Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"

профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

**Расчетно-графическое задание по дисциплине  
 Защита информации**

**Вариант 3**

Выполнил:

Cтудент гр. ИП-911 \_\_\_\_\_\_\_\_\_\_\_\_\_/Белоусова Е. Е./

ФИО студента

«19» ноября 2022 г.

Проверил:

Доцент кафедры ПМиК \_\_\_\_\_\_\_\_\_\_\_\_\_/Ракитский А. А. /

ФИО преподавателя

«19» ноября 2022 г Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Новосибирск 2022 г.

Оглавление

[Постановка задачи 3](#_Toc119682613)

[Теория 3](#_Toc119682614)

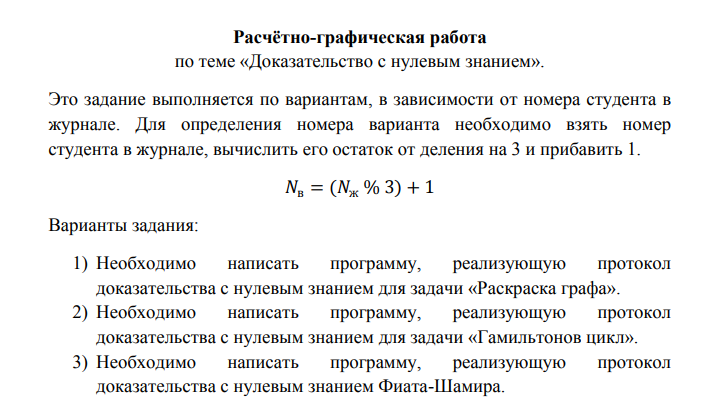
[**Предварительные действия:** 4](#_Toc119682615)

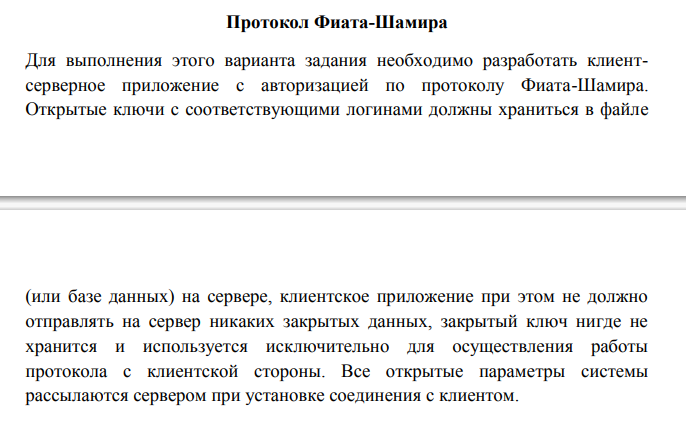
[**Основные действия:** 4](#_Toc119682616)

[Результаты работы 6](#_Toc119682617)

[Исходный код 8](#_Toc119682618)

# Постановка задачи





# Теория

Доказательство с нулевым разглашением (информации) в криптографии (англ. Zero-knowledge proof) — интерактивный криптографический протокол, позволяющий одной из взаимодействующих сторон убедиться в достоверности какого-либо утверждения (обычно математического), не имея при этом никакой другой информации от второй. Причём последнее условие является необходимым, так как обычно доказать, что сторона обладает определёнными сведениями в большинстве случаев тривиально, если она имеет право просто раскрыть информацию. Вся сложность состоит в том, чтобы доказать, что у одной из сторон есть информация, не раскрывая её содержание. Протокол должен учитывать, что доказывающий сможет убедить проверяющего только в случае, если утверждение действительно доказано. В противном случае сделать это будет невозможно, или крайне маловероятно из-за вычислительной сложности.

Данный криптографический протокол должен обладать тремя свойствами:

1. Полнота: если утверждение действительно верно, то Доказывающий убедит в этом Проверяющего с любой наперед заданной точностью.
2. Корректность: если утверждение неверно, то любой, даже «нечестный», Доказывающий не сможет убедить Проверяющего за исключением пренебрежимо малой вероятности.
3. Нулевое разглашение: если утверждение верно, то любой, даже «нечестный», Проверяющий не узнает ничего кроме самого факта, что утверждение верно.

Доказательства с нулевым разглашением не являются доказательствами в математическом смысле этого термина, потому что есть некоторая небольшая вероятность, что обманом доказывающая сторона сможет убедить проверяющего в ложном утверждении (ошибка корректности). Иными словами, доказательства с нулевым разглашением — это вероятностные доказательства, а не детерминированные. Тем не менее, есть методы, позволяющие уменьшить ошибку корректности до пренебрежимо малых значений.

