Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Вычислительной математики и информационных технологий

Отчет

учебной практики

(технологической (проектно-технологической) практики)

Обучающийся Гофуров\_Исломбек\_Элмурод\_угли, гр. 09-033 \_\_\_\_\_\_\_

(ФИО студента) (Группа) (Подпись)

Руководитель практики от КФУ:

Старший преподаватель кафедры САИТ Долгов Д.А.

Оценка за практику \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Дата сдачи отчета «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2022 г.

Казань – 2022

Оглавление

1. [ВВЕДЕНИЕ 3](#_Toc103934794)
2. [ТЕОРИТИЧЕСКАЯ ЧАСТЬ 4](#_Toc103934795)

[Поставленные задачи 4](#_Toc103934796)

[Язык программирования 4](#_Toc103934797)

[История RSA 4](#_Toc103934798)

[Описание алгоритма шифрования RSA 4](#_Toc103934799)

1. [Ход работы 7](#_Toc103934800)

[Проблемы в реализации 9](#_Toc103934801)

1. [ЗАКЛЮЧЕНИЕ 10](#_Toc103934802)
2. [СПИСОК ЛИТЕРАТУРЫ 11](#_Toc103934803)

ВВЕДЕНИЕ

Проектно-технологическая практика проходила на кафедре системного анализа и информационных технологий Института вычислительной математики и информационных технологий КФУ с 09 февраля 2022 по 07 июня 2022 года.

Целью практики является получение компетенций в области информационной безопасности и криптографии, в том числе изучение основных теоретико-числовых и криптографических алгоритмов.

В данном отчёте рассказывается о выбранном алгоритме шифрования, об истории данного алгоритма, о теоритической части данного алгоритма, о ходе работы и о возникших проблемах при реализации.

Для выбора были представлены такие алгоритмы как «частотный анализ», «RSA», «атака на RSA». После долгих дискуссий был выбран алгоритм шифрования с открытым ключом «RSA».

ТЕОРИТИЧЕСКАЯ ЧАСТЬ

Поставленные задачи

Получить навыки создания алгоритма шифрования RSA и написание полноценного алгоритма на одном из языков программирования.

Язык программирования

Для написания данного алгоритма шифрования было решено использовать компилируемый, статически типизированный язык программирования общего назначения С++ (читается си-плюс-плюс).

Для написания кода была выбрана такая среда разработки для языков программирования как CLion. Была выбрана по причине лёгкого и удобного написания и редактирования кода.

История RSA

Наряду с обычным шифрованием на основе засекреченного ключа в последние годы все большее признание получают системы шифрования с открытым ключом. В таких системах используются два ключа. Информация шифруется с помощью открытого ключа, а расшифровывается с использованием секретного ключа. В настоящее время наиболее эффективным и распространенным алгоритмом шифрования с открытым ключом является алгоритм RSA. Данный алгоритм стал первой системой подходящей как для шифрования, так и для цифровой подписи. Идея алгоритма с открытым и закрытым ключом приписывается Уитфилду Диффи и Мартину Хеллману, опубликовавшие эту концепцию в 1976 году. А уже в августе 1977 года в колонке «Математические игры» Мартина Гарднера в журнале Scientific American с разрешения Рональда Ривеста появилось первое описание криптосистемы RSA.

Описание алгоритма шифрования RSA

Скажем Стив хочет получать зашифрованные сообщения, которые только она могла бы расшифровать и прочесть. В этом случае Стив подбирает два простых числа (желательно больших) «p» и «q». Данные числа она не кому не говорит а вот их произведение она публикует.

Скажем произведение это N = p \* q .

N- во многих источниках известен как «модуль алгоритма»

После он должен найти по функции Эйлера «pi» (читается как: “фи”)

Pi = (p-1)\*(q-1)

Далее Стив подбирает такое «Е», удовлетворяющую условию НОД (Е, pi)=1

\*так же число «Е» должно быть простым

Пара «N» и «Е» - доступны всем.

