

Vcalc Server

1.0

Создано системой Doxygen 1.9.8

1	Список файлов	1
1.1	Файлы	1
2	Файлы	3
2.1	Файл server.cpp	3
2.1.1	Подробное описание	4
2.1.2	Функции	4
2.1.2.1	checkAuth()	4
2.1.2.2	handleClient()	4
2.1.2.3	isHexString()	4
2.1.2.4	loadUsers()	4
2.1.2.5	logMsg()	4
2.1.2.6	main()	5
2.1.2.7	parseAuthString()	5
2.1.2.8	readAll()	5
2.1.2.9	readLittleEndian32()	5
2.1.2.10	writeAll()	6
2.1.2.11	writeLittleEndian32()	6
2.2	server.cpp	6
2.3	Файл sha256.cpp	10
2.3.1	Подробное описание	11
2.3.2	Функции	11
2.3.2.1	bsig0()	11
2.3.2.2	bsig1()	11
2.3.2.3	ch()	11
2.3.2.4	maj()	12
2.3.2.5	ROTR()	12
2.3.2.6	sha256()	12
2.3.2.7	SHR()	13
2.3.2.8	ssig0()	13
2.3.2.9	ssig1()	13
2.3.3	Переменные	13
2.3.3.1	k	13
2.4	sha256.cpp	14
2.5	Файл sha256.hpp	15
2.5.1	Подробное описание	15
2.5.2	Функции	16
2.5.2.1	sha256()	16
2.6	sha256.hpp	16
2.7	Файл tests/test_auth.cpp	17
2.7.1	Подробное описание	17
2.7.2	Функции	17
2.7.2.1	checkAuth()	17

2.7.2.2 main()	17
2.7.2.3 SUITE()	17
2.8 test_auth.cpp	18
2.9 Файл tests/test_cli.cpp	20
2.9.1 Подробное описание	20
2.9.2 Функции	20
2.9.2.1 cleanup_argv()	20
2.9.2.2 create_argv()	20
2.9.2.3 create_test_file()	20
2.9.2.4 main()	21
2.9.2.5 main_server()	21
2.9.2.6 SUITE()	21
2.10 test_cli.cpp	21
2.11 Файл tests/test_func.cpp	23
2.11.1 Подробное описание	23
2.11.2 Функции	24
2.11.2.1 check_server_running()	24
2.11.2.2 create_test_users()	24
2.11.2.3 main()	24
2.11.2.4 run_command()	24
2.11.2.5 stop_server()	24
2.11.2.6 SUITE()	24
2.12 test_func.cpp	25
2.13 Файл tests/test_protocol.cpp	27
2.13.1 Подробное описание	27
2.13.2 Функции	27
2.13.2.1 main()	27
2.13.2.2 SUITE()	28
2.14 test_protocol.cpp	28
2.15 Файл tests/test_sha256.cpp	29
2.15.1 Подробное описание	29
2.15.2 Функции	30
2.15.2.1 main()	30
2.15.2.2 SUITE()	30
2.16 test_sha256.cpp	30
2.17 Файл tests/test_vectors.cpp	32
2.17.1 Подробное описание	32
2.17.2 Функции	32
2.17.2.1 calculateSumOfSquares()	32
2.17.2.2 floatToLittleEndian()	32
2.17.2.3 littleEndianToFloat()	32
2.17.2.4 main()	33
2.17.2.5 SUITE()	33

2.18 test_vectors.cpp	33
---------------------------------	----

Глава 1

Список файлов

1.1 Файлы

Полный список файлов.

server.cpp	Сервер аутентификации и вычислений vcalc	3
sha256.cpp	Реализация алгоритма SHA-256	10
sha256.hpp	Заголовочный файл для реализации SHA-256	15
tests/ test_auth.cpp	Тесты аутентификации с использованием UnitTest++	17
tests/ test_cli.cpp	Тест интерфейса командной строки с использованием UnitTest++	20
tests/ test_func.cpp	Функциональные тесты сервера vcalc	23
tests/ test_protocol.cpp	Тесты протокола обмена с использованием UnitTest++	27
tests/ test_sha256.cpp	Тесты SHA-256 с использованием UnitTest++	29
tests/ test_vectors.cpp	Тесты вычисления векторов с использованием UnitTest++	32

Глава 2

Файлы

2.1 Файл server.cpp

Сервер аутентификации и вычислений vcalc.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <cstring>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/socket.h>
#include <ctime>
#include <iomanip>
#include "sha256.hpp"
#include <boost/program_options.hpp>
```

Функции

- void [logMsg](#) (const string &file, const string &msg)
- vector< pair< string, string > > [loadUsers](#) (const string &file)
- bool [checkAuth](#) (const string &login, const string &salt, const string &hash, const vector< pair< string, string > > &users)
- bool [readAll](#) (int sock, void *buf, size_t len)
- bool [writeAll](#) (int sock, const void *buf, size_t len)
- uint32_t [readLittleEndian32](#) (const uint8_t *bytes)
- void [writeLittleEndian32](#) (uint32_t value, uint8_t *bytes)
- bool [isHexString](#) (const string &str)
Проверяет, является ли строка шестнадцатеричной
- bool [parseAuthString](#) (const string &authStr, string &login, string &salt, string &hash)
Парсит строку аутентификации, поддерживая оба формата
- void [handleClient](#) (int sock, const vector< pair< string, string > > &users, const string &logFile)
- int [main](#) (int argc, char *argv[])

2.1.1 Подробное описание

Сервер аутентификации и вычислений vcalc.

См. определение в файле [server.cpp](#)

2.1.2 Функции

2.1.2.1 checkAuth()

```
bool checkAuth (
    const string & login,
    const string & salt,
    const string & hash,
    const vector< pair< string, string > > & users )
```

См. определение в файле [server.cpp](#) строка 42

2.1.2.2 handleClient()

```
void handleClient (
    int sock,
    const vector< pair< string, string > > & users,
    const string & logFile )
```

См. определение в файле [server.cpp](#) строка 157

2.1.2.3 isHexString()

```
bool isHexString (
    const string & str )
```

Проверяет, является ли строка шестнадцатеричной

См. определение в файле [server.cpp](#) строка 94

2.1.2.4 loadUsers()

```
vector< pair< string, string > > loadUsers (
    const string & file )
```

См. определение в файле [server.cpp](#) строка 30

2.1.2.5 logMsg()

```
void logMsg (
    const string & file,
    const string & msg )
```

См. определение в файле [server.cpp](#) строка 22

2.1.2.6 main()

```
int main (
    int argc,
    char * argv[] )
```

См. определение в файле [server.cpp](#) строка [253](#)

2.1.2.7 parseAuthString()

```
bool parseAuthString (
    const string & authStr,
    string & login,
    string & salt,
    string & hash )
```

Парсит строку аутентификации, поддерживая оба формата

Аргументы

authStr	Строка аутентификации
login	[out] Извлеченный логин
salt	[out] Извлеченная соль (16 hex символов)
hash	[out] Извлеченный хэш (64 hex символа)

Возвращает

true если успешно, false если ошибка

См. определение в файле [server.cpp](#) строка [113](#)

2.1.2.8 readAll()

```
bool readAll (
    int sock,
    void * buf,
    size_t len )
```

См. определение в файле [server.cpp](#) строка [60](#)

2.1.2.9 readLittleEndian32()

```
uint32_t readLittleEndian32 (
    const uint8_t * bytes )
```

См. определение в файле [server.cpp](#) строка [80](#)

2.1.2.10 writeAll()

```
bool writeAll (
    int sock,
    const void * buf,
    size_t len )
```

См. определение в файле [server.cpp](#) строка 70

2.1.2.11 writeLittleEndian32()

```
void writeLittleEndian32 (
    uint32_t value,
    uint8_t * bytes )
```

См. определение в файле [server.cpp](#) строка 84

2.2 server.cpp

[См. документацию.](#)

```
00001
00006 #include <iostream>
00007 #include <fstream>
00008 #include <vector>
00009 #include <string>
00010 #include <cstring>
00011 #include <arpa/inet.h>
00012 #include <unistd.h>
00013 #include <sys/socket.h>
00014 #include <ctime>
00015 #include <iomanip>
00016 #include "sha256.hpp"
00017 #include <boost/program_options.hpp>
00018
00019 namespace po = boost::program_options;
00020 using namespace std;
00021
00022 void logMsg(const string &file, const string &msg) {
00023     ofstream f(file, ios::app);
00024     if (!f) return;
00025     time_t t = time(nullptr);
00026     tm *tm = localtime(&t);
00027     f << put_time(tm, "%Y-%m-%d %H:%M:%S") << " | " << msg << endl;
00028 }
00029
00030 vector<pair<string,string>> loadUsers(const string &file) {
00031     vector<pair<string,string>> users;
00032     ifstream f(file);
00033     string line;
00034     while (getline(f, line)) {
00035         size_t p = line.find(':');
00036         if (p != string::npos)
00037             users.push_back({line.substr(0, p), line.substr(p + 1)});
00038     }
00039     return users;
00040 }
00041
00042 bool checkAuth(const string &login, const string &salt, const string &hash,
00043               const vector<pair<string,string>> &users) {
00044     for (const auto &[l, p] : users) {
00045         if (l == login) {
00046             string data = salt + p;
00047             uint8_t digest[32];
00048             sha256((uint8_t*)data.c_str(), data.size(), digest);
00049
00050             char hex[65];
00051             for (int i = 0; i < 32; i++) sprintf(hex + i*2, "%02X", digest[i]);
00052             hex[64] = '\0';
00053
00054             return string(hex) == hash;
00055         }
00056     }
```

