Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра Программной Инженерии

Практическое занятие №4

По дисциплине «Основы информационной безопасности»

На тему **«Криптографическая защита информации»**

Выполнил:

Студент 2 курса 3 группы

Шутро А.С.

Преподаватель: Ржеутская Н.В.

Минск 2023

**Теоретическое введение**

Криптография - наука о методах обеспечения конфиденциальности (невозможности прочтения информации посторонним) и аутентичности (целостности и подлинности авторства) информации.

Изначально криптография изучала методы шифрования информации – обратимого преобразования открытого (исходного) текста на основе секретного алгоритма и/или ключа в шифрованный текст (шифротекст). Традиционная криптография образует раздел симметричных криптосистем, в которых зашифрование и расшифрование проводится с использованием одного и того же секретного ключа.

Помимо этого современная криптография включает в себя асимметричные криптосистемы, системы электронной цифровой подписи, хеш-функции, управление ключами, получение скрытой информации, квантовую криптографию.

Шифрованием (encryption) называют процесс преобразования открытых данных (plaintext) в зашифрованные (шифртекст, ciphertext) или зашифрованных данных в открытые по определенным правилам с применением ключей.

В полиалфавитных подстановках для замены некоторого символа исходного сообщения в каждом случае его появления последовательно используются различные символы из некоторого набора. Понятно, что этот набор не бесконечен, через какое-то количество символов его нужно использовать снова. В этом слабость чисто полиалфавитных шифров.

В современных криптографических системах, как правило, используют оба способа шифрования (замены и перестановки). Такой шифратор называют составным (product cipher). Oн более стойкий, чем шифратор, использующий только замены или перестановки.

В асимметричных алгоритмах шифрования (или криптографии с открытым ключом) для зашифровывания информации используют один ключ (открытый), а для расшифровывания - другой (секретный). Эти ключи различны и не могут быть получены один из другого.

Схема обмена информацией такова:

· получатель вычисляет открытый и секретный ключи, секретный ключ хранит в тайне, открытый же делает доступным (сообщает отправителю, группе пользователей сети, публикует);

· отправитель, используя открытый ключ получателя, зашифровывает сообщение, которое пересылается получателю;

· получатель получает сообщение и расшифровывает его, используя свой секретный ключ.

**Практическая работа 4.1**

**Задание к выполнению**

1. Изучить теоретические сведения по данной теме.
2. Зашифровать сообщение с использованием шифра Цезаря, Трисемуса, Плейфейра и Вижинера и полученного секретного ключа (по номеру варианта и ключевому слову «Защита»). В качестве сообщения использовать свою Фамилию Имя Отчество.
3. Расшифровать следующие сообщения:

Сообщение: **Ие михежцчжшйч сшихуцчб**

Способ**: Расшифровать с помощью шифра Цезаря. Ключ 5**

#include <iostream>

#include <vector>

#include <typeinfo>

#include <unordered\_map>

#include <string>

#define watch(x) std::cout << typeid(x).name() << " " << #x << " = " << x << std::endl;

//2

std::vector<std::vector<char>> generateKeyMatrix(const std::string& key, size\_t width, size\_t height, char startLatter, char finishLatter) {

std::vector<std::vector<char>> matrix(height, std::vector<char>(width, ' '));

std::string uniqueKey = "";

for (char ch : key) {

if (ch != ' ' && uniqueKey.find(ch) == std::string::npos)

uniqueKey += ch;

}

int row = 0, col = 0;

for (char ch : uniqueKey) {

matrix[row][col] = ch;

col++;

if (col == width) {

col = 0;

row++;

}

}

char currentChar = startLatter;

for (row = 0; row < height; row++) {

for (col = 0; col < width; col++) {

if (matrix[row][col] == ' ') {

while (uniqueKey.find(currentChar) != std::string::npos ||

currentChar == finishLatter) {

currentChar++;

}

matrix[row][col] = currentChar;

currentChar++;

}

}

}

return matrix;

}

void findCoordinates(const std::vector<std::vector<char>>& matrix, char ch, int& row, int& col) {

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

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

if (matrix[row][col] == ch) {

return;

}

}

}

}

std::string trisEncrypt(const std::string& text, const std::vector<std::vector<char>>& matrix, size\_t width) {

std::string encryptedText = "";

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

char firstChar = text[i];

int row1 = 0, col1 = 0;

findCoordinates(matrix, firstChar, row1, col1);

encryptedText += matrix[(row1 + 1) % width][col1];

}

return encryptedText;

