Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект**

**“Мессенджер мгновенных сообщений”**

**по курсу**

**«Операционные системы»**

Студент: Пищик Е.С.

Группа: М8О–206Б–19

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

**Задача: Создание клиента для передачи мгновенных личных сообщений.**

На основе технологии sockets. Создать собственный клиент и сервер быстрых сообщений, который бы работал в рамках сети. Сервер или клиент передачи мгновенных сообщений должны обладать дополнительным функционалом — шифрование, эффективное хранение истории. Шифрование – RSA, эффективное хранение истории – sqlite таблица.

**Общие сведения о программе**

Для работы используются стандартные сокеты, программа собирается при помощи Makefile. Есть 2 исполняемых файла – client, server. Сервер постоянно принимает сообщения от клиентов, которые подключаются к нему по ip. Каждый клиент имеет nickname. Клиент также может посмотреть историю сообщений при помощи команды history.

**Сокеты**

Для работы используются стандартные сокеты, в самом начале создаем struct sockaddr\_in в которую будем записывать информацию, при помощи вызова socket(arg1, arg2, arg3) – создаем сокет, setsockopt – устанавливаем созданному сокету опции переиспользования адресса и порта.

При помощи bind связываем наш сокет с созданной ранее структурой.

listen – ставит сокет в режим прослушивания, 2-ой аргумент – максимальная длина очереди на прослушивание. accept – создает копию нашего сокета и убирает его из очереди на прослушивание, send – отправляет данные, укзанного размера на сокет. inet\_pton(int af, const char\* src, void\* dst) - данная функция преобразует строку символов src в сетевой адрес, затем копирует полученную структуру с адресом в dst. htons и htonl - переводят данные из узлового порядка расположения байтов в сетевой и наоборот. connect – создает соединение на сокете.

**RSA**

RSA – алгоритм ассиметричного шифрования. Суть алгоритма: создать 2 ключа – приватный и публичный, публичный отправляется client-ам, затем каким то образом сообщение шифруется при помощи этого публичного ключа и отправляется на сервер в зашифрованном виде, на сервере при помощи приватного ключа сообщение расшифровывается. Алгоритм RSA использует операцию возведения в степень и взятие остатка – операция к которой очень сложно получить обратное преобразование, не зная приватного ключа. Сам алгоритм получения ключей и шифрования/дешифрования:

1. выбираем 2 случайных числа p и q – достаточно большие.
2. считаем n = p\*q
3. считаем f = (p-1)\*(q-1)
4. выбираем еще 2 числа e и k, так чтобы d было целым
5. считаем d = (k\*f+1)/e
6. публичный ключ – e, n
7. приватный ключ – d,n

Шифрование – crypt = m^e%n, где m – шифруемое число.

Дешифрование - decrypt = crypt^d%n.

**SQLITE**

sqlite является встраиваемой СУБД, для своей программы я скачал sqlite3.c и sqlie3.h и просто добавил их в проект. При помощи sqlie\_open\_v2 – открываем базу данных с флагами на чтение/запись, в многопоточном режиме, создавая, если ее нет. Дальше записываем в std::string нужный нам запрос для создания таблицы, при помощи sqlite3\_exec, в открытую базу данных поступает запрос на создание таблицы(если не существует)(CREATE TABLE IF NOT EXISTS TABLENAME) с 2 колонками MSG и NICKNAME типа – TEXT. Добавление происходит аналогичным образом при помощи sqlite3\_exec, но другого запроса(INSERT INTO ...), если мы хотим вывести всю записанную информацию в таблице на экран, мы также делаем sqlite3\_exec, но в качестве 3 аргумента передаем название функции, куда будут переданны данные, а запрос выглядит следующим образом – SELECT \* FROM TABLENAME. Моя функция callback – выводит на экран название колонки и хранящееся сообщение. Также важно не забывать после использования закрывать БД, даже если произошла ошибка в sqlite\_open\_v2.