```

00056     }
00057     return false;
00058 }
00059
00060 bool readAll(int sock, void *buf, size_t len) {
00061     size_t got = 0;
00062     while (got < len) {
00063         ssize_t n = read(sock, (char*)buf + got, len - got);
00064         if (n <= 0) return false;
00065         got += n;
00066     }
00067     return true;
00068 }
00069
00070 bool writeAll(int sock, const void *buf, size_t len) {
00071     size_t sent = 0;
00072     while (sent < len) {
00073         ssize_t n = write(sock, (char*)buf + sent, len - sent);
00074         if (n <= 0) return false;
00075         sent += n;
00076     }
00077     return true;
00078 }
00079
00080 uint32_t readLittleEndian32(const uint8_t* bytes) {
00081     return (bytes[0] << 0) | (bytes[1] << 8) | (bytes[2] << 16) | (bytes[3] << 24);
00082 }
00083
00084 void writeLittleEndian32(uint32_t value, uint8_t* bytes) {
00085     bytes[0] = (value >> 0) & 0xFF;
00086     bytes[1] = (value >> 8) & 0xFF;
00087     bytes[2] = (value >> 16) & 0xFF;
00088     bytes[3] = (value >> 24) & 0xFF;
00089 }
00090
00091 bool isHexString(const string& str) {
00092     for (char c : str) {
00093         if (!(c >= '0' && c <= '9' ||
00094             c >= 'A' && c <= 'F' ||
00095             c >= 'a' && c <= 'f')) {
00096             return false;
00097         }
00098     }
00099     return true;
00100 }
00101
00102 bool parseAuthString(const string& authStr, string& login, string& salt, string& hash) {
00103     // Подсчитываем количество двоеточий
00104     size_t colonCount = 0;
00105     for (char c : authStr) {
00106         if (c == ':') colonCount++;
00107     }
00108     // логин:соль:хэш
00109     if (colonCount == 2) {
00110         size_t firstColon = authStr.find(':');
00111         size_t secondColon = authStr.find(':', firstColon + 1);
00112         if (firstColon == string::npos || secondColon == string::npos ||
00113             firstColon == 0 || secondColon == firstColon + 1) {
00114             return false;
00115         }
00116         login = authStr.substr(0, firstColon);
00117         salt = authStr.substr(firstColon + 1, secondColon - firstColon - 1);
00118         hash = authStr.substr(secondColon + 1);
00119     }
00120     // логин(4) + соль(16) + хэш(64)
00121     else if (colonCount == 0 && authStr.length() == 84) {
00122         login = authStr.substr(0, 4);
00123         salt = authStr.substr(4, 16);
00124         hash = authStr.substr(20, 64);
00125     }
00126     else {
00127         return false;
00128     }
00129     // Проверка форматов
00130     if (login.empty() || salt.length() != 16 || hash.length() != 64) {
00131         return false;
00132     }
00133     // Проверка hex формата
00134     if (!isHexString(salt) || !isHexString(hash)) {
00135         return false;
00136     }
00137 }
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153

```

```

00154     return true;
00155 }
00156
00157 void handleClient(int sock, const vector<pair<string,string> &users, const string &logFile) {
00158     logMsg(logFile, "Клиент подключен");
00159
00160     // Аутентификация
00161     char auth[256];
00162     ssize_t n = read(sock, auth, sizeof(auth)-1);
00163     if (n <= 0) {
00164         logMsg(logFile, "Ошибка чтения аутентификации");
00165         close(sock);
00166         return;
00167     }
00168     auth[n] = '\0';
00169
00170     string authStr(auth);
00171
00172     // Парсинг строки аутентификации
00173     string login, salt, hash;
00174     if (!parseAuthString(authStr, login, salt, hash)) {
00175         writeAll(sock, "ERR", 3);
00176         logMsg(logFile, "Неверный формат аутентификации: " +
00177             (authStr.length() > 50 ? authStr.substr(0, 50) + "..." : authStr));
00178         close(sock);
00179         return;
00180     }
00181
00182     // Логируем формат (новый или старый)
00183     size_t colonCount = 0;
00184     for (char c : authStr) if (c == ':') colonCount++;
00185     string format = (colonCount == 2) ? "новый (логин:соль:хэш)" :
00186         (colonCount == 0 && authStr.length() == 84) ? "старый (логин4+соль16+хэш64)" : "неизвестный";
00187
00188     logMsg(logFile, "Аутентификация: " + login + " (формат: " + format + ")");
00189
00190     if (!checkAuth(login, salt, hash, users)) {
00191         writeAll(sock, "ERR", 3);
00192         logMsg(logFile, "Аутентификация отклонена: " + login);
00193         close(sock);
00194         return;
00195     }
00196
00197     writeAll(sock, "OK", 2);
00198     logMsg(logFile, "Клиент аутентифицирован: " + login);
00199
00200     // Обработка векторов
00201     uint8_t buffer[4];
00202     if (!readAll(sock, buffer, 4)) {
00203         logMsg(logFile, "Ошибка чтения количества векторов");
00204         close(sock);
00205         return;
00206     }
00207
00208     uint32_t numVectors = readLittleEndian32(buffer);
00209
00210     for (uint32_t i = 0; i < numVectors; i++) {
00211         if (!readAll(sock, buffer, 4)) {
00212             logMsg(logFile, "Ошибка чтения размера вектора");
00213             close(sock);
00214             return;
00215         }
00216
00217         uint32_t vectorSize = readLittleEndian32(buffer);
00218         float sum = 0.0f;
00219
00220         for (uint32_t j = 0; j < vectorSize; j++) {
00221             if (!readAll(sock, buffer, 4)) {
00222                 logMsg(logFile, "Ошибка чтения данных вектора");
00223                 close(sock);
00224                 return;
00225             }
00226
00227             uint32_t val = readLittleEndian32(buffer);
00228             float f;
00229             memcpy(&f, &val, sizeof(float));
00230             sum += f * f;
00231         }
00232
00233         logMsg(logFile, "Вектор " + to_string(i+1) + ": сумма квадратов = " + to_string(sum));
00234
00235         uint32_t resultBits;
00236         memcpy(&resultBits, &sum, sizeof(float));
00237         uint8_t resultBuffer[4];
00238         writeLittleEndian32(resultBits, resultBuffer);
00239
00240         if (!writeAll(sock, resultBuffer, 4)) {

```

```

00241         logMsg(logFile, "Ошибка отправки результата");
00242         close(sock);
00243         return;
00244     }
00245 }
00246
00247 logMsg(logFile, "Вычисления завершены для " + to_string(numVectors) + " векторов");
00248 close(sock);
00249 }
00250
00251 // Основная логика сервера
00252 #ifndef TEST_MODE
00253 int main(int argc, char *argv[]) {
00254     #else
00255     extern "C" int main_server(int argc, char *argv[]) {
00256     #endif
00257         string userFile = "users.txt";
00258         string logFile = "server.log";
00259         int port = 33333;
00260
00261         po::options_description desc("Сервер vcalc v1.0\n\nИспользование: server [options]\n\nДоступные опции");
00262         desc.add_options()
00263             ("help,h", "Показать справку")
00264             ("database,d", po::value<string>(&userFile)->default_value("users.txt"), "Файл с базой пользователей")
00265             ("log,l", po::value<string>(&logFile)->default_value("server.log"), "Файл логов")
00266             ("port,p", po::value<int>(&port)->default_value(33333), "Порт сервера");
00267
00268         po::variables_map vm;
00269         try {
00270             po::store(po::parse_command_line(argc, argv, desc), vm);
00271             po::notify(vm);
00272         } catch (exception &e) {
00273             #ifndef TEST_MODE
00274                 return 1;
00275             #else
00276                 cerr << "Ошибка: " << e.what() << endl << desc << endl;
00277                 return 1;
00278             #endif
00279         }
00280
00281         if (vm.count("help")) {
00282             #ifndef TEST_MODE
00283                 return 0;
00284             #else
00285                 cout << desc << endl;
00286                 return 0;
00287             #endif
00288         }
00289
00290         if (port <= 0 || port > 65535) {
00291             #ifndef TEST_MODE
00292                 return 1;
00293             #else
00294                 cerr << "Ошибка: Порт должен быть в диапазоне 1-65535" << endl;
00295                 logMsg(logFile, "ОШИБКА: Неверный порт " + to_string(port));
00296                 return 1;
00297             #endif
00298         }
00299
00300         #ifndef TEST_MODE
00301         logMsg(logFile, "=== Запуск сервера ===");
00302         #endif
00303
00304         auto users = loadUsers(userFile);
00305         if (users.empty()) {
00306             #ifndef TEST_MODE
00307                 return 1;
00308             #else
00309                 cerr << "Ошибка: Нет пользователей" << endl;
00310                 return 1;
00311             #endif
00312         }
00313
00314         #ifndef TEST_MODE
00315         return 0;
00316         #endif
00317
00318         int sock = socket(AF_INET, SOCK_STREAM, 0);
00319         if (sock < 0) {
00320             perror("Ошибка сокета");
00321             return 1;
00322         }
00323
00324         int opt = 1;
00325         setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
00326
00327         sockaddr_in addr = {};

```

