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.**

Выполнил:

студент гр. 153503

Бобко И.В.

Проверил:

Лещенко Е.А.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc157598705)

[2 Теоретические сведения 4](#_Toc157598706)

[3 Полученные результаты 6](#_Toc157598712)

[Выводы 8](#_Toc157598713)

[Приложение А (обязательное) Листинг кода 9](#_Toc157598714)

[Приложение Б (обязательное) Блок-схема алгоритма 12](#_Toc157598715)

# **1 ЦЕЛЬ РАБОТЫ**

Целью выполнения данной лабораторной работы является создание приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую Алгоритм DES.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ** 

**Протокол Kerberos**

Kerberos является сетевым протоколом аутентификации, который используется для установления идентичности пользователей и сервисов в сети без необходимости передавать секретные данные, такие как пароли, в открытом виде. Он разработан в Массачусетском технологическом институте (MIT) и назван в честь Кербера, собаки-хранителя подземного мира в греческой мифологии, что символизирует его функцию как "хранителя" сетевой безопасности.

Принцип работы Kerberos основан на модели доверенного третьего лица. В этой модели, все участники сети доверяют центральному узлу, называемому Керберос-сервером или Ключевым Распределительным Центром (Key Distribution Center, KDC), который отвечает за аутентификацию пользователей и выдачу токенов доступа, известных как "билеты" (tickets), которые используются для подтверждения идентичности при обращении к сетевым ресурсам.

Процесс аутентификации в Kerberos происходит в несколько этапов:

1. Аутентификация пользователя при помощи KD. Пользователь запрашивает доступ к сервису, отправляя свой идентификатор на KDC. В ответ KDC выдает "билет для получения билета" (Ticket-Granting Ticket, TGT), который шифруется секретным ключом, известным только пользователю и KDC.
2. Получение билета доступа к сервису. Пользователь отправляет TGT обратно на KDC вместе с запросом на доступ к конкретному сервису. KDC проверяет TGT и, в случае успеха, выдает билет доступа к запрашиваемому сервису. Этот билет шифруется ключом, известным только сервису и KDC.
3. Аутентификация пользователя у сервиса. Пользователь предъявляет билет доступа сервису. Сервис проверяет билет с использованием своего секретного ключа и, если проверка проходит успешно, предоставляет доступ пользователю.

Этот процесс обеспечивает безопасность аутентификации в сети, минимизируя количество случаев, когда пользователь должен предъявлять свой секретный ключ или пароль. Благодаря использованию сильного криптографического шифрования и тщательному управлению ключами, Kerberos эффективно снижает риск перехвата и раскрытия учетных данных.

Kerberos широко применяется в различных операционных системах и сетевых приложениях для обеспечения безопасного взаимодействия в компьютерных сетях, включая корпоративные среды и Интернет. Несмотря на свои преимущества, Kerberos требует тщательной настройки и управления, включая синхронизацию времени между всеми узлами сети и безопасное хранение ключевой информации.

**Алгоритм DES**

DES (Data Encryption Standard) является одним из наиболее известных и исторически значимых алгоритмов симметричного шифрования. Разработанный в начале 1970-х годов и официально принятый в качестве федерального стандарта шифрования в США в 1977 году, DES долгое время служил основным средством защиты цифровой информации в различных сферах, от банковских систем до государственных коммуникаций.

DES использует симметричный ключ, то есть тот же ключ применяется как для шифрования, так и для расшифровки данных. Основная идея симметричного шифрования заключается в том, что если у вас есть правильный ключ, вы можете легко преобразовать зашифрованный текст обратно в исходный текст и наоборот.

Алгоритм шифрует данные блоками по 64 бита, используя ключ шифрования длиной 56 бит (плюс 8 бит паритета, которые не используются в процессе шифрования, что делает общую длину ключа 64 бита). DES работает в несколько раундов, каждый из которых включает в себя перестановки и подстановки, основанные на ключе шифрования. Эти операции, называемые "функцией Фейстеля", повторяются 16 раз для каждого блока данных, что делает процесс шифрования достаточно сложным для взлома методом прямого перебора в те времена.