**Описание протокола Фиата-Шамира:**

**A** доказывает **B** знание **s** в течение **t** раундов. Раунд называют также аккредитацией. Каждая аккредитация состоит из 3х этапов.

### **Предварительные действия:**

Сервером выбираются простые числа **p** и **q** (держатся в секрете), и вычисляется число n, которое при подключении к клиенту сразу ему отправляется.

Претендент выбирает **s** взаимно-простое с **n** где **s** принадлежит промежутку **[1, n - 1].**

Затем претендент вычисляет число v = s2 и публикует его на сервере в качестве открытого ключа.

### **Основные действия:**

Следующие действия последовательно и независимо выполняются **t** раз. **В** считает знание доказанным, если все **t** раундов прошли успешно.

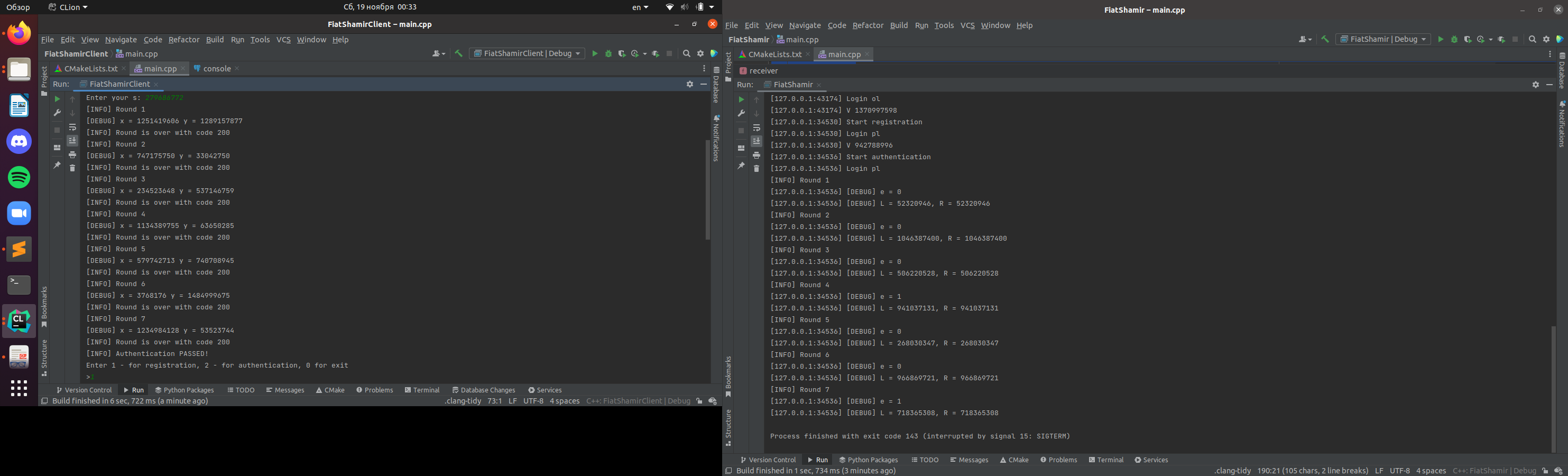
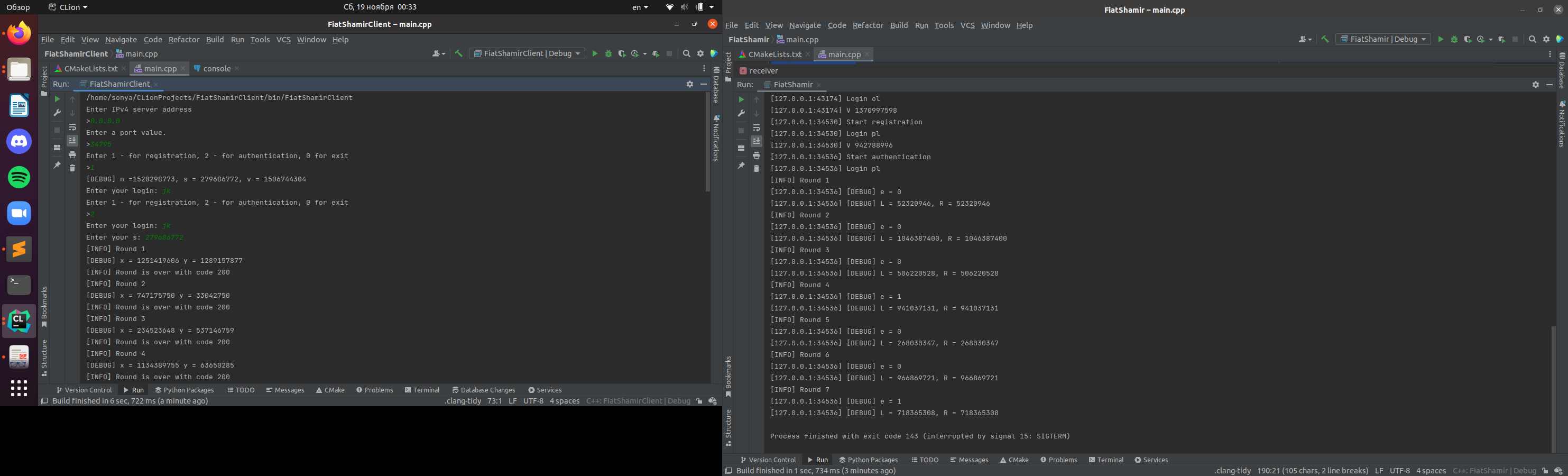
* **А** выбирает случайное **r**, такое, что {\displaystyle r\in [1,n-1]}**r** принадлежит промежутку **[1, n-1]** и отсылает {\displaystyle x=r^{2}{\pmod {n}}}стороне **B**
* **B** случайно выбирает **e равное либо 0, либо 1** и отсылает его **A**
* **А** вычисляет **у** и отправляет его обратно к **B**. Если **e** = 0, то {\displaystyle y=r}**y** = **r**, иначе {\displaystyle y=r\*s{\pmod {n}}} **y** = **r** \* **s** (mod **n**)
* Если **y**=0, то **B** отвергает доказательство или, другими словами, **А** не удалось доказать знание **s**. В противном случае, сторона **B** проверяет, действительно ли {\displaystyle y^{2}=x\*v^{e}{\pmod {n}}}**y**2 = **x** \* **v**e и, если это так, то происходит переход к следующему раунду протокола.