```

00328     addr.sin_family = AF_INET;
00329     addr.sin_port = htons(port);
00330     addr.sin_addr.s_addr = INADDR_ANY;
00331
00332     if (bind(sock, (sockaddr*)&addr, sizeof(addr)) < 0) {
00333         perror("Ошибка привязки");
00334         close(sock);
00335         return 1;
00336     }
00337
00338     if (listen(sock, 5) < 0) {
00339         perror("Ошибка прослушивания");
00340         close(sock);
00341         return 1;
00342     }
00343
00344     cout << "Сервер запущен на порту " << port << endl;
00345
00346     while (true) {
00347         sockaddr_in client;
00348         socklen_t len = sizeof(client);
00349         int clientSock = accept(sock, (sockaddr*)&client, &len);
00350         if (clientSock < 0) continue;
00351         handleClient(clientSock, users, logFile);
00352     }
00353
00354     close(sock);
00355     return 0;
00356 }

```

2.3 Файл sha256.cpp

Реализация алгоритма SHA-256.

```

#include "sha256.hpp"
#include <stdint>
#include <cstdlib>
#include <cstring>

```

Функции

- static uint32_t **ROTR** (uint32_t x, int n)
Циклический сдвиг вправо
- static uint32_t **SHR** (uint32_t x, int n)
Логический сдвиг вправо
- static uint32_t **ch** (uint32_t x, uint32_t y, uint32_t z)
Функция Ch(x, y, z)
- static uint32_t **maj** (uint32_t x, uint32_t y, uint32_t z)
Функция Maj(x, y, z)
- static uint32_t **bsig0** (uint32_t x)
Функция Sigma0(x)
- static uint32_t **bsig1** (uint32_t x)
Функция Sigma1(x)
- static uint32_t **ssig0** (uint32_t x)
Функция sigma0(x)
- static uint32_t **ssig1** (uint32_t x)
Функция sigma1(x)
- void **sha256** (const uint8_t *data, size_t len, uint8_t out[32])
Вычисляет хэш SHA-256 для входных данных

Переменные

- `static const uint32_t k [64]`

Константы SHA-256 (первые 32 бита дробных частей кубических корней первых 64 простых чисел)

2.3.1 Подробное описание

Реализация алгоритма SHA-256.

Автор

[Ваше имя]

Дата

2024

Алгоритм хеширования SHA-256 согласно стандарту FIPS 180-4. Используется для аутентификации пользователей.

См. определение в файле [sha256.cpp](#)

2.3.2 Функции

2.3.2.1 `bsig0()`

```
static uint32_t bsig0 (  
    uint32_t x )    [inline], [static]
```

Функция `Sigma0(x)`

См. определение в файле [sha256.cpp](#) строка 61

2.3.2.2 `bsig1()`

```
static uint32_t bsig1 (  
    uint32_t x )    [inline], [static]
```

Функция `Sigma1(x)`

См. определение в файле [sha256.cpp](#) строка 68

2.3.2.3 `ch()`

```
static uint32_t ch (  
    uint32_t x,  
    uint32_t y,  
    uint32_t z )    [inline], [static]
```

Функция `Ch(x, y, z)`

См. определение в файле [sha256.cpp](#) строка 47

2.3.2.4 maj()

```
static uint32_t maj (
    uint32_t x,
    uint32_t y,
    uint32_t z ) [inline], [static]
```

Функция Maj(x, y, z)

См. определение в файле [sha256.cpp](#) строка 54

2.3.2.5 ROTR()

```
static uint32_t ROTR (
    uint32_t x,
    int n ) [inline], [static]
```

Циклический сдвиг вправо

См. определение в файле [sha256.cpp](#) строка 19

2.3.2.6 sha256()

```
void sha256 (
    const uint8_t * data,
    size_t len,
    uint8_t out[32] )
```

Вычисляет хэш SHA-256 для входных данных

Вычисляет хэш SHA-256 для данных

Аргументы

data	Входные данные для хеширования
len	Длина входных данных в байтах
out	Выходной массив для хэша (32 байта)

Реализация алгоритма SHA-256:

1. Инициализация хэш-значений
2. Подготовка сообщения с дополнением
3. Обработка блоков по 512 бит
4. Генерация итогового хэша

Заметки

Функция выделяет динамическую память для временного буфера

См. определение в файле [sha256.cpp](#) строка 101

2.3.2.7 SHR()

```
static uint32_t SHR (
    uint32_t x,
    int n )    [inline], [static]
```

Логический сдвиг вправо

См. определение в файле [sha256.cpp](#) строка 26

2.3.2.8 ssig0()

```
static uint32_t ssig0 (
    uint32_t x )    [inline], [static]
```

Функция sigma0(x)

См. определение в файле [sha256.cpp](#) строка 75

2.3.2.9 ssig1()

```
static uint32_t ssig1 (
    uint32_t x )    [inline], [static]
```

Функция sigma1(x)

См. определение в файле [sha256.cpp](#) строка 82

2.3.3 Переменные

2.3.3.1 k

```
const uint32_t k[64]    [static]
```

Инициализатор

```
= {
    0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,0x3956c25b,0x59f111f1,0x923f82a4,0xab1c5ed5,
    0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,0x72be5d74,0x80deb1fe,0x9bdc06a7,0xc19bf174,
    0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240ca1cc,0x2de92c6f,0x4a7484aa,0x5cb0a9dc,0x76f988da,
    0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,0xc6e00bf3,0xd5a79147,0x06ca6351,0x14292967,
    0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,0x650a7354,0x766a0abb,0x81c2c92e,0x92722c85,
    0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,0xd192e819,0xd6990624,0xf40e3585,0x106aa070,
    0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,0x391c0cb3,0x4ed8aa4a,0x5b9cca4f,0x682e6ff3,
    0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,0x90b efffa,0xa4506ceb,0xbef9a3f7,0xc67178f2
}
```

Константы SHA-256 (первые 32 бита дробных частей кубических корней первых 64 простых чисел)

См. определение в файле [sha256.cpp](#) строка 33

2.4 sha256.cpp

См. документацию.

```
00001
00011 #include "sha256.hpp"
00012 #include <stdint>
00013 #include <cstdlib>
00014 #include <cstring>
00015
00019 static inline uint32_t ROTR(uint32_t x, int n) {
00020     return (x » n) | (x « (32-n));
00021 }
00022
00026 static inline uint32_t SHR(uint32_t x, int n) {
00027     return x » n;
00028 }
00029
00033 static const uint32_t k[64] = {
00034     0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
00035     0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
00036     0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
00037     0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
00038     0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
00039     0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
00040     0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
00041     0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90bafffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
00042 };
00043
00047 static inline uint32_t ch(uint32_t x, uint32_t y, uint32_t z){
00048     return (x & y) ^ (~x & z);
00049 }
00050
00054 static inline uint32_t maj(uint32_t x, uint32_t y, uint32_t z){
00055     return (x & y) ^ (x & z) ^ (y & z);
00056 }
00057
00061 static inline uint32_t bsig0(uint32_t x){
00062     return ROTR(x,2) ^ ROTR(x,13) ^ ROTR(x,22);
00063 }
00064
00068 static inline uint32_t bsig1(uint32_t x){
00069     return ROTR(x,6) ^ ROTR(x,11) ^ ROTR(x,25);
00070 }
00071
00075 static inline uint32_t ssig0(uint32_t x){
00076     return ROTR(x,7) ^ ROTR(x,18) ^ SHR(x,3);
00077 }
00078
00082 static inline uint32_t ssig1(uint32_t x){
00083     return ROTR(x,17) ^ ROTR(x,19) ^ SHR(x,10);
00084 }
00085
00101 void sha256(const uint8_t *data, size_t len, uint8_t out[32]) {
00102     // Инициализация начальных хэш-значений
00103     uint32_t h[8] = {
00104         0x6a09e667UL, 0xbb67ae85UL, 0x3c6ef372UL, 0xa54ff53aUL,
00105         0x510e527fUL, 0x9b05688cUL, 0x1f83d9abUL, 0x5be0cd19UL
00106     };
00107
00108     // Подготовка сообщения (дополнение до кратного 512 битам)
00109     size_t num_blocks = ((len + 9) + 63) / 64;
00110     uint8_t *msg = (uint8_t*)calloc(num_blocks*64, 1);
00111     memcpy(msg, data, len);
00112
00113     // Добавление бита '1'
00114     msg[len] = 0x80;
00115
00116     // Добавление длины сообщения в битах (64 бита, старший байт вперед)
00117     uint64_t bit_len = (uint64_t)len * 8;
00118     uint8_t *p = msg + num_blocks*64 - 8;
00119     for (int i = 7; i >= 0; i--) {
00120         p[i] = bit_len & 0xFF;
00121         bit_len »= 8;
00122     }
00123
00124     // Обработка каждого 512-битного блока
00125     for (size_t b = 0; b < num_blocks; b++) {
00126         uint32_t w[64];
00127         const uint8_t *chunk = msg + b*64;
00128
00129         // Подготовка расписания сообщений
00130         for (int t = 0; t < 16; t++) {
00131             w[t] = (chunk[t*4] « 24) | (chunk[t*4+1] « 16) |
00132                 (chunk[t*4+2] « 8) | (chunk[t*4+3]);
00133         }
00134     }
00135 }
```