Несмотря на уязвимость к атакам полным перебором, DES по-прежнему используется в некоторых приложениях, часто в усовершенствованной форме, такой как Triple DES (3DES), которая значительно увеличивает безопасность за счет последовательного применения алгоритма три раза к каждому блоку данных с использованием двух или трех различных ключей. Это увеличивает длину ключа и значительно усложняет задачу для потенциального злоумышленника, пытающегося взломать шифр.

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

В ходе выполнения данной лабораторной работы было создано приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую Алгоритм DES. Для наглядной работы протокола, имеется вывод информации о каждом шаге что на стороне серверов, так и на стороне клиента.

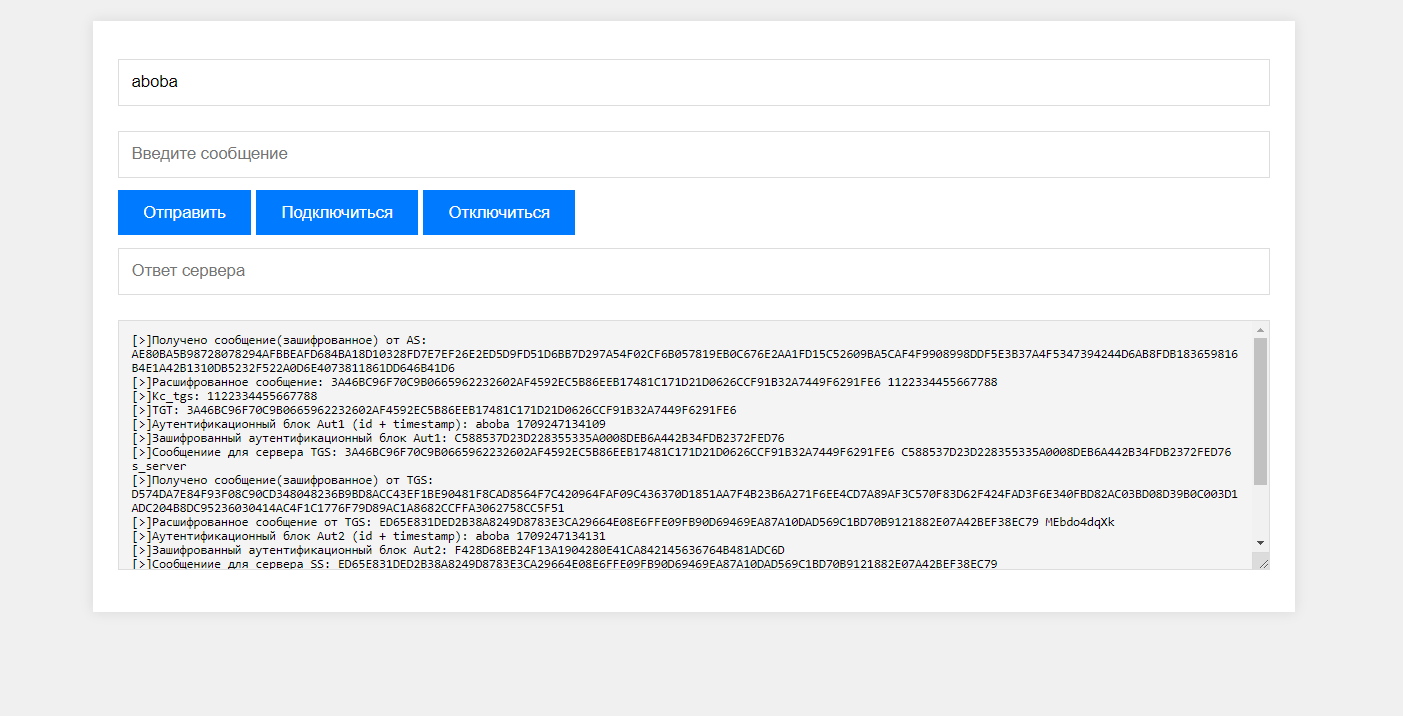


Рисунок 1 – Интерфейс клиента

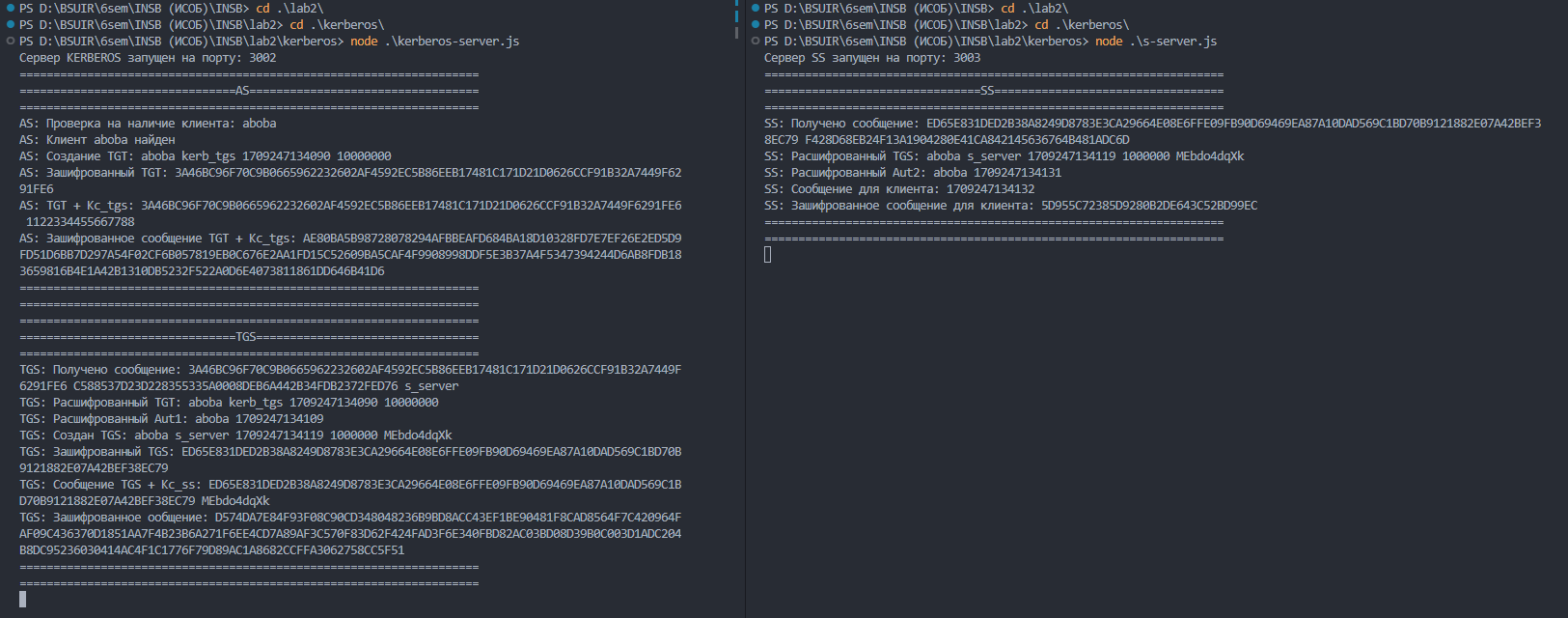