Выбор **е** предполагает, что если сторона **А** действительно знает секрет, то она всегда сможет правильно ответить, вне зависимости от выбранного **e**. Допустим, что **А** хочет обмануть **B**. В этом случае **А**, может отреагировать только на конкретное значение **e**.

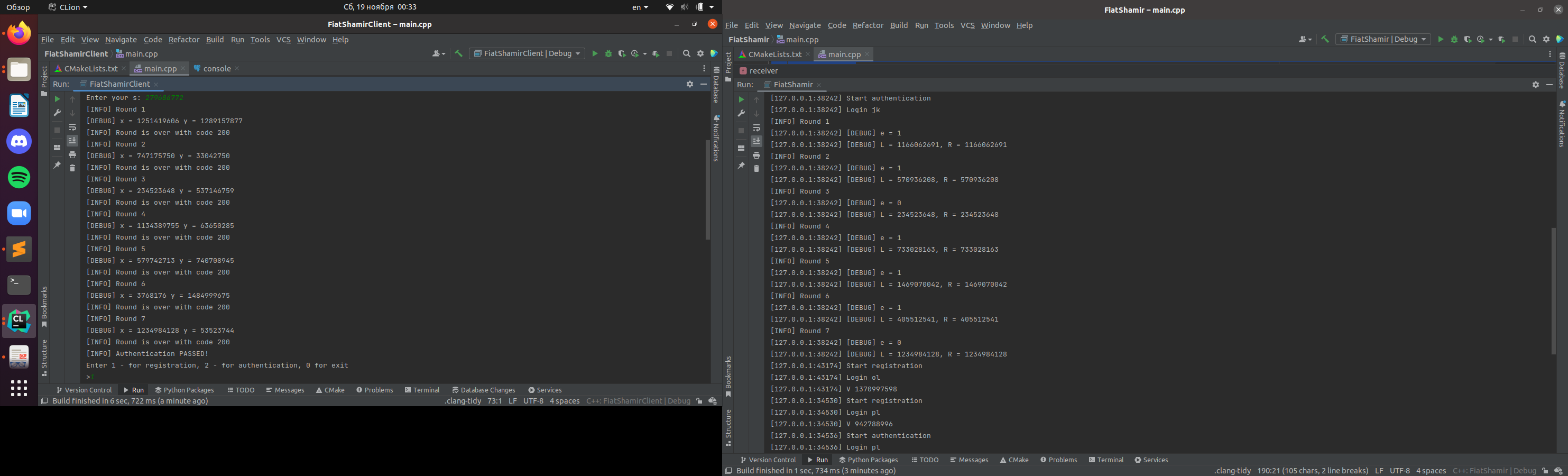
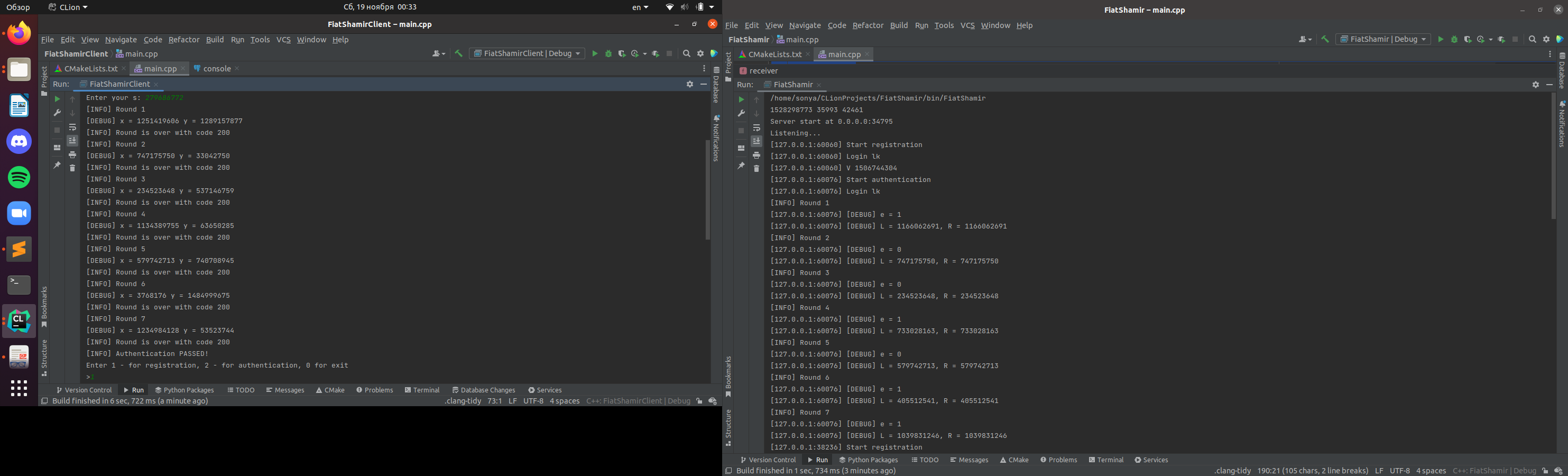
Проблема заключается в том, что **А** изначально не знает какое **e** он получит и поэтому не может со 100 % вероятностью выслать на сторону **В** нужные для обмана **r** и **х,** поэтому вероятность зависит от **e**. Чтобы снизить вероятность жульничества **t** выбирают достаточно большим. Таким образом, **B** удостоверяется в знании **А** тогда и только тогда, когда все **t** раундов прошли успешно.

# Результаты работы

Со стороны клиента:



Со стороны сервера:



# Исходный код

//Server.cpp

#include <arpa/inet.h>

#include <bits/stdc++.h>

#include <iostream>

#include <netinet/in.h>

#include <ostream>

#include <unistd.h>

#include <pthread.h>

#include <map>

#define MAX(x, y) (x)>(y)?(x):(y)

#define MIN(x, y) (x)<(y)?(x):(y)

#define ll long long

#define MAX\_SIZE 256

#define ROUNDS 7

pthread\_mutex\_t mutex;

ll n;

const std::string db = "base.txt";

std::map<std::string, ll> logins;

ll modularExponentiation(ll x, ll ex, ll p) {

ll result = 1;

for (ll na = abs(ex); na > 0; na >>= 1) {

if (na % 2 == 1) {

result = (result \* x) % p;

}

x = (x \* x) % p;

}

return result % p;

}

ll gcd(ll a, ll b, ll \*x, ll \*y) {

ll U\_array[] = {MAX(a, b), 1, 0};

ll V\_array[] = {MIN(a, b), 0, 1};

ll T\_array[3];

ll q, \*swop\_p, \*U, \*V, \*T;

q = MAX(a, b);

if (q != a) {

swop\_p = x;

x = y;

y = swop\_p;