```

00134
00135     for (int t = 16; t < 64; t++) {
00136         w[t] = ssig1(w[t-2]) + w[t-7] + ssig0(w[t-15]) + w[t-16];
00137     }
00138
00139     // Инициализация рабочих переменных
00140     uint32_t a = h[0];
00141     uint32_t b0 = h[1];
00142     uint32_t c = h[2];
00143     uint32_t d = h[3];
00144     uint32_t e = h[4];
00145     uint32_t f = h[5];
00146     uint32_t g = h[6];
00147     uint32_t hh = h[7];
00148
00149     // Основной цикл сжатия
00150     for (int t = 0; t < 64; t++) {
00151         uint32_t T1 = hh + bsig1(e) + ch(e, f, g) + k[t] + w[t];
00152         uint32_t T2 = bsig0(a) + maj(a, b0, c);
00153         hh = g;
00154         g = f;
00155         f = e;
00156         e = d + T1;
00157         d = c;
00158         c = b0;
00159         b0 = a;
00160         a = T1 + T2;
00161     }
00162
00163     // Обновление хэш-значений
00164     h[0] += a;
00165     h[1] += b0;
00166     h[2] += c;
00167     h[3] += d;
00168     h[4] += e;
00169     h[5] += f;
00170     h[6] += g;
00171     h[7] += hh;
00172 }
00173
00174 // Освобождение памяти
00175 free(msg);
00176
00177 // Преобразование хэш-значений в байтовый массив (big-endian)
00178 for (int i = 0; i < 8; i++) {
00179     out[i*4] = (h[i] >> 24) & 0xFF;
00180     out[i*4+1] = (h[i] >> 16) & 0xFF;
00181     out[i*4+2] = (h[i] >> 8) & 0xFF;
00182     out[i*4+3] = (h[i] & 0xFF);
00183 }
00184 }

```

2.5 Файл sha256.hpp

Заголовочный файл для реализации SHA-256.

```

#include <stdint>
#include <stddef>

```

Функции

- void [sha256](#) (const uint8_t *data, size_t len, uint8_t out[32])
Вычисляет хэш SHA-256 для данных

2.5.1 Подробное описание

Заголовочный файл для реализации SHA-256.

См. определение в файле [sha256.hpp](#)

2.5.2 Функции

2.5.2.1 sha256()

```
void sha256 (
    const uint8_t * data,
    size_t len,
    uint8_t out[32] )
```

Вычисляет хэш SHA-256 для данных

Аргументы

data	Указатель на входные данные
len	Длина данных в байтах
out	Массив для записи результата (32 байта)

Вычисляет хэш SHA-256 для данных

Аргументы

data	Входные данные для хеширования
len	Длина входных данных в байтах
out	Выходной массив для хэша (32 байта)

Реализация алгоритма SHA-256:

1. Инициализация хэш-значений
2. Подготовка сообщения с дополнением
3. Обработка блоков по 512 бит
4. Генерация итогового хэша

Заметки

Функция выделяет динамическую память для временного буфера

См. определение в файле [sha256.cpp](#) строка 101

2.6 sha256.hpp

[См. документацию.](#)

```
00001
00006 #pragma once
00007 #include <cstdint>
00008 #include <cstddef>
00009
00016 void sha256(const uint8_t *data, size_t len, uint8_t out[32]);
```

2.7 Файл tests/test_auth.cpp

Тесты аутентификации с использованием `UnitTest++`.

```
#include <UnitTest++/UnitTest++.h>
#include <vector>
#include <string>
#include <cstring>
#include "../sha256.hpp"
```

Функции

- bool `checkAuth` (const std::string &login, const std::string &salt, const std::string &hash, const std::vector< std::pair< std::string, std::string > > &users)
- `SUITE` (AuthTests)
- int `main` ()

2.7.1 Подробное описание

Тесты аутентификации с использованием `UnitTest++`.

См. определение в файле `test_auth.cpp`

2.7.2 Функции

2.7.2.1 `checkAuth()`

```
bool checkAuth (
    const std::string & login,
    const std::string & salt,
    const std::string & hash,
    const std::vector< std::pair< std::string, std::string > > & users )
```

См. определение в файле `test_auth.cpp` строка 12

2.7.2.2 `main()`

```
int main ( )
```

См. определение в файле `test_auth.cpp` строка 170

2.7.2.3 `SUITE()`

```
SUITE (
    AuthTests )
```

См. определение в файле `test_auth.cpp` строка 33

2.8 test_auth.cpp

См. документацию.

```

00001
00006 #include <UnitTest++/UnitTest++.h>
00007 #include <vector>
00008 #include <string>
00009 #include <cstring>
00010 #include "../sha256.hpp"
00011
00012 bool checkAuth(const std::string& login, const std::string& salt,
00013               const std::string& hash,
00014               const std::vector<std::pair<std::string, std::string>> & users) {
00015     for (const auto& [l, p] : users) {
00016         if (l == login) {
00017             std::string data = salt + p;
00018             uint8_t digest[32];
00019             sha256((uint8_t*)data.c_str(), data.size(), digest);
00020
00021             char hex[65];
00022             for (int i = 0; i < 32; i++) {
00023                 sprintf(hex + i*2, "%02X", digest[i]);
00024             }
00025             hex[64] = '\0';
00026
00027             return std::string(hex) == hash;
00028         }
00029     }
00030     return false;
00031 }
00032
00033 SUITE(AuthTests) {
00034     std::vector<std::pair<std::string, std::string>> users = {
00035         {"user", "P@ssW0rd"},
00036         {"admin", "Admin123"},
00037         {"test", "Test456"}
00038     };
00039
00040     // Тест 1: Правильная аутентификация
00041     TEST(CorrectAuthentication) {
00042         std::string login = "user";
00043         std::string salt = "A1B2C3D4E5F67890";
00044         std::string password = "P@ssW0rd";
00045         std::string data = salt + password;
00046
00047         uint8_t digest[32];
00048         sha256((uint8_t*)data.c_str(), data.size(), digest);
00049
00050         char hex[65];
00051         for (int i = 0; i < 32; i++) {
00052             sprintf(hex + i*2, "%02X", digest[i]);
00053         }
00054
00055         CHECK(checkAuth(login, salt, std::string(hex), users));
00056     }
00057
00058     // Тест 2: Неправильный пароль
00059     TEST(WrongPassword) {
00060         std::string wrong_hash = "0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF";
00061         CHECK(!checkAuth("user", "A1B2C3D4E5F67890", wrong_hash, users));
00062     }
00063
00064     // Тест 3: Несуществующий пользователь
00065     TEST(NonExistentUser) {
00066         std::string dummy_hash = "0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF";
00067         CHECK(!checkAuth("nonexistent", "A1B2C3D4E5F67890", dummy_hash, users));
00068     }
00069
00070     // Тест 4: Пустая база пользователей
00071     TEST(EmptyUserDatabase) {
00072         std::vector<std::pair<std::string, std::string>> empty_users;
00073         std::string dummy_hash = "0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF";
00074         CHECK(!checkAuth("user", "A1B2C3D4E5F67890", dummy_hash, empty_users));
00075     }
00076
00077     // Тест 5: Несколько пользователей
00078     TEST(MultipleUsers) {
00079         std::vector<std::pair<std::string, std::string>> multi_users = {
00080             {"alice", "password1"},
00081             {"bob", "password2"},
00082             {"charlie", "password3"}
00083         };
00084
00085         bool all_correct = true;
00086         for (const auto& [l, p] : multi_users) {

```

