Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЕТ

к лабораторной работе №2

на тему

**ИДЕНТИФИКАЦИЯ И АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЕЙ. ПРОТОКОЛ KERBEROS.**

Студент В. М. Вергасов

Преподаватель Е. А. Лещенко

Минск 2024

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc159003347)

[1 Результат выполнения 4](#_Toc159003348)

[Заключение 6](#_Toc159003349)

[Приложение А (обязательное) Листинг кода 7](#_Toc159003350)

**ВВЕДЕНИЕ**

Лабораторная работа ставит перед собой задачу разработки программных средств для реализации протокола Kerberos 5 и алгоритма шифрования DES.

Основная цель работы – изучение принципов работы указанных алгоритмов и их реализация на языке программирования *C++*. В ходе работы будет осуществлен анализ методов шифрования, разработка алгоритмов шифрования и дешифрования, а также создание программного продукта, позволяющего осуществлять шифрование и общение по протоколу Kerberos 5.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате работы было создано три приложения для реализации протокола Kerberos 5. Вывод программы-клиента указан на рисунке 1, вывод программы-сервера авторизации приведён на рисунке 2, а вывод конечного сервера указан на рисунке 3.

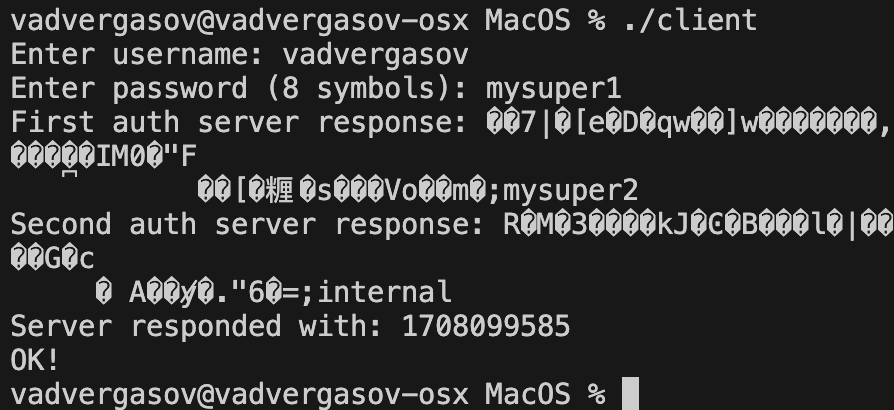


Рисунок 1 – Вывод клиента

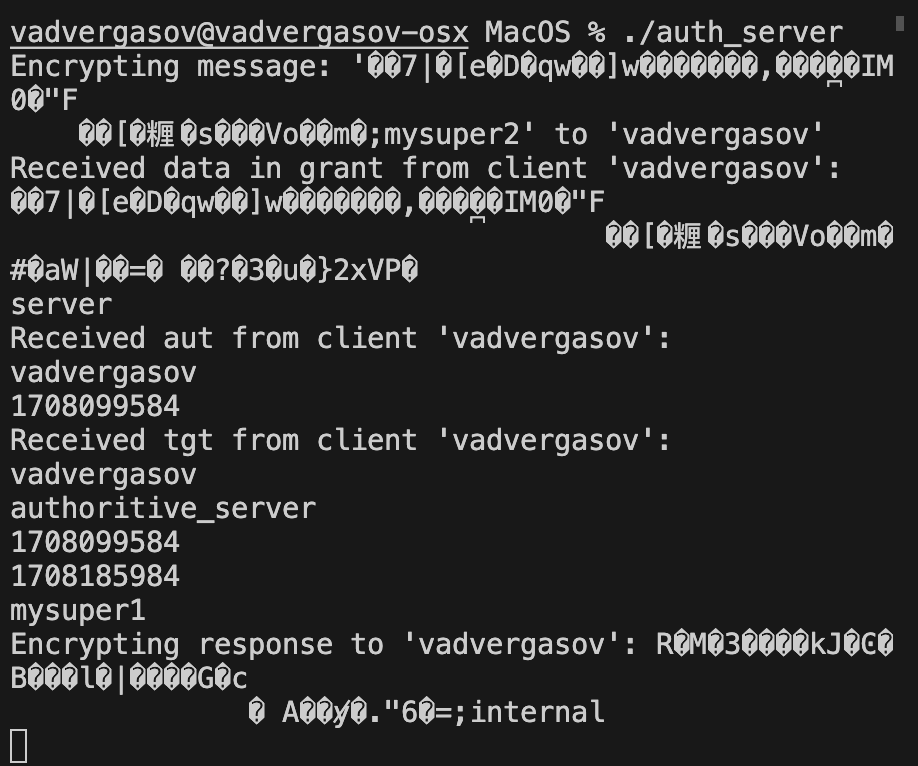


Рисунок 2 – Вывод сервера авторизации

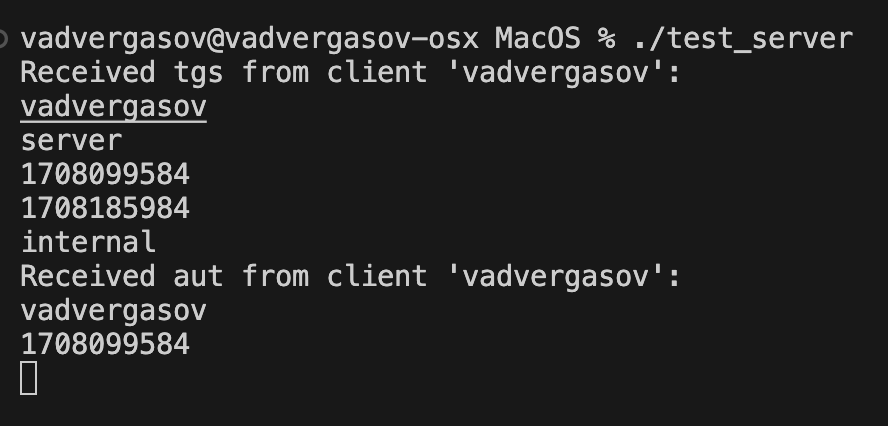


Рисунок 3 – Вывод сервера