**Основные файлы программы**

**Makefile:**

**GPP = g++**

**GCC = gcc**

**GPPFLAGS = -std=c++11**

**GCCFLAGS =**

**SERVNAME = server**

**CLIENNAME = client**

**all: server.o client.o help\_functions.o crypt.o history.o sqlite.o**

**$(GPP) $(GPPFLAGS) server.o crypt.o help\_functions.o \**

**history.o sqlite.o -o $(SERVNAME) -ldl -lpthread**

**$(GPP) $(GPPFLAGS) client.o crypt.o help\_functions.o \**

**history.o sqlite.o -o $(CLIENNAME) -ldl -lpthread**

**client.o: crypt.hpp help\_functions.hpp**

**$(GPP) $(GPPFLAGS) -c client.cpp -o client.o**

**server.o: crypt.hpp history.hpp help\_functions.hpp sqlite3.h**

**$(GPP) $(GPPFLAGS) -c server.cpp -o server.o**

**help\_functions.o: help\_functions.hpp**

**$(GPP) $(GPPFLAGS) -c help\_functions.cpp -o help\_functions.o**

**crypt.o: crypt.hpp**

**$(GPP) $(GPPFLAGS) -c crypt.cpp -o crypt.o**

**history.o: history.hpp**

**$(GPP) $(GPPFLAGS) -c history.cpp -o history.o**

**sqlite.o: sqlite3.h**

**$(GCC) $(GCCFLAGS) -c sqlite3.c -o sqlite.o**

**clean:**

**rm \*.o $(SERVNAME) $(CLIENNAME) history.db server.cpp:**

**#include <unistd.h>**

**#include <stdio.h>**

**#include <sys/socket.h>**

**#include <stdlib.h>**

**#include <netinet/in.h>**

**#include <netdb.h>**

**#include <arpa/inet.h>**

**#include <string.h>**

**#include <iostream>**

**#include "crypt.hpp"**

**#include "help\_functions.hpp"**

**#include "history.hpp"**

**#include "sqlite3.h"**

**ll\* encrypted(public\_key key, bool& flag, ll& pid)**

**{**

**std::string str;**

**ll len = 0;**

**size\_t j = 0;**

**getline(std::cin, str, '\n');**

**if(str == "exit") { flag = true; }**

**else { flag = false; }**

**char\* my\_str = (char\*)str.c\_str();**

**len = str.size();**

**ll\* enc = (ll\*)my\_malloc((len+2)\*sizeof(ll));**

**enc[0] = (ll)len;**

**enc[1] = (ll)pid;**

**for(size\_t i = 2; i < len+2; ++i) { enc[i] = (ll)0; }**

**for (size\_t i = 2; i < len+2; ++i) { enc[i] = (ll)encrypt(key, my\_str[i-2]); }**

**return enc;**

**}**

**void get\_host(char\* name, char\* ip)**

**{**

**char namebuff[MAXMSGLEN];**

**char\* ipbuff;**

**struct hostent \*host;**

**if(gethostname(namebuff, sizeof(namebuff)) == -1)**

**{**

**perror("get host name error\n");**

**exit(-1);**

**}**

**host = gethostbyname(namebuff);**

**if(host == NULL)**

**{**

**perror("get host by name error\n");**

**exit(-1);**

**}**

**ipbuff = inet\_ntoa(\*((struct in\_addr\*)host->h\_addr\_list[0]));**

**std::cout << "Host name " << namebuff << "\n";**

**std::cout << "Host ip " << ipbuff << "\n\n";**

**strcpy(name, namebuff);**

**strcpy(ip, ipbuff);**

**}**

**std::pair<std::string,std::string> decrypted(private\_key key, int sock)**

**{**

**std::string str;**

**std::string name\_str;**

**ll len = 0;**

**ll len\_name = 0;**

**read(sock, &len, sizeof(ll));**

**ll\* dec = (ll\*)my\_malloc(len\*sizeof(ll));**

**read(sock, &len\_name, sizeof(ll));**

**ll\* dec\_name = (ll\*)my\_malloc(len\_name\*sizeof(ll));**

**read(sock, dec, len\*sizeof(ll));**

**read(sock, dec\_name, len\_name\*(sizeof(ll)));**

**for(ll i = 0; i < len; ++i) { str += (char)decrypt(key, dec[i]); }**

**for(ll i = 0; i < len\_name; ++i) { name\_str += (char)decrypt(key, dec\_name[i]); }**