```

00087         std::string s = "SALT1234567890AB";
00088         std::string d = s + p;
00089         uint8_t dig[32];
00090         sha256((uint8_t*)d.c_str(), d.size(), dig);
00091         char h[65];
00092         for (int i = 0; i < 32; i++) {
00093             sprintf(h + i*2, "%02X", dig[i]);
00094         }
00095         if (!checkAuth(l, s, h, multi_users)) {
00096             all_correct = false;
00097             break;
00098         }
00099     }
00100     CHECK(all_correct);
00101 }
00102
00103 // Тест 6: Формат соли
00104 TEST(SaltFormat) {
00105     std::string salt = "0123456789ABCDEF";
00106     CHECK_EQUAL(16, salt.length());
00107
00108     for (char c : salt) {
00109         CHECK((c >= '0' && c <= '9') || (c >= 'A' && c <= 'F') || (c >= 'a' && c <= 'f'));
00110     }
00111 }
00112
00113 // Тест 7: Формат хэша
00114 TEST(HashFormat) {
00115     std::string hash = "0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF";
00116     CHECK_EQUAL(64, hash.length());
00117
00118     for (char c : hash) {
00119         CHECK((c >= '0' && c <= '9') || (c >= 'A' && c <= 'F') || (c >= 'a' && c <= 'f'));
00120     }
00121 }
00122
00123 // Тест 8: Разная соль для одного пароля
00124 TEST(DifferentSaltDifferentHash) {
00125     std::string pass = "mypassword";
00126     std::string salt1 = "1111111111111111";
00127     std::string salt2 = "2222222222222222";
00128
00129     std::string d1 = salt1 + pass;
00130     std::string d2 = salt2 + pass;
00131
00132     uint8_t dig1[32], dig2[32];
00133     sha256((uint8_t*)d1.c_str(), d1.size(), dig1);
00134     sha256((uint8_t*)d2.c_str(), d2.size(), dig2);
00135
00136     CHECK(memcmp(dig1, dig2, 32) != 0);
00137 }
00138
00139 // Тест 9: Одинаковая соль и пароль дают одинаковый хэш
00140 TEST(SameInputSameHash) {
00141     std::string pass = "password123";
00142     std::string salt = "3333333333333333";
00143     std::string d1 = salt + pass;
00144     std::string d2 = salt + pass;
00145
00146     uint8_t dig1[32], dig2[32];
00147     sha256((uint8_t*)d1.c_str(), d1.size(), dig1);
00148     sha256((uint8_t*)d2.c_str(), d2.size(), dig2);
00149
00150     CHECK_ARRAY_EQUAL(dig1, dig2, 32);
00151 }
00152
00153 // Тест 10: Чувствительность к регистру пароля
00154 TEST(CaseSensitivePassword) {
00155     std::string pass_lower = "password";
00156     std::string pass_upper = "PASSWORD";
00157     std::string salt = "4444444444444444";
00158
00159     std::string d1 = salt + pass_lower;
00160     std::string d2 = salt + pass_upper;
00161
00162     uint8_t dig1[32], dig2[32];
00163     sha256((uint8_t*)d1.c_str(), d1.size(), dig1);
00164     sha256((uint8_t*)d2.c_str(), d2.size(), dig2);
00165
00166     CHECK(memcmp(dig1, dig2, 32) != 0);
00167 }
00168 }
00169
00170 int main() {
00171     return UnitTest::RunAllTests();
00172 }

```

2.9 Файл tests/test_cli.cpp

Тест интерфейса командной строки с использованием `UnitTest++`.

```
#include <UnitTest++/UnitTest++.h>
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
#include <cstring>
#include <vector>
#include <sstream>
```

Функции

- `int main_server (int argc, char *argv[])`
- `void create_test_file (const string &filename, const string &content)`
- `vector< char * > create_argv (const vector< string > &args)`
- `void cleanup_argv (vector< char * > &argv)`
- `SUITE (CLITests)`
- `int main ()`

2.9.1 Подробное описание

Тест интерфейса командной строки с использованием `UnitTest++`.

См. определение в файле `test_cli.cpp`

2.9.2 Функции

2.9.2.1 cleanup_argv()

```
void cleanup_argv (
    vector< char * > & argv )
```

См. определение в файле `test_cli.cpp` строка 43

2.9.2.2 create_argv()

```
vector< char * > create_argv (
    const vector< string > & args )
```

См. определение в файле `test_cli.cpp` строка 28

2.9.2.3 create_test_file()

```
void create_test_file (
    const string & filename,
    const string & content )
```

См. определение в файле `test_cli.cpp` строка 21

2.9.2.4 main()

```
int main ( )
```

См. определение в файле [test_cli.cpp](#) строка 161

2.9.2.5 main_server()

```
int main_server (
    int argc,
    char * argv[] )
```

2.9.2.6 SUITE()

```
SUITE (
    CLITests )
```

См. определение в файле [test_cli.cpp](#) строка 49

2.10 test_cli.cpp

[См. документацию.](#)

```
00001
00006 #include <UnitTest++/UnitTest++.h>
00007 #include <iostream>
00008 #include <string>
00009 #include <fstream>
00010 #include <cstdlib>
00011 #include <cstring>
00012 #include <vector>
00013 #include <sstream>
00014
00015 // Объявление функции main_server из server.cpp
00016 extern "C" int main_server(int argc, char* argv[]);
00017
00018 using namespace std;
00019
00020 // Вспомогательная функция для создания тестовых файлов
00021 void create_test_file(const string& filename, const string& content) {
00022     ofstream file(filename);
00023     file << content;
00024     file.close();
00025 }
00026
00027 // Вспомогательная функция для преобразования vector<string> в char* argv[]
00028 vector<char*> create_argv(const vector<string>& args) {
00029     vector<char*> argv;
00030     argv.push_back(const_cast<char*>("server")); // argv[0]
00031
00032     for (const auto& arg : args) {
00033         char* arg_copy = new char[arg.size() + 1];
00034         strcpy(arg_copy, arg.c_str());
00035         argv.push_back(arg_copy);
00036     }
00037
00038     argv.push_back(nullptr); // NULL terminator
00039     return argv;
00040 }
00041
00042 // Вспомогательная функция для очистки argv
00043 void cleanup_argv(vector<char*>& argv) {
00044     for (size_t i = 1; i < argv.size() - 1; i++) {
00045         delete[] argv[i];
00046     }
00047 }
00048
00049 SUITE(CLITests) {
```

```

00050
00051 struct Setup {
00052     Setup() {
00053         // Создаем тестовые файлы пользователей
00054         create_test_file("test_users.txt", "user:pass\nadmin:admin123\n");
00055     }
00056
00057     ~Setup() {
00058         // Удаляем временные файлы
00059         remove("test_users.txt");
00060         remove("empty_users.txt");
00061         remove("test.log");
00062         remove("server.log");
00063     }
00064 };
00065
00066 // Тест 1: Опция --help
00067 TEST_FIXTURE(Setup, TestHelpFlag) {
00068     vector<string> args = {"--help"};
00069     vector<char*> argv = create_argv(args);
00070     int result = main_server(args.size() + 1, argv.data());
00071     cleanup_argv(argv);
00072     CHECK_EQUAL(0, result);
00073 }
00074
00075 // Тест 2: Опция -h (краткая справка)
00076 TEST_FIXTURE(Setup, TestShortHelpFlag) {
00077     vector<string> args = {"-h"};
00078     vector<char*> argv = create_argv(args);
00079     int result = main_server(args.size() + 1, argv.data());
00080     cleanup_argv(argv);
00081     CHECK_EQUAL(0, result);
00082 }
00083
00084 // Тест 3: Неизвестный аргумент
00085 TEST_FIXTURE(Setup, TestInvalidArgument) {
00086     vector<string> args = {"--invalid-arg"};
00087     vector<char*> argv = create_argv(args);
00088     int result = main_server(args.size() + 1, argv.data());
00089     cleanup_argv(argv);
00090     CHECK(result != 0);
00091 }
00092
00093 // Тест 4: Неверный порт (строка)
00094 TEST_FIXTURE(Setup, TestInvalidPortString) {
00095     vector<string> args = {"-p", "not_a_number", "-d", "test_users.txt"};
00096     vector<char*> argv = create_argv(args);
00097     int result = main_server(args.size() + 1, argv.data());
00098     cleanup_argv(argv);
00099     CHECK(result != 0);
00100 }
00101
00102 // Тест 5: Неверный порт (ноль)
00103 TEST_FIXTURE(Setup, TestInvalidPortZero) {
00104     vector<string> args = {"-p", "0", "-d", "test_users.txt"};
00105     vector<char*> argv = create_argv(args);
00106     int result = main_server(args.size() + 1, argv.data());
00107     cleanup_argv(argv);
00108     CHECK(result != 0);
00109 }
00110
00111 // Тест 6: Неверный порт (слишком большой)
00112 TEST_FIXTURE(Setup, TestInvalidPortTooLarge) {
00113     vector<string> args = {"-p", "70000", "-d", "test_users.txt"};
00114     vector<char*> argv = create_argv(args);
00115     int result = main_server(args.size() + 1, argv.data());
00116     cleanup_argv(argv);
00117     CHECK(result != 0);
00118 }
00119
00120 // Тест 7: Корректный порт с --help
00121 TEST_FIXTURE(Setup, TestValidPortWithHelp) {
00122     vector<string> args = {"-p", "8080", "-d", "test_users.txt", "--help"};
00123     vector<char*> argv = create_argv(args);
00124     int result = main_server(args.size() + 1, argv.data());
00125     cleanup_argv(argv);
00126     CHECK_EQUAL(0, result);
00127 }
00128
00129 // Тест 8: Отсутствующий файл пользователей
00130 TEST_FIXTURE(Setup, TestMissingDatabaseFile) {
00131     vector<string> args = {"--database=nonexistent.txt"};
00132     vector<char*> argv = create_argv(args);
00133     int result = main_server(args.size() + 1, argv.data());
00134     cleanup_argv(argv);
00135     CHECK(result != 0);
00136 }

```