}

U = U\_array;

V = V\_array;

T = T\_array;

while (V[0] != 0) {

q = U[0] / V[0];

T[0] = U[0] % V[0];

T[1] = U[1] - q \* V[1];

T[2] = U[2] - q \* V[2];

swop\_p = U;

U = V;

V = T;

T = swop\_p;

}

if (x != NULL) {

\*x = U[1];

}

if (y != NULL) {

\*y = U[2];

}

return U[0];

}

ll inversion(ll \*c, ll \*d, ll p) {

ll x, y;

ll b\_c, b\_d, b\_p;

do {

\*c = rand() + 1;

} while (gcd(\*c, p, &x, &y) != 1);

b\_c = \*c;

b\_p = p;

\*d = x < 0 ? p + x : x;

b\_d = \*d;

return (b\_c \* b\_d) % b\_p;

}

bool ferma(ll x) {

ll a;

if (!(x % 2)) {

return false;

}

for (int i = 0; i < 100; i++) {

a = (rand() % (x - 2)) + 2;

if (gcd(a, x, NULL, NULL) != 1)

return false;

if (modularExponentiation(a, x - 1, x) != 1)

return false;

}

return true;

}

ll \*eucleadian(ll a, ll b) {

ll static u[]{a, 1, 0}, v[]{b, 0, 1}, t[]{0, 0, 0};

ll q = 0;

while (v[0]) {

q = u[0] / v[0];

t[0] = u[0] % v[0];

t[1] = u[1] - q \* v[1];

t[2] = u[2] - q \* v[2];

u[0] = v[0];

u[1] = v[1];

u[2] = v[2];

v[0] = t[0];

v[1] = t[1];

v[2] = t[2];

}

//cout << u[0] << " " << u[1] << " " << u[2] << endl;

return u;

}

bool isPrime(ll p, int k) {

if (p == 2) {

return true;

}

if (!(p & 1)) {

return false;

}

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

ll a = rand() % (p - 2) + 1;

if (eucleadian(a, p)[0] != 1 || modularExponentiation(a, p - 1, p) != 1) {

//cout << a << p << endl;

return false;

}

}

return true;

}

ll primeNumberRandom(ll left, ll right) {

ll cnt = 0;

while (1) {

ll x = (rand() \* (ll) rand() + rand()) % (right - left) + left;

x |= 1;

//std::cout << x << std::endl;

if (isPrime(x, 100)) {

return x;

}

}

}

void initSocket(int &serverSocket) {

serverSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (serverSocket < 0) {

std::cerr << "socket:" << errno << std::endl;

exit(EXIT\_FAILURE);

}

}

void initStruct(sockaddr\_in &serverStruct) {

serverStruct.sin\_family = AF\_INET;

serverStruct.sin\_addr.s\_addr = htonl(INADDR\_ANY);

serverStruct.sin\_port = htons(0);

}

std::string getIPAndPort(const sockaddr\_in &structure);

std::string getIPAndPort(const int &fd, const bool &client = true);

void \*receiver(void \*clientSocketPtr) {

int clientSocket = (int) reinterpret\_cast<std::intptr\_t>(clientSocketPtr);

char buff[MAX\_SIZE];

recv(clientSocket, buff, MAX\_SIZE, 0);

std::string prefix("[" + getIPAndPort(clientSocket) + "] ");

if (buff[0] == '1') {

std::cout << prefix << "Start registration" << std::endl;

send(clientSocket, std::to\_string(n).c\_str(), MAX\_SIZE, 0);

recv(clientSocket, buff, MAX\_SIZE, 0);

std::cout << prefix << "Login " << buff << std::endl;

std::string login(buff);

recv(clientSocket, buff, MAX\_SIZE, 0);

std::cout << prefix << "V " << buff << std::endl;

ll v = atoll(buff);

logins[login] = v;

pthread\_mutex\_lock(&mutex);

std::ofstream out(db, std::ios::app);

out << login << " " << v << std::endl;

out.close();

pthread\_mutex\_unlock(&mutex);

