Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЁТ

к лабораторной работе №2

на тему

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

Студент: гр.153502

Толстой Д.В.

Руководитель: Лещенко Е. А.

Минск 2024

**СОДЕРЖАНИЕ**

# Введение 3

# Демонстрация программного средства 4

# Заключение 9

Приложение А Схема протокола Kerberos 10

Приложение Б Схема алгоритма DES 11

Приложение В Листинг кода 12

**ВВЕДЕНИЕ**

Протокол *Kerberos* является одной из реализаций протокола аутентификации с использованием третьей стороны, призванной уменьшить количество сообщений, которыми обмениваются стороны.

Задачи лабораторной работы:

* изучение теоретических сведений о *Kerberos* и *DES*;
* создание приложения, реализующего протокол распределения ключей Kerberos, включая процедуру, реализующую алгоритм *DES*.

**2 ДЕМОНСТРАЦИЯ ПРОГРАММНОГО СРЕДСТВА**

Программное средство разделено на 4 части:

* клиент;
* сервер аутентификации;
* сервер выдачи разрешений;
* сервер, доступ к которому хочет получить клиент.

Первоначально клиент посылает незашифрованное сообщение со своим идентификатором серверу аутентификации. При этом в консоль клиента выводится соответствующее сообщение (рисунок 1).

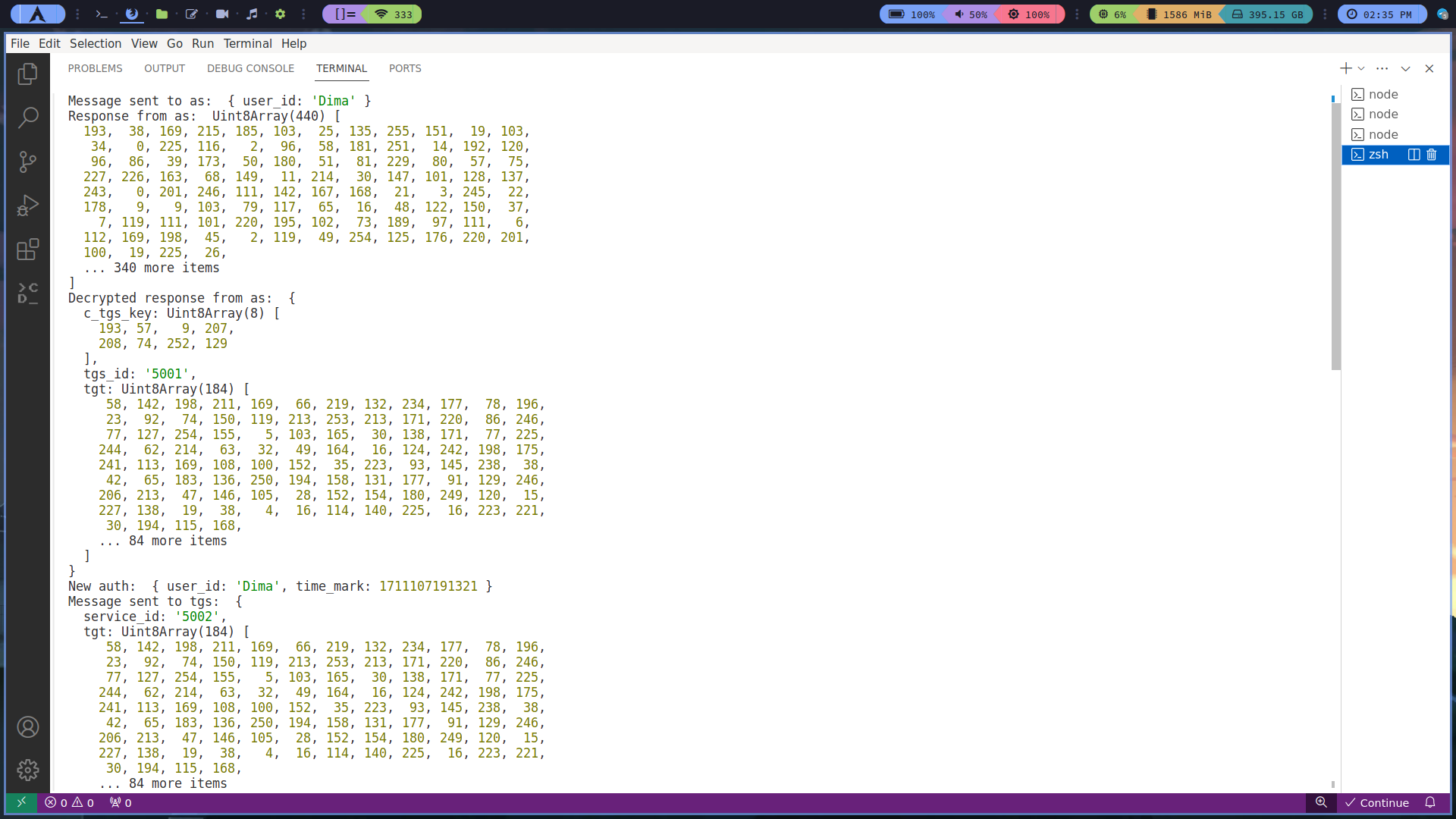


Рисунок 1 – Сообщение об отправке идентификатора

Когда сервер получает сообщение, он проверяет, имеется ли в списке клиентов сервера полученный идентификатор. Если такой идентификатор имеется, сервер отправляет клиенту билет для доступа к серверу выдачи разрешений, зашифрованные на ключе клиента, и ключ для взаимодействия с сервером выдачи разрешений (рисунок 2).

