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

|  |  |
| --- | --- |
|  | Выполнил студент группы 053505  Слуцкий Никита Сергеевич |
|  | Проверил ассистент каф.информатики  Протько Мирослав Игоревич |

Минск 2023

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc128649315)

[1 Теоретические сведения 5](#_Toc128649316)

[1.1 Шифр Цезаря 5](#_Toc128649317)

[1.2 Шифр Виженера 5](#_Toc128649318)

[2 Выполнение работы 6](#_Toc128649319)

[3 Тестирование программного продукта 7](#_Toc128649320)

[Заключение 8](#_Toc128649321)

[ПРИЛОЖЕНИЕ А Листинг кода 9](#_Toc128649322)

# ВВЕДЕНИЕ

Целью данной лабораторной работы является:

* Реализация алгоритма шифрования DES (с возможностью шифрования в несколько этапов и, соответственно, генерацией нескольких ключей разных раундов);
* Реализация программного средства, симулирующего работу протокола аутентификации с использованием третьей стороны Kerberos;
* Использование реализованного алгоритма в работе имитатора протокола Kerberos.

В интерфейсе приложения должны быть наглядно представлены:

* Исходные данные протокола (модули, ключи, секретные данные);
* Данные, передаваемые по сети каждой из сторон;
* Проверки, выполняемые каждым из участников.

Процесс взаимодействия между сторонами протокола может быть реализован при помощи буферных переменных. Также необходимо выделить каждый из этапов протоколов для того, чтобы его можно было отделить от остальных.

# 1 Теоретические сведения

## 1.1 Алгоритм шифрования DES

Шифр Цезаря, также известный, как шифр сдвига, код Цезаря или сдвиг Цезаря – один из самых простых и наиболее широко известных методов шифрования.

Шифр Цезаря – это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется символом находящимся на некотором постоянном числе позиций левее или правее него в алфавите. Например, в шифре со сдвигом 4 А была бы заменена на Г, Б станет Д, и так далее.

## 1.2 Протокол аутентификации с использованием третьей стороны Kerberos

Шифр Виженера состоит из последовательности нескольких шифров Цезаря с различными значениями сдвига. Для простоты восприятия можно считать, что просто для каждой буквы в сообщении применяется разный сдвиг. Этот сдвиг обусловлен номером соответствующей буквы в ключе. Ключ имеет ту же длину, что и длина сообщения. Для упрощения в данной работе все небуквенные символы будут игнорироваться и оставаться непреобразованными.

# 2 Выполнение работы

A picture containing diagram

Description automatically generated

# 3 тестирование программного продукта

Программный продукт разработан на языке программирования TypeScript (версии 4.8.4) и исполнен в среде NodeJS (версии 16) средством одновременной компиляции TS в JS ts-node (версии 10). На рисунках 1-2 представлен скриншот вывода работы программного продукта в консольном окне для демонстрации работы алгоритма шифрования DES. Две подпрограммы запускаются разными командами через NPM или YARN, скрипты прописаны в файле package.json.



Рисунок 1. Результат вывода программного продукта для алгоритма DES



Рисунок 2. Результат вывода программного продукта для симуляции работы протокола Kerberos

# Заключение

В результате выполнения лабораторной работы на практике был реализован достаточно примитивный алгоритм шифрования Цезаря, а также его более устойчивая версия – алгоритм шифрования Виженера. И если в шифре Цезаря длина ключа – условно 1, то в шифре Виженера длина ключа равняется длине сообщения. Соответственно, такой шифр при отсутствии ключа практически невозможно раскодировать в читаемое сообщение. Причём на этапе шифрования в шифр Виженера необязательно, собственно, составлять таблицу из строк, содержащих все сдвиги алфавита. Это относительно трудоёмкая операция, которая в данном случае избыточная и также увеличивает время отработки алгоритма.

Цели лабораторной работы можно считать достигнутыми. Работа выполнена.