Рисунок 2 – Работа серверов

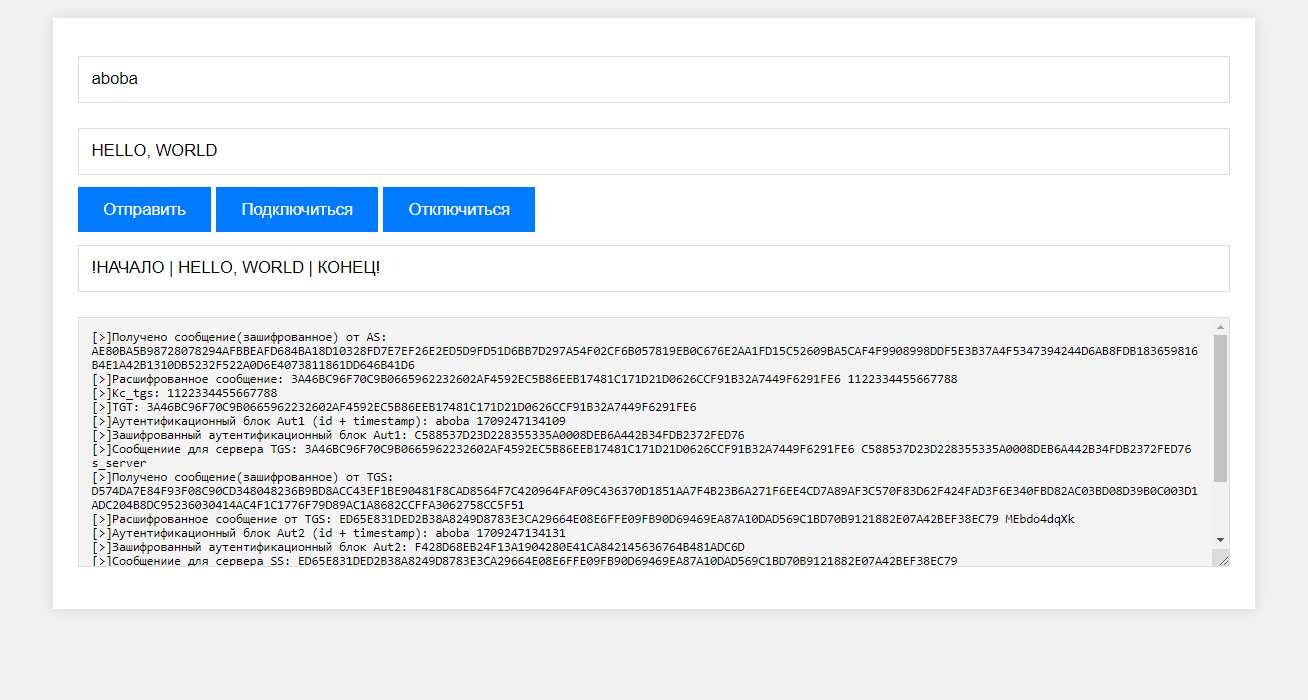


Рисунок 3 – Отправка и получение сообщения на клиенте после подключения

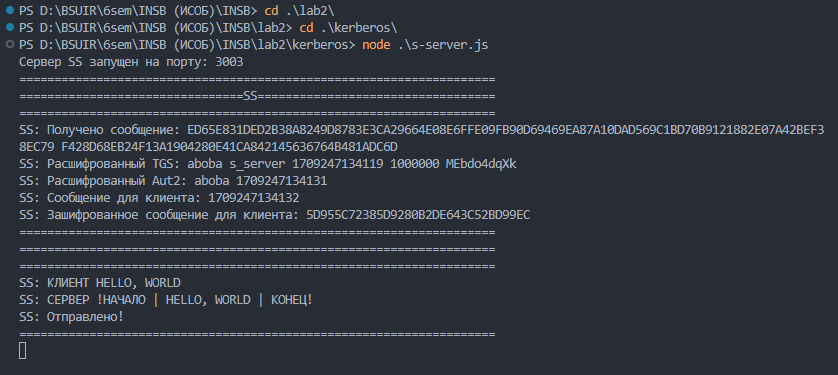


Рисунок 4 – Получение и отправка сообщения на сервере после подключения

# **ВЫВОДЫ**

В результате выполнения данной лабораторной работы было создано приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую Алгоритм DES.