Так же можно посмотреть как выглядит общение программ для стороннего человека с помощью программы Wireshark, как показано на рисунке 4.

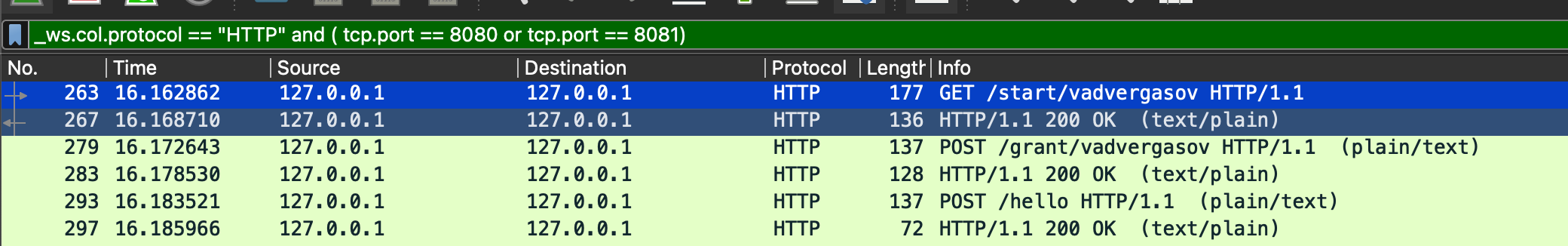


Рисунок 4 – Пакеты, передаваемые по сети

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной лабораторной работы были разработаны программные средства для реализации протокола Kerberos и алгоритма шифрования DES на языке программирования *C++*.

При выполнении работы были использованы теоретические сведения о принципах работы протокола Kerberos 5 и алгоритма DES. Так же использовалась программа Wireshark для анализа трафика между приложениями.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *des.h*

#pragma once

#include <bitset>

#include <string>

using namespace std;

class DES {

public:

bitset<32> F(bitset<32> R, bitset<48> k); // функция F

bitset<28> leftShift(const bitset<28>& bits, int rotateCount); // функция сдвига влево

void generateKeys(); // Генерировать 16 48-битных подключей

bitset<64> change(bitset<64> temp);

bitset<64> DES\_encrypt(bitset<64>& plain); // шифрование DES

bitset<64> DES\_decrypt(bitset<64>& cipher); // Расшифровка DES

void get\_s(string str) {

s = str;

} // Получить открытый текст

void setKey(string key\_t) {

k = key\_t;

} // Получаем ключ

bitset<64> charToBit(const char s[8]);

string encryptAnyString(string plaintext);

string decryptAnyString(string cipher);

DES() = default;