} else {

ll x, e, y;

std::cout << prefix << "Start authentication" << std::endl;

send(clientSocket, std::to\_string(n).c\_str(), MAX\_SIZE, 0);

recv(clientSocket, buff, MAX\_SIZE, 0);

std::cout << prefix << "Login " << buff << std::endl;

std::string login(buff);

send(clientSocket, std::to\_string(ROUNDS).c\_str(), MAX\_SIZE, 0);

for (int i = 0; i < ROUNDS; i++) {

std::cout << "[INFO] Round " << i + 1 << std::endl;

recv(clientSocket, buff, MAX\_SIZE, 0);

x = atoll(buff);

e = rand() % 2;

std::cout << prefix << "[DEBUG] e = " << e << std::endl;

send(clientSocket, std::to\_string(e).c\_str(), MAX\_SIZE, 0);

recv(clientSocket, buff, MAX\_SIZE, 0);

y = atoll(buff);

ll l = modularExponentiation(y, 2, n);

ll r = (x \* modularExponentiation(logins[login], e, n)) % n;

std::cout << prefix << "[DEBUG] L = " << l << ", R = " << r << std::endl;

if (l == r) {

send(clientSocket, std::to\_string(200).c\_str(), MAX\_SIZE, 0);

std::cout << prefix << "Authentication PASSED!" << std::endl;

} else {

send(clientSocket, std::to\_string(215).c\_str(), MAX\_SIZE, 0);

std::cout << prefix << "Authentication FAILED!" << std::endl;

break;

}

}

}

close(clientSocket);

}

void \*listenConnections(void \*serverSocketPtr) {

int serverSocket = (int) reinterpret\_cast<std::intptr\_t>(serverSocketPtr);

listen(serverSocket, SOMAXCONN);

int clientSocket;

std::cout << "Listening..." << std::endl;

while ((clientSocket = accept(serverSocket, nullptr, nullptr)) > 0) {

pthread\_t receiverThread;

pthread\_create(&receiverThread, nullptr, receiver, (void \*) clientSocket);

pthread\_detach(receiverThread);

}

return nullptr;

}

void readDbFromFile() {

std::ifstream file(db);

std::string key;

ll value;

while (file >> key >> value) {

logins[key] = value;

}

file.close();

}

int main() {

int serverSocket;

srand(42);

pthread\_t mTh;

readDbFromFile();

sockaddr\_in serverStructure;

initStruct(serverStructure);

initSocket(serverSocket);

ll p, q;

p = primeNumberRandom(10000, 45000);

q = primeNumberRandom(10000, 45000);

n = p \* q;

std::cout << n << " " << p << " " << q << std::endl;

if (bind(serverSocket, (sockaddr \*) &serverStructure, sizeof(serverStructure)) < 0) {

std::cerr << "bind:" << errno << std::endl;

exit(EXIT\_FAILURE);

}

std::cout << "Server start at " + getIPAndPort(serverSocket, false) << std::endl;

pthread\_create(&mTh, nullptr, listenConnections, (void \*) serverSocket);

pthread\_detach(mTh);

std::string str;

while (str != "stop" && str != "exit") {

std::cin >> str;

}

close(serverSocket);

pthread\_mutex\_destroy(&mutex);

pthread\_cancel(mTh);

return 0;

}

std::string getIPAndPort(const sockaddr\_in &structure) {

unsigned int port = ntohs(structure.sin\_port);

std::string clientIP(inet\_ntoa(structure.sin\_addr));

return clientIP + ":" + std::to\_string(port);

}

std::string getIPAndPort(const int &fd, const bool &client) {

sockaddr\_in structure{};

socklen\_t size = sizeof(structure);

if (client)

getpeername(fd, (sockaddr \*) &structure, &size);

else

getsockname(fd, (sockaddr \*) &structure, &size);

return getIPAndPort(structure);

}

//Client.cpp

#include <arpa/inet.h>

#include <bits/stdc++.h>

#include <netinet/in.h>

#include <unistd.h>

#include <iostream>

#include <fstream>

#include "rapidjson/document.h"

#include "rapidjson/writer.h"

#include "rapidjson/stringbuffer.h"

#include "rapidjson/ostreamwrapper.h"

#include "rapidjson/istreamwrapper.h"

#include <vector>

#include <algorithm>

#define MAX(x, y) (x)>(y)?(x):(y)

#define MIN(x, y) (x)<(y)?(x):(y)

#define MAX\_SIZE 256

#define ll long long

using namespace rapidjson;

int setServerPort();

std::string setServerIP();

ll modularExponentiation(ll x, ll ex, ll p) {

ll result = 1;

for (ll na = abs(ex); na > 0; na >>= 1) {

if (na % 2 == 1) {

result = (result \* x) % p;

}

x = (x \* x) % p;

}

return result % p;