```

00137
00138 // Тест 9: Пустой файл пользователей
00139 TEST_FIXTURE(Setup, TestEmptyDatabaseFile) {
00140     create_test_file("empty_users.txt", "");
00141
00142     vector<string> args = {"-d", "empty_users.txt"};
00143     vector<char*> argv = create_argv(args);
00144     int result = main_server(args.size() + 1, argv.data());
00145     cleanup_argv(argv);
00146     CHECK(result != 0);
00147
00148     remove("empty_users.txt");
00149 }
00150
00151 // Тест 10: Все опции с --help
00152 TEST_FIXTURE(Setup, TestAllOptionsWithHelp) {
00153     vector<string> args = {"--database=test_users.txt", "--log=test.log", "--port=8080", "--help"};
00154     vector<char*> argv = create_argv(args);
00155     int result = main_server(args.size() + 1, argv.data());
00156     cleanup_argv(argv);
00157     CHECK_EQUAL(0, result);
00158 }
00159 }
00160
00161 int main() {
00162     return UnitTest::RunAllTests();
00163 }

```

2.11 Файл tests/test_func.cpp

Функциональные тесты сервера vcalc.

```

#include <UnitTest++/UnitTest++.h>
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <vector>
#include <thread>
#include <chrono>

```

Функции

- int run_command (const string &cmd, int timeout_seconds=5)
- bool check_server_running (int port)
- void stop_server (int port)
- void create_test_users (const string &filename)
- SUITE (SimpleFunctionalTests)
- int main ()

2.11.1 Подробное описание

Функциональные тесты сервера vcalc.

См. определение в файле test_func.cpp

2.11.2 Функции

2.11.2.1 check_server_running()

```
bool check_server_running (
    int port )
```

См. определение в файле [test_func.cpp](#) строка 25

2.11.2.2 create_test_users()

```
void create_test_users (
    const string & filename )
```

См. определение в файле [test_func.cpp](#) строка 48

2.11.2.3 main()

```
int main ( )
```

См. определение в файле [test_func.cpp](#) строка 194

2.11.2.4 run_command()

```
int run_command (
    const string & cmd,
    int timeout_seconds = 5 )
```

См. определение в файле [test_func.cpp](#) строка 19

2.11.2.5 stop_server()

```
void stop_server (
    int port )
```

См. определение в файле [test_func.cpp](#) строка 40

2.11.2.6 SUITE()

```
SUITE (
    SimpleFunctionalTests )
```

См. определение в файле [test_func.cpp](#) строка 56

2.12 test_func.cpp

[См. документацию.](#)

```

00001
00006 #include <UnitTest++/UnitTest++.h>
00007 #include <iostream>
00008 #include <cstdlib>
00009 #include <string>
00010 #include <fstream>
00011 #include <vector>
00012 #include <thread>
00013 #include <chrono>
00014
00015 using namespace std;
00016 using namespace std::chrono;
00017
00018 // Запуск команды с таймаутом
00019 int run_command(const string& cmd, int timeout_seconds = 5) {
00020     string timed_cmd = "timeout " + to_string(timeout_seconds) + " " + cmd;
00021     return system(timed_cmd.c_str());
00022 }
00023
00024 // Проверка занятости порта
00025 bool check_server_running(int port) {
00026     string cmd = "lsof -ti:" + to_string(port) + " 2>/dev/null | wc -l";
00027     FILE* pipe = popen(cmd.c_str(), "r");
00028     if (!pipe) return false;
00029
00030     char buffer[16];
00031     if (fgets(buffer, sizeof(buffer), pipe) != nullptr) {
00032         // ignore
00033     }
00034     pclose(pipe);
00035
00036     return atoi(buffer) > 0;
00037 }
00038
00039 // Остановка сервера на порту
00040 void stop_server(int port) {
00041     string cmd = "kill -f `./server.*-p " + to_string(port) + "' 2>/dev/null";
00042     int result = system(cmd.c_str());
00043     (void)result;
00044     this_thread::sleep_for(seconds(1));
00045 }
00046
00047 // Создание тестового файла пользователей
00048 void create_test_users(const string& filename) {
00049     ofstream file(filename);
00050     file << "user1:password1" << endl;
00051     file << "user2:password2" << endl;
00052     file << "admin:admin123" << endl;
00053     file.close();
00054 }
00055
00056 SUITE(SimpleFunctionalTests) {
00057     TEST(ServerStartStop) {
00058         cout << "\n[1] Тест запуска и остановки сервера" << endl;
00059
00060         int port = 33444;
00061
00062         create_test_users("test_users.txt");
00063
00064         string start_cmd = "./server -d test_users.txt -l test.log -p " + to_string(port) + " 2>&1";
00065
00066         pid_t pid = fork();
00067         if (pid == 0) {
00068             execl("/bin/sh", "sh", "-c", start_cmd.c_str(), NULL);
00069             exit(0);
00070         }
00071
00072         this_thread::sleep_for(seconds(2));
00073
00074         CHECK(check_server_running(port));
00075         cout << " Сервер запущен на порту " << port << endl;
00076
00077         stop_server(port);
00078
00079         CHECK(!check_server_running(port));
00080         cout << " Сервер корректно остановлен" << endl;
00081
00082         system("rm -f test_users.txt test.log 2>/dev/null");
00083     }
00084 }
00085
00086 TEST(InvalidArgumentsNoCrash) {

```

```

00087     cout << "\n[2] Тест некорректных аргументов" << endl;
00088
00089     vector<string> bad_args = {
00090         "/server --invalid-option",
00091         "/server -p not_a_number",
00092         "/server -p 0",
00093         "/server -p -100",
00094         "/server -d nonexistent_file.txt"
00095     };
00096
00097     for (const auto& cmd : bad_args) {
00098         string full_cmd = cmd + " 2>&1";
00099         int result = run_command(full_cmd, 2);
00100
00101         CHECK(result != 124); // Код таймаута
00102
00103         if (result != 124) {
00104             cout << " Команда '" << cmd << "' обработана без падения" << endl;
00105         } else {
00106             cout << " Команда '" << cmd << "' зависла" << endl;
00107         }
00108     }
00109 }
00110
00111
00112 TEST(HelpOption) {
00113     cout << "\n[3] Тест справки" << endl;
00114
00115     string cmd = "/server --help 2>&1";
00116     int result = run_command(cmd, 2);
00117     CHECK_EQUAL(0, result);
00118
00119     cout << " Справка отображается корректно" << endl;
00120 }
00121
00122
00123 TEST(DifferentPorts) {
00124     cout << "\n[4] Тест разных портов" << endl;
00125
00126     vector<int> ports = {33555, 33666, 33777};
00127
00128     for (int port : ports) {
00129         string user_file = "test_users_" + to_string(port) + ".txt";
00130         string log_file = "test_" + to_string(port) + ".log";
00131
00132         create_test_users(user_file);
00133
00134         string start_cmd = "/server -d " + user_file + " -l " + log_file +
00135             " -p " + to_string(port) + " 2>&1 &";
00136         system(start_cmd.c_str());
00137
00138         this_thread::sleep_for(seconds(2));
00139
00140         bool running = check_server_running(port);
00141         if (running) {
00142             cout << " Сервер запущен на порту " << port << endl;
00143
00144             stop_server(port);
00145             CHECK(!check_server_running(port));
00146         } else {
00147             cout << " Сервер не запустился на порту " << port << endl;
00148         }
00149
00150         string cleanup = "rm -f " + user_file + " " + log_file + " 2>/dev/null";
00151         system(cleanup.c_str());
00152     }
00153 }
00154
00155
00156 TEST(MultipleServers) {
00157     cout << "\n[5] Тест нескольких серверов одновременно" << endl;
00158
00159     int port1 = 33888;
00160     int port2 = 33999;
00161
00162     create_test_users("multi1.txt");
00163     string cmd1 = "/server -d multi1.txt -l multi1.log -p " +
00164         to_string(port1) + " 2>&1 &";
00165     system(cmd1.c_str());
00166
00167     create_test_users("multi2.txt");
00168     string cmd2 = "/server -d multi2.txt -l multi2.log -p " +
00169         to_string(port2) + " 2>&1 &";
00170     system(cmd2.c_str());
00171
00172     this_thread::sleep_for(seconds(3));
00173

```

```

00174     bool running1 = check_server_running(port1);
00175     bool running2 = check_server_running(port2);
00176
00177     if (running1 && running2) {
00178         cout << "    Два сервера запущены одновременно" << endl;
00179
00180         stop_server(port1);
00181         stop_server(port2);
00182
00183         CHECK(!check_server_running(port1));
00184         CHECK(!check_server_running(port2));
00185     } else {
00186         cout << "    Не все серверы запустились" << endl;
00187     }
00188
00189     system("rm -f multi1.txt multi2.txt multi1.log multi2.log 2>/dev/null");
00190 }
00191 }
00192
00193
00194 int main() {
00195     cout << "===== " << endl;
00196     cout << "ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ СЕРВЕРА VCALC" << endl;
00197     cout << "===== " << endl;
00198
00199     if (system("test -f ./server 2>/dev/null") != 0) {
00200         cerr << "ОШИБКА: Сначала соберите сервер (make all)" << endl;
00201         return 1;
00202     }
00203
00204     system("pkill -f './server' 2>/dev/null");
00205     this_thread::sleep_for(seconds(2));
00206
00207     return UnitTest::RunAllTests();
00208 }

```

2.13 Файл tests/test_protocol.cpp

Тесты протокола обмена с использованием UnitTest++.

