Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №2

“Идентификация и аутентификация пользователей. Протокол Kerberos.”

Выполнил:

студент гр. 853504

Кузьма В. В.

Проверил:

Протько М.И.

Минск 2021

**Содержание**

[**1.** **Цель работы** 3](#_Toc65156180)

[**2. Краткие теоретические сведения** 4](#_Toc65156181)

[**3. Реализация задачи** 7](#_Toc65156182)

[**Приложение. Текст программ** 8](#_Toc65156183)

# 

# **Цель работы**

* Изучить теоретические сведения.
* Создать приложение, реализующее протокол распределения ключей **Kerberos**, включая процедуру, релаизующую Алгоритм **DES**.

В интерфейсе приложения должны быть наглядно представлены:

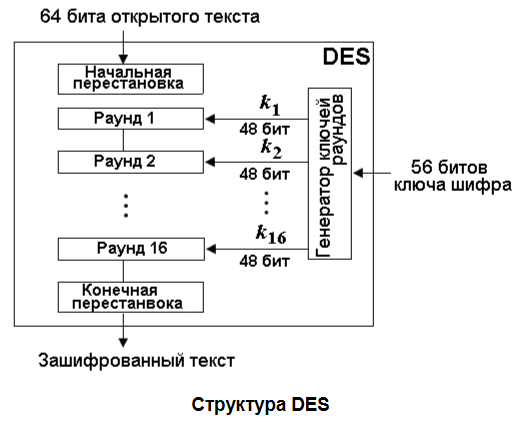
1. Исходные данные протокола (модули, ключи, секретные данные и т.п.);
2. Данные, передаваемые по сети каждой из сторон;
3. Проверки, выполняемые каждым из участников.

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.

# **2. Краткие теоретические сведения**

**Стандарт шифрования данных** ***DES*** (DATA ENCRYPTION STANDARD) – блочный шифр с симметричными ключами, разработан Национальным Институтом Стандартов и Технологии (NIST –National Institute of Standards and Technology).

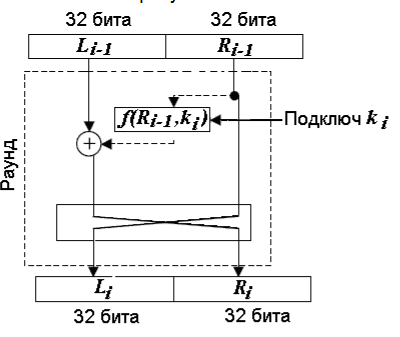
Для шифрования **DES** принимает 64-битовый открытыйтекст и порождает 64-битовый зашифрованный текст и наоборот, получив 64 бита зашифрованного текста, он выдает 64 бита расшифрованного. В обоих случаях для шифрования и дешифрования применяется один и тот же 56-битовый ключ.



Процесс шифрования состоит из двух перестановок, которые называют начальнойи финальной (конечной) перестановками, и 16 раундов **Фейстеля**. Каждый раунд использует различные сгенерированные 48-битовые ключи.

На вход каждой из нихпоступает 64 бита, которые затем переставляются в соответствии с заданными таблицами. Эти перестановки взаимно обратны. Другими словами, 58-й бит на входе начальной перестановке переходит в 1-ую позицию на выходе из нее. А финальная перестановка 1-ый входной бит переведет в 58-ую позицию на выходе.

**DES** использует 16 раундов. Каждый раунд DES применяет шифр Фейстеля, как это показано на рисунке

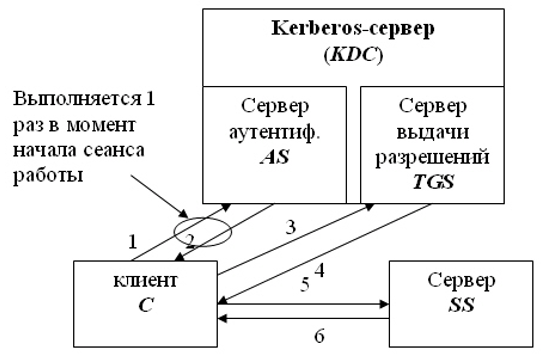


После этого для полученного 48-битного значения выполняется операция XOR с 48-битным подключом Ki. Затем полученное 48-битное значение подается на вход функции подстановки, результатом которой является 32-битное значение.

Подстановка состоит из восьми S-boxes, каждый из которых на входе получает 6 бит, а на выходе создает 4 бита. Эти преобразования определяются специальными таблицами. Первый и последний биты входного значения S-box определяют номер строки в таблице, средние 4 бита определяют номер столбца. Пересечение строки и столбца определяет 4-битный выход. Например, если входом является 011011, то номер строки равен 01 (строка 1) и номер столбца равен 1101 (столбец 13). Значение в строке 1 и столбце 13 равно 5, т.е. выходом является 0101.