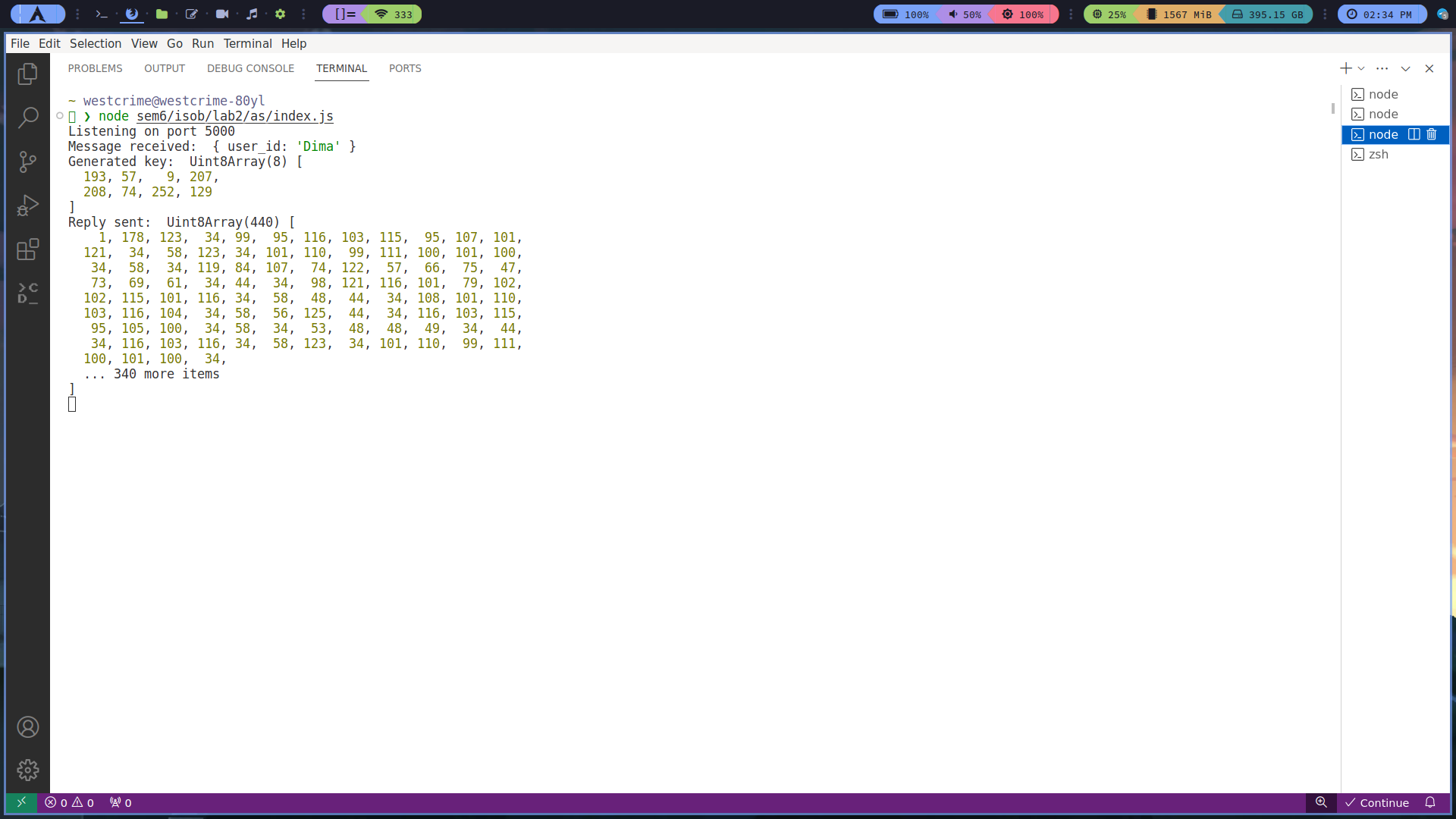


Рисунок 2 – Отправка ответа сервером аутентификации клиенту

После получения ответа от сервера аутентификации и его расшифровки (рисунок 3), клиент отправляет полученный билет вместе с аутентификационным блоком серверу разрешений (рисунок 4)

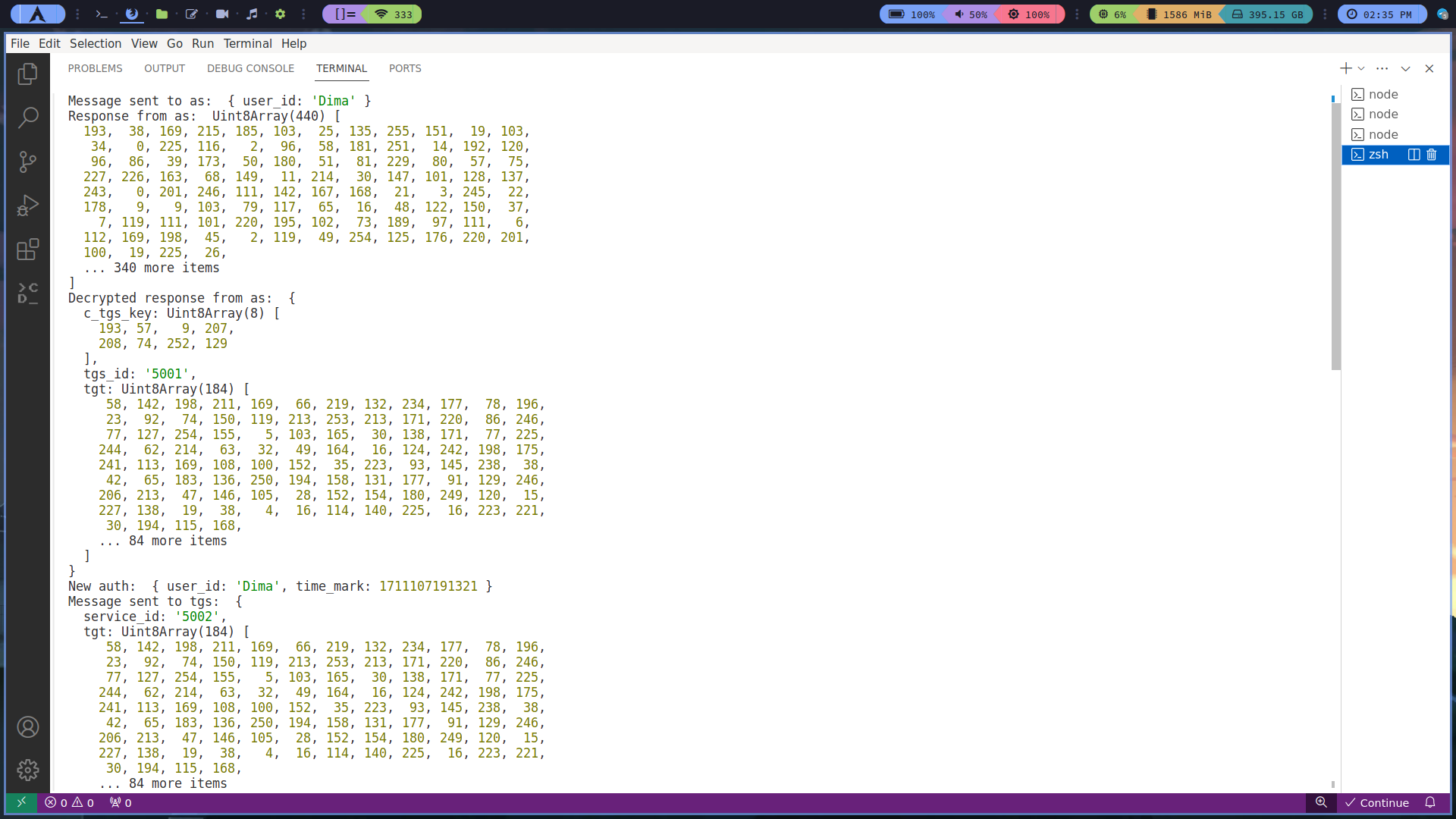
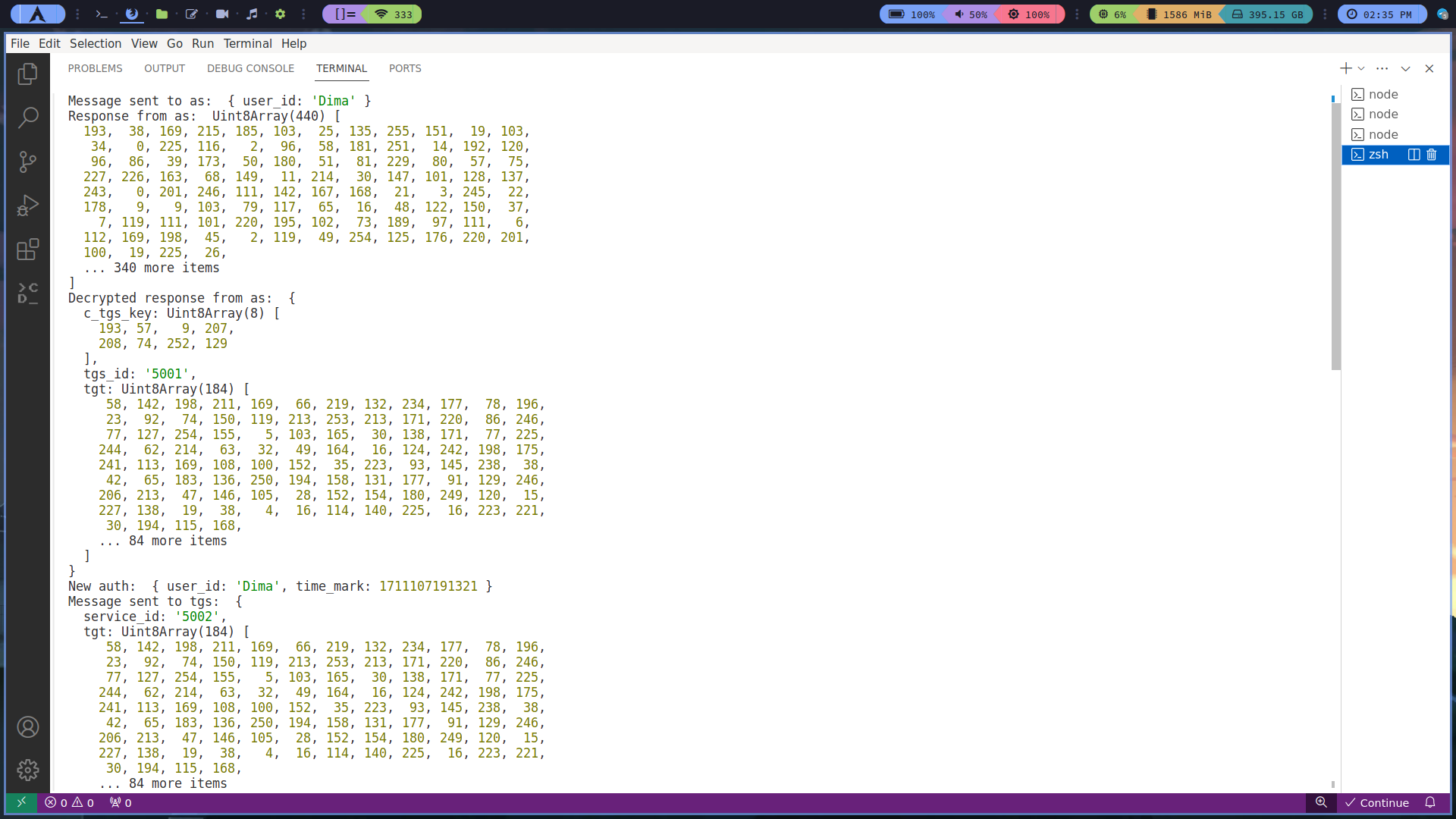