**free(dec);**

**free(dec\_name);**

**return std::make\_pair(str,name\_str);**

**}**

**ll\* fill\_buff(keys key)**

**{**

**ll\* buff = (ll\*)my\_malloc(2\*sizeof(ll));**

**buff[0] = key.publ.first;**

**buff[1] = key.publ.second;**

**return buff;**

**}**

**int main()**

**{**

**History hist;**

**struct sockaddr\_in address;**

**int opt = 1;**

**int addrlen = sizeof(address);**

**char\* ip;**

**char hostname[MAXMSGLEN];**

**srand(time(NULL));**

**keys key = generate\_keys();**

**ll\* buff = fill\_buff(key);**

**int server\_fd = my\_socket(AF\_INET, SOCK\_STREAM, 0);**

**my\_setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT, &opt, sizeof(opt));**

**address.sin\_family = AF\_INET;**

**inet\_aton(ip, &address.sin\_addr);**

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

**address.sin\_port = htons(PORT);**

**get\_host(hostname, ip);**

**my\_bind(server\_fd, (struct sockaddr\*)&address, sizeof(address));**

**my\_listen(server\_fd, 10);**

**while(true)**

**{**

**int new\_socket = my\_accept(server\_fd,(struct sockaddr\*)&address,(socklen\_t\*)&addrlen);**

**my\_send(new\_socket, buff, 2\*sizeof(ll), 0);**

**std::pair<std::string,std::string> msg = decrypted(key.priv, new\_socket);**

**hist.add(msg.first, msg.second);**

**hist.print();**

**std::cout << "\n";**

**}**

**free(buff);**

**return 0;**

**}**

**client.cpp:**

**#include <iostream>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <sys/socket.h>**

**#include <arpa/inet.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include "crypt.hpp"**

**#include "history.hpp"**

**#include "sqlite3.h"**

**#include "help\_functions.hpp"**

**ll\* encrypted(public\_key key, bool& flag, std::string& name\_str, History& hist)**

**{**

**std::string str;**

**ll len = 0;**

**ll len\_name = 0;**

**size\_t j = 0;**

**getline(std::cin, str, '\n');**

**if(str == "exit") { flag = true; }**

**else { flag = false; }**

**if(str == "history")**

**{**

**std::cout << "\nMessages history\n";**

**hist.print();**

**std::cout << "\n";**

**}**

**char\* my\_str = (char\*)str.c\_str();**

**len = str.size();**

**len\_name = name\_str.size();**

**ll\* enc = (ll\*)my\_malloc((len+len\_name+2)\*sizeof(ll));**

**enc[0] = (ll)len;**

**enc[1] = (ll)len\_name;**

**for(size\_t i = 2; i < len+len\_name+2; ++i) { enc[i] = (ll)0; }**

**for (size\_t i = 2; i < len+2; ++i) { enc[i] = (ll)encrypt(key, my\_str[i-2]); }**

**for(size\_t i = len+2; i < len+len\_name+2; ++i) { enc[i] = (ll)encrypt(key, name\_str[i-len-2]); }**

**return enc;**

**}**

**public\_key publ\_key(int sock)**

**{**

**ll\* buff = (ll\*)my\_malloc(2\*sizeof(ll));**

**my\_read(sock, buff, 2\*sizeof(ll));**

**public\_key new\_k(buff[0], buff[1]);**

**free(buff);**

**return new\_k;**

**}**

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

**{**

**History hist;**

**std::string name;**

**if(argc != 2)**

**{**

**perror("Invalid count of arguments\n");**

**exit(-1);**

**}**

**struct sockaddr\_in serv\_addr;**

**serv\_addr.sin\_family = AF\_INET;**

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