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

Протокол **Kerberos**, достаточно гибкий и имеющий возможности тонкой настройки под конкретные применения, существует в нескольких версиях. Мы рассмотрим упрощенный механизм аутентификации, реализованный с помощью протокола Kerberos.



Чтобы запустить собственно процедуру аутентификации, необходимо обратиться к специальному серверу аутентификации с запросом, содержащим логин пользователя. Если сервер не находит автора запроса в своей базе данных, запрос отклоняется.

Если все шаги выполнены правильно и все проверки прошли успешно, то стороны взаимодействия **C** и **SS**, во-первых, удостоверились в подлинности друг друга, а во-вторых, получили ключ шифрования для защиты сеанса связи - ключ **KC**\_**SS**.

Нужно отметить, что в процессе сеанса работы клиент проходит шаги 1) и 2) только один раз. Когда нужно получить билет на доступ к другому серверу (назовем его **SS1** ), клиент **С** обращается к серверу выдачи разрешений **TGS** с уже имеющимся у него билетом, т.е. протокол выполняется начиная с шага 3).

В алгоритме Kerberos могут применяться различные алгоритмы блочного симметричного шифрования. Для целей настоящей работы будем использовать алгоритм **DES**.

## **3. Реализация задачи**

Программу реализовал на языке программирования **Python** в **VSC**. Результат программы:



Рис 1. Результат работы программы (**Клиент** | **Сервер KDC** | **Сервер SS**).

# 

# **Приложение. Текст программ**

***Библиотека DES:***

import textwrap

import binascii

import datetime

# Create 16 Subkeys, each of which is 48-bits long

def get\_keys48(key64: str) -> list:

key56 = get\_pc\_1\_permutation(key64)

left28 = [key56[: len(key56) // 2]]

right28 = [key56[len(key56) // 2 :]]

left\_shifts = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

for i in range( len(left\_shifts) ):

l\_key28, r\_key28 = left28[i], right28[i]

for q in range(left\_shifts[i]):

l\_key28, r\_key28 = l\_key28[1:] + l\_key28[0], r\_key28[1:] + r\_key28[0]

left28.append(l\_key28)

right28.append(r\_key28)

keys48 = list()

for i in range( len(left28) ):

keys48.append( get\_pc\_2\_permutation(left28[i] + right28[i]) )

return keys48[1:17]

# Encode each 64-bit block of data

def encode\_text64(text64: str, keys48: list) -> str:

text64\_ip = get\_ip\_permutation(text64)

left32\_ip = [text64\_ip[: len(text64\_ip) // 2]]

right32\_ip = [text64\_ip[len(text64\_ip) // 2 :]]

for i in range(1, 17):

left32\_ip.append(right32\_ip[-1])

right48\_e = get\_e\_permutation(right32\_ip[-1])

r\_key48 = get\_xor(right48\_e, keys48[i-1])

r\_key32 = str()

b\_keys6 = textwrap.wrap(r\_key48, 6)

for q in range( len(b\_keys6) ):

s\_row = int(str.encode(b\_keys6[q][0] + b\_keys6[q][-1]), base=2)

s\_col = int(str.encode(b\_keys6[q][1:5]), base=2)

r\_key32 += get\_s\_convertation(q, s\_row, s\_col)

p\_key32 = get\_p\_permutation(r\_key32)

right32\_ip.append( get\_xor(left32\_ip[i-1], p\_key32) )

return bit\_to\_hex(get\_ip\_reverse\_permutation(right32\_ip[-1] + left32\_ip[-1]))

# Usefull Functions

def get\_permutation(key: str, table: list) -> str:

return ''.join( [key[table[i] - 1] for i in range( len(table) )] )

def get\_xor(key\_a: str, key\_b: str) -> str:

return ''.join( [(str(int(key\_a[i]) + int(key\_b[i]))) for i in range ( len(key\_a) )] ).replace('2','0')

def bit\_to\_hex(key: str) -> str:

convert\_dict = { '0000' : '0', '0001' : '1', '0010' : '2', '0011' : '3', '0100' : '4',

'0101' : '5', '0110' : '6', '0111' : '7', '1000' : '8', '1001' : '9',

'1010' : 'a', '1011' : 'b', '1100' : 'c', '1101' : 'd', '1110' : 'e', '1111' : 'f'}

return ''.join([convert\_dict[element] for element in textwrap.wrap(key, 4)])

def hex\_to\_bit(key: str) -> str:

convert\_dict = { '0' : '0000', '1' : '0001', '2' : '0010', '3' : '0011', '4' : '0100',