~DES() = default;

private:

string s; // обычный текст

string k; // Ключ

bitset<64> key; // 64-битный ключ

bitset<48> subkey[16]; // ключ для каждого колеса

/\* Таблица замены инициализации IP \*/

int IP[64] =

{

58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6,

64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7};

/\* Таблица перестановок обратной инициализации \*/

int IP\_1[64] =

{

40, 8, 48, 16, 56, 24, 64, 32,

39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30,

37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28,

35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26,

33, 1, 41, 9, 49, 17, 57, 25};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Таблица, используемая функцией пароля F \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Замена расширения электронного блока\*/

/\*Расширить 32 бит до 48 бит\*/

int E[48] =

{

32, 1, 2, 3, 4, 5,

4, 5, 6, 7, 8, 9,

8, 9, 10, 11, 12, 13,

12, 13, 14, 15, 16, 17,

16, 17, 18, 19, 20, 21,

20, 21, 22, 23, 24, 25,

24, 25, 26, 27, 28, 29,

28, 29, 30, 31, 32, 1};

/\* S - бокс вместо сжатия \*/

/\* Каждый блок S представляет собой таблицу замены 16x4 \*/

/\* 48 бит, сжатый до 32 бит \*/

int S\_BOX[8][4][16] = {

{{14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},

{0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8},

{4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},

{15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}},

{{15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},

{3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},

{0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},

{13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}},

{{10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},

{13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},

{13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},

{1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12}},

{{7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},

{13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},

{10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},

{3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}},

{{2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},

{14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},

{4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},

{11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3}},

{{12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},

{10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},

{9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},

{4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13}},

{{4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},

{13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},

{1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},

{6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12}},

{{13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},

{1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2},

{7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},

{2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}}};

/\* Замена P - бокса \*/

/\* Аналогично инициализации IP \*/

int P[32] =

{

16, 7, 20, 21,

29, 12, 28, 17,

1, 15, 23, 26,

5, 18, 31, 10,

2, 8, 24, 14,

32, 27, 3, 9,

19, 13, 30, 6,

22, 11, 4, 25};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Таблица, используемая для генерации ключа \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Выбор замены PC - 1 \*/

/\* 64 - битный ключ становится 56 - битным \*/ int PC\_1[56] =

{

57, 49, 41, 33, 25, 17, 9,

1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27,

19, 11, 3, 60, 52, 44, 36,

63, 55, 47, 39, 31, 23, 15,

7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29,

21, 13, 5, 28, 20, 12, 4};

/\* Выбор замены PC - 2 \* / / \*56 - битный ключ, сжатый в 48 - битный подключ \*/

int PC\_2[48] =

{

14, 17, 11, 24, 1, 5,

3, 28, 15, 6, 21, 10,

23, 19, 12, 4, 26, 8,

16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55,

30, 40, 51, 45, 33, 48,

44, 49, 39, 56, 34, 53,

46, 42, 50, 36, 29, 32};

/\* Циклический сдвиг влево \*/

int shiftBits[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

};

Листинг 2 – Файл *des.cpp*

#include "des.h"

#include <iostream>

#include <sstream>

using namespace std;

bitset<32> DES::F(bitset<32> R, bitset<48> k) {

// Расширение электронного блока

bitset<48> expandR;

for (int i = 0; i < 48; i++)

expandR[47 - i] = R[32 - E[i]]; //expandR[i] = R[E[i] - 1];

// XOR

expandR = expandR ^ k;

// вместо этого S-поле

bitset<32> output;

int x = 0;

for (int i = 0; i < 48; i = i + 6) {

int row = expandR[i] \* 2 + expandR[i + 5];

int col = expandR[i + 1] \* 8 + expandR[i + 2] \* 4 + expandR[i + 3] \* 2 + expandR[i + 4];

int num = S\_BOX[i / 6][row][col];

bitset<4> temp(num);

output[x + 3] = temp[0];

output[x + 2] = temp[1];

output[x + 1] = temp[2];

output[x] = temp[3];

x += 4;

}

// Замена P-бокса

bitset<32> tmp = output;

for (int i = 0; i < 32; i++)

output[i] = tmp[P[i] - 1];

return output;

}

// Функция сдвига влево

bitset<28> DES::leftShift(const bitset<28>& bits, int rotateCount) {

return (bits << rotateCount) | (bits >> (bits.size() - rotateCount));

}

void DES::generateKeys() {

bitset<56> real\_key;

bitset<28> left;

bitset<28> right;

bitset<48> compressKey;

