

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:

Студент гр. ИП-911

«19» ноября 2022 г.

_____/Белоусова Е. Е./
ФИО студента

Проверил:

Доцент кафедры ПМиК

«19» ноября 2022 г

_____/Ракитский А. А. /
ФИО преподавателя

Оценка _____

Новосибирск 2022 г.

Оглавление

Постановка задачи	3
Теория	3
Предварительные действия:	4
Основные действия:	4
Результаты работы	6
Исходный код	8

Постановка задачи

Расчётно-графическая работа по теме «Доказательство с нулевым знанием».

Это задание выполняется по вариантам, в зависимости от номера студента в журнале. Для определения номера варианта необходимо взять номер студента в журнале, вычислить его остаток от деления на 3 и прибавить 1.

$$N_v = (N_{\text{ж}} \% 3) + 1$$

Варианты задания:

- 1) Необходимо написать программу, реализующую протокол доказательства с нулевым знанием для задачи «Раскраска графа».
- 2) Необходимо написать программу, реализующую протокол доказательства с нулевым знанием для задачи «Гамильтонов цикл».
- 3) Необходимо написать программу, реализующую протокол доказательства с нулевым знанием Фиата-Шамира.

Протокол Фиата-Шамира

Для выполнения этого варианта задания необходимо разработать клиент-серверное приложение с авторизацией по протоколу Фиата-Шамира. Открытые ключи с соответствующими логинами должны храниться в файле

(или базе данных) на сервере, клиентское приложение при этом не должно отправлять на сервер никаких закрытых данных, закрытый ключ нигде не хранится и используется исключительно для осуществления работы протокола с клиентской стороны. Все открытые параметры системы рассылаются сервером при установке соединения с клиентом.

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 $v = s^2$ и публикует его на сервере в качестве открытого ключа.

Основные действия:

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

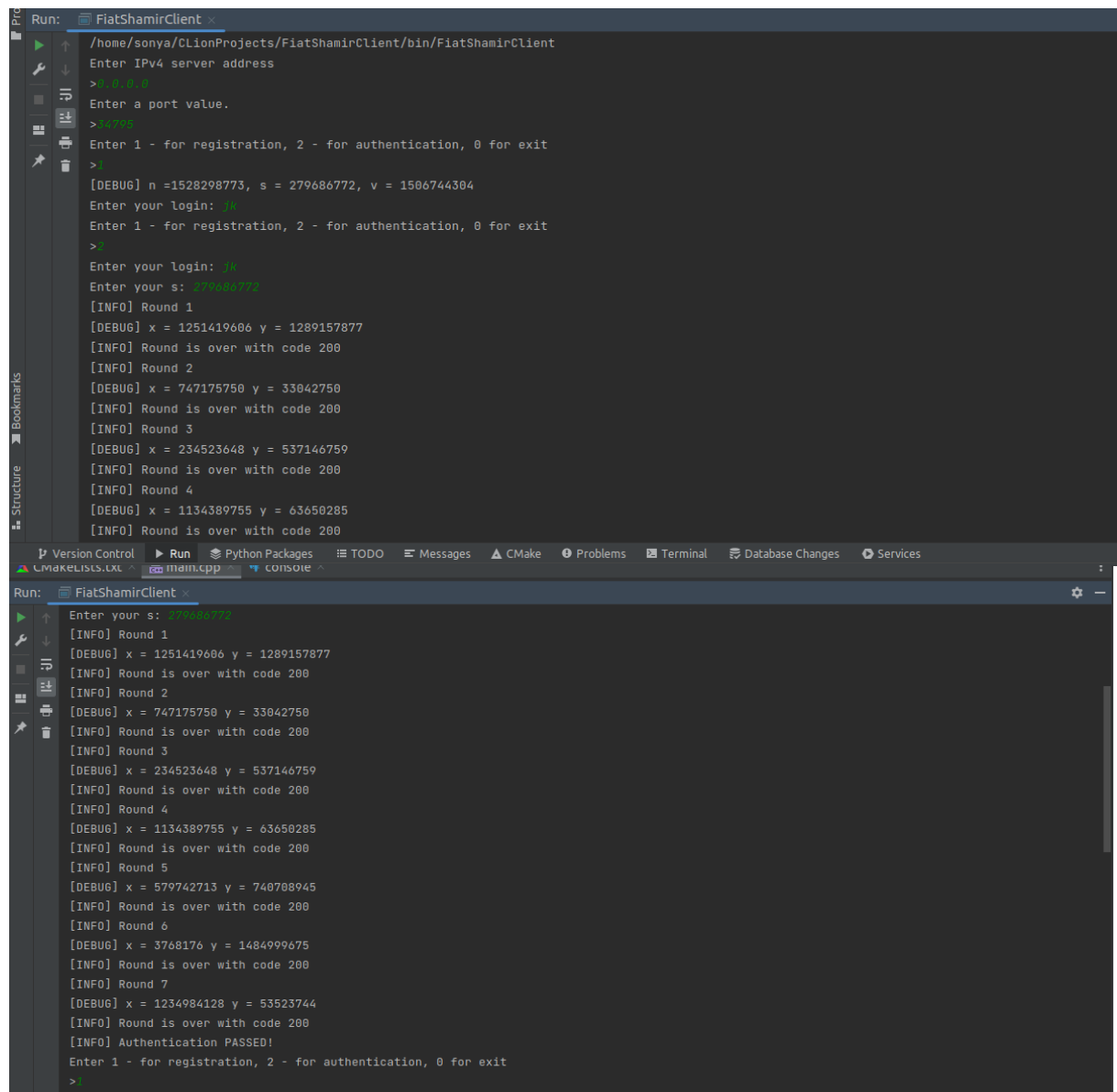
- **A** выбирает случайное **r**, такое, что **r** принадлежит промежутку **[1, n-1]** и отправляет стороне **B**
- **B** случайно выбирает **e** равное либо 0, либо 1 и отправляет его **A**
- **A** вычисляет **y** и отправляет его обратно к **B**. Если **e = 0**, то $y = r$, иначе $y = r * s \pmod n$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