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг **–** Файл *script.js*

import NumeralSystemConverter from "./nsc.js";

import DESCrypto from "./des.js";

const sendBtn = document.getElementById('sendBtn');

const connectBtn = document.getElementById('connectBtn');

const disconnectBtn = document.getElementById('disconnectBtn');

const usernameInput = document.getElementById('username');

const messageInput = document.getElementById('message');

const textArea = document.querySelector('.tech-info');

const responseInput = document.getElementById('response');

const nsc = NumeralSystemConverter;

const des = DESCrypto;

const user = {

id: 'aboba',

key : 'key123client',

ss\_id : 's\_server',

local\_key\_c\_ss: '',

startProtocol: async function (logfunc=console.log) {

try {

// AS ================================

let response = await fetch(`http://localhost:3002/AS/${user.id}`, {

method: 'GET',

});

let data = await response.json();

if (!response.ok){

throw data.msg;

}

let as\_msg = data.msg;

logfunc(`Получено сообщение(зашифрованное) от AS: ${as\_msg}`);

as\_msg = nsc.binaryToText(des.decrypt(

nsc.hexToBinary(as\_msg),

nsc.textToBinary(user.key)

));

logfunc(`Расшифрованное сообщение: ${as\_msg}`);

const [tgt, key\_c\_tgs] = as\_msg.split(' ');

logfunc(`Kc\_tgs: ${key\_c\_tgs}`);

logfunc(`TGT: ${tgt}`);

let timestamp = new Date().getTime();

let aut1 = user.id + " " + timestamp;

logfunc(`Аутентификационный блок Aut1 (id + timestamp): ${aut1}`);

aut1 = nsc.binaryToHex(des.encrypt(

nsc.textToBinary(aut1),

nsc.textToBinary(key\_c\_tgs)

));

logfunc(`Зашифрованный аутентификационный блок Aut1: ${aut1}`);

let msgForTGS = tgt + " " + aut1 + " " + user.ss\_id;

logfunc(`Сообщениие для сервера TGS: ${msgForTGS}`)

// TGS ===============================

response = await fetch(`http://localhost:3002/TGS/${msgForTGS}`, {

method: 'GET',

});

data = await response.json();

if (!response.ok){

throw data.msg;

}

let tgs\_msg = data.msg

logfunc(`Получено сообщение(зашифрованное) от TGS: ${tgs\_msg}`);

tgs\_msg = nsc.binaryToText(des.decrypt(

nsc.hexToBinary(tgs\_msg),

nsc.textToBinary(key\_c\_tgs)

));

logfunc(`Расшифрованное сообщение от TGS: ${tgs\_msg}`);

const [tgs, key\_c\_ss] = tgs\_msg.split(' ');

timestamp = new Date().getTime();

let aut2 = user.id + ' ' + timestamp;

logfunc(`Aутентификационный блок Aut2 (id + timestamp): ${aut2}`);

aut2 = nsc.binaryToHex(des.encrypt(

nsc.textToBinary(aut2),

nsc.textToBinary(key\_c\_ss)

));

logfunc(`Зашифрованный аутентификационный блок Aut2: ${aut2}`);

let msgForSS = tgs + ' ' + aut2;

logfunc(`Сообщениие для сервера SS: ${msgForSS}`);

// SS ==============================

response = await fetch(`http://localhost:3003/SS/${msgForSS}`, {

method: 'GET',

});

data = await response.json();

if (!response.ok){

throw data.msg;

}

let ss\_msg = data.msg

logfunc(`Получено сообщение(зашифрованное) от SS: ${ss\_msg}`);

ss\_msg = nsc.binaryToText(des.decrypt(

nsc.hexToBinary(ss\_msg),

nsc.textToBinary(key\_c\_ss)

));

logfunc(`Расшифрованное сообщение от SS: ${ss\_msg}`);

if ((parseInt(ss\_msg) - 1) != parseInt(timestamp)) {

throw 'Сервер не доказал свою подлинность!'