Рисунок 3 – Получение ответа от сервера аутентификации



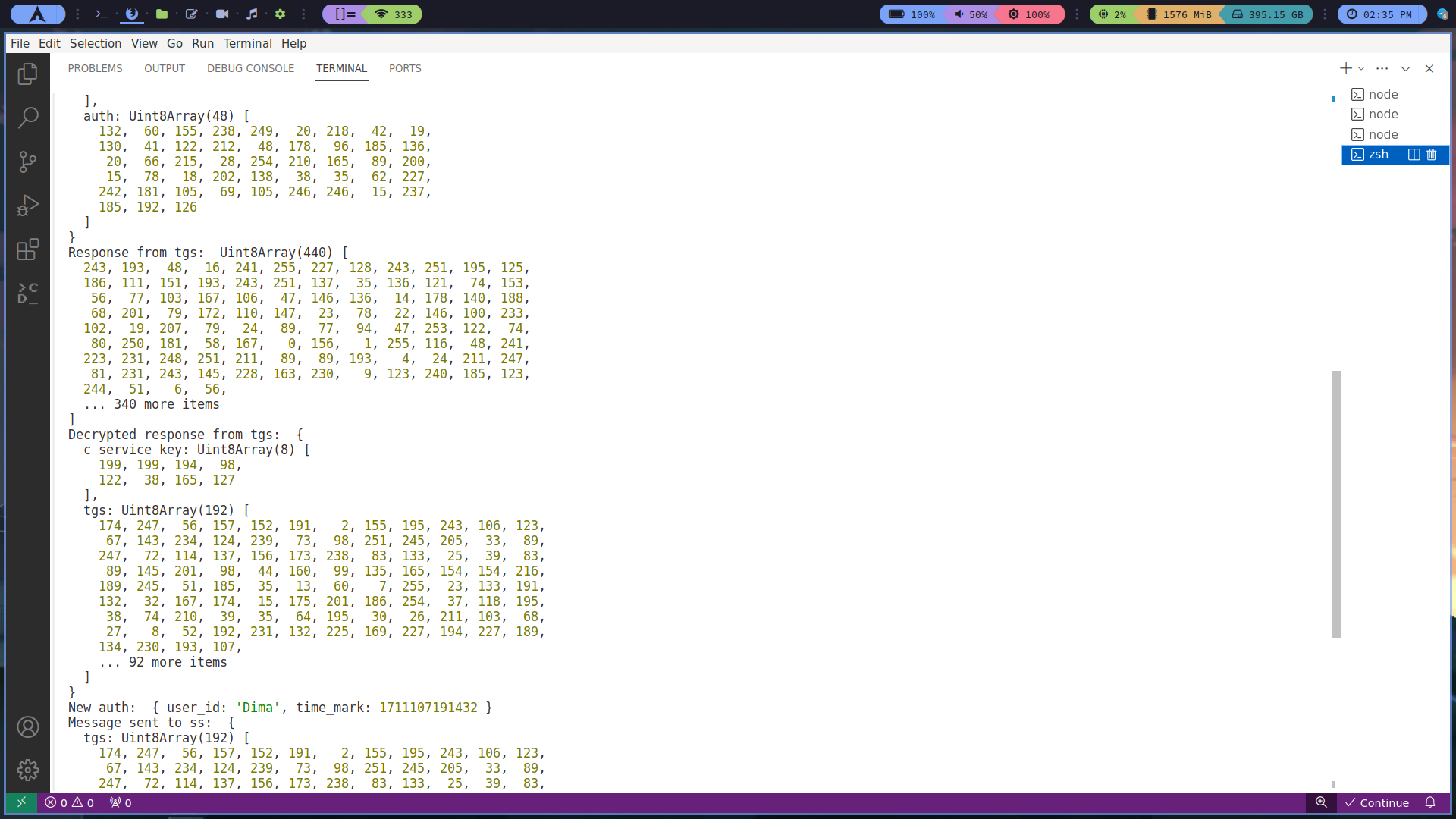


Рисунок 4 – Отправка сообщения серверу выдачи разрешений

Когда сервер выдачи разрешений получает сообщение, он проверяет соответствие идентификатора в билете идентификатору в аутентификационном блоке, а также удостоверяется, что срок действия билета не истек. В случае выполнения перечисленных условий, сервер посылает клиенту билет для доступа к желаемому серверу и ключ для взаимодействия с ним (рисунок 5).