// Во-первых, после выбора и замены ПК-1 удаляем 8-битный бит четности исходного ключа

// И переставляем

for (int i = 0; i < 56; i++)

real\_key[i] = key[PC\_1[i] - 1];

for (int round = 0; round < 16; round++) {

for (int i = 0; i < 28; i++)

left[i] = real\_key[i];

for (int i = 28; i < 56; i++)

right[i - 28] = real\_key[i];

// Сдвиг влево

left = leftShift(left, shiftBits[round]);

right = leftShift(right, shiftBits[round]);

// Подключаем, заменяем и выбираем ПК-2 для перестановки и сжатия

for (int i = 0; i < 28; i++)

real\_key[i] = left[i];

for (int i = 28; i < 56; i++)

real\_key[i] = right[i - 28];

for (int i = 0; i < 48; i++) {

int m = PC\_2[i];

compressKey[i] = real\_key[m - 1];

}

subkey[round] = compressKey;

}

}

// Функция инструмента: выполнить двоичное обратное преобразование

bitset<64> DES::change(bitset<64> temp) {

bitset<64> bits;

bitset<8> n;

for (int i = 0; i < 64; i = i + 8) {

for (int j = 0; j < 8; j++) {

bits[i + j] = temp[i + 7 - j];

}

}

return bits;

}

bitset<64> DES::DES\_encrypt(bitset<64>& plain) {

bitset<64> cipher;

bitset<64> currentBits;

bitset<32> left;

bitset<32> right;

bitset<32> newLeft;

// Первоначальная замена IP

for (int i = 0; i < 64; i++)

currentBits[i] = plain[IP[i] - 1]; //

for (int i = 0; i < 32; i++)

left[i] = currentBits[i];

for (int i = 32; i < 64; i++)

right[i - 32] = currentBits[i];

// Вводим 16 раундов изменения

for (int round = 0; round < 16; round++) {

newLeft = right;

right = left ^ F(right, subkey[round]);

left = newLeft;

}

// Слияние

for (int i = 0; i < 32; i++)

cipher[i] = right[i];

for (int i = 32; i < 64; i++)

cipher[i] = left[i - 32];

// Обратная инициализация замены

currentBits = cipher;

for (int i = 0; i < 64; i++)

cipher[i] = currentBits[IP\_1[i] - 1];

return cipher;

}

bitset<64> DES::DES\_decrypt(bitset<64>& cipher) {

bitset<64> plain;

bitset<64> currentBits;

bitset<32> left;

bitset<32> right;

bitset<32> newLeft;

// Заменить IP

for (int i = 0; i < 64; i++)

currentBits[i] = cipher[IP[i] - 1];

for (int i = 0; i < 32; i++)

left[i] = currentBits[i];

for (int i = 32; i < 64; i++)

right[i - 32] = currentBits[i];

// Вводим 16 итераций (подключи применяются в обратном порядке)

for (int round = 0; round < 16; round++) {

newLeft = right;

right = left ^ F(right, subkey[15 - round]);

left = newLeft;

}

// Слияние

for (int i = 0; i < 32; i++)

plain[i] = right[i];

for (int i = 32; i < 64; i++)

plain[i] = left[i - 32];

// Обратная инициализация замены

currentBits = plain;

for (int i = 0; i < 64; i++)

plain[i] = currentBits[IP\_1[i] - 1];

return plain;

}

bitset<64> DES::charToBit(const char s[8]) {

return bitset<64>(\*reinterpret\_cast<const unsigned long long\*>(s));

}

string DES::encryptAnyString(string plaintext) {

key = charToBit(k.c\_str());

stringstream result;

plaintext += string(8 - (plaintext.size() % 8), '\0');

for (size\_t current = 0; current < plaintext.size() / 8; current++) {

string tmp = plaintext.substr(8 \* current, 8);

bitset<64> plain = charToBit(tmp.c\_str());

generateKeys();

bitset<64> cipher = DES\_encrypt(plain);

result << string(reinterpret\_cast<char \*>(std::addressof(cipher)), 8);

}

return result.str();

}

string DES::decryptAnyString(string cipher) {

key = charToBit(k.c\_str());

stringstream result;

for (size\_t current = 0; current < cipher.size() / 8; current++) {

bitset<64> cipher\_set = charToBit(cipher.substr(8 \* current, 8).c\_str());

generateKeys();

bitset<64> plain = DES\_decrypt(cipher\_set);

plain = change(plain);

string res = plain.to\_string();

for (int i = 0; i < 8; i++) {

string tmp = plain.to\_string().substr(0 + 8 \* i, 8);

reverse(tmp.begin(), tmp.end());

res.replace(res.begin() + i \* 8, res.begin() + 8 \* (i + 1), tmp);