}

ll gcd(ll a, ll b, ll \*x, ll \*y) {

ll U\_array[] = {MAX(a, b), 1, 0};

ll V\_array[] = {MIN(a, b), 0, 1};

ll T\_array[3];

ll q, \*swop\_p, \*U, \*V, \*T;

q = MAX(a, b);

if (q != a) {

swop\_p = x;

x = y;

y = swop\_p;

}

U = U\_array;

V = V\_array;

T = T\_array;

while (V[0] != 0) {

q = U[0] / V[0];

T[0] = U[0] % V[0];

T[1] = U[1] - q \* V[1];

T[2] = U[2] - q \* V[2];

swop\_p = U;

U = V;

V = T;

T = swop\_p;

}

if (x != NULL) {

\*x = U[1];

}

if (y != NULL) {

\*y = U[2];

}

return U[0];

}

ll inversion(ll \*c, ll \*d, ll p) {

ll x, y;

ll b\_c, b\_d, b\_p;

do {

\*c = rand() + 1;

} while (gcd(\*c, p, &x, &y) != 1);

b\_c = \*c;

b\_p = p;

\*d = x < 0 ? p + x : x;

b\_d = \*d;

return (b\_c \* b\_d) % b\_p;

}

bool ferma(ll x) {

ll a;

if (!(x % 2)) {

return false;

}

for (int i = 0; i < 100; i++) {

a = (rand() % (x - 2)) + 2;

if (gcd(a, x, NULL, NULL) != 1)

return false;

if (modularExponentiation(a, x - 1, x) != 1)

return false;

}

return true;

}

ll \*eucleadian(ll a, ll b) {

ll static u[]{a, 1, 0}, v[]{b, 0, 1}, t[]{0, 0, 0};

ll q = 0;

while (v[0]) {

q = u[0] / v[0];

t[0] = u[0] % v[0];

t[1] = u[1] - q \* v[1];

t[2] = u[2] - q \* v[2];

u[0] = v[0];

u[1] = v[1];

u[2] = v[2];

v[0] = t[0];

v[1] = t[1];

v[2] = t[2];

}

//cout << u[0] << " " << u[1] << " " << u[2] << endl;

return u;

}

bool isPrime(ll p, int k) {

if (p == 2) {

return true;

}

if (!(p & 1)) {

return false;

}

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

ll a = rand() % (p - 2) + 1;

if (eucleadian(a, p)[0] != 1 || modularExponentiation(a, p - 1, p) != 1) {

//cout << a << p << endl;

return false;

}

}

return true;

}

ll primeNumberRandom(ll left, ll right) {

ll cnt = 0;

while (1) {

ll x = (rand() \* (ll) rand() + rand()) % (right - left) + left;

x |= 1;

//std::cout << x << std::endl;

if (isPrime(x, 100)) {

return x;

}

}

}

ll coprime(ll n) {

ll s, x, y;

s = (rand() \* rand() + rand()) % n;

while (gcd(s, n, 0, 0) != 1) {

s--;

}

if (s < 0) {

s \*= -1;

}

return s;

}

void initSocket(int &clientSocket) {

clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (clientSocket < 0) {

std::cerr << "socket:" << errno << std::endl;

exit(EXIT\_FAILURE);

}

}

void initStruct(sockaddr\_in &serverStruct) {

serverStruct.sin\_family = AF\_INET;

serverStruct.sin\_addr.s\_addr = inet\_addr(setServerIP().c\_str());

serverStruct.sin\_port = htons(setServerPort());

}

std::vector<std::string> possible\_logins;

const std::string logins = "logins.txt";

void readDbFromFile() {

std::ifstream file(logins);

std::string key;

while (file >> key) {

possible\_logins.push\_back(key);

}

file.close();

}

int main() {

srand(42);

int clientSocket;

sockaddr\_in server;

initStruct(server);

char buff[MAX\_SIZE];

int choose;

readDbFromFile();

while (1) {

std::cout << "Enter 1 - for registration, 2 - for authentication, 0 for exit \n>";

std::cin >> choose;

initSocket(clientSocket);

if (connect(clientSocket, (struct sockaddr \*) &server, sizeof(server)) < 0) {

std::cerr << "connect:" << errno << std::endl;

exit(EXIT\_FAILURE);

}

ll s, v, n;

if (choose == 1) {

//сообщаем серверу, что будет регистрация проходить

send(clientSocket, std::to\_string(choose).c\_str(), MAX\_SIZE, MSG\_NOSIGNAL);