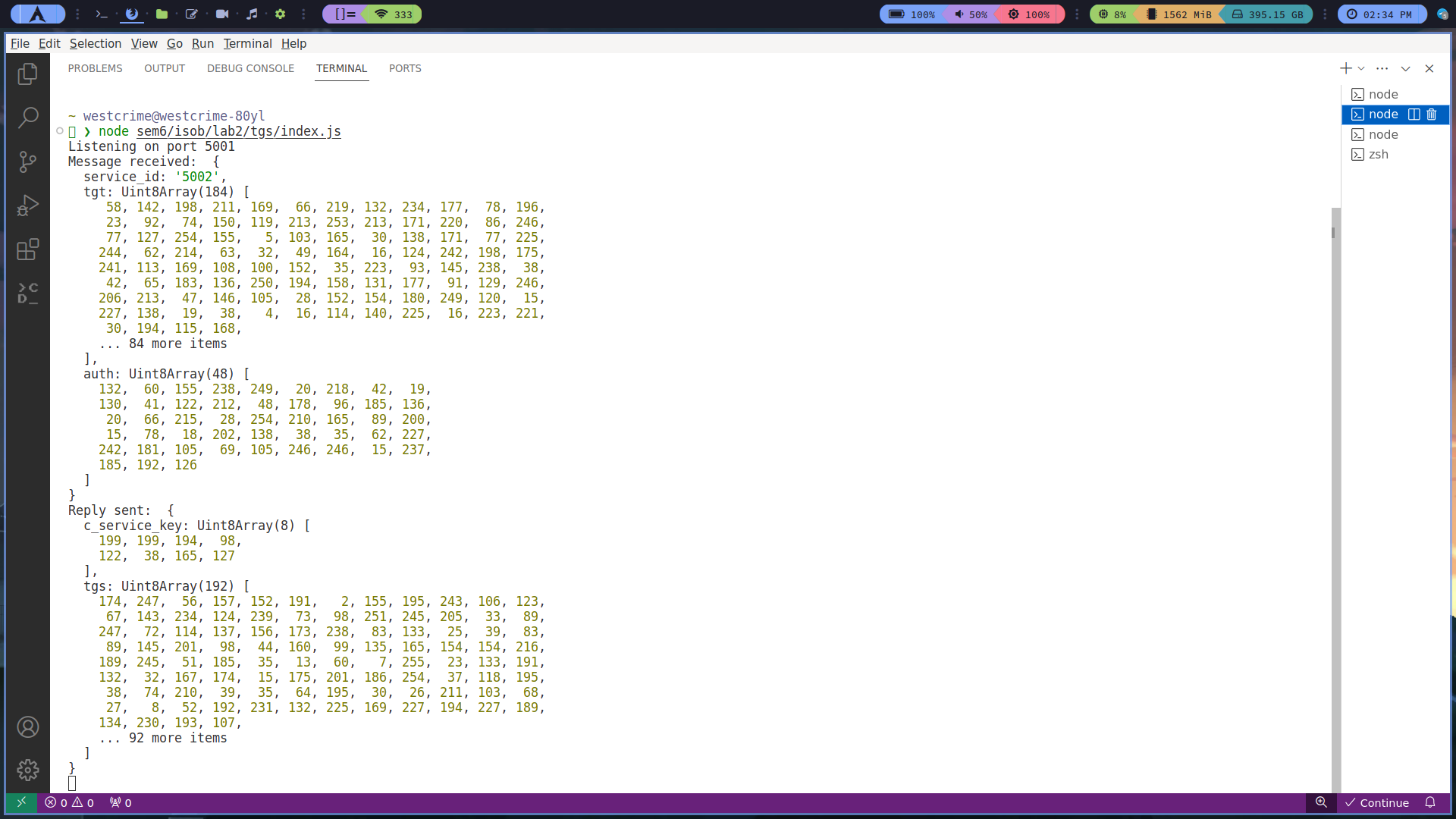


Рисунок 5 – Получение и отправка сообщений сервером выдачи разрешений

После получения ответа от сервера выдачи разрешений клиент пересылает полученный билет с новым аутентификационным блоком серверу, доступ к которому хотел получить изначально (рисунок 6).

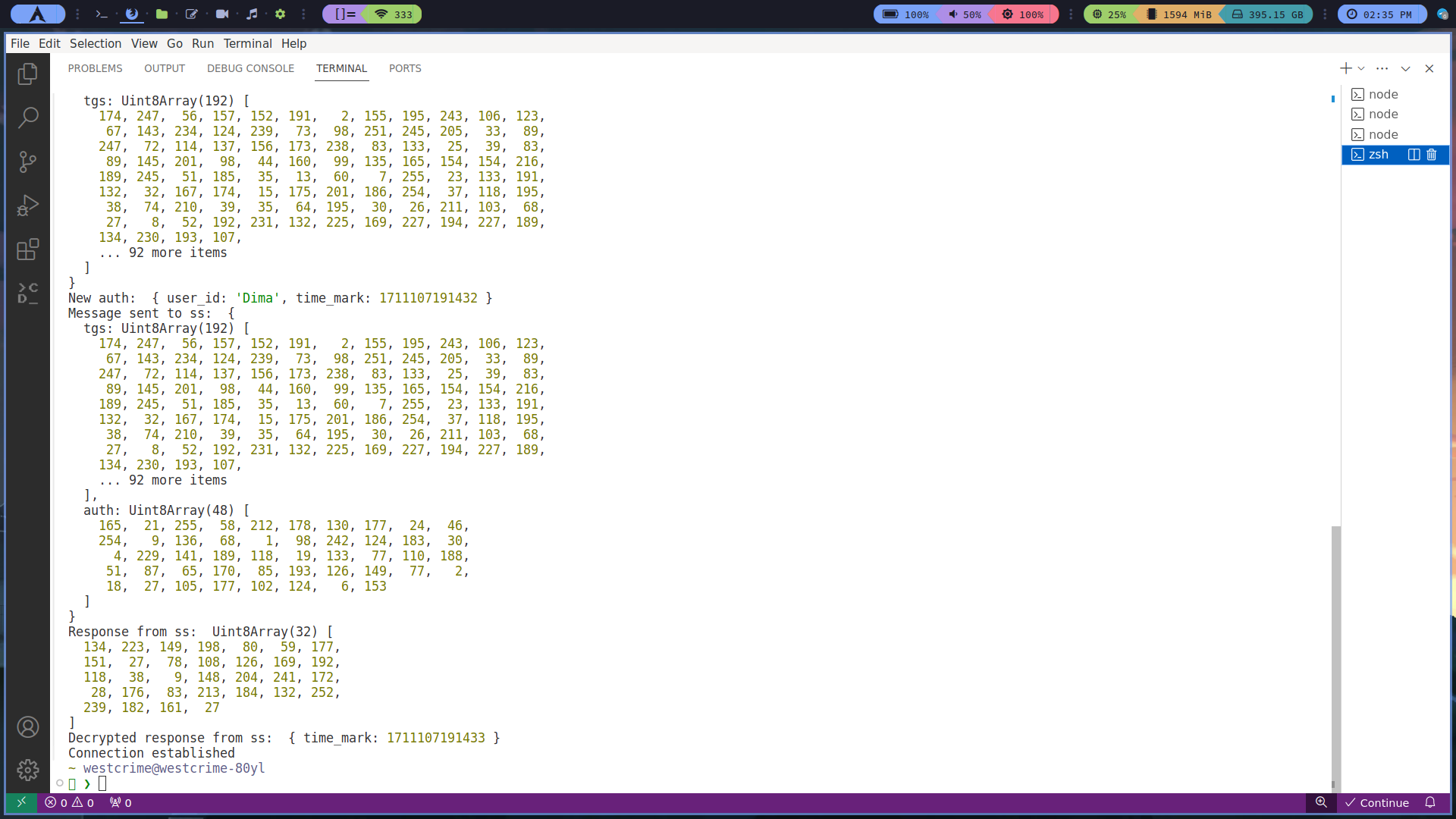


Рисунок 6 – отправка сообщения целевому серверу

После получения сообщения от клиента целевой сервер проводит проверки, аналогичные проверкам сервера выдачи разрешений, после чего отправляет клиенту увеличенное на 1 значение временной метки из полученного аутентификационного блока клиенту (рисунок 7).