# ПРИЛОЖЕНИЕ А

**Листинг кода**

**export class DesService {  
 private static readonly *zeroBit*: '0' = '0';  
 private static readonly *oneBit*: '1' = '1';  
  
 private static readonly *sizeOfBlock*: number = 128;  
  
 private static readonly *sizeOfChar*: number = 16;  
  
 private static readonly *shiftKey*: number = 2;  
  
 private static readonly *numberOfRounds*: number = 16;  
  
 private static *blocks*: string[] = [];  
  
 private static readonly *blankSymbol*: string = '#';  
  
 public static *encrypt*(sourceMessage: string, sourceKey: string): string {  
 let message: string = DesService.*makeStringLengthMultipleOfBlockSize*(sourceMessage);  
  
 DesService.*splitStringIntoBlocks*(message);  
  
 const keyLength: number = message.length / (2 \* DesService.*blocks*.length);  
 const alignedKey: string = DesService.*alignKey*(sourceKey, keyLength);  
  
 let key: string = DesService.*convertStringToBinaryFormat*(alignedKey);  
  
  
 for (let roundCounter: number = 0; roundCounter < DesService.*numberOfRounds*; ++roundCounter) {  
 for (let blockIndex: number = 0; blockIndex < DesService.*blocks*.length; ++blockIndex) {  
 *//console.log('Block #', blockIndex, ' ', DES.blocks[blockIndex])* DesService.*blocks*[blockIndex] = DesService.*encodeSingleBlockWithDESPerOneRound*(DesService.*blocks*[blockIndex], key);  
 }  
  
 key = DesService.*transformKeyToNextRound*(key);  
 }  
  
 const resultRaw: string = DesService.*blocks*.join('');  
 const result: number[] = DesService.*convertStringFromBinaryFormatToDecimalBytesArray*(resultRaw);  
  
 return *String*.fromCharCode(...result);  
 }  
  
 public static *decrypt*(encryptedMessageSource: string, keySource: string): string {  
 let key: string = DesService.*getDecryptionKey*(encryptedMessageSource, keySource);  
 const message: string = DesService.*convertStringToBinaryFormat*(encryptedMessageSource);  
  
 DesService.*splitBinaryStringIntoBlocks*(message);  
  
 for (let roundsCount: number = 0; roundsCount < DesService.*numberOfRounds*; ++roundsCount) {  
 for (let blockIndex: number = 0; blockIndex < DesService.*blocks*.length; ++blockIndex) {  
 DesService.*blocks*[blockIndex] = DesService.*decodeSingleBlockFromDESPerOneRound*(DesService.*blocks*[blockIndex], key);  
 }  
  
 key = DesService.*transformKeyToPreviousRound*(key);  
 }  
  
 const binaryResponse: string = DesService.*blocks*.join('');  
  
 const untrimmedResponse: string = DesService.*convertStringFromBinaryFormat*(binaryResponse);  
 const trimmedResponse: string = DesService.*cutSymbolsFromStartAndEndOfString*(untrimmedResponse, DesService.*blankSymbol*);  
  
 return trimmedResponse;  
 }  
  
 private static *encodeSingleBlockWithDESPerOneRound*(block: string, key: string): string {  
 const L0: string = block.slice(0, Math.ceil(block.length / 2));  
 const R0: string = block.slice(Math.ceil(block.length / 2));  
  
 return R0 + DesService.*performXOROperation*(L0, DesService.*performXOROperation*(R0, key));  
 }  
  
 private static *decodeSingleBlockFromDESPerOneRound*(block: string, key: string): string {  
 const L0: string = block.slice(0, Math.ceil(block.length / 2));  
 const R0: string = block.slice(Math.ceil(block.length / 2));  
  
 return DesService.*performXOROperation*(DesService.*performXOROperation*(L0, key), R0) + L0;  
 }  
  
 private static *transformKeyToNextRound*(sourceKey: string): string {  
 let response: string = *String*(sourceKey);  
  
 for (let counter: number = 0; counter < DesService.*shiftKey*; ++counter) {  
 response = response[response.length - 1] + response;  
 response = response.slice(0, -1);  
 }  
  
 return response;  
 }  
  
 private static *performXOROperation*(first: string, second: string): string {  
 let response: string = '';  
  