Для получения секретного ключа Стив применяет расширенный алгоритм Евклида к таким числам как «Е» и «pi». При помощи данного алгоритма он получает расшифровывающую экспоненту «d». Она должна удовлетворять следующему соотношению: E \* d= 1(mod (pi))

Секретным ключом являются следующие два числа, «d» и «pi»

Скажем Джэк хочет отправить зашифрованное сообщение своему другу Стиву, он должен свериться с открытым ключом и представить сообщение в виде числа «m», строго меньшего модуля N алгоритма. Зашифрованный текст должен получится по следующей формуле:

С= m^(E) \* (mod N)

Стив получив зашифрованное сообщение от Джэка расшифрует его по следующему алгоритму:

m=c^(d) \* (mod N)

Для реального примера возьмём очень маленькие числа. Допустим что p= 7 и q= 11. В таком случае N= 77, а pi= 60 ((p-1)\*(q-1)). В качестве открытой экспоненты «Е» возьмём число 37, Е= 37 так как НОД(37, 60)=1.

Применим расширенный алгоритм Евклида и найдём «d», он равен 13.

37\*13=481=1 (mod 60)

Скажем что нам нужно зашифровать сообщение, численное представление которого имеет вид: m=2.

С=m^(E)\*(mod N)

2^(37)\*(mod 77)=51

Алгоритм расшифровывания проходит так же,

M=c^(d)\* (mod N)

51^(13) \* (mod 77)=2

Ход работы

После сравнения определённого количества алгоритмов шифрования с открытым и закрытым ключами, было решено реализовать такой алгоритм шифрования как RSA.

Для реализации был выбран статически типизированный язык программирования общего назначения С++.

Для написания самого кода было решено использовать такую среду разработки для языков программирования как CLion.

Для упрощения реализации алгоритма и увеличения скорости шифрования было решено добавить дополнительно отдельные алгоритмы. Были добавлены такие алгоритмы как алгоритм быстрого возведение в степень, классический (простой) и расширенный алгоритм Евклида, алгоритм Миллера-Рабина, функция Эйлера.

Структура кода устроена следующим образом. Есть два класса «class RSA» и «class Crypt:RSA» (который наследует от первого класса).

В классе «RSA» в области «protected» прописано всё необходимое для реализации алгоритма, есть такие переменные как p, q, mod, pi, e, d и требуемые алгоритмы для генерации закрытых и открытых ключей (bool nod\_evklid(…), bool isPrime(…), bool miller\_test(), void create\_keys (…), void create\_prime\_numbers(…), static long powerFast(…),static long int decrypt(…),int gcd(…) ).

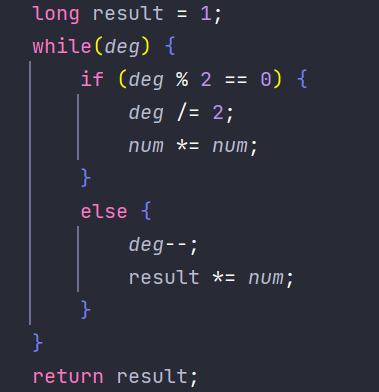
* «p», «q» - два оригинальных простых числа
* «mod» - произведение «p» и «q»
* «pi» - результат работы функции Эйлера
* «е» - число, взаимно простое с «pi»
* «d» - число обратное по «e» оп модулю «pi»
* «bool nod\_evklid(…)» - проверка, является ли число простым
* «bool miller\_test(…)» - алгоритм Миллера-Рабина
* «void create\_prime\_numbers()»- генерация простых чисел «p» и «q»
* «void create\_keys (…)» - вычисление открытых и закрытых ключей
* «static long powerFast(…)» - алгоритм быстрого возведения в степень
* «static long int decrypt(…)» - возвращает результат d\*e % mod
* «int gcd(…)»- расширенный алгоритм Евклида

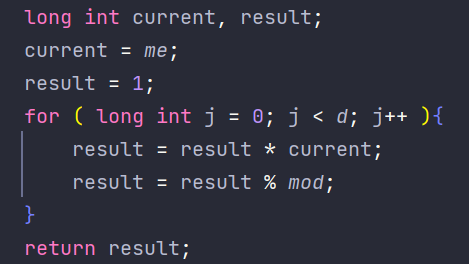
В классе «class Crypt:RSA»