**serv\_addr.sin\_port = htons(PORT);**

**bool flag = false;**

**std::cout << "Please enter your nick\n";**

**std::cin >> name;**

**while(true)**

**{**

**int sock = my\_socket(AF\_INET, SOCK\_STREAM, 0);**

**my\_inet\_pton(AF\_INET, argv[1], &serv\_addr.sin\_addr);**

**my\_connect(sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr));**

**public\_key new\_k = publ\_key(sock);**

**ll\* enc = encrypted(new\_k, flag, name, hist);**

**my\_send(sock, enc, (enc[0]+enc[1]+2)\*sizeof(ll), 0);**

**free(enc);**

**if(flag) { break; }**

**}**

**return 0;**

**}**

**crypt.cpp:**

**#include <iostream>**

**#include <time.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <stdio.h>**

**#include <iomanip>**

**#include "crypt.hpp"**

**#define LIMIT 10000**

**ll modpow(ll n, ll p, ll mod)**

**{**

**ll res = 1;**

**for (;p;p >>= 1)**

**{**

**if (p & 1) { res = (1LL \* res \* n) % mod; }**

**n = (1LL \* n \* n) % mod;**

**}**

**return res;**

**}**

**bool rabin\_miller(ll n)**

**{**

**bool flag = true;**

**for (ll i = 1; i <= 5 && flag; ++i)**

**{**

**ll a = (ll)(rand() + 1);**

**ll res = modpow(a, n - 1, n);**

**flag &= (res == 1);**

**}**

**return flag;**

**}**

**ll gen\_p()**

**{**

**ll gen = (ll)(rand() % LIMIT);**

**while (!rabin\_miller(gen)) { gen = (ll)(rand() % LIMIT); }**

**return gen;**

**}**

**ll gcd(ll a, ll b)**

**{**

**while (b)**

**{**

**ll r = a % b;**

**a = b;**

**b = r;**

**}**

**return a;**

**}**

**ll gen\_c(ll n)**

**{**

**ll gen = (ll)(rand() % LIMIT);**

**while (gcd(n, gen) != 1) { gen = (ll)(rand() % LIMIT); }**

**return gen;**

**}**

**std::pair<ll,ll> euclid(ll a, ll b)**

**{**

**if(!b) { return std::pair<ll,ll>(1, 0); }**

**std::pair<ll,ll> result = euclid(b, a % b);**

**return std::pair<ll,ll>(result.second, result.first - (a / b) \* result.second);**

**}**

**ll mod\_inv(ll n, ll mod)**

**{**

**ll inv = euclid(n, mod).first;**

**while(inv < 0) { inv += mod; }**

**return inv;**

**}**

**keys generate\_keys()**

**{**

**keys res;**

**ll p = gen\_p();**

**ll q = gen\_p();**

**ll n = p \* q;**

**ll f = (p - 1) \* (q - 1);**

**ll e = gen\_c(f);**

**res.publ = std::make\_pair(n, e);**

**ll d = mod\_inv(e, f);**

**res.priv = std::make\_pair(n, d);**

**return res;**

**}**

**ll encrypt(public\_key key, ll val)**

**{ return modpow(val, key.second, key.first); }**

**ll decrypt(private\_key key, ll val)**

**{ return modpow(val, key.second, key.first); }**

**crypt.hpp:**

**#pragma once**

**#include <iostream>**

**typedef long long ll;**

**typedef std::pair<ll, ll> public\_key;**

**typedef std::pair<ll, ll> private\_key;**

**struct keys**

**{**

**public\_key publ;**

**private\_key priv;**

**};**

**keys generate\_keys();**

**ll encrypt(public\_key, ll);**

**ll decrypt(private\_key, ll);**

**help\_functions.cpp:**

**#include <iostream>**

**#include <unistd.