j < alp.size(); j++)

if (word[i] == alp[j])

result+=alp[(j+key)%alp.size()];

return result;

}

Шифрует строку word в шифр Цезаря со сдвигом key. Возвращает зашифрованную строку.

string caesarDecryption(string word, const int key)

{

string alp = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

string result = "";

for (auto i = 0; i < word.size(); i++)

for (auto j = 0; j < alp.size(); j++)

if (word[i] == alp[j])

result += alp[(alp.size() + (j - key)) % alp.size()];

return result;

}

Дешифрует строку word по ключу key. Возвращает дешифрованную строку.

void polibiusEncryption(const bool sq[][sizeCyph], const string word, vector <int>& v1, vector <int>& v2, const int a, const int b)

{

string alp = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

vector <int> wordnum = {};

for (auto i = 0; i < word.size(); i++)

for (auto j = 0; j < alp.size(); j++)

if (word[i] == alp[j])

wordnum.push\_back(j); //Преобразуем исходную строку в вектор чисел по номерам букв\*/

auto c = 0; //Счетчик добавленных символов

while (c < wordnum.size()) {

auto tmp0 = rand() % sizeCyph;

auto tmp1 = rand() % sizeCyph;

if (sq[tmp0][tmp1]) { //Кодируем подлинные знаки в квадрат Полибия

v1.push\_back((wordnum[c] / polibius) + a); //Строка

v2.push\_back((wordnum[c] % polibius) + b); //Столбец

c++;

}

else { //Иначе шум

v1.push\_back(tmp0); //Строка

v2.push\_back(tmp1); //Столбец

}

}

}

Здесь следует рассмотреть подробнее. Вектора v1 и v2, передаваемые по ссылке, отвечают за координаты символов в рамках шифра Полибия. Как было сказано ранее, в самом простом варианте символ В будет записано как (0,1). В рамках данной программы это бы означало, что v1[i] = 0, v2[i] = 1, где i – это номер символа В в сообщении.

Однако мы не просто перезаписываем их таким образом, а со сдвигом на a строк и b столбцов.

В рамках функции нумеруем все символы строки word в вектор wordnum. А=0, B =1, …Y=23, Z=24.

Затем идет цикл while, продолжающийся пока не будет зашифровано все слово. В рамках цикла while через rand() генерируются случайные координаты из таблицы. Если они совпадают с областью, в которую мы шифруем сообщение – в вектора через push\_back() записывается символ из шифруемого сообщения. В противном случае записываются ничего не значащие случайно сгенерированные значения.

Отдельно рассмотрим формулу добавления.

v1.push\_back((wordnum[c] / polibius) + a); //Строка

v2.push\_back((wordnum[c] % polibius) + b); //Столбец

Допустим, мы шифруем букву H со смещением на a по строке и b по столбцу. До смещения эта буква имела бы координаты (1,2), являясь 7-ой по счету с нуля в латинском алфавите. Таким образом, строка была бы равна (7/5)=1, а столбец (7%5=2). 7 в формуле соответствует wordnum[i], 5 – значение константы polibius, и уже дополнительно мы смещаем на a знаков по строке и на b знаков по столбцам.

Функция void, но изменяет переданные по ссылке значения векторов v1 и v2.

string polibiusDecryption(const bool sq[][sizeCyph], const vector <int> v1, const vector <int> v2, const int a, const int b)

{

string alpModified = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

string result = "";

for (int i = 0; i < v1.size(); i++) {

if (sq[v1[i]][v2[i]]) {

result += alpModified[(v1[i] - a) \* polibius + (v2[i] - b)];

}

}

return result;

}

Дешифратор. Дешифруются только значения, соответствующие таблице истинности sq. Возвращается дешифрованная строка.

int findNgram(string word)

{

int result = 0;

string ngrams[] = { "THE", "HIS", "ITI", "HER", "SHE", "YOU", "THI", "WOR", "TES" };

for (auto i : ngrams) {

auto tmp = word.find(i);

if (tmp<word.size()) { //В противном случае выведется максимальное значение

result++;

}

}

return result;

}

Поиск N-грамм. Для примера использован ряд употребляемых в английском триграмм, при желании список может быть расширен, но в рамках программы этого более чем достаточного. Функция возвращает количество найденных триграмм.

void bruteforce(bool sq[][sizeCyph], const vector <int> v1, const vector <int> v2)

{

vector <int> result = {};

int c = 0;

for (auto key0=0;key0<(sizeCyph-polibius); key0++) //Не выйти за рамки

for (auto key1=0; key1 <(sizeCyph - polibius); key1++) //Не выйти за рамки

for (auto key2=0; key2 < (pow(polibius,2)); key2++) {

formSquare(sq, key0, key1);

//cout << "KEY0 " << key0 << " KEY1 " << key1 << endl;

string pd = polibiusDecryption(sq, v1, v2, key0, key1);

string cd = caesarDecryption(pd, key2);

if (findNgram(cd) > c) {

c = findNgram(cd);

result = { key0,key1,key2 };

}

}

cout<< endl;

if (c) {

cout << "BRUTEFORCED, KEY0, KEY1 AND KEY 2 IS " << endl;

for (auto el : result)

cout << el << endl;

}

else {

cout<< "BRUTEFORCE FAILED"<<endl;

}  
}

Взламывает шифр методом грубой силы. key0 – сдвиг по строкам, key1 – сдвиг по столбцам, key2 – сдвиг шифра Цезаря.