* в области «public» прописаны следующие функции: explicit Crypt(…), void encryption(…), void decryption(…), void print\_encrypted\_line(…), void print\_decrypted\_line(…), ~Crypt(…)
* в области «private» прописаны: arr\_en, line, s, elems\_queue а также такие функции как rsa\_algorythm(), print(…).
* «arr\_en» - хранит уникальный алфавит
* «line» - хранит не зашифрованную строку
* «s» - хранит зашифрованно-расшифрованную строку
* «elems\_queue» - очередь для хранения зашифрованной строки
* «rsa\_algorythm()» - создание открытых и закрытых ключей RSA
* «print()» - вывод элементов очереди «elems\_queue»
* «Crypt(…)» - конструктор класса
* «void encryption(…)» - запускает алгоритм шифрования текста
* «void decryption(…)» - запускает алгоритм расшифровки текста
* «void print\_encrypted\_line(…)» - вывод зашифрованной строки
* «void print\_decrypted\_line(…)» - вывод расшифрованной строки
* «Crypt(…)» - деструктор класса

Проблемы в реализации

При написании кода возникли некоторые проблемы. Благодаря дискуссии с руководителем практики и часовых поисков решений в интернете они были решены.



* По началу некоторые переменные были глобальными, другие функции могли использовать их, решили мы эту проблему обернув всё это в два класса
* Функция «std::pow» - работала плохо с большими числами, она не могла возводить в степень большие числа, решили посредством добавления алгоритма быстрого возведения в степень.
* Уравнение следующего вида «d\*e % mod» тоже становилось большим числом и выходило из допустимой памяти, решено было по очереди возводить в степень число «d» пока не достигнет числа «e» и на каждом шагу выполнять следующую операцию: «% mod»

ЗАКЛЮЧЕНИЕ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

СПИСОК ЛИТЕРАТУРЫ

Брюс Шнайер: Прикладная криптография.

Ш.Т. ИШМУХАМЕТОВ: МЕТОДЫ ФАКТОРИЗАЦИИ НАТУРАЛЬНЫХ ЧИСЕЛ

О. Н. ВАСИЛЕНКО: ТЕОРЕТИКО-ЧИСЛОВЫЕ АЛГОРИТМЫ В КРИПТОГРАФИИ

Криптография : Dan Boneh <https://ru.coursera.org/learn/crypto>

Alex Kolos RSA алгоритм. https://www.youtube.com/watch?v=xqPgE-hPIfE&t=245s

ПРИЛОЖЕНИЕ

#include <iostream>

#include <utility>

#include <vector>

#include <bits/stdc++.h>

#include <queue>

class RSA{

protected:

int p, q, mod, pi, e, d;

static bool nod\_evklid(int a, int b);

bool isPrime(int n);

bool miller\_test(int w, int n);

static int power(int x, unsigned int y, int p);

void create\_prime\_numbers();//cin>>p>>q

void create\_keys();//p, q, mod, pi, e, d

static long powerFast(long num, long deg);

static long int decrypt(long int me, long int d, long int mod);//d\*e % mod

int gcd (int a, int b, int &x, int& y);

};

class Crypt:RSA{

private:

std::vector<char> arr\_en = {' ','b','a','j','x','w','f','g','c','t','h','k','l','z','n','o','s','q','r','p','i','u','v','e','d','y','m'};

std::string line;

std::string s;

std::queue<int>elems\_queue;

public:

explicit Crypt(std::string str) : RSA(){

line=std::move(str);

rsa\_algorithm();

}

void encryption();

void decryption();

void print\_encrypted\_line();

void print\_decrypted\_line();

~Crypt()= default;

private:

void rsa\_algorithm();

static void print(std::queue<int> a);

};

int main() {

srand(time(nullptr));

std::string str="text me something";

Crypt me(str);

me.encryption();

me.print\_encrypted\_line();

me.decryption();

me.print\_decrypted\_line();

return 0;

}

void Crypt::rsa\_algorithm(){

std::cout<<"do you want to enter two prime numbers? [y, n]\n";

char yes\_no;

std::cin>>yes\_no;

//yes\_no='n';

switch (yes\_no) {

case 'y':

std::cout<<"p= ";std::cin>>p;

std::cout<<"q= ";std::cin>>q;

break;

case 'n':

create\_prime\_numbers();

break;

default:

std::cerr<<"error of entering prime numbers\n";

return;

}

create\_keys();

}

void Crypt::print\_decrypted\_line(){

std::cout<<"\ndecrypted line: "<<s<<"\n";

}

void Crypt::print\_encrypted\_line(){

std::cout<<"\nencrypted line:\n";

print(elems\_queue);