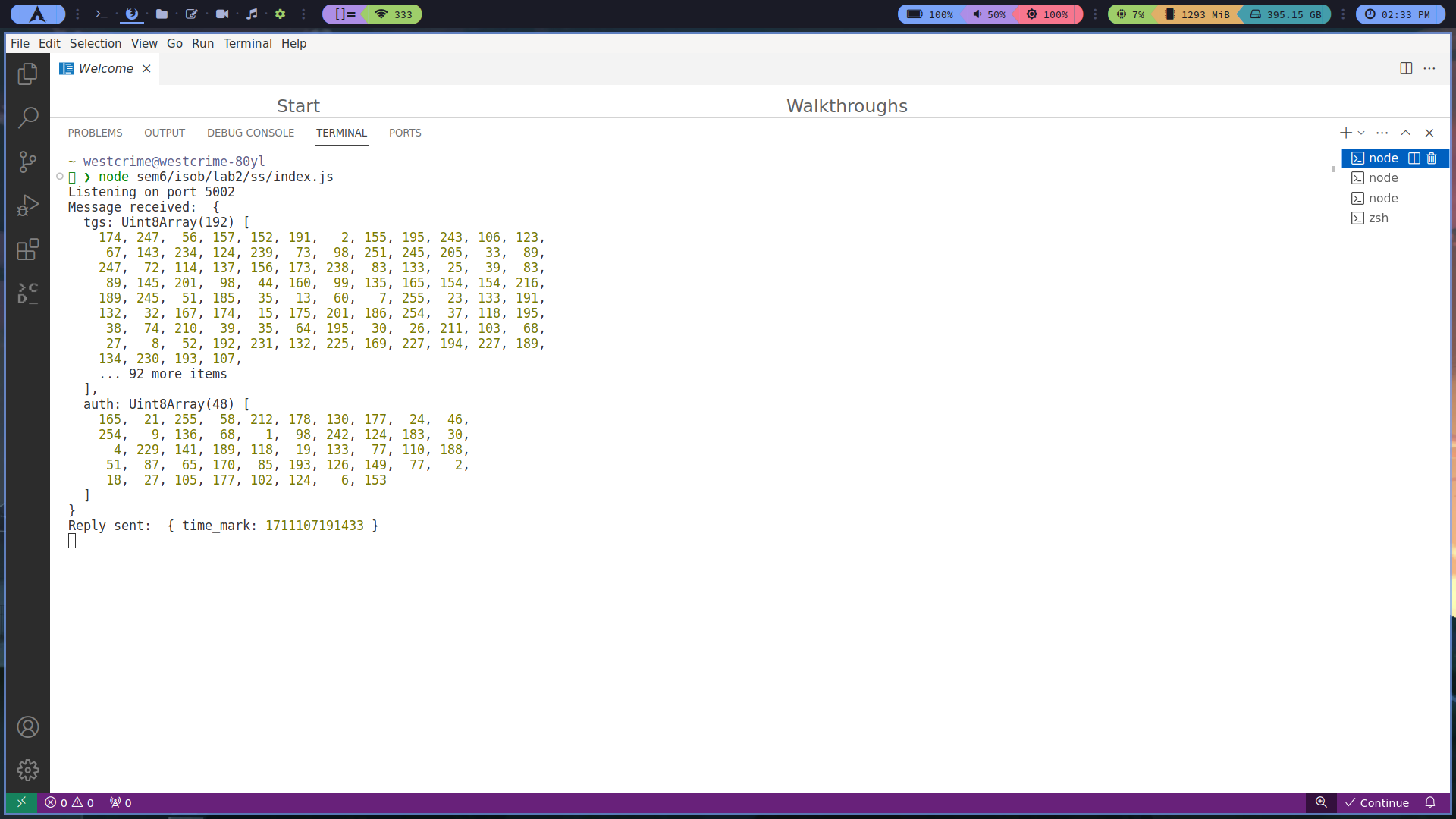


Рисунок 7 – Получение и отправка сообщений целевым сервером

После всех этих действий клиенту остается только удостовериться в подлинности сервера, проверив полученную временную метку. Тогда соединение считается установленным (рисунок 8).

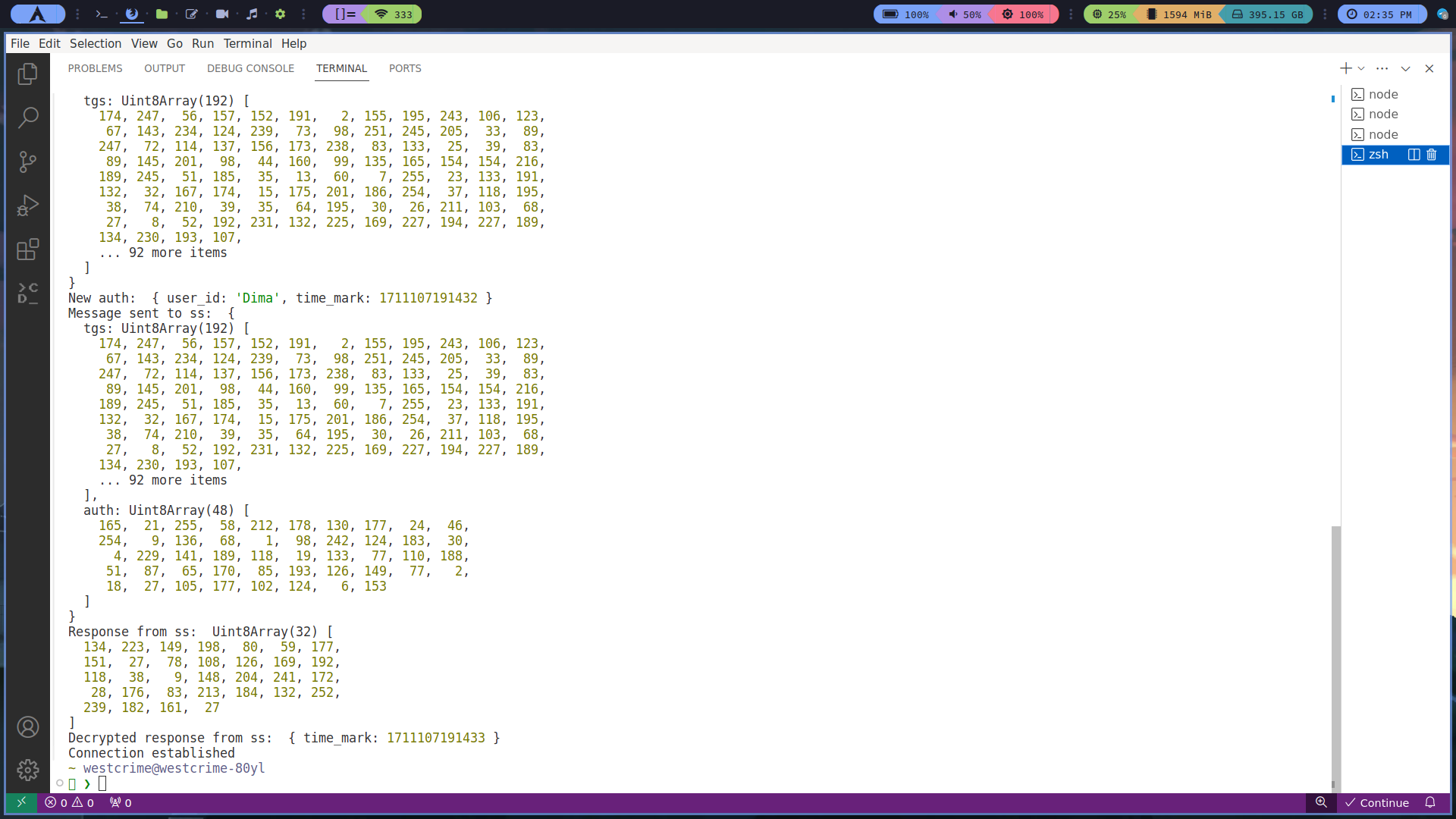


Рисунок 8 – Получение клиентом сообщения от целевого сервера

**ЗАКЛЮЧЕНИЕ**

В ходе данной лабораторной работы было создано приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую алгоритм DES.