}

//3

std::string playfairEncrypt(const std::string& plaintext, const std::vector<std::vector<char>>& matrix) {

std::string encryptedText = "";

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

char firstChar = plaintext[i];

char secondChar = (i + 1 < plaintext.length()) ? plaintext[i + 1] : 'X';

int row1, col1, row2, col2;

findCoordinates(matrix, firstChar, row1, col1);

findCoordinates(matrix, secondChar, row2, col2);

if (row1 == row2) {

encryptedText += matrix[row1][(col1 + 1) % 5];

encryptedText += matrix[row2][(col2 + 1) % 5];

}

else if (col1 == col2) {

encryptedText += matrix[(row1 + 1) % 5][col1];

encryptedText += matrix[(row2 + 1) % 5][col2];

}

else {

encryptedText += matrix[row1][col2];

encryptedText += matrix[row2][col1];

}

}

return encryptedText;

}

//4

int mislower(int \_C) {

return (\_C >= (char)'а' && \_C <= (char)'я') ? \_C : 0;

}

int mtolower(int \_C) {

return mislower(\_C) > 0 ? \_C : (\_C + ((char)'я' - (char)'а'));

}

std::string encryptVigenere(const std::string& plaintext, const std::string& key) {

std::string ciphertext = "";

int keyLength = key.length();

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

char plainChar = plaintext[i];

char keyChar = key[i % keyLength];

if ((plainChar >= (char)'а' && plainChar <= (char)'я') || (plainChar >= (char)'А' && plainChar <= (char)'Я')) {

char base = mislower(plainChar) > 0 ? char('а') : char('А');

char shift = mtolower(keyChar) - char('а');

char encryptedChar = static\_cast<char>((plainChar - base + shift) % ((char)'я' - (char)'а') + base);

ciphertext += encryptedChar;

}

else {

ciphertext += plainChar;

}

}

return ciphertext;

}

#define METHOD(name) std::cout << std::endl << "========================- " << name << " -========================" << std::endl;

int main(int argc, char\* argv[]) {

setlocale(LC\_ALL, "Rus");

// 1

METHOD("Цезарь");

int shift = 4;

std::string text\_C = "Шутро Александр Сергеевич";

std::cout << "Исходный текст: " << text\_C << std::endl;

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

text\_C[i] += shift;

}

std::cout << "Зашифрованный текст: " << text\_C << std::endl;

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

text\_C[i] -= shift;

}

std::cout << "Расшифрованный текст: " << text\_C << std::endl;

// 2

METHOD("Трисемуса");

std::string key\_T = "KEYWORD";

std::vector<std::vector<char>> keyMatrix\_T = generateKeyMatrix(key\_T, 5, 5, 'A', 'J');

for (auto line : keyMatrix\_T) {

for (auto element : line) {

std::cout << element;

}

std::cout << std::endl;

}

std::string plaintext\_T = "SHUTRO";

std::string encryptedText\_T = trisEncrypt(plaintext\_T, keyMatrix\_T, 5);

std::cout << "Исходный текст: " << plaintext\_T << std::endl;

std::cout << "Зашифрованный текст: " << encryptedText\_T << std::endl;

// 3

METHOD("Плейфейра");

std::string key\_P = "KEYWORD";

std::vector<std::vector<char>> keyMatrix\_P = generateKeyMatrix(key\_P, 5, 5, 'A', 'J');

std::string plaintext\_P = "SHUTRO";

std::string ciphertext\_P = playfairEncrypt(plaintext\_P, keyMatrix\_P);

std::cout << "Исходный текст: " << plaintext\_P << std::endl;

std::cout << "Зашифрованный текст: " << ciphertext\_P << std::endl;

// 4

METHOD("Вижинера");

std::string \_key = "Защита";

std::string \_plaintext = "Шутро Александр Сергеевич";

std::string \_ciphertext = encryptVigenere(\_plaintext, \_key);

std::cout << "Исходный текст: " << \_plaintext << std::endl;

std::cout << "Зашифрованный текст: " << \_ciphertext << std::endl;

//3 задание

std::string text\_C = "ие михежцчжшйч сшихуцчб";

std::cout << "Зашифрованный текст: " << text\_C << std::endl;

std::string text = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";

for(int shift = 1; shift < 33; shift++) {

std::string text\_D = text\_C;

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

if (text\_D[i] == ' ') {

}

else {

int index\_text = 0;

//нахожу индекс по своему алфавиту

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

if (text\_D[i] == text[x])

index\_text = x;