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <sys/socket.h>**

**#include <netinet/in.h>**

**#include <arpa/inet.h>**

**#include <string.h>**

**#include <netdb.h>**

**#include "help\_functions.hpp"**

**void\* my\_malloc(size\_t size)**

**{**

**void\* mal = malloc(size);**

**if(mal == NULL)**

**{**

**perror("malloc error\n");**

**exit(-1);**

**}**

**return mal;**

**}**

**int my\_socket(int domain, int type, int protocol)**

**{**

**int fd = socket(domain, type, protocol);**

**if(fd == 0)**

**{**

**perror("socket failed\n");**

**exit(-1);**

**}**

**return fd;**

**}**

**int my\_inet\_pton(int af, const char\* src, void\* dst)**

**{**

**int ip = inet\_pton(af, src, dst);**

**if(ip <= 0)**

**{**

**perror("invalid address / address not supported\n");**

**exit(-1);**

**}**

**return ip;**

**}**

**ssize\_t my\_send(int s, const void\* msg, size\_t len, int flags)**

**{**

**ssize\_t var = send(s, msg, len, flags);**

**if(var == -1)**

**{**

**perror("send error\n");**

**exit(-1);**

**}**

**return var;**

**}**

**int my\_connect(int sockfd,struct sockaddr\* serv\_addr, socklen\_t addrlen)**

**{**

**int var = connect(sockfd, serv\_addr, addrlen);**

**if(var == -1)**

**{**

**perror("connection failed\n");**

**exit(-1);**

**}**

**return var;**

**}**

**int my\_read(int fd, void\* buff, size\_t count)**

**{**

**int var = read(fd, buff, count);**

**if(var != count)**

**{**

**perror("read error\n");**

**exit(-1);**

**}**

**return var;**

**}**

**int my\_listen(int s, int backlog)**

**{**

**int var = listen(s, backlog);**

**if(var == -1)**

**{**

**perror("listen error\n");**

**exit(-1);**

**}**

**return var;**

**}**

**int my\_bind(int sockfd, struct sockaddr\* my\_addr, socklen\_t addrlen)**

**{**

**int var = bind(sockfd, my\_addr, addrlen);**

**if(var == -1)**

**{**

**perror("bind failed\n");**

**exit(-1);**

**}**

**return var;**

**}**

**int my\_setsockopt(int sockfd, int lev, int optname, const void\* optval, socklen\_t optlen)**

**{**

**int var = setsockopt(sockfd, lev, optname, optval, optlen);**

**if(var == -1)**

**{**

**perror("setsockopt error\n");**

**exit(-1);**

**}**

**return var;**

**}**

**int my\_accept(int s, struct sockaddr\* addr, socklen\_t\* addrlen)**

**{**

**int var = accept(s, addr, addrlen);**

**if(var == -1)**

**{**

**perror("accept error\n");**

**exit(-1);**

**}**

**return var;**

**}**

**help\_functions.hpp:**

**#pragma once**

**#ifndef PORT**

**#define PORT 8080**

**#endif**

**#ifndef MAXMSGLEN**

**#define MAXMSGLEN 300**

**#endif**

**void\* my\_malloc(size\_t);**

**int my\_socket(int,int,int);**

**int my\_inet\_pton(int,const char\*,void\*);**

**ssize\_t my\_send(int,const void\*,size\_t,int);**

**int my\_connect(int,struct sockaddr\*,socklen\_t);**

**int my\_read(int,void\*,size\_t);**

**int my\_listen(int,int);**

**int my\_bind(int,struct sockaddr\*,socklen\_t);**

**int my\_setsockopt(int,int,int,const void\*,socklen\_t);**

**int my\_accept(int,struct sockaddr\*,socklen\_t\*);**

**void get\_host(char\*,char\*);**

**history.cpp:**

**#include <iostream>**

**#include <string>**

**#include "history.hpp"**

**History::History()**

**{**

**if(sqlite3\_open\_v2(DATABASE, &db,**

**SQLITE\_OPEN\_NOMUTEX | SQLITE\_OPEN\_READWRITE | SQLITE\_OPEN\_CREATE, NULL))**

**{**

**perror("open/create database error\n");**

**sqlite3\_close(db);**

**exit(-1);**

**}**

**std::string sql\_request = "CREATE TABLE IF NOT EXISTS HISTORY";**

**sql\_request += "(MSG TEXT NOT NULL, NICKNAME TEXT NOT NULL);";**