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

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

Результаты работы

Со стороны клиента:



```
Run: FiatShamirClient x
/home/sonya/CLionProjects/FiatShamirClient/bin/FiatShamirClient
Enter IPv4 server address
> 0.0.0.0
Enter a port value.
> 44770
Enter 1 - for registration, 2 - for authentication, 0 for exit
> 1
[DEBUG] n =1528298773, s = 279686772, v = 1506744304
Enter your login: 44
Enter 1 - for registration, 2 - for authentication, 0 for exit
> 2
Enter your login: 44
Enter your s: 279686772
[INFO] Round 1
[DEBUG] x = 1251419606 y = 1289157877
[INFO] Round is over with code 200
[INFO] Round 2
[DEBUG] x = 747175750 y = 33042750
[INFO] Round is over with code 200
[INFO] Round 3
[DEBUG] x = 234523648 y = 537146759
[INFO] Round is over with code 200
[INFO] Round 4
[DEBUG] x = 1134389755 y = 63650285
[INFO] Round is over with code 200
[INFO] Round 5
[DEBUG] x = 579742713 y = 740708945
[INFO] Round is over with code 200
[INFO] Round 6
[DEBUG] x = 3768176 y = 1484999675
[INFO] Round is over with code 200
[INFO] Round 7
[DEBUG] x = 1234984128 y = 53523744
[INFO] Round is over with code 200
[INFO] Authentication PASSED!
Enter 1 - for registration, 2 - for authentication, 0 for exit
> 0
```

Со стороны сервера:

```
receiver
un: FiatShamir x
/home/sonya/CLionProjects/FiatShamir/bin/FiatShamir
1528298773 35993 42461
Server start at 0.0.0.0:34795
Listening...
[127.0.0.1:60060] Start registration
[127.0.0.1:60060] Login lk
[127.0.0.1:60060] V 1506744304
[127.0.0.1:60076] Start authentication
[127.0.0.1:60076] Login lk
[INFO] Round 1
[127.0.0.1:60076] [DEBUG] e = 1
[127.0.0.1:60076] [DEBUG] L = 1166062691, R = 1166062691
[INFO] Round 2
[127.0.0.1:60076] [DEBUG] e = 0
[127.0.0.1:60076] [DEBUG] L = 747175750, R = 747175750
[INFO] Round 3
[127.0.0.1:60076] [DEBUG] e = 0
[127.0.0.1:60076] [DEBUG] L = 234523648, R = 234523648
[INFO] Round 4
[127.0.0.1:60076] [DEBUG] e = 1
[127.0.0.1:60076] [DEBUG] L = 733028163, R = 733028163
[INFO] Round 5
[127.0.0.1:60076] [DEBUG] e = 0
[127.0.0.1:60076] [DEBUG] L = 579742713, R = 579742713
[INFO] Round 6
[127.0.0.1:60076] [DEBUG] e = 1
[127.0.0.1:60076] [DEBUG] L = 405512541, R = 405512541
[INFO] Round 7
[127.0.0.1:60076] [DEBUG] e = 1
[127.0.0.1:60076] [DEBUG] L = 1039831246, R = 1039831246
[127.0.0.1:38236] Start registration
[127.0.0.1:38242] Start authentication
[127.0.0.1:38242] Login jk
[INFO] Round 1
[127.0.0.1:38242] [DEBUG] e = 1
[127.0.0.1:38242] [DEBUG] L = 1166062691, R = 1166062691
[INFO] Round 2
[127.0.0.1:38242] [DEBUG] e = 1
[127.0.0.1:38242] [DEBUG] L = 570936208, R = 570936208
[INFO] Round 3
[127.0.0.1:38242] [DEBUG] e = 0
[127.0.0.1:38242] [DEBUG] L = 234523648, R = 234523648
[INFO] Round 4
[127.0.0.1:38242] [DEBUG] e = 1
[127.0.0.1:38242] [DEBUG] L = 733028163, R = 733028163
[INFO] Round 5
[127.0.0.1:38242] [DEBUG] e = 1
[127.0.0.1:38242] [DEBUG] L = 1469070042, R = 1469070042
[INFO] Round 6
[127.0.0.1:38242] [DEBUG] e = 1
[127.0.0.1:38242] [DEBUG] L = 405512541, R = 405512541
[INFO] Round 7
[127.0.0.1:38242] [DEBUG] e = 0
[127.0.0.1:38242] [DEBUG] L = 1234984128, R = 1234984128
[127.0.0.1:43174] Start registration
[127.0.0.1:43174] Login ol
[127.0.0.1:43174] V 1370997598
[127.0.0.1:34530] Start registration
[127.0.0.1:34530] Login pl
[127.0.0.1:34530] V 942788996
[127.0.0.1:34536] Start authentication
[127.0.0.1:34536] Login pl
```

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;

}

```

```

// 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 *eucledian(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 (eucledian(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 *eucledian(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 (eucledian(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 x

    send(clientSocket, std::to_string(x).c_str(), MAX_SIZE, MSG_NOSIGNAL);

    //получаем e

    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;
}
```