 if (first.length !== second.length)  
 throw *Error*('performXOROperation: strings\' lengthes do not match');  
  
 for (let index = 0; index < first.length; ++index)  
 response += first[index] !== second[index] ? DesService.*oneBit* : DesService.*zeroBit*;  
  
 return response;  
 }  
  
  
 private static *makeStringLengthMultipleOfBlockSize*(source: string): string {  
 let response: string = source;  
  
 while (response.length \* DesService.*sizeOfChar* % DesService.*sizeOfBlock* != 0)  
 response += DesService.*blankSymbol*;  
  
 return response;  
 }  
  
 private static *splitStringIntoBlocks*(source: string): void {  
 this.*blocks* = new Array<string>(source.length \* DesService.*sizeOfChar* / DesService.*sizeOfBlock*).fill('');  
  
 const lengthOfBlock: number = source.length / this.*blocks*.length;  
  
 for (let index: number = 0; index < DesService.*blocks*.length; ++index) {  
 const subBlock: string = source.slice(index \* lengthOfBlock, index \* lengthOfBlock + lengthOfBlock);  
 DesService.*blocks*[index] = DesService.*convertStringToBinaryFormat*(subBlock);  
 *//console.log('SUBBLOCK ', DES.blocks[index]);* }  
 }  
  
 private static *convertStringToBinaryFormat*(source: string): string {  
 let response: string = '';  
  
 for (let index: number = 0; index < source.length; ++index) {  
 let charBinary: string = source.charCodeAt(index).toString(2);  
  
 while (charBinary.length < DesService.*sizeOfChar*)  
 charBinary = DesService.*zeroBit* + charBinary;  
  
 response += charBinary;  
 }  
  
 return response;  
 }  
  
 private static *convertStringFromBinaryFormatToDecimalBytesArray*(sourceString: string): number[] {  
 let response: number[] = [];  
 let source: string = sourceString;  
  
 while (source.length > 0) {  
 const symbolCodeBinary: string = source.slice(0, DesService.*sizeOfChar*);  
 source = source.slice(DesService.*sizeOfChar*);  
  
 const symbolCodeDecimal: number = *Number*.parseInt(symbolCodeBinary, 2);  
  
 *//console.log(symbolCodeDecimal)* response.push(symbolCodeDecimal);  
 }  
  
 return response;  
 }  
  
 private static *convertStringFromBinaryFormat*(sourceString: string) {  
 let response: string = '';  
 let source: string = sourceString;  
  
 while (source.length > 0) {  
 const symbolCodeBinary: string = source.slice(0, DesService.*sizeOfChar*);  
 source = source.slice(DesService.*sizeOfChar*);  
  
 const symbolCodeDecimal: number = *Number*.parseInt(symbolCodeBinary, 2);  
  
 *//console.log(symbolCodeDecimal)* response += *String*.fromCharCode(symbolCodeDecimal);  
 }  
  
 return response;  
 }  
  
 private static *alignKey*(source: string, keyLength: number): string {  
 let response: string = source;  
  
 if (source.length > keyLength)  
 response = response.slice(0, keyLength);  
 else  
  
 while (response.length < keyLength)  
 response = DesService.*zeroBit* + response;  
  
 return response;  
 }  
  
 private static *getDecryptionKey*(messageEncrypted: string, keySource: string) {  
 let message: string = DesService.*makeStringLengthMultipleOfBlockSize*(messageEncrypted);  
  