'5' : '0101', '6' : '0110', '7' : '0111', '8' : '1000', '9' : '1001',

'a' : '1010', 'b' : '1011', 'c' : '1100', 'd' : '1101', 'e' : '1110', 'f' : '1111'}

return ''.join([convert\_dict[element] for element in key])

def get\_hex\_key(text: str) -> str:

text = textwrap.wrap(text.encode().hex(), 16)

if len(text[0]) != 16:

text[0] = '0' \* (16 - len(text[0])) + text[0]

return text[0]

def time\_compare(str\_date\_a: str, str\_date\_b: str, str\_period: str) -> bool:

date\_a = datetime.datetime.strptime(str\_date\_a, '%Y-%m-%d %H:%M:%S.%f')

date\_b = datetime.datetime.strptime(str\_date\_b, '%Y-%m-%d %H:%M:%S.%f')

time\_period = datetime.timedelta(minutes=int(str\_period))

return date\_a + time\_period >= date\_b

# Const permutation tables

def get\_pc\_1\_permutation(key: str) -> str:

permut\_table = [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 ]

return get\_permutation(key, permut\_table)

def get\_pc\_2\_permutation(key: str) -> str:

permut\_table = [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 ]

return get\_permutation(key, permut\_table)

def get\_ip\_permutation(key: str) -> str:

permut\_table = [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]

return get\_permutation(key, permut\_table)

def get\_e\_permutation(key: str) -> str:

permut\_table = [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 ]

return get\_permutation(key, permut\_table)

def get\_s\_permutation(key: str) -> str:

permut\_table = [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 ]

return get\_permutation(key, permut\_table)

def get\_ip\_reverse\_permutation(key: str) -> str:

permut\_table = [ 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 ]

return get\_permutation(key, permut\_table)

def get\_s\_convertation(iteration: int, row: int, col: int) -> str:

permut\_table = [[[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]]]

convert\_dict = { 0 : '0000', 1 : '0001', 2 : '0010', 3 : '0011', 4 : '0100', 5 : '0101',

6 : '0110', 7 : '0111', 8 : '1000', 9 : '1001', 10 : '1010', 11 : '1011',

12 : '1100', 13 : '1101', 14 : '1110', 15 : '1111'}

return convert\_dict[permut\_table[iteration][row][col]]

def get\_p\_permutation(key: str) -> str:

permut\_table = [ 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 ]

return get\_permutation(key, permut\_table)

# MAIN ENCODE/DECODE

def des\_encode(text: str, key: str, is\_hex\_hey: bool):

if not is\_hex\_hey:

key = get\_hex\_key(key)

keys48 = get\_keys48( hex\_to\_bit(key) )

text = textwrap.wrap(text.encode().hex(), 16)

if len(text[-1]) != 16:

text[-1] = '0' \* (16 - len(text[-1])) + text[-1]

hex\_list = list()

for hex\_element in text:

hex\_list.append( encode\_text64(hex\_to\_bit(hex\_element), keys48) )

return ''.join(hex\_list)

def des\_decode(text: str, key: str, is\_hex\_hey: bool):

if not is\_hex\_hey:

key = get\_hex\_key(key)

keys48 = get\_keys48( hex\_to\_bit(key) )[::-1]

text = textwrap.wrap(text, 16)

hex\_list = list()

for hex\_element in text:

hex\_list.append( encode\_text64(hex\_to\_bit(hex\_element), keys48) )

text\_decode = ''

for hex\_element in hex\_list:

hex\_str = textwrap.wrap(hex\_element, 2)

for hex\_char in hex\_str:

text\_decode += binascii.unhexlify(hex\_char.encode()).decode().rstrip('\x00')

return text\_decode

***Клиент:***

import socket

import uuid

from des import des\_encode, des\_decode, get\_hex\_key

from datetime import datetime

client\_addr = ('127.0.0.1', 20000)

kdc\_addr = ('127.0.0.1', 20001)

ss\_addr = ('127.0.0.1', 20002)

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sock.bind(client\_addr)

if \_\_name\_\_ == "\_\_main\_\_":

print('\n-= 1 =-')

login = str('1|' + input('Введите ваш логин: ') )

print('C -> AS:', login)

sock.sendto(login.encode(), kdc\_addr)

print('\n-= 2 =-')

res\_encode = sock.recvfrom(1024)[0].decode()

print('AS -> C:', res\_encode)

print('\n-= 3 =-')

password = input('Введите ваш пароль: ')

res = des\_decode(res\_encode, password, False)

TGT\_encode, K\_c\_tgs = res.split('|')