```

#include <UnitTest++/UnitTest++.h>
#include <string>
#include <cstring>
#include <cstdlib>

```

Функции

- SUITE (ProtocolTests)
- int main ()

2.13.1 Подробное описание

Тесты протокола обмена с использованием UnitTest++.

См. определение в файле [test_protocol.cpp](#)

2.13.2 Функции

2.13.2.1 main()

```
int main ( )
```

См. определение в файле [test_protocol.cpp](#) строка 114

2.13.2.2 SUITE()

```
SUITE (
    ProtocolTests )
```

См. определение в файле `test_protocol.cpp` строка 11

2.14 test_protocol.cpp

См. документацию.

```
00001
00006 #include <UnitTest++/UnitTest++.h>
00007 #include <string>
00008 #include <cstring>
00009 #include <cstdlib>
00010
00011 SUITE(ProtocolTests) {
00012     // Тест 1: Формат сообщения аутентификации
00013     TEST(AuthMessageFormat) {
00014         std::string login = "user";
00015         std::string salt = "A1B2C3D4E5F67890";
00016         std::string hash(64, 'A');
00017         std::string auth_msg = login + salt + hash;
00018
00019         CHECK_EQUAL(4, login.length());
00020         CHECK_EQUAL(16, salt.length());
00021         CHECK_EQUAL(64, hash.length());
00022         CHECK_EQUAL(84, auth_msg.length());
00023     }
00024
00025     // Тест 2: Коды ответов сервера
00026     TEST(ServerResponseCodes) {
00027         std::string ok = "OK";
00028         std::string err = "ERR";
00029
00030         CHECK_EQUAL(2, ok.length());
00031         CHECK_EQUAL(3, err.length());
00032     }
00033
00034     // Тест 3: Размеры данных
00035     TEST(DataSizes) {
00036         CHECK_EQUAL(4, sizeof(uint32_t));
00037         CHECK_EQUAL(4, sizeof(float));
00038     }
00039
00040     // Тест 4: Формат соли
00041     TEST(SaltFormat) {
00042         std::string salt = "0123456789ABCDEF";
00043
00044         CHECK_EQUAL(16, salt.length());
00045
00046         for (char c : salt) {
00047             CHECK((c >= '0' && c <= '9') || (c >= 'A' && c <= 'F') || (c >= 'a' && c <= 'f'));
00048         }
00049     }
00050
00051     // Тест 5: Формат хэша
00052     TEST(HashFormat) {
00053         std::string hash = "0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF";
00054
00055         CHECK_EQUAL(64, hash.length());
00056
00057         for (char c : hash) {
00058             CHECK((c >= '0' && c <= '9') || (c >= 'A' && c <= 'F') || (c >= 'a' && c <= 'f'));
00059         }
00060     }
00061
00062     // Тест 6: Хранение пароля в файле
00063     TEST(UserFileFormat) {
00064         std::string line = "user:P@ssW0rd";
00065         size_t colon_pos = line.find(':');
00066         std::string stored_login = line.substr(0, colon_pos);
00067         std::string stored_password = line.substr(colon_pos + 1);
00068
00069         CHECK_EQUAL("user", stored_login);
00070         CHECK_EQUAL("P@ssW0rd", stored_password);
00071     }
00072 }
```

```

00073 // Тест 7: Обнаружение короткого сообщения
00074 TEST(ShortAuthMessage) {
00075     std::string wrong_msg = "short";
00076     bool auth_error = false;
00077
00078     if (wrong_msg.length() < 84) {
00079         auth_error = true;
00080     }
00081
00082     CHECK(auth_error);
00083 }
00084
00085 // Тест 8: Нулевое количество векторов
00086 TEST(ZeroVectors) {
00087     uint32_t zero_vectors = 0;
00088     CHECK_EQ(4, sizeof(zero_vectors));
00089 }
00090
00091 // Тест 9: Последовательность данных протокола
00092 TEST(ProtocolSequence) {
00093     uint32_t num_vectors = 2;
00094     uint32_t vector_size = 3;
00095     float vector_data[] = {1.0f, 2.0f, 3.0f};
00096     float result = 14.0f;
00097
00098     CHECK_EQ(4, sizeof(num_vectors));
00099     CHECK_EQ(4, sizeof(vector_size));
00100     CHECK_EQ(12, sizeof(vector_data));
00101     CHECK_EQ(4, sizeof(result));
00102 }
00103
00104 // Тест 10: Максимальные размеры данных
00105 TEST(MaxDataSizes) {
00106     uint32_t max_size = 1000;
00107     float sample_data[1000];
00108
00109     CHECK_EQ(4, sizeof(max_size));
00110     CHECK_EQ(4000, sizeof(sample_data));
00111 }
00112 }
00113
00114 int main() {
00115     return UnitTest::RunAllTests();
00116 }

```

2.15 Файл tests/test_sha256.cpp

Тесты SHA-256 с использованием UnitTest++.

```

#include <UnitTest++/UnitTest++.h>
#include <cstring>
#include "../sha256.hpp"

```

Функции

- SUITE (SHA256Tests)
- int main ()

2.15.1 Подробное описание

Тесты SHA-256 с использованием UnitTest++.

См. определение в файле `test_sha256.cpp`

2.15.2 Функции

2.15.2.1 main()

```
int main ( )
```

См. определение в файле `test_sha256.cpp` строка 136

2.15.2.2 SUITE()

```
SUITE (
    SHA256Tests )
```

См. определение в файле `test_sha256.cpp` строка 10

2.16 test_sha256.cpp

См. документацию.

```
00001
00006 #include <UnitTest++/UnitTest++.h>
00007 #include <cstring>
00008 #include "../sha256.hpp"
00009
00010 SUITE(SHA256Tests) {
00011     // Тест 1: Пустая строка
00012     TEST(EmptyString) {
00013         const char* empty = "";
00014         uint8_t hash[32];
00015         sha256((uint8_t*)empty, 0, hash);
00016
00017         const char* expected = "E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855";
00018         char hex[65];
00019         for (int i = 0; i < 32; i++) {
00020             sprintf(hex + i*2, "%02X", hash[i]);
00021         }
00022
00023         CHECK_EQUAL(std::string(expected), std::string(hex));
00024     }
00025
00026     // Тест 2: Строка "abc"
00027     TEST(ABCString) {
00028         const char* abc = "abc";
00029         uint8_t hash[32];
00030         sha256((uint8_t*)abc, 3, hash);
00031
00032         const char* expected = "BA7816BF8F01CFEA414140DE5DAE2223B00361A396177A9CB410FF61F20015AD";
00033         char hex[65];
00034         for (int i = 0; i < 32; i++) {
00035             sprintf(hex + i*2, "%02X", hash[i]);
00036         }
00037
00038         CHECK_EQUAL(std::string(expected), std::string(hex));
00039     }
00040
00041     // Тест 3: Длинная строка
00042     TEST(LongString) {
00043         const char* long_text =
00044             "abcdefghijklmnopqrstuvwxyz0123456789abcdefghijklmnopqrstuvwxyz0123456789abcdefghijklmnopqrstuvwxyz0123456789";
00045         uint8_t hash[32];
00046         sha256((uint8_t*)long_text, strlen(long_text), hash);
00047
00048         const char* expected = "CF5B16A778AF8380036CE59E7B0492370B249B11E8F07A51AFAC45037AFEE9D1";
00049         char hex[65];
00050         for (int i = 0; i < 32; i++) {
00051             sprintf(hex + i*2, "%02X", hash[i]);
00052         }
00053
00054         CHECK_EQUAL(std::string(expected), std::string(hex));
00055     }
00056     // Тест 4: Консистентность
```