}

logfunc('Подлючено! Сервер доказал подлинность!');

user.local\_key\_c\_ss = key\_c\_ss;

} catch (error) {

logfunc(`[ERROR]: ${error}`);

}

},

};

async function getTextFromServer(text, logfunc=console.log) {

try{

text = nsc.binaryToHex(des.encrypt(

nsc.textToBinary(text),

nsc.textToBinary(user.local\_key\_c\_ss)

));

let response = await fetch(`http://localhost:3003/gettext/${text}`, {

method: 'GET',

});

let data = await response.json();

if (!response.ok){

throw data.msg;

}

let result = nsc.binaryToText(des.decrypt(

nsc.hexToBinary(data.msg),

nsc.textToBinary(user.local\_key\_c\_ss)

));

result = result.replace(/\u0000/g, '');

return result;

} catch (error) {

if (user.local\_key\_c\_ss === '') {

return `not connected?`;

}

logfunc(error);

}

}

async function connectToServer() {

user.id = usernameInput.value;

await user.startProtocol(arealog);

}

async function sendMessage(logfunc=console.log) {

let text = messageInput.value;

try {

const result = await getTextFromServer(text, arealog);

responseInput.value = result;

} catch (error) {

logfunc(error);

}

}

function arealog(text) {

textArea.value += '[>]' + text + '\n';

}

connectBtn.addEventListener('click', connectToServer);

sendBtn.addEventListener('click', sendMessage);

disconnectBtn.addEventListener('click', () => {user.local\_key\_c\_ss = ''});

Листинг **–** Файл *kerberos-server.js*

const express = require('express');

const axios = require('axios');

const cors = require('cors');

const nsc = require('./nsc');

const des = require('./des');

const app = express();

const port = 3002;

app.use(

cors({

origin: 'http://127.0.0.1:5500',

})

);

const kerberosServer = {

id: 'kerb\_tgs',

key\_as\_tgs: 'FEDCBA0987654321',

key\_c\_tgs: '1122334455667788',

key\_tgs\_ss: '1234567890ABCDEF',

clients: [

{

id: 'aboba',

c\_key: 'key123client'

},

{

id: 'daddy',

c\_key: 'sugar7daddy'

}

],

};

function generateRandomKey(length) {

const charset = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';

let key = '';

for (let i = 0; i < length; i++) {

const randomIndex = Math.floor(Math.random() \* charset.length);

key += charset[randomIndex];

}

return key;

}

app.get('/AS/:id', (req, res) => {

const id = req.params.id;

console.log('====================================================================');

console.log('================================AS==================================');

console.log('====================================================================');

console.log(`AS: Проверка на наличие клиента: ${id}`);

const client = kerberosServer.clients.find(client => client.id === id);

if (!client) {

console.log(`AS: Клиент ${id} не найден`);

return res.status(404).json({msg: `AS: Клиент ${id} не найден`});

}

console.log(`AS: Клиент ${client.id} найден`);

let timestamp = new Date().getTime();

let TGT = client.id + " " + kerberosServer.id + " " + timestamp + " " + "10000000";

console.log(`AS: Создание TGT: ${TGT}`);

TGT = nsc.binaryToHex(des.encrypt(

nsc.textToBinary(TGT),

nsc.textToBinary(kerberosServer.key\_as\_tgs)

));

console.log(`AS: Зашифрованный TGT: ${TGT}`);

let msgForClient = TGT + " " + kerberosServer.key\_c\_tgs;

console.log(`AS: TGT + Kc\_tgs: ${msgForClient}`);

msgForClient = nsc.binaryToHex(des.encrypt(

nsc.textToBinary(msgForClient),

nsc.textToBinary(client.c\_key)

));

console.log(`AS: Зашифрованное сообщение TGT + Kc\_tgs: ${msgForClient}`);

console.log('====================================================================');

console.log('====================================================================');

return res.status(200).json({ msg: msgForClient});

});

app.get('/TGS/:msg', (req, res) => {

console.log('====================================================================');

console.log('================================TGS=================================');

console.log('====================================================================');

const msg = req.params.msg;

console.log(`TGS: Получено сообщение: ${msg}`);

let [tgt, aut1, serverId] = msg.split(' ');

tgt = nsc.binaryToText(des.decrypt(

nsc.hexToBinary(tgt),

nsc.textToBinary(kerberosServer.key\_as\_tgs)

));

console.log(`TGS: Расшифрованный TGT: ${tgt}`);

aut1 = nsc.binaryToText(des.decrypt(

nsc.hexToBinary(aut1),

nsc.textToBinary(kerberosServer.key\_c\_tgs)