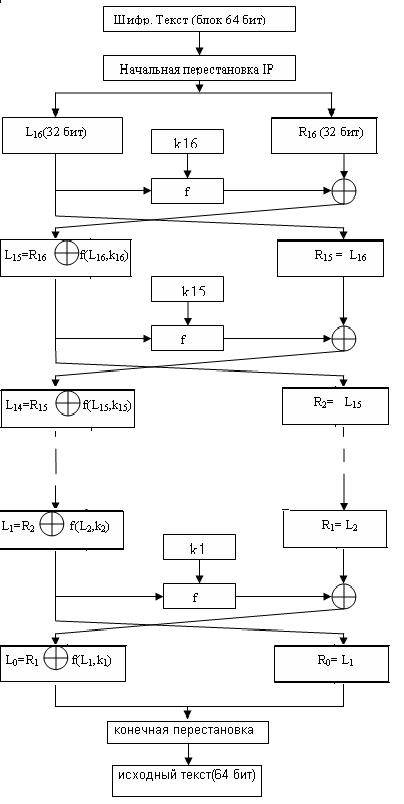
**ПРИЛОЖЕНИЕ А**

**Схема протокола Kerberos**



**ПРИЛОЖЕНИЕ Б**

**Схема алгоритма DES**



**ПРИЛОЖЕНИЕ В**

**Листинг кода**

**Клиент**

const { getSubKeys, decrypt, encrypt, pack, unpack, typeson } = require("des");

const user\_id = 'Dima';

//const user\_id = "Undefined";

const service\_id = '5002';

const user\_key = new Uint8Array([ 2, 3, 4, 5, 5, 4, 3, 2 ]);

const as\_id = '5000';

const headers = {

'Content-Type': 'application/json'

};

let service\_keys = new Map();

function genAuth() {

let auth = {

user\_id: user\_id,

time\_mark: Date.now()

}

console.log('New auth: ', auth);

return auth

}

async function connectServer(message, server\_id) {

return await fetch(`http://localhost:${server\_id}/server/authentication`, {

method: 'post',

headers: headers,

body: JSON.stringify(typeson.encapsulate(message))

}).then((response) => response.text()). then((response) => {

response = typeson.revive(JSON.parse(response));

return response;

});

}

async function prepareAsMessage() {

let asMessage = {

user\_id: user\_id

};

console.log('Message sent to as: ', asMessage);

let asResponse = await connectServer(asMessage, as\_id);

console.log('Response from as: ', asResponse);

let asDecrResponse = unpack(decrypt(asResponse, getSubKeys(user\_key)));

console.log('Decrypted response from as: ', asDecrResponse);

return asDecrResponse;

}

async function prepareTgsMessage(asDecrResponse) {

let auth = genAuth();

let tgt = asDecrResponse.tgt;

let tgs\_id = asDecrResponse.tgs\_id;

let tgsMessage = {

service\_id: service\_id,

tgt: tgt,

auth: encrypt(pack(auth), getSubKeys(asDecrResponse.c\_tgs\_key))

};

console.log('Message sent to tgs: ', tgsMessage);

let tgsResponse = await connectServer(tgsMessage, tgs\_id);

console.log('Response from tgs: ', tgsResponse);

let tgsDecrResponse = unpack(decrypt(tgsResponse, getSubKeys(asDecrResponse.c\_tgs\_key)));

console.log('Decrypted response from tgs: ', tgsDecrResponse);

return tgsDecrResponse;

}

async function prepareSsMessage(tgsDecrResponse) {

let auth = genAuth();

let tgs = tgsDecrResponse.tgs;

let ssMessage = {

tgs: tgs,

auth: encrypt(pack(auth), getSubKeys(tgsDecrResponse.c\_service\_key))

};

console.log('Message sent to ss: ', ssMessage);

let ssResponse = await connectServer(ssMessage, service\_id);

console.log('Response from ss: ', ssResponse);

let ssDecrResponse = unpack(decrypt(ssResponse, getSubKeys(tgsDecrResponse.c\_service\_key)));

console.log('Decrypted response from ss: ', ssDecrResponse);

if (ssDecrResponse.time\_mark != auth.time\_mark + 1) {

console.log('Service cannot be trusted');

} else {

console.log('Connection established');

service\_keys.set(service\_id, tgsDecrResponse.c\_service\_key);

}

}

async function connect() {

await prepareSsMessage(await prepareTgsMessage(await prepareAsMessage()));

}

connect();

**Сервер аутентификации**

const { getSubKeys, decrypt, encrypt, pack, unpack, typeson } = require("des");

const express = require("express");

const app = express();

const port = process.env.PORT || 5000;

let user\_keys = new Map(); // user\_id, key

user\_keys.set('Dima', new Uint8Array([ 2, 3, 4, 5, 5, 4, 3, 2]));

let as\_tgs\_key = new Uint8Array([ 1, 2, 3, 4, 5, 6, 7, 8 ]);

let tgs\_id = '5001'; // port of tgs

const availability\_term = 5 \* 1000; // millisecons

app.use(express.json());

app.listen(port, () => console.log(`Listening on port ${port}`));

function genUserKey() {

const max = 255;

return new Uint8Array([ Math.floor(Math.random() \* max),

Math.floor(Math.random() \* max),

Math.floor(Math.random() \* max),

Math.floor(Math.random() \* max),

Math.floor(Math.random() \* max),

Math.floor(Math.random() \* max),

Math.floor(Math.random() \* max),

Math.floor(Math.random() \* max) ]);

}

app.post('/server/authentication', (req, res) => {

let body = typeson.revive(req.body);

console.log('Message received: ', body);

let user\_id = body.user\_id;

if (user\_keys.has(user\_id)) {

let c\_tgs\_key = genUserKey();

console.log('Generated key: ', c\_tgs\_key);

let tgt = {

c\_tgs\_key: c\_tgs\_key,

user\_id: user\_id,

available\_until: Date.now() + availability\_term,

time\_mark: Date.now()

}

tgt = encrypt(pack(tgt), getSubKeys(as\_tgs\_key));

let response = {

c\_tgs\_key: c\_tgs\_key,

tgs\_id: tgs\_id,

tgt: tgt

};

console.log('Reply sent: ', pack(response));

response = encrypt(pack(response), getSubKeys(user\_keys.get(user\_id)));

res.send(JSON.stringify(typeson.encapsulate(response)));

} else {

console.log('No such user');

}

})

**Сервер выдачи разрешений**

const { getSubKeys, decrypt, encrypt, pack, unpack, typeson } = require("des");

const express = require("express");

const app = express();

const port = process.env.PORT || 5001;

let tgs\_service\_keys = new Map(); // service\_id, key

tgs\_service\_keys.set('5002', new Uint8Array([ 8, 7, 6, 5, 4, 3, 2, 1]));

let as\_tgs\_key = new Uint8Array([ 1, 2, 3, 4, 5, 6, 7, 8 ]);

const availability\_term = 5 \* 1000 \* 60; // millisecons

const user\_mark\_expiration\_term = 2 \* 1000 \* 60; // millisecons

app.use(express.json());

app.listen(port, () => console.log(`Listening on port ${port}`));

// function sleepFor(sleepDuration){

// var now = new Date().getTime();

// while(new Date().getTime() < now + sleepDuration);

// }

app.post('/server/authentication', (req, res) => {

let body = typeson.revive(req.body);

console.log('Message received: ', body);

let service\_id = body.service\_id;

let tgt = body.tgt;

let auth = body.auth;

tgt = unpack(decrypt(tgt, getSubKeys(as\_tgs\_key)));

if (typeof(tgt) !== 'object'){

console.log('Cannot decrypt value!');

return;