}

//проверка на выход за text.size()

if (index\_text + shift +1 <= text.size()) {

text\_D[i] = text[index\_text + shift];

}

else {

text\_D[i] = text[index\_text + shift - text.size()];

}

}

}

std::cout << shift << " - " << "Расшифрованный текст: " << text\_D << std::endl;

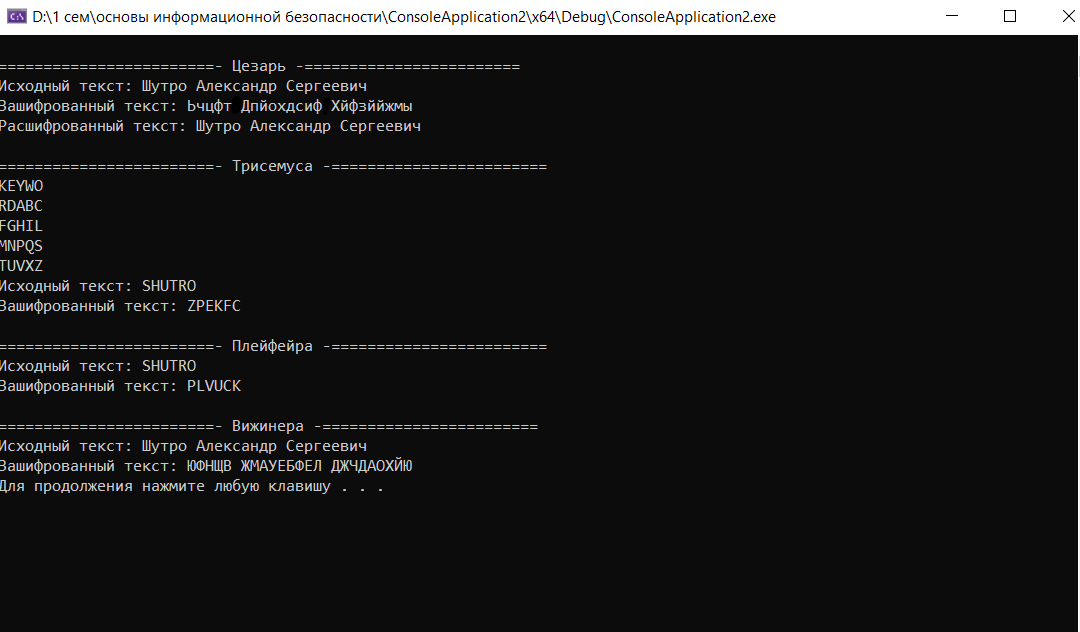
}

system("pause");

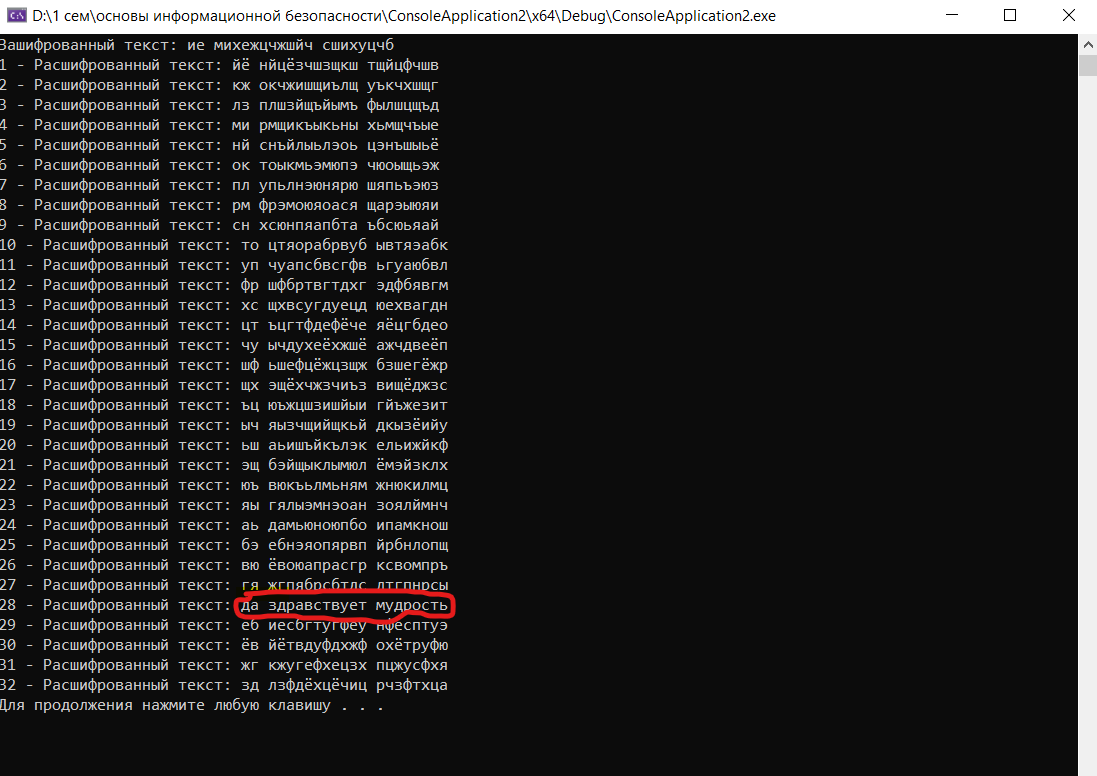
return 0;