**if(sqlite3\_exec(db, sql\_request.c\_str(), NULL, NULL, NULL))**

**{**

**perror("table create error\n");**

**}**

**}**

**int callback(void\* data, int argc, char\*\* argv, char\*\* azcolname)**

**{**

**for(int i = 0; i < argc; ++i)**

**{**

**std::cout << azcolname[i] << ": " << argv[i] << "\n";**

**}**

**return 0;**

**}**

**void History::add(std::string& msg, std::string& nickname)**

**{**

**std::string sql\_request = "INSERT INTO HISTORY(\'MSG\',\'NICKNAME\') VALUES(";**

**sql\_request += "\'" + msg + "\'" + ",\'" + nickname + "\'" + ");";**

**if(sqlite3\_exec(db, sql\_request.c\_str(), NULL, NULL, NULL))**

**{**

**perror("insert error\n");**

**}**

**}**

**void History::print()**

**{**

**std::string sql\_request = "SELECT \* FROM HISTORY;";**

**if(sqlite3\_exec(db, sql\_request.c\_str(), callback, NULL, NULL))**

**{**

**perror("select error\n");**

**}**

**}**

**History::~History()**

**{**

**sqlite3\_close(db);**

**}**

**history.hpp:**

**#pragma once**

**#include "sqlite3.h"**

**#ifndef DATABASE**

**#define DATABASE "history.db"**

**#endif**

**class History**

**{**

**private:**

**sqlite3\* db;**

**public:**

**void add(std::string&, std::string&);**

**void print();**

**~History();**

**History();**

**};**

**sqlite3.c**

**sqlite3.h**

**Пример работы**

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~/OS/KP$ make

g++ -std=c++11 -c server.cpp -o server.o

g++ -std=c++11 -c client.cpp -o client.o

g++ -std=c++11 -c help\_functions.cpp -o help\_functions.o

g++ -std=c++11 -c crypt.cpp -o crypt.o

g++ -std=c++11 -c history.cpp -o history.o

gcc -c sqlite3.c -o sqlite.o

g++ -std=c++11 server.o crypt.o help\_functions.o \

history.o sqlite.o -o server -ldl -lpthread

g++ -std=c++11 client.o crypt.o help\_functions.o \

history.o sqlite.o -o client -ldl –lpthread

**server**

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~/OS/KP$ ./server

Host name pe4eniks-HP-Laptop-14-dk0xxx

Host ip 127.0.1.1

MSG:

NICKNAME: pe4eniks

MSG:

NICKNAME: pe4eniks

MSG: hello

NICKNAME: pe4eniks

MSG:

NICKNAME: pe4eniks

MSG: hello

NICKNAME: pe4eniks

MSG: how r u?

NICKNAME: pe4eniks

MSG:

NICKNAME: pe4eniks

MSG: hello

NICKNAME: pe4eniks

MSG: how r u?

NICKNAME: pe4eniks

MSG: im ok

NICKNAME: pe4eniks

MSG:

NICKNAME: pe4eniks

MSG: hello

NICKNAME: pe4eniks

MSG: how r u?

NICKNAME: pe4eniks

MSG: im ok

NICKNAME: pe4eniks

MSG: history

NICKNAME: pe4eniks

**client**

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~/OS/KP$ ./client 127.0.1.1

Please enter your nick

pe4eniks

hello

how r u?

im ok

history

Messages history

MSG:

NICKNAME: pe4eniks

MSG: hello

NICKNAME: pe4eniks

MSG: how r u?

NICKNAME: pe4eniks

MSG: im ok

NICKNAME: pe4eniks

**Вывод**

При помощи сокетов можно удобно организовать работу архитектуры клиент-сервер. При помощи алгоритмов шифрования отправленные данные будут защищены, а историю сообщений можно хранить например в БД. Существует множество удобных СУБД – одна из них sqlite, большим ее преимуществом является простота встраивоемости. Как один из алгоритмов ассиметричного шифрования можно использовать RSA. Используя все эти технологии вместе, я написал простой мессенджер для передачи мгновенных сообщений.