}

//sleepFor(10000);

if (tgt.available\_until < Date.now()) {

console.log('Term of tgt expired!');

return;

}

auth = unpack(decrypt(auth, getSubKeys(tgt.c\_tgs\_key)));

if (Date.now() - auth.time\_mark > user\_mark\_expiration\_term) {

console.log('User mark is too old!');

return;

}

if (auth.user\_id !== tgt.user\_id) {

console.log('User undefined!');

return;

}

if (tgs\_service\_keys.has(service\_id)) {

let c\_service\_key = tgs\_service\_keys.get(service\_id);

let tgs = {

c\_service\_key: c\_service\_key,

user\_id: auth.user\_id,

available\_until: Date.now() + availability\_term,

time\_mark: Date.now()

}

tgs = encrypt(pack(tgs), getSubKeys(tgs\_service\_keys.get(service\_id)));

let response = {

c\_service\_key: c\_service\_key,

tgs: tgs

};

console.log('Reply sent: ', response);

response = encrypt(pack(response), getSubKeys(tgt.c\_tgs\_key));

res.send(JSON.stringify(typeson.encapsulate(response)));

} else {

console.log('Undefined service!');

}

})

**Целевой сервер**

const { getSubKeys, decrypt, encrypt, pack, unpack, typeson } = require("des");

const express = require("express");

const app = express();

const port = process.env.PORT || 5002;

let tgs\_service\_key = new Uint8Array([ 8, 7, 6, 5, 4, 3, 2, 1]);

const user\_mark\_expiration\_term = 2 \* 1000 \* 60; // millisecons

let user\_keys = new Map();

app.use(express.json());

app.listen(port, () => console.log(`Listening on port ${port}`));

app.post('/server/authentication', (req, res) => {

let body = typeson.revive(req.body);

console.log('Message received: ', body);

let tgs = body.tgs;

let auth = body.auth;

tgs = unpack(decrypt(tgs, getSubKeys(tgs\_service\_key)));

if (typeof(tgs) !== 'object'){

console.log('Cannot decrypt value!');

return;

}

if (tgs.available\_until < Date.now()) {

console.log('Term of tgt expired!');

return;

}

auth = unpack(decrypt(auth, getSubKeys(tgs.c\_service\_key)));

if (Date.now() - auth.time\_mark > user\_mark\_expiration\_term) {

console.log('User mark is too old!');

return;

}

if (auth.user\_id !== tgs.user\_id) {

console.log('User undefined!');

return;

}

user\_keys.set(auth.user\_id, tgs.c\_service\_key);

let response = {

time\_mark: auth.time\_mark + 1

}

console.log('Reply sent: ', response);

response = encrypt(pack(response), getSubKeys(user\_keys.get(auth.user\_id)));

res.send(JSON.stringify(typeson.encapsulate(response)));

})

**DES**

const {Typeson, builtin} = require('typeson-registry');

const bits\_in\_byte = 8;

const bits\_in\_tetrade = 4;

const key\_size\_in\_bytes = 8;

const value\_size\_in\_bytes = 8;

const round\_key\_size\_in\_bytes = 7;

const subkey\_size\_in\_bytes = 6;

const part\_size\_in\_bytes = 4;

const round\_count = 16;

const round\_shifts = [ 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 ];

const subkey\_indexes = [ 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 S\_boxes = [ [ [ 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 = [ 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 permutation\_indexes = [ 4, 0, 5, 1, 6, 2, 7, 3 ];

const typeson = new Typeson().register([

builtin

]);

function shiftLeft(bytes, shift) {

if (shift > bits\_in\_byte) {

throw new Error('Invalid shift');

}

const ones = 0b11111111;

const bits\_not\_remembered = bits\_in\_byte - shift;

const remembered\_bits = (ones << bits\_not\_remembered) & ones;

let tmp = 0;

for (let i = bytes.length - 1; i >= 0; i--) {

let new\_byte = (bytes[i] << shift) | tmp;

tmp = (bytes[i] & remembered\_bits) >> bits\_not\_remembered;

bytes[i] = new\_byte;

}

bytes[bytes.length - 1] |= tmp;

}

function shiftRight(bytes, shift) {

if (shift > bits\_in\_byte) {

throw new Error('Invalid shift');

}

const ones = 0b11111111;

const bits\_not\_remembered = bits\_in\_byte - shift;

const remembered\_bits = (ones >> bits\_not\_remembered) & ones;

let tmp = 0;

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

let new\_byte = (bytes[i] >> shift) | tmp;

tmp = (bytes[i] & remembered\_bits) << bits\_not\_remembered;

bytes[i] = new\_byte;

}

bytes[0] |= tmp;

}

function getBit(bytes, index) {

let bit\_pos = index % bits\_in\_byte;

let byte\_pos = Math.floor(index / bits\_in\_byte);

let byte\_with\_bit\_set = 0b10000000 >> bit\_pos;

return !!(bytes[byte\_pos] & byte\_with\_bit\_set)

}

function setBit(bytes, index, value) {

let bit\_pos = index % bits\_in\_byte;

let byte\_pos = Math.floor(index / bits\_in\_byte);

let byte\_with\_bit\_set = 0b10000000 >> bit\_pos;

if (value) {

bytes[byte\_pos] |= byte\_with\_bit\_set;

} else {

bytes[byte\_pos] &= ~byte\_with\_bit\_set;

}

}

function permutedChoice(key) {

let byte\_with\_bit\_set = 0b10000000;

let result = new Uint8Array(round\_key\_size\_in\_bytes).fill(0);

for (let i = 0; i < round\_key\_size\_in\_bytes; i++)

{

for (let j = 0; j < key\_size\_in\_bytes; j++)

{

result[round\_key\_size\_in\_bytes - 1 - i] |= ((key[j] & byte\_with\_bit\_set) << i) >> (value\_size\_in\_bytes - 1 - j);

}

byte\_with\_bit\_set >>= 1;

}

let tmp = result[4];

result[4] = result[6];

result[6] = tmp;

shiftLeft(result.subarray(3, 7), 4);

return result;

}

function shiftRoundKey(lastKey, shift\_size) {

const D\_last\_bits = (0b00001111 >> (bits\_in\_tetrade - shift\_size)) << bits\_in\_tetrade;

const C\_last\_bits = 0b11111111 >> (bits\_in\_byte - shift\_size);

const D\_last\_byte = 3;

const C\_last\_byte = 6;

shiftLeft(lastKey, shift\_size);

let tmp = lastKey[D\_last\_byte] & D\_last\_bits

lastKey[D\_last\_byte] = (lastKey[D\_last\_byte] & ~D\_last\_bits) | ((lastKey[C\_last\_byte] & C\_last\_bits) << bits\_in\_tetrade);

lastKey[C\_last\_byte] = (lastKey[C\_last\_byte] & ~C\_last\_bits) | (tmp >> bits\_in\_tetrade);

}

function getSubKey(roundKey) {

const round\_key\_size\_in\_bits = round\_key\_size\_in\_bytes \* bits\_in\_byte;

let result = new Uint8Array(subkey\_size\_in\_bytes);

for (let i = 0; i < subkey\_size\_in\_bytes \* bits\_in\_byte; i++)

{

let index = (round\_key\_size\_in\_bits / 2 + subkey\_indexes[i] - 1) % round\_key\_size\_in\_bits;

setBit(result, i, getBit(roundKey, index));

}

return result;