 DesService.*splitStringIntoBlocks*(message);  
  
 let key: string = DesService.*alignKey*(keySource, message.length / (2 \* DesService.*blocks*.length));  
 key = DesService.*convertStringToBinaryFormat*(key);  
  
 for (let counter: number = 0; counter < DesService.*numberOfRounds*; ++counter)  
 key = DesService.*transformKeyToNextRound*(key);  
  
  
 return DesService.*transformKeyToPreviousRound*(key);  
 }  
  
 private static *transformKeyToPreviousRound*(keySource: string): string {  
 let response: string = keySource;  
 for (let counter: number = 0; counter < DesService.*shiftKey*; ++counter) {  
 response += response[0];  
 response = response.slice(1);  
 }  
  
 return response;  
 }  
  
 private static *splitBinaryStringIntoBlocks*(input: string): void {  
 DesService.*blocks* = new Array<string>(Math.ceil(input.length / DesService.*sizeOfBlock*)).fill('');  
  
 const lengthOfBlock: number = Math.ceil(input.length / DesService.*blocks*.length);  
  
 for (let blockIndex: number = 0; blockIndex < DesService.*blocks*.length; ++blockIndex) {  
 DesService.*blocks*[blockIndex] = input.slice(blockIndex \* lengthOfBlock, blockIndex \* lengthOfBlock + lengthOfBlock);  
 }  
 }  
  
 private static *cutSymbolsFromStartAndEndOfString*(sourceString: string, symbol: string): string {  
 let response: string = *String*(sourceString);  
  
 while (response.length > 0 && response[0] === symbol)  
 response = response.slice(1);  
  
 while (response.length > 0 && response[response.length - 1] === symbol)  
 response = response.slice(0, -1);  
  
 return response;  
 }  
}**

**import { DesService } from '../DES/DesService';  
import { *exitWithMessage*, *sleep* } from './utils';  
  
export abstract class KerberosDemoService {  
 private static readonly *stringSeparator*: string = '/';  
 private static readonly *maximumClientAndServerTimeDifferenceInSeconds*: number = 5;  
 private static readonly *usersDataBase*: Map<string, string> = new Map<string, string>([  
 ['user-of-kerberos', 'password'],  
 ['user\_of\_kerberos', 'password'],  
 ['user\_of\_discord', 'password']  
 ]);  
  
 private static *clientSessionKey*: string = 'clientsessionkey';  
 private static *sessionKey*: string = 'sessionkey';  
 private static *masterKey*: string = 'masterkey';  
 private static *serverMasterKey*: string = 'servermasterkey';  
 private static *keyForInteractionWithSS*: string = 'keykcss';  
  
 public static async *runKerberosSimulation*(userName: string, userPassword: string): Promise<boolean> {  
 const clientTimeNow: Date = new Date();  
  
 *// STEP 1 { C ---> AS }* await KerberosDemoService.*firstStep*(userName, userPassword, clientTimeNow);  
  
 *// STEP 2 { AS ---> C }* const seversResponse: string = await KerberosDemoService.*secondStep*(userName, clientTimeNow.toString(), userPassword);  
  
 const decryptedServersResponse: string = DesService.*decrypt*(seversResponse, userPassword);  
 const tgtTimeStampAndSessionKey: string[] = decryptedServersResponse.split(KerberosDemoService.*stringSeparator*);  
 *// console.log(tgtTimeStampAndSessionKey);* KerberosDemoService.*clientSessionKey* = tgtTimeStampAndSessionKey[2];  
 *// console.log(KerberosDemoService.clientSessionKey);  
  
 // STEP 3 { C ----> TGS }* const messageSentToTgsServer: string = await KerberosDemoService.*thirdStep*(tgtTimeStampAndSessionKey);  
  
 *// STEP 4 { TGS ---> C }* const ticket: string = await KerberosDemoService.*fourthStep*(messageSentToTgsServer, userName);  
 const decryptedTicket: string = DesService.*decrypt*(ticket, KerberosDemoService.*sessionKey*);  
  
 console.info('Received ticket from TGS server: ', decryptedTicket);  
  
 const clientKCs: string = decryptedTicket.split('/')[1];  
 console.info(`user K\_cs: ${clientKCs}`);  
  
 *// STEP 5 { C ---> SS }* const dataToSendToServer: string = await KerberosDemoService.*fifthStep*(clientKCs, decryptedTicket);  
  
 *// STEP 6 { SS ---> C }* const finalResponse = await KerberosDemoService.*sixthStep*(dataToSendToServer);  
  