}

Результат шифров Цезаря, Трисемуса, Плейфейра, Вижинера



Результат расшифровки



**Практическая работа 4.2**

**RSA (Rivest-Shamir-Adleman):**

**Генерация ключей:** В начале процесса генерируются два простых числа, p и q. Затем вычисляется модуль n = p \* q и значение функции Эйлера φ(n) = (p-1)(q-1). Далее выбирается открытая экспонента e, которая должна быть взаимно простой с φ(n). Закрытая экспонента d вычисляется так, чтобы (e \* d) mod φ(n) = 1.

**Шифрование:** Для зашифрования сообщения M отправитель использует открытый ключ (n, e). Сообщение M возводится в степень e по модулю n: C = M^e mod n**.**

**Дешифрование:** Получатель использует свой закрытый ключ (n, d) для дешифрования. Шифротекст C возводится в степень d по модулю n: M = C^d mod n**.**

**Подпись:** Чтобы создать цифровую подпись сообщения M, отправитель использует свой закрытый ключ (n, d). Подпись S вычисляется как S = M^d mod n. Получатель может проверить подпись, используя открытый ключ отправителя и сравнив S с M.

**Принцип работы RSA основан на сложности факторизации больших чисел.**

**Диффи-Хеллман (Diffie-Hellman):**

**Генерация параметров:** Две стороны (например, Алиса и Боб) согласовывают общие параметры: большое простое число p и примитивный корень по модулю p, g. Параметры p и g являются общедоступными и известны обеим сторонам**.**

**Выбор секретных ключей:** Каждая сторона генерирует свой секретный ключ (x для Алисы и y для Боба).

**Вычисление общего ключа:** Алиса и Боб вычисляют общие открытые ключи: Алиса вычисляет g^x mod p, а Боб вычисляет g^y mod p. Эти значения обмениваются между ними. Затем обе стороны вычисляют общий секретный ключ, который равен (g^x)^y mod p = (g^y)^x mod p**.**

**Принцип работы Диффи-Хеллмана основан на трудности вычисления дискретного логарифма.**

**Эль-Гамаля (ElGamal):**

**Генерация ключей:** Генерируются параметры - большое простое число p и примитивный корень по модулю p, g. Затем генерируется закрытый ключ x, который является случайным числом в пределах [1, p-1]. Открытый ключ y вычисляется как g^x mod p.

**Шифрование:** Для зашифрования сообщения M отправитель выбирает случайное k и вычисляет a = g^k mod p и b = (y^k \* M) mod p. Затем отправляет пару (a, b) получателю.

**Дешифрование:** Получатель использует свой закрытый ключ x для дешифрования. Он вычисляет s = a^x mod p и затем находит обратное значение s^(-1) mod p. Исходное сообщение восстанавливается как M = (b \* s^(-1)) mod p.

**Принцип работы Эль-Гамаля основан на сложности дискретного логарифмирования и обратной операции в конечном поле.**

#include <iostream>

#include <cmath>

// Функция для нахождения НОД двух чисел

int gcd(int a, int b) {

if (b == 0) {

return a;

}

return gcd(b, a % b);

}

// Функция для вычисления обратного элемента по модулю

int mod\_inverse(int a, int m) {

for (int x = 1; x < m; x++) {

if ((a \* x) % m == 1) {

return x;

}

}

return -1; // обратного элемента не существует

}

// Функция для вычисления (base^exponent) % modulus

int mod\_pow(int base, int exponent, int modulus) {

int result = 1;

base = base % modulus;

while (exponent > 0) {

if (exponent % 2 == 1) {

result = (result \* base) % modulus;