}

plain = bitset<64>(res);

result << string(reinterpret\_cast<char \*>(&plain), 8);

}

return result.str();

}

Листинг 3 – Файл *common.h*

#include <string>

#include <vector>

namespace Utils {

std::vector<std::string> Split(const std::string& s, char delim);

};

Листинг 4 – Файл *common.cpp*

#include "common.h"

#include <sstream>

std::vector<std::string> Utils::Split(const std::string &s, char delim) {

std::vector<std::string> result;

std::stringstream ss(s);

std::string item;

while(std::getline(ss, item, delim)) {

result.push\_back(item);

}

return result;

}

Листинг 5 – Файл *client.cpp*

#include <iostream>

#include <des.h>

#include <httplib.h>

#include <common.h>

#include <chrono>

using namespace std;

using namespace httplib;

constexpr const char\* server = "server";

int main() {

Client cli("http://127.0.0.1:8080");

string username, password;

cout << "Enter username: ";

cin >> username;

while (password.size() != 8) {

cout << "Enter password (8 symbols): ";

cin >> password;

}

auto response = cli.Get("/start/" + username)->body;

DES c;

c.setKey(password);

string result = c.decryptAnyString(response);

cout << result << endl;

string tgs\_key = Utils::Split(result, ';').back().substr(0, 8);

string aut = username + ";" + to\_string(chrono::system\_clock::to\_time\_t(chrono::system\_clock::now()));

c.setKey(tgs\_key);

string aut\_encrypted = c.encryptAnyString(aut);

string request\_to\_tgs = Utils::Split(result, ';').front() + ";" + aut\_encrypted + ";" + server;

result = cli.Post("/grant/" + username, request\_to\_tgs, "plain/text")->body;

result = c.decryptAnyString(result);

cout << result << endl;

Client hello("http://localhost:8081");

auto tgs = Utils::Split(result, ';').front();

auto current\_time = chrono::system\_clock::to\_time\_t(chrono::system\_clock::now());

aut = username + ";" + to\_string(current\_time);

c.setKey(string(Utils::Split(result, ';').back()));

response = hello.Post("/hello", tgs + ";" + c.encryptAnyString(aut), "plain/text")->body;

cout << "Server responded with: " << c.decryptAnyString(response) << endl;

if (std::atoll(c.decryptAnyString(response).c\_str()) != current\_time + 1) {

cout << "Server verification failed!" << endl;

return 1;

}

cout << "OK!" << endl;

return 0;

}

Листинг 6 – Файл *auth\_server.cpp*

#include <httplib.h>

#include <map>

#include <iostream>

#include <chrono>

#include <des.h>

#include <common.h>

using namespace httplib;

using namespace std;

using namespace chrono\_literals;

struct ClientInfo {

string key;

string tgs\_key;

string c\_ss;

vector<string> allowed\_servers;

};

struct ServerInfo {

string tgs\_key;

};

const string MASTER\_PASSWORD = "password";

const string TGS = "authoritive\_server";

map<string, ClientInfo> clients = {

{"vadvergasov", {

.key = "mysuper1",

.tgs\_key = "mysuper2",

.c\_ss = "internal",

.allowed\_servers = {

"server",

},

}},

};

map<string, ServerInfo> servers = {

{"server", {

.tgs\_key = "serverkc",

}}};

void initMessage(const Request& request, Response& response) {

string id = request.path\_params.at("id");

if (clients.find(id) == clients.end()) {

response.status = 403;

response.set\_content("Forbidden", 10, "text/plain");

return;

}

DES c;

auto current\_time = chrono::system\_clock::now();

auto end\_time = current\_time + 24h;

string TGT = id + ";" + TGS + ";" + to\_string(chrono::system\_clock::to\_time\_t(current\_time)) + ";" + to\_string(chrono::system\_clock::to\_time\_t(end\_time)) + ";" + clients[id].key;

c.setKey(MASTER\_PASSWORD);

string TGT\_encrypted = c.encryptAnyString(TGT);

string answer = TGT\_encrypted + ";" + clients[id].tgs\_key;

c.setKey(clients[id].key);

cout << "Encrypting message: '" << answer << "' to '" << id << "'" << endl;

string encrypted = c.encryptAnyString(answer);