//получаем n

recv(clientSocket, buff, MAX\_SIZE, MSG\_NOSIGNAL);

n = atoll(buff);

s = coprime(n);

v = modularExponentiation(s, 2, n);

std::cout << "[DEBUG] n =" << n << ", s = " << s << ", v = " << v << std::endl;

std::cout << "Enter your login: ";

std::string login;

std::cin >> login;

//отправляем логин и открытый ключ v

send(clientSocket, login.c\_str(), MAX\_SIZE, MSG\_NOSIGNAL);

send(clientSocket, std::to\_string(v).c\_str(), MAX\_SIZE, MSG\_NOSIGNAL);

//менеджер паролей :D

// Document d;

// rapidjson::Document::AllocatorType &allocator = d.GetAllocator();

// d.SetObject();

// d.AddMember("v", (int64\_t) v, allocator);

// d.AddMember("s", (int64\_t) s, allocator);

// std::ofstream fout(login + ".json", std::ios::binary);

// OStreamWrapper out(fout);

// Writer<OStreamWrapper> writer(out);

// d.Accept(writer);

// fout.close();

close(clientSocket);

std::ofstream loginsout(logins, std::ios::app);

loginsout << login << std::endl;

loginsout.close();

possible\_logins.push\_back(login);

} else if (choose == 2) {

ll x, r, y;

std::string login;

int e;

std::cout << "Enter your login: ";

std::cin >> login;

if (std::find(possible\_logins.begin(), possible\_logins.end(), login) == possible\_logins.end()) {

std::cout << "[WARN] You should register your profile first!" << std::endl;

continue;

}

//сообщаем серверу о том, что сейчас будет проходить аутентификация

send(clientSocket, std::to\_string(choose).c\_str(), MAX\_SIZE, MSG\_NOSIGNAL);

//получаем n

recv(clientSocket, buff, MAX\_SIZE, MSG\_NOSIGNAL);

n = atoll(buff);

//так делать нельзя, но чтобы было легче проверять

// std::ifstream ifs(login + ".json");

// IStreamWrapper isw{ifs};

// Document doc{};

// doc.ParseStream(isw);

// StringBuffer buffer{};

// Writer<StringBuffer> writer{buffer};

// doc.Accept(writer);

// ll s = doc["s"].GetInt64();

// ifs.close();

// std::cout << "[DEBUG] Your s is " << s << std::endl;

std::cout << "Enter your s: ";

std::cin >> s;

//отправляем логин

send(clientSocket, login.c\_str(), MAX\_SIZE, MSG\_NOSIGNAL);

//получаем кол-во раундов

recv(clientSocket, buff, MAX\_SIZE, MSG\_NOSIGNAL);

int rounds = atoi(buff);

for (int i = 0; i < rounds; i++) {

std::cout << "[INFO] Round " << i + 1 << std::endl;

r = rand() % n + 1;

x = modularExponentiation(r, 2, n);

//отправляем х

send(clientSocket, std::to\_string(x).c\_str(), MAX\_SIZE, MSG\_NOSIGNAL);

//получаем е

recv(clientSocket, buff, MAX\_SIZE, MSG\_NOSIGNAL);

e = atoi(buff);

y = (r \* modularExponentiation(s, e, n)) % n;

std::cout << "[DEBUG] x = " << x << " y = " << y << std::endl;

send(clientSocket, std::to\_string(y).c\_str(), MAX\_SIZE, MSG\_NOSIGNAL);

recv(clientSocket, buff, MAX\_SIZE, MSG\_NOSIGNAL);

std::cout << "[INFO] Round is over with code " << buff << std::endl;

int code = atoi(buff);

if (code != 200) {

std::cout << "[INFO] Authentication FAILED!" << std::endl;

break;

}

}

std::cout << "[INFO] Authentication PASSED!" << std::endl;

close(clientSocket);

} else {

break;

}

}

return 0;

}

std::string setServerIP() {

std::string ip;

unsigned char buf[sizeof(struct in6\_addr)];

do {

std::cout << "Enter IPv4 server address " << std::endl;

std::cout << ">";

getline(std::cin, ip);

} while (inet\_pton(AF\_INET, ip.c\_str(), buf) != 1);

return ip;

}

int setServerPort() {

int port;

do {

std::cout << "Enter a port value." << std::endl;

std::cout << ">";

std::cin >> port;

} while (port <= 0 || port > 65535);

return port;

}