```

00057 TEST(Consistency) {
00058     const char* data = "test data";
00059     uint8_t hash1[32], hash2[32];
00060
00061     sha256((uint8_t*)data, strlen(data), hash1);
00062     sha256((uint8_t*)data, strlen(data), hash2);
00063
00064     CHECK_ARRAY_EQUAL(hash1, hash2, 32);
00065 }
00066
00067 // Тест 5: Разные длины данных
00068 TEST(DifferentLengths) {
00069     bool lengths_ok = true;
00070     const char* test_strings[] = {"a", "ab", "abc", "abcd", "abcde", "abcdef"};
00071
00072     for (const char* str : test_strings) {
00073         uint8_t h[32];
00074         sha256((uint8_t*)str, strlen(str), h);
00075
00076         bool all_zero = true;
00077         for (int i = 0; i < 32; i++) {
00078             if (h[i] != 0) {
00079                 all_zero = false;
00080                 break;
00081             }
00082         }
00083
00084         if (all_zero) {
00085             lengths_ok = false;
00086             break;
00087         }
00088     }
00089
00090     CHECK(lengths_ok);
00091 }
00092
00093 // Тест 6: Разные данные дают разные хэши
00094 TEST(DifferentDataDifferentHash) {
00095     const char* str1 = "hello";
00096     const char* str2 = "world";
00097     uint8_t hash1[32], hash2[32];
00098
00099     sha256((uint8_t*)str1, strlen(str1), hash1);
00100     sha256((uint8_t*)str2, strlen(str2), hash2);
00101
00102     CHECK(memcmp(hash1, hash2, 32) != 0);
00103 }
00104
00105 // Тест 7: Один символ разницы меняет хэш
00106 TEST(OneCharacterDifference) {
00107     const char* str1 = "test1";
00108     const char* str2 = "test2";
00109     uint8_t hash1[32], hash2[32];
00110
00111     sha256((uint8_t*)str1, strlen(str1), hash1);
00112     sha256((uint8_t*)str2, strlen(str2), hash2);
00113
00114     CHECK(memcmp(hash1, hash2, 32) != 0);
00115 }
00116
00117 // Тест 8: Хэш не нулевой для непустых данных
00118 TEST(HashNotEmpty) {
00119     const char* non_empty = "x";
00120     uint8_t hash[32];
00121
00122     sha256((uint8_t*)non_empty, strlen(non_empty), hash);
00123
00124     bool all_zero = true;
00125     for (int i = 0; i < 32; i++) {
00126         if (hash[i] != 0) {
00127             all_zero = false;
00128             break;
00129         }
00130     }
00131
00132     CHECK(!all_zero);
00133 }
00134 }
00135
00136 int main() {
00137     return UnitTest::RunAllTests();
00138 }

```

2.17 Файл tests/test_vectors.cpp

Тесты вычисления векторов с использованием UnitTest++.

```
#include <UnitTest++/UnitTest++.h>
#include <vector>
#include <cstring>
#include <cmath>
#include <cstdlib>
```

Функции

- float [calculateSumOfSquares](#) (const std::vector< float > &vec)
- uint32_t [floatToLittleEndian](#) (float f)
- float [littleEndianToFloat](#) (uint32_t val)
- SUITE (VectorTests)
- int [main](#) ()

2.17.1 Подробное описание

Тесты вычисления векторов с использованием UnitTest++.

См. определение в файле [test_vectors.cpp](#)

2.17.2 Функции

2.17.2.1 calculateSumOfSquares()

```
float calculateSumOfSquares (
    const std::vector< float > & vec )
```

См. определение в файле [test_vectors.cpp](#) строка 12

2.17.2.2 floatToLittleEndian()

```
uint32_t floatToLittleEndian (
    float f )
```

См. определение в файле [test_vectors.cpp](#) строка 20

2.17.2.3 littleEndianToFloat()

```
float littleEndianToFloat (
    uint32_t val )
```

См. определение в файле [test_vectors.cpp](#) строка 32

2.17.2.4 main()

```
int main ( )
```

См. определение в файле `test_vectors.cpp` строка 139

2.17.2.5 SUITE()

```
SUITE (
    VectorTests )
```

См. определение в файле `test_vectors.cpp` строка 44

2.18 test_vectors.cpp

См. документацию.

```
00001
00006 #include <UnitTest++/UnitTest++.h>
00007 #include <vector>
00008 #include <cstring>
00009 #include <cmath>
00010 #include <stdint> // Добавил этот include
00011
00012 float calculateSumOfSquares(const std::vector<float>& vec) {
00013     float sum = 0.0f;
00014     for (float val : vec) {
00015         sum += val * val;
00016     }
00017     return sum;
00018 }
00019
00020 uint32_t floatToLittleEndian(float f) {
00021     uint32_t bits;
00022     memcpy(&bits, &f, sizeof(float));
00023     uint8_t* bytes = (uint8_t*)&bits;
00024     uint32_t result = 0;
00025     result |= (bytes[0] << 0);
00026     result |= (bytes[1] << 8);
00027     result |= (bytes[2] << 16);
00028     result |= (bytes[3] << 24);
00029     return result;
00030 }
00031
00032 float littleEndianToFloat(uint32_t val) {
00033     uint8_t* bytes = (uint8_t*)&val;
00034     uint32_t bits = 0;
00035     bits |= (bytes[0] << 0);
00036     bits |= (bytes[1] << 8);
00037     bits |= (bytes[2] << 16);
00038     bits |= (bytes[3] << 24);
00039     float f;
00040     memcpy(&f, &bits, sizeof(float));
00041     return f;
00042 }
00043
00044 SUITE(VectorTests) {
00045     // Тест 1: Простой вектор
00046     TEST(SimpleVector) {
00047         std::vector<float> vec = {1.0f, 2.0f, 3.0f, 4.0f};
00048         float result = calculateSumOfSquares(vec);
00049         CHECK_CLOSE(30.0f, result, 0.0001f);
00050     }
00051
00052     // Тест 2: Отрицательные значения
00053     TEST(NegativeValues) {
00054         std::vector<float> vec = {-1.0f, -2.0f, 3.0f, 4.0f};
00055         float result = calculateSumOfSquares(vec);
00056         CHECK_CLOSE(30.0f, result, 0.0001f);
00057     }
00058
00059     // Тест 3: Пустой вектор
00060     TEST(EmptyVector) {
00061         std::vector<float> vec = {};
```

```

00062     float result = calculateSumOfSquares(vec);
00063     CHECK_EQUAL(0.0f, result);
00064 }
00065
00066 // Тест 4: Один элемент
00067 TEST(SingleElement) {
00068     std::vector<float> vec = {5.0f};
00069     float result = calculateSumOfSquares(vec);
00070     CHECK_CLOSE(25.0f, result, 0.0001f);
00071 }
00072
00073 // Тест 5: Большие числа
00074 TEST(LargeNumbers) {
00075     std::vector<float> vec = {100.0f, 200.0f, 300.0f};
00076     float result = calculateSumOfSquares(vec);
00077     float expected = 10000.0f + 40000.0f + 90000.0f;
00078     CHECK_CLOSE(expected, result, 0.1f);
00079 }
00080
00081 // Тест 6: Дробные числа
00082 TEST(FractionalNumbers) {
00083     std::vector<float> vec = {0.5f, 1.5f, 2.5f};
00084     float result = calculateSumOfSquares(vec);
00085     float expected = 0.25f + 2.25f + 6.25f;
00086     CHECK_CLOSE(expected, result, 0.0001f);
00087 }
00088
00089 // Тест 7: Конвертация endian
00090 TEST(EndianConversion) {
00091     float original = 123.456f;
00092     uint32_t le = floatToLittleEndian(original);
00093     float converted = littleEndianToFloat(le);
00094     CHECK_CLOSE(original, converted, 0.0001f);
00095 }
00096
00097 // Тест 8: Все нули
00098 TEST(AllZeros) {
00099     std::vector<float> vec(5, 0.0f);
00100     float result = calculateSumOfSquares(vec);
00101     CHECK_EQUAL(0.0f, result);
00102 }
00103
00104 // Тест 9: Очень маленькие числа
00105 TEST(VerySmallNumbers) {
00106     std::vector<float> vec = {0.001f, 0.002f, 0.003f};
00107     float result = calculateSumOfSquares(vec);
00108     float expected = 0.000001f + 0.000004f + 0.000009f;
00109     CHECK_CLOSE(expected, result, 1e-10f);
00110 }
00111
00112 // Тест 10: Смешанные значения
00113 TEST(MixedPositiveNegative) {
00114     std::vector<float> vec = {-2.0f, 0.0f, 2.0f, -4.0f, 4.0f};
00115     float result = calculateSumOfSquares(vec);
00116     float expected = 4.0f + 0.0f + 4.0f + 16.0f + 16.0f;
00117     CHECK_CLOSE(expected, result, 0.0001f);
00118 }
00119
00120 // Тест 11: Много элементов
00121 TEST(ManyElements) {
00122     std::vector<float> vec;
00123     for (int i = 0; i < 100; i++) {
00124         vec.push_back(1.0f);
00125     }
00126     float result = calculateSumOfSquares(vec);
00127     CHECK_CLOSE(100.0f, result, 0.0001f);
00128 }
00129
00130 // Тест 12: Конвертация отрицательного числа
00131 TEST(NegativeEndianConversion) {
00132     float negative = -123.456f;
00133     uint32_t le = floatToLittleEndian(negative);
00134     float converted = littleEndianToFloat(le);
00135     CHECK_CLOSE(negative, converted, 0.0001f);
00136 }
00137 }
00138
00139 int main() {
00140     return UnitTest::RunAllTests();
00141 }

```