));

console.log(`TGS: Расшифрованный Aut1: ${aut1}`);

const [id\_tgt, tgs\_tgt, t\_tgt, p\_tgt] = tgt.split(' ');

const [id\_aut1, t\_aut1] = aut1.split(' ');

if (id\_tgt != id\_aut1) {

console.log(`TGS: Ошибка аунтификации`);

return res.status(403).json({msg: `TGS: Ошибка аунтификации`});

}

if (tgs\_tgt != kerberosServer.id) {

console.log(`TGS: Неверный TGS`);

return res.status(403).json({msg: `TGS: Неверный TGS`});

}

if (parseInt(t\_aut1) - parseInt(t\_tgt) > parseInt(p\_tgt)) {

console.log(`TGS: Время действия билета истекло`);

return res.status(403).json({msg: `TGS: Время действия билета истекло`});

}

let timestamp = new Date().getTime();

let key\_c\_ss = generateRandomKey(10);

let tgs = id\_tgt + ' ' + serverId + ' ' + timestamp + ' ' + '1000000' + ' ' + key\_c\_ss;

console.log(`TGS: Создан TGS: ${tgs}`);

tgs = nsc.binaryToHex(des.encrypt(

nsc.textToBinary(tgs),

nsc.textToBinary(kerberosServer.key\_tgs\_ss)

));

console.log(`TGS: Зашифрованный TGS: ${tgs}`);

let msgForClient = tgs + ' ' + key\_c\_ss;

console.log(`TGS: Сообщение TGS + Kc\_ss: ${msgForClient}`);

msgForClient = nsc.binaryToHex(des.encrypt(

nsc.textToBinary(msgForClient),

nsc.textToBinary(kerberosServer.key\_c\_tgs)

));

console.log(`TGS: Зашифрованное ообщение: ${msgForClient}`);

console.log('====================================================================');

console.log('====================================================================');

return res.status(200).json({msg: msgForClient});

});

app.listen(port, () => {

console.log(`Сервер KERBEROS запущен на порту: ${port}`);

});

Листинг **–** Файл *s-server.js*

const express = require('express');

const axios = require('axios');

const cors = require('cors');

const nsc = require('./nsc');

const des = require('./des');

const app = express();

const port = 3003;

const key\_tgs\_ss = '1234567890ABCDEF';

const server\_id = 's\_server';

let local\_key\_c\_ss = '';

app.use(

cors({

origin: 'http://127.0.0.1:5500',

})

);

app.get('/gettext/:msg' , (req, res) => {

try{

const msg = req.params.msg;

let msgForClient = nsc.binaryToText(des.decrypt(

nsc.hexToBinary(msg),

nsc.textToBinary(local\_key\_c\_ss)

));

console.log('====================================================================');

console.log(`SS: КЛИЕНТ ${msgForClient}`);

msgForClient = '!НАЧАЛО | ' + msgForClient + ' | КОНЕЦ!';

console.log(`SS: СЕРВЕР ${msgForClient}`);

msgForClient = nsc.binaryToHex(des.encrypt(

nsc.textToBinary(msgForClient),

nsc.textToBinary(local\_key\_c\_ss)

));

res.status(200).json({msg: msgForClient});

console.log(`SS: Отправлено!`);

console.log('====================================================================');

} catch (error) {

if (local\_key\_c\_ss === '') {

res.status(403).json({msg: 'SS: Нет доступа'});

} else {

res.status(500).json({msg: 'SS: Ошибка'});

console.log(`[ERROR]: SS: ${error}`);

}

}

});

app.get('/SS/:msg', (req, res) => {

console.log('====================================================================');

console.log('================================SS==================================');

console.log('====================================================================');

const msg = req.params.msg;

console.log(`SS: Получено сообщение: ${msg}`);

let [tgs, aut2] = msg.split(' ');

tgs = nsc.binaryToText(des.decrypt(

nsc.hexToBinary(tgs),

nsc.textToBinary(key\_tgs\_ss)