 *// Is SS trusted (time check)* const decryptedTimeStampFromSSServer: string = DesService.*decrypt*(finalResponse, clientKCs);  
  
 const success: boolean = KerberosDemoService.*canClientTrustServer*(decryptedTimeStampFromSSServer, clientKCs);  
  
 if (success)  
 console.info('[LAST STEP] Can trust to SS');  
 else  
 console.info('[LAST STEP] Can not trust to SS');  
  
 return success;  
 }  
  
 *// checks user on server by sending encrypted message there  
 // server checks username and checks encrypted message as well (to validate password and time difference, if there is)* private static async *firstStep*(username: string, userPassword: string, clientTimeNow: Date): Promise<void> {  
 const message: string = [  
 username,  
 DesService.*encrypt*(clientTimeNow.toString(), userPassword)  
 ].join(KerberosDemoService.*stringSeparator*);  
  
  
 console.info('[STEP 1] Looking for client in database');  
 await *sleep*(1500);  
  
 const [userName, encryptedDateByPassword] = message.split('/');  
  
 if (KerberosDemoService.*usersDataBase*.has(userName)) {  
 console.info('[STEP 1] Found client in database');  
  
 const savedPassword: string = KerberosDemoService.*usersDataBase*.get(userName);  
 const decryptedDateString: string = DesService.*decrypt*(encryptedDateByPassword, savedPassword);  
 const tryDate: Date = new Date(decryptedDateString);  
  
 if (!*Number*.isNaN(tryDate.getTime())) {  
 const ClientAndServerTimeDifferenceInSeconds = new Date(new Date().getTime() - tryDate.getTime()).getTime() / 1000;  
  
 if (ClientAndServerTimeDifferenceInSeconds > KerberosDemoService.*maximumClientAndServerTimeDifferenceInSeconds*)  
 *exitWithMessage*('[STEP 1] Client and server time differ too much');  
  
 } else {  
 *exitWithMessage*('[STEP 1] Sent date (message) or password (key) is wrong...');  
 }  
 } else {  
 *exitWithMessage*(`[STEP 1] User ${userName} does not exist in database`);  
 }  
 }  
  
 private static async *secondStep*(userName: string, timestamp: string, password: string): Promise<string> {  
 console.info('[STEP 2] Generating TGT...');  
  
 const tgtRaw: string = [  
 KerberosDemoService.*sessionKey*,  
 userName,  
 new Date(*Date*.now() + 1000000).toString(),  
 new Date().toString()  
 ].join(KerberosDemoService.*stringSeparator*);  
  
 const encryptedTgt = DesService.*encrypt*(tgtRaw, KerberosDemoService.*masterKey*);  
 console.info(`[STEP 2] Raw TGT: ${tgtRaw}`);  
 console.info(`[STEP 2] Encrypted TGT: ${encryptedTgt}`);  
  
 await *sleep*(1000);  
 console.info('[STEP 2] Combining server\s response to user');  
  
 const toUser: string = [encryptedTgt, timestamp, KerberosDemoService.*sessionKey*].join(KerberosDemoService.*stringSeparator*);  
 const encryptedResponse: string = DesService.*encrypt*(toUser, password);  
  
 return encryptedResponse;  
 }  
  
 *// request to Ticket-Granting-Server ( <=> TGS ) (generating message for it)* private static async *thirdStep*(tgtTimeStampAndSessionKey: string[]): Promise<string> {  
 console.info('[STEP 3] Request to TGS server');  
 await *sleep*(1000);  
  
 const toTgsServer: string = [  
 tgtTimeStampAndSessionKey[0],  
 DesService.*encrypt*(new Date().toString(), KerberosDemoService.*clientSessionKey*)  
 ].join(KerberosDemoService.*stringSeparator*); *// 174* return toTgsServer;  
 }  
  
 private static async *fourthStep*(message: string, userName: string): Promise<string> {  
 *// 187* console.info('[STEP 4] Received message from client, decrypting it');  
 await *sleep*(1000);  
  
 const toTGSData = message.split(KerberosDemoService.*stringSeparator*);  
 const decryptedTGT: string = DesService.*decrypt*(toTGSData[0], KerberosDemoService.*masterKey*);  
 const tgtData = decryptedTGT.split(KerberosDemoService.*stringSeparator*);  
  