}

function expansion(vec32) {

const block\_size\_in\_bits = 6;

const left\_bits = 0b00111111;

const right\_bits = 0b11111100;

let leftArr = new Uint8Array(vec32);

let rightArr = new Uint8Array(vec32);

shiftLeft(leftArr, 1);

shiftRight(rightArr, 1);

let result = new Uint8Array(subkey\_size\_in\_bytes);

for (let i = 0; i < vec32.length; i++)

{

let current\_byte = Math.floor(2 \* block\_size\_in\_bits \* i / bits\_in\_byte);

let current\_bit = 2 \* block\_size\_in\_bits \* i % bits\_in\_byte;

let left\_val = leftArr[i] & left\_bits;

let right\_val = rightArr[i] & right\_bits;

if (current\_bit === bits\_in\_tetrade) {

result[current\_byte] |= right\_val >> bits\_in\_tetrade;

result[current\_byte + 1] = (right\_val << bits\_in\_tetrade) | left\_val;

} else {

result[current\_byte] = right\_val | (left\_val >> bits\_in\_tetrade);

result[current\_byte + 1] = left\_val << bits\_in\_tetrade;

}

}

return result;

}

function shrinking(vec48) {

const block\_size\_in\_bits = 6;

let result = new Uint8Array(part\_size\_in\_bytes);

for (let i = 0; i < result.length; i++)

{

let current\_byte = Math.floor(2 \* i \* block\_size\_in\_bits / bits\_in\_byte);

let current\_bit = 2 \* i \* block\_size\_in\_bits % bits\_in\_byte;

let first\_row;

let first\_col;

let last\_row;

let last\_col;

if (current\_bit === bits\_in\_tetrade) {

first\_row = ((vec48[current\_byte] & 0b00001000) >> 2) | ((vec48[current\_byte + 1] & 0b01000000) >> 6);

first\_col = ((vec48[current\_byte] & 0b00000111) << 1) | ((vec48[current\_byte + 1] & 0b10000000) >> 7);

last\_row = ((vec48[current\_byte + 1] & 0b00100000) >> 4) | (vec48[current\_byte + 1] & 0b00000001);

last\_col = (vec48[current\_byte + 1] & 0b00011110) >> 1;

} else {

first\_row = ((vec48[current\_byte] & 0b10000000) >> 6) | ((vec48[current\_byte] & 0b00000100) >> 2);

first\_col = (vec48[current\_byte] & 0b01111000) >> 3;

last\_row = (vec48[current\_byte] & 0b00000010) | ((vec48[current\_byte + 1] & 0b00010000) >> 4);

last\_col = ((vec48[current\_byte] & 0b00000001) << 3) | (vec48[current\_byte + 1] & 0b11100000) >> 5;

}

let first\_val = S\_boxes[i \* 2][first\_row][first\_col];

let last\_val = S\_boxes[i \* 2 + 1][last\_row][last\_col];

result[i] = (first\_val << 4) | last\_val;

}

return result;

}

function pPermutation(vec32) {

let result = new Uint8Array(part\_size\_in\_bytes);

for (let i = 0; i < part\_size\_in\_bytes \* bits\_in\_byte; i++)

{

setBit(result, i, getBit(vec32, P[i] - 1));

}

return result;

}

function startingPermutation(value) {

let byte\_with\_bit\_set = 0b10000000;

let result = new Uint8Array(value\_size\_in\_bytes).fill(0);

for (let i = 0; i < value\_size\_in\_bytes; i++)

{

for (let j = 0; j < value\_size\_in\_bytes; j++)

{

result[permutation\_indexes[i]] |= ((value[j] & byte\_with\_bit\_set) << i) >> (bits\_in\_byte - 1 - j);

}

byte\_with\_bit\_set >>= 1;

}

return result;

}

function endingPermutation(value) {

let byte\_with\_bit\_set = 0b00000001;

let result = new Uint8Array(value\_size\_in\_bytes).fill(0);

for (let i = 0; i < value\_size\_in\_bytes; i++)

{

for (let j = 0; j < value\_size\_in\_bytes; j++)

{

result[i] |= ((value[permutation\_indexes[j]] & byte\_with\_bit\_set) >> i) << (bits\_in\_byte - 1 - j);

}

byte\_with\_bit\_set <<= 1;

}

return result;

}

function xor(vec1, vec2) {

let result = new Uint8Array(vec1.length);

for (let i = 0; i < result.length; i++)

{

result[i] = vec1[i] ^ vec2[i];

}

return result;

}

function feistel(vec32, subkey) {

let result = expansion(vec32);

result = xor(result, subkey);

result = shrinking(result);

return pPermutation(result);

}

function encrypt\_single(value64, subkeys) {

value64 = startingPermutation(value64);

let left\_part = value64.slice(0, Math.floor(value\_size\_in\_bytes / 2));

let right\_part = value64.slice(Math.floor(value\_size\_in\_bytes / 2), value\_size\_in\_bytes);

for (let i = 0; i < round\_count; i++)

{

let tmp = left\_part;

left\_part = right\_part;

right\_part = xor(tmp, feistel(right\_part, subkeys[i]));

}

let result = endingPermutation(new Uint8Array([ ...right\_part, ...left\_part ]));

return result;

}

function decrypt\_single(value64, subkeys) {

value64 = startingPermutation(value64);

let left\_part = value64.slice(0, Math.floor(value\_size\_in\_bytes / 2));

let right\_part = value64.slice(Math.floor(value\_size\_in\_bytes / 2), value\_size\_in\_bytes);

for (let i = round\_count - 1; i >= 0; i--)

{

let tmp = left\_part;

left\_part = right\_part;

right\_part = xor(tmp, feistel(right\_part, subkeys[i]));

}

let result = endingPermutation(new Uint8Array([ ...right\_part, ...left\_part ]));

return result;

}

function getSubKeys(key) {

let roundKey = permutedChoice(key);

let result = new Array(round\_count);

for (let i = 0; i < result.length; i++)

{

shiftRoundKey(roundKey, round\_shifts[i]);

result[i] = getSubKey(roundKey);

}

return result;

}

function pack(value) {

value = JSON.stringify(typeson.encapsulate(value));

value = new TextEncoder().encode(value);

let additional\_bytes\_count = Math.ceil((value.length + 1) / value\_size\_in\_bytes) \* value\_size\_in\_bytes - value.length - 2;

let additional\_bytes = new Uint8Array(additional\_bytes\_count).fill(0);

return new Uint8Array([ value.length / 2\*\*bits\_in\_byte, value.length, ...value, ...additional\_bytes ]);

}

function unpack(value) {

value = new Uint8Array(value.slice(2, 2 + ((value[0] \* 2\*\*bits\_in\_byte) | value[1])));

value = new TextDecoder().decode(value);

return typeson.revive(JSON.parse(value));

}

function encrypt(value, subkeys) {

if (value.length % value\_size\_in\_bytes != 0) {

throw new Error('Invalid size');

}

let result = new Uint8Array(value.length);

for (let i = 0; i < value.length; i += value\_size\_in\_bytes)

{

let block = encrypt\_single(value.slice(i, i + value\_size\_in\_bytes), subkeys);

for (let j = 0; j < value\_size\_in\_bytes; j++)

{

result[i + j] = block[j];

}

}

return result;

}

function decrypt(value, subkeys) {

if (value.length % value\_size\_in\_bytes != 0) {

throw new Error('Invalid size');

}

let result = new Uint8Array(value.length);

for (let i = 0; i < value.length; i += value\_size\_in\_bytes)

{

let block = decrypt\_single(value.slice(i, i + value\_size\_in\_bytes), subkeys);

for (let j = 0; j < value\_size\_in\_bytes; j++)

{

result[i + j] = block[j];

}

}

return result;

}

module.exports = { getSubKeys, decrypt, encrypt, pack, unpack, typeson };