// cout << "Sending message: '" << encrypted << "' to '" << id << "'" << endl;

response.set\_content(encrypted.c\_str(), encrypted.size(), "text/plain");

response.status = 200;

}

void grant(const Request& request, Response& response) {

string id = request.path\_params.at("id");

if (clients.find(id) == clients.end()) {

response.status = 403;

response.set\_content("Forbidden", 10, "text/plain");

return;

}

auto data = Utils::Split(request.body, ';');

cout << "Received data in grant from client '" << id << "':" << endl;

for (const auto& cur : data) {

cout << cur << endl;

}

DES c;

c.setKey(clients[id].tgs\_key);

auto aut = Utils::Split(c.decryptAnyString(data[1]), ';');

cout << "Received aut from client '" << id << "':" << endl;

for (const auto& cur : aut) {

cout << cur << endl;

}

string server\_id = data.back().c\_str();

if (

find(clients[id].allowed\_servers.begin(), clients[id].allowed\_servers.end(), server\_id) == clients[id].allowed\_servers.end() ||

aut.front().substr(0, 8) == id) {

response.status = 403;

const char\* resp = "Forbidden (incorrect user)";

response.set\_content(resp, strlen(resp), "text/plain");

return;

}

auto ticket\_creation = chrono::system\_clock::from\_time\_t(std::atoll(aut.back().c\_str()));

c.setKey(MASTER\_PASSWORD);

auto tgt = Utils::Split(c.decryptAnyString(data.front()), ';');

cout << "Received tgt from client '" << id << "':" << endl;

for (const auto& cur : tgt) {

cout << cur << endl;

}

if (

chrono::system\_clock::from\_time\_t(std::atoll(tgt[2].c\_str())) > ticket\_creation &&

chrono::system\_clock::from\_time\_t(std::atoll(tgt[3].c\_str())) < ticket\_creation) {

response.status = 403;

const char\* resp = "Forbidden (ticket expired)";

response.set\_content(resp, strlen(resp), "text/plain");

return;

}

c.setKey(servers[server\_id].tgs\_key);

auto current\_time = chrono::system\_clock::now();

auto end\_time = current\_time + 24h;

string tgs\_server = id + ";" + server\_id + ";" + to\_string(chrono::system\_clock::to\_time\_t(current\_time)) + ";" + to\_string(chrono::system\_clock::to\_time\_t(end\_time)) + ";" + clients[id].c\_ss;

string tgs\_server\_encrypted = c.encryptAnyString(tgs\_server);

string answer = tgs\_server\_encrypted + ";" + clients[id].c\_ss;

cout << "Encrypting response to '" << id << "': " << answer << endl;

c.setKey(clients[id].tgs\_key);

string answer\_encrypted = c.encryptAnyString(answer);

response.body = answer\_encrypted;

response.status = 200;

}

int main() {

Server svc;

svc.Get("/start/:id", initMessage);

svc.Post("/grant/:id", grant);

svc.listen("0.0.0.0", 8080);

return 0;

}

Листинг 7 – Файл *test\_server.cpp*

#include <httplib.h>

#include <des.h>

#include <common.h>

#include <iostream>

#include <chrono>

using namespace httplib;

using namespace std;

constexpr const char\* key = "serverkc";

int main() {

Server svc;

DES c;

c.setKey(key);

svc.Post("/hello", [&](const Request& request, Response& response) {

auto tgs = Utils::Split(c.decryptAnyString(Utils::Split(request.body, ';').front()), ';');

string c\_ss = tgs.back();

string client\_id = tgs.front();

cout << "Received tgs from client '" << client\_id << "': ";

for (const auto& cur : tgs) {

cout << cur << endl;

}

c.setKey(c\_ss);

auto aut = Utils::Split(c.decryptAnyString(Utils::Split(request.body, ';').back()), ';');

cout << "Received aut from client '" << client\_id << "': ";

for (const auto& cur : aut) {

cout << cur << endl;

}

if (client\_id != string(aut.front())) {

response.status = 403;

const char\* resp = "Forbidden (incorrect user)";

response.set\_content(resp, strlen(resp), "text/plain");

return;

}

string answer = "";

response.status = 200;

response.body = c.encryptAnyString(to\_string(std::atoll(aut.back().c\_str()) + 1));

});

svc.listen("0.0.0.0", 8081);

return 0;

}