Перебирает все возможные комбинации, в случае нахождения комбинации с б**о**льшим количеством N-грамм запоминает значения ключей. Печатает ключи, соответствующие максимальному количеству содержащихся в сообщении N-грамм. В случае, если не найдена ни одна N-грамма, печатает сообщение о неудаче.

Возвращаясь к main

int main()

{

string word = "ABCDHIKMTHEWORDISSHEYZ";

auto a = 3;

cout << "A "<< a << " THIS IS 1ST POLIBIUS KEY" << endl;

auto b = 2;

cout << "B " << b << " THIS IS 2ND POLIBIUS KEY"<< endl;

auto c = 5;

cout << "C " << c << " THIS IS CAESAR KEY" << endl;

word = toSquare(word);

cout << word << endl;

word = caesarEncryption(word, c);

cout << "ENCRYPTED TO CAESAR" << endl << word << endl;

bool sq[sizeCyph][sizeCyph];

formSquare(sq, a, b);

showSquare(sq);

vector <int> v1 = {};

vector <int> v2 = {};

polibiusEncryption(sq, word, v1, v2, a, b);

cout << "ENCRYPTED TO CAESAR & POLIBIUS " << endl;

for (auto el : v1)

cout << el << " ";

cout << endl;

for (auto el : v2)

cout << el << " ";;

cout << endl<< "DECRYPTED FROM POLIBIUS" << endl;

cout << polibiusDecryption(sq, v1, v2, a, b) << endl;

cout << "DECRYPTED FROM POLIBIUS & CAESAR" << endl;

cout << caesarDecryption(polibiusDecryption(sq, v1, v2, a, b), c) << endl;

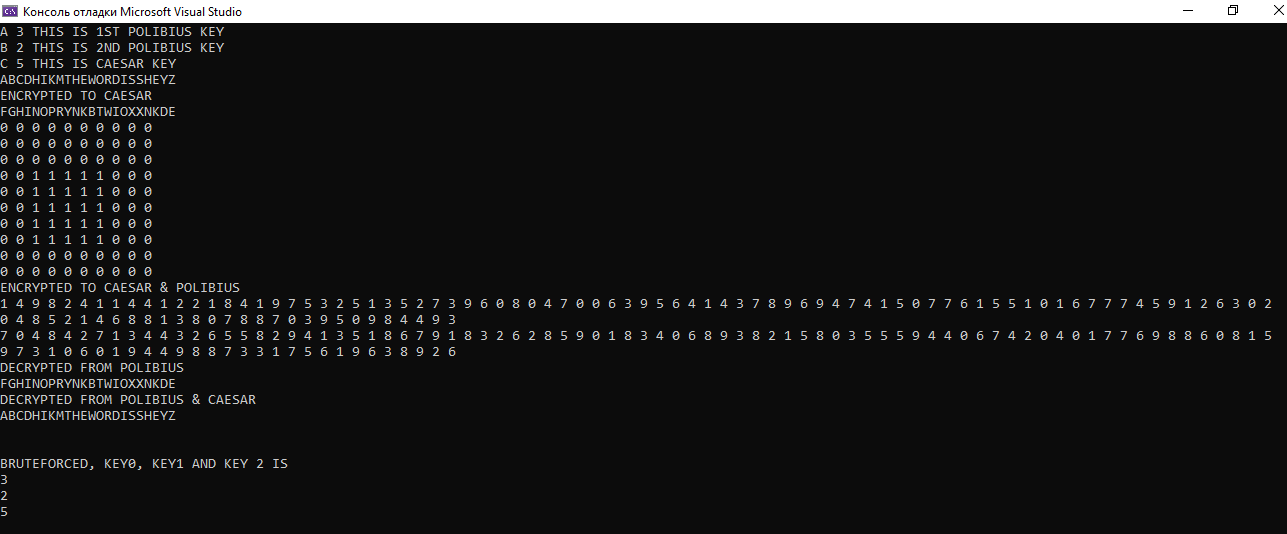
cout << endl;

bruteforce(sq, v1, v2);

return 0;

}

Сначала объявляется шифруемое сообщение word. Затем объявляются ключи a (сдвиг по строкам) и b (сдвиг по столбцам), а также с (сдвиг по шифру Цезаря). По значениям a и b формируется таблица истинности sq через функцию formSquare(), которая после этого печатается через showSquare().

Строка word модифицируется с помощью toSquare() и шифруется через caesarEncryption(), затем зашифрованное таким образом слово шифруется через polibiusEncryption(). Затем печатается зашифрованное сообщение – то есть печатаются вектора v1 и v2. После чего мы дешифруем сообщения через polibiusDecryption() и caesarDecryption() указывая ключи и, наконец, взламываем шифр через bruteforce().

Программа выполнена.

Список литературы

1. Введение в криптографию и шифрование https://habr.com/ru/company/yandex/blog/324866/
2. Квадрат Полибия <https://questhint.ru/kvadrat-polibiya/>
3. Квадрат Полибия <https://ru.wikipedia.org/wiki/Квадрат_Полибия>
4. Частотный анализ <https://habr.com/ru/post/513926/>
5. Поиск N-грамм <https://habr.com/ru/post/114997/>