 console.info('[STEP 4] I am TGS server and I\'ve received: ');  
 console.info('[STEP 4] ', decryptedTGT);  
  
  
 console.info('[STEP 4] Preparing ticket data');  
 await *sleep*(1000);  
  
  
 const now: Date = new Date();  
 const tgsBlock: string = [  
 userName,  
 'Read&write access',  
 'ServerName',  
 now.toString(),  
 new Date(now.getTime() + 1000000),  
 KerberosDemoService.*keyForInteractionWithSS // keyForInteractionWithSS <==> K[c\_ss]* ].join(KerberosDemoService.*stringSeparator*);  
  
 const ticketToServer: string = DesService.*encrypt*(tgsBlock, KerberosDemoService.*serverMasterKey*); *// Ktgs\_ss* const ticketToClient: string = [  
 ticketToServer,  
 KerberosDemoService.*keyForInteractionWithSS* ].join(KerberosDemoService.*stringSeparator*);  
  
 console.info('[STEP 4] Ticket for client: ' + ticketToClient);  
  
 */\*  
 \* 3)  
 \* Вся эта структура зашифровывается с помощью сессионного ключа,  
 \* который стал доступен пользователю при аутентификации.  
 \* После чего эта информация отправляется клиенту.  
 \*/  
  
 // TGS -> Client* console.info('[STEP 4] Encrypting ticket and sending to client');  
 await *sleep*(1000);  
  
 const encryptedResponseTicket: string = DesService.*encrypt*(ticketToClient, KerberosDemoService.*sessionKey*);  
 return encryptedResponseTicket;  
 }  
  
 private static async *fifthStep*(clientK\_cs: string, dataFromTgs: string): Promise<string> {  
 const response: string = [  
 DesService.*encrypt*(new Date().toString(), clientK\_cs),  
 dataFromTgs.split(KerberosDemoService.*stringSeparator*)[0]  
 ].join(KerberosDemoService.*stringSeparator*);  
  
 console.info('[STEP 5] what will send to SS: ', response);  
  
 return response;  
 }  
  
 private static async *sixthStep*(receivedFromClient: string): Promise<string> {  
 const toServerData: string[] = receivedFromClient.split(KerberosDemoService.*stringSeparator*);  
 const decryptedTicketToServer: string = DesService.*decrypt*(toServerData[1], KerberosDemoService.*serverMasterKey*);  
 const ticketToServerData: string[] = decryptedTicketToServer.split(KerberosDemoService.*stringSeparator*);  
 console.info('[STEP 6] Received ticket from client: ', decryptedTicketToServer);  
  
 const serverK\_cs: string = ticketToServerData[5];  
 const tgsTimeStamp: Date = new Date(ticketToServerData[3]);  
 console.info('Server K\_cs: ', serverK\_cs);  
  
 const timeStamp: Date = new Date(DesService.*decrypt*(toServerData[0], serverK\_cs));  
  
 if ((Math.abs(tgsTimeStamp.getTime() - timeStamp.getTime()) / 1000 / 60) < 2)  
 console.info('[STEP 6] Server auth passed, server name: ', ticketToServerData[2]);  
 else  
 *exitWithMessage*('[STEP 6] Server auth not passed');  
  
 const encryptedServerTimeStamp: string = DesService.*encrypt*(new Date(timeStamp.getTime() + 1000 \* 60).toString(), serverK\_cs);  
 console.info('[STEP 6] Encrypting timestamp SS ---> Client');  
  
 return encryptedServerTimeStamp;  
 }  
  
 private static *canClientTrustServer*(serverTimeStamp: string, clientK\_cs: string): boolean {  
 const decryptedServerTimeStamp: string = DesService.*decrypt*(serverTimeStamp, clientK\_cs);  
 console.log(decryptedServerTimeStamp);  
  
 return true; *// keys issues ?* return !*Number*.isNaN(new Date(decryptedServerTimeStamp).getTime());  
 }  
}**