}

void Crypt::print(std::queue<int> a){

while(!a.empty()){

std::cout<<a.front()<<" ";

a.pop();

}

std::cout<<"\n";

}

void Crypt::encryption() {

std::vector<char>::iterator it;

for(char & i : line)

{

it = find(arr\_en.begin(), arr\_en.end(), i);

int elem= it-arr\_en.begin();

int result=powerFast(elem, e) % mod;

elems\_queue.push(result);

}

}

void Crypt::decryption(){

std::queue<int> que;

while(!elems\_queue.empty()){

int me=elems\_queue.front();

elems\_queue.pop();

int t=decrypt(me, d, mod);

que.push(t);

}

//закидываю расшифрованные элементы из очереди в строку

while(!que.empty()){

int me=que.front();

que.pop();

s.push\_back(arr\_en[me]);

}

}

int RSA::power(int x, unsigned int y, int p){

int res = 1; // Initialize result

x = x % p; // Update x if it is more than or

// equal to p

while (y > 0){

// If y is odd, multiply x with result

if (y & 1)

res = (res\*x) % p;

// y must be even now

y = y>>1; // y = y/2

x = (x\*x) % p;

}

return res;

}

int RSA::gcd(int a, int b, int &x, int& y) {

if (b < a){

int t = a;

a = b;

b = t;

}

if (a == 0){

x = 0;

y = 1;

return b;

}

int u = gcd(b % a, a, x, y);

int newY = x;

int newX = y - (b / a) \* x;

x = newX;

y = newY;

return u;

}

long RSA::powerFast(long num, long deg) {

long result = 1;

while(deg) {

if (deg % 2 == 0) {

deg /= 2;

num \*= num;

}

else {

deg--;

result \*= num;

}

}

return result;

}

bool RSA::nod\_evklid(int a, int b){

auto cur=1;

for(int i=2; i<=a && i<=b; i++)

if(a%i==0 && b%i==0)

cur++;

if(cur==1)//если cur==1 то оно взаимопростое

return true;

return false;

}

void RSA::create\_keys(){

mod=p\*q;

pi=(p-1)\*(q-1);

std::vector<int> prime\_numbers ={2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499};

for(auto cur: prime\_numbers){

e=cur;

if(e<pi && nod\_evklid(e, pi))

break;

}

int a1;

gcd(e, pi, d, a1);

if(d<0){

d+=pi;

}

std::cout<<"p= "<<p<<" q= "<<q<<" pi= "<<pi<<std::endl;

std::cout<<"public key\n{e, mod} {"<<e<<", "<<mod<<"}";

std::cout<<"\nprivate key\n{d, mod} {"<<d<<", "<<mod<<"}";

}

void RSA::create\_prime\_numbers() {

do{

int A=1, B=99;

p=rand()%(B-A+1) + A;//[A, B]

q=rand()%(B-A+1) + A;

} while (p==q);

while(true){

if(isPrime(p)){

if(isPrime(q)){

break;

}

else{

while(!isPrime(q)){

q+=1;

}

std::cout<<"q=true\n";

break;

}

}else{

while(!isPrime(p)){

p+=1;

}

if(isPrime(q)){

break;

}

else{

while(!isPrime(q)){

q+=1;

}

break;

}

}

}

}

bool RSA::miller\_test(int w, int n){

// Pick a random number in [2..n-2]

// Corner cases make sure that n > 4

int a = 2 + rand() % (n - 4);

// Compute a^d % n

int x = power(a, w, n);

if (x == 1 || x == n-1)

return true;

while (w != n-1){

x = (x \* x) % n;

w \*= 2;

if (x == 1) return false;

if (x == n-1) return true;

}

return false;

}

bool RSA::isPrime(int n){

int k=4;

// Corner cases

if (n <= 1 || n == 4) return false;

if (n <= 3) return true;

// Find r such that n = 2^d \* r + 1 for some r >= 1

int d = n - 1;

while (d % 2 == 0)

d /= 2;

// Iterate given number of 'k' times

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

if (!miller\_test(d, n))

return false;

return true;

}

long int RSA::decrypt(long int me, long int d, long int mod){

long int current, result;

current = me;

result = 1;

for ( long int j = 0; j < d; j++ ){

result = result \* current;

result = result % mod;

}

return result;

}