));

console.log(`SS: Расшифрованный TGS: ${tgs}`);

let [id\_tgs, server\_id\_tgs, t\_tgs, p\_tgs, key\_c\_ss] = tgs.split(' ');

if (server\_id\_tgs != server\_id) {

return res.status(403).json({msg: 'Неверный сервер'});

}

aut2 = nsc.binaryToText(des.decrypt(

nsc.hexToBinary(aut2),

nsc.textToBinary(key\_c\_ss)

));

console.log(`SS: Расшифрованный Aut2: ${aut2}`);

let [id\_aut2, t\_aut2] = aut2.split(' ');

if (id\_tgs != id\_aut2) {

res.status(404).json({msg: `Клиент не найден`});

}

let msgForClient = (parseInt(t\_aut2)) + 1;

console.log(`SS: Сообщение для клиента: ${msgForClient}`);

msgForClient = nsc.binaryToHex(des.encrypt(

nsc.textToBinary(msgForClient),

nsc.textToBinary(key\_c\_ss)

));

console.log(`SS: Зашифрованное сообщение для клиента: ${msgForClient}`);

local\_key\_c\_ss = key\_c\_ss;

console.log('====================================================================');

console.log('====================================================================');

return res.status(200).json({msg: msgForClient});

});

app.listen(port, () => {

console.log(`Сервер SS запущен на порту: ${port}`);

});

Листинг **–** Файл *des.js*

const initial\_perm = [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];

const ext\_func = [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];

const transf\_s = [

[

[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]

]

];

const p\_per = [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];

const gen\_key = [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];

const shift\_table = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1];

const final\_key\_process = [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];

const final\_perm = [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];

class DES {

static encrypt(message\_bin, key\_bin, en=true) {

if (message\_bin.length % 64 !== 0) {

message\_bin += '0'.repeat(64 - (message\_bin.length % 64));

}

if (key\_bin.length % 56 !== 0) {

key\_bin += '0'.repeat(64 - (key\_bin.length % 56));

}

let result = '';

for (let q = 0; q < message\_bin.length / 64; q++) {

let pt = message\_bin.substring(q \* 64, q \* 64 + 64);

pt = this.permute(pt, initial\_perm, 64);

let left = pt.substring(0, 32);

let right = pt.substring(32);

let p\_key = this.permute(key\_bin, gen\_key, 56);

let sp\_keys = [];

shift\_table.forEach( (n) => {

sp\_keys.push(this.shift(p\_key, n));

});

if (!en) {

sp\_keys.reverse();

}

for (let i = 0; i < 16; i++) {

p\_key = sp\_keys[i];

p\_key = this.permute(p\_key, final\_key\_process, 48);

let right\_ex = this.permute(right, ext\_func, 48);

let result = this.xor(p\_key, right\_ex);

let transf\_s\_result = '';

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

let row = parseInt(result[j \* 6] + result[j \* 6 + 5], 2);

let col = parseInt(result[j \* 6 + 1] + result[j \* 6 + 4]);

transf\_s\_result += (transf\_s[j][row][col]).toString(2).padStart(4, '0');

}

let p\_per\_result = this.permute(transf\_s\_result, p\_per, 32);

result = this.xor(p\_per\_result, left);

left = result;

if (i !== 15) {

[left, right] = [right, left];

}

}

let combine = left + right;

result += this.permute(combine, final\_perm, 64);

}

return result;

}

static decrypt(message, key) {

return this.encrypt(message, key, false);

}

static permute(bytes, template, n) {

return template.reduce( (res, next) => {

return res + bytes.charAt(next - 1);

}, '');

}

static shift(key, n) {

for (let i = 0; i < n; i++) {

let fl = key[0];

let fr = key[28];

key = key.substring(1, 28) + fl + key.substring(29) + fr;

}

return key;

}

static xor(a, b) {

return a.split('').reduce((result, char, index) => {

return result + (char === b.charAt(index) ? '0' : '1');

}, '');

}

}

module.exports = DES;

# **ПРИЛОЖЕНИЕ Б (обязательное) Блок-схема алгоритма**