}

base = (base \* base) % modulus;

exponent /= 2;

}

return result;

}

// Алгоритм RSA

struct RSAKeys {

int n; // модуль

int e; // открытая экспонента

int d; // закрытая экспонента

};

RSAKeys generateRSAKeys(int p, int q) {

RSAKeys keys;

keys.n = p \* q;

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

// Выбор открытой экспоненты e (обычно фиксированное значение, например, 65537)

keys.e = 65537;

// Вычисление закрытой экспоненты d

keys.d = mod\_inverse(keys.e, phi);

return keys;

}

int rsa\_encrypt(int plaintext, int e, int n) {

return mod\_pow(plaintext, e, n);

}

int rsa\_decrypt(int ciphertext, int d, int n) {

return mod\_pow(ciphertext, d, n);

}

// Алгоритм Диффи-Хеллмана

int diffie\_hellman(int p, int g, int private\_key) {

return mod\_pow(g, private\_key, p);

}

// Алгоритм Эль-Гамаля

struct ElGamalKeys {

int p; // большое простое число

int g; // примитивный корень по модулю p

int y; // открытый ключ (g^x mod p)

int x; // закрытый ключ

};

ElGamalKeys generateElGamalKeys(int p, int g, int x) {

ElGamalKeys keys;

keys.p = p;

keys.g = g;

keys.x = x;

keys.y = mod\_pow(g, x, p);

return keys;

}

struct ElGamalCiphertext {

int a;

int b;

};

ElGamalCiphertext elgamal\_encrypt(int plaintext, int p, int g, int y) {

int k = rand() % (p - 2) + 1; // случайное k, 1 <= k <= p-2

int a = mod\_pow(g, k, p);

int b = (mod\_pow(y, k, p) \* plaintext) % p;

return { a, b };

}

int elgamal\_decrypt(ElGamalCiphertext ciphertext, int p, int x) {

int a = ciphertext.a;

int b = ciphertext.b;

int s = mod\_pow(a, x, p);

int plaintext = (b \* mod\_inverse(s, p)) % p;

return plaintext;

}

int main() {

// RSA

int p = 61;

int q = 53;

RSAKeys rsaKeys = generateRSAKeys(p, q);

int plaintext\_rsa = 42;

int encrypted\_rsa = rsa\_encrypt(plaintext\_rsa, rsaKeys.e, rsaKeys.n);

int decrypted\_rsa = rsa\_decrypt(encrypted\_rsa, rsaKeys.d, rsaKeys.n);

std::cout << "RSA:" << std::endl;

std::cout << "Original: " << plaintext\_rsa << std::endl;

std::cout << "Encrypted: " << encrypted\_rsa << std::endl;

std::cout << "Decrypted: " << decrypted\_rsa << std::endl;

// Диффи-Хеллман

int dh\_p = 23;

int dh\_g = 5;

int dh\_privateAlice = 6;

int dh\_privateBob = 15;

int dh\_publicAlice = diffie\_hellman(dh\_p, dh\_g, dh\_privateAlice);

int dh\_publicBob = diffie\_hellman(dh\_p, dh\_g, dh\_privateBob);

int dh\_sharedSecretAlice = diffie\_hellman(dh\_p, dh\_publicBob, dh\_privateAlice);

int dh\_sharedSecretBob = diffie\_hellman(dh\_p, dh\_publicAlice, dh\_privateBob);

std::cout << "Diffie-Hellman:" << std::endl;

std::cout << "Shared Secret (Alice): " << dh\_sharedSecretAlice << std::endl;

std::cout << "Shared Secret (Bob): " << dh\_sharedSecretBob << std::endl;

// Эль-Гамаля

int elgamal\_p = 29;

int elgamal\_g = 2;

int elgamal\_x = 15;

ElGamalKeys elgamalKeys = generateElGamalKeys(elgamal\_p, elgamal\_g, elgamal\_x);

int plaintext\_elgamal = 18;

ElGamalCiphertext elgamalCiphertext = elgamal\_encrypt(plaintext\_elgamal, elgamalKeys.p, elgamalKeys.g, elgamalKeys.y);

int elgamalDecrypted = elgamal\_decrypt(elgamalCiphertext, elgamalKeys.p, elgamalKeys.x);

std::cout << "ElGamal:" << std::endl;

std::cout << "Original: " << plaintext\_elgamal << std::endl;

std::cout << "Decrypted: " << elgamalDecrypted << std::endl;

return 0;

}