AUT1 = login + '|' + str( datetime.now() )

AUT1\_encode = des\_encode(AUT1, K\_c\_tgs, True)

res = '3|' + TGT\_encode + '|' + AUT1\_encode

print('C -> TGS:', res)

sock.sendto(res.encode(), kdc\_addr)

print('\n-= 4 =-')

res\_encode = sock.recvfrom(1024)[0].decode()

print('TGS -> C:', res\_encode)

print('\n-= 5 =-')

res = des\_decode(res\_encode, K\_c\_tgs, True)

TGS\_encode, K\_c\_ss = res.split('|')

AUT2 = login + '|' + str( datetime.now() )

AUT2\_encode = des\_encode(AUT2, K\_c\_ss, True)

res = '5|' + TGS\_encode + '|' + AUT2\_encode

print('C -> SS:', res)

sock.sendto(res.encode(), ss\_addr)

print('\n-= 6 =-')

res\_encode = sock.recvfrom(1024)[0].decode()

print('SS -> C:', res\_encode)

res = des\_decode(res\_encode, K\_c\_ss, True)

if AUT2.split('|')[2] == res[:-1]:

print('\*Подлинность SS подтверждена!\*')

print('Ключ для защиты сеанса связи Kc\_ss:', K\_c\_ss)

else:

print('\*Истекло время ключа или неправильный идентификатор\*')

print()

sock.close()

***Сервер KDC:***

import socket

import uuid

import datetime

from des import des\_encode, des\_decode, get\_hex\_key, time\_compare

client\_addr = ('127.0.0.1', 20000)

kdc\_addr = ('127.0.0.1', 20001)

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sock.bind(kdc\_addr)

# AS data

tgs = uuid.uuid4()

K\_as\_tgs = uuid.uuid4()

p1 = 10

# c: [K\_c, K\_c\_tgs]

db = {'vlad': ['70617373776f7264', get\_hex\_key( str(uuid.uuid4()) )] }

# TGS data

ss = '2323d4e8-4b32-44b2-b035-d0a6bf24daef'

K\_tgs\_ss = '3030353861396231'

p2 = 10

# c: Kc\_ss

db\_ss = {'vlad': '6231353064313266'}

if \_\_name\_\_ == "\_\_main\_\_":

while True:

try:

data = sock.recvfrom(1024)[0].decode()

if data[:2] == '1|':

print('\n-= 2 =-')

login = data[2:]

TGT = login + '|' + str(tgs) + '|' + str( datetime.datetime.now() ) + '|' + str(p1) + '|' + db[login][1]

print(' TGT(d):', TGT)

TGT\_encode = des\_encode(TGT, str(K\_as\_tgs), False)

print(' TGT(e):', TGT\_encode, '\n')

res = TGT\_encode + '|' + db[login][1]

print(' res(d):', res)

res\_encode = des\_encode(res, db[login][0], True)

print(' res(e):', res\_encode, '\n')

sock.sendto(res\_encode.encode(), client\_addr)

if data[:2] == '3|':

print('-= 3 =-')

data = data.split('|')

TGT\_encode = data[1]

print(' TGT(e):', TGT\_encode)

TGT = des\_decode(TGT\_encode, str(K\_as\_tgs), False)

print(' TGT(d):', TGT, '\n')

TGT = TGT.split('|')

AUT = des\_decode(data[2], TGT[4], True)

print(' AUT(d):', AUT, '\n')

AUT = AUT.split('|')[1:]

if TGT[0] == AUT[0] and time\_compare(TGT[2], AUT[1], TGT[3]):

print('-= 4 =-')

TGS = TGT[0] + '|' + ss + '|' + str(datetime.datetime.now()) + '|' + str(p2) + '|' + db\_ss[TGT[0]]

print(' TGS(d):', TGS)

TGS\_encode = des\_encode(TGS, K\_tgs\_ss, True)

print(' TGS(e):', TGS\_encode, '\n')

res = TGS\_encode + '|' + db\_ss[TGT[0]]

print(' res(d):', res)

res\_encode = des\_encode(res, db[TGT[0]][1], True)

print(' res(e):', res\_encode, '\n')

sock.sendto(res\_encode.encode(), client\_addr)

else:

print('\*Истекло время ключа или неправильный идентификатор\*')

except socket.error:

pass

sock.close()

***Сервер SS:***

import socket

import uuid

from des import des\_encode, des\_decode, get\_hex\_key, time\_compare

client\_addr = ('127.0.0.1', 20000)

ss\_addr = ('127.0.0.1', 20002)

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sock.bind(ss\_addr)

ss = '2323d4e8-4b32-44b2-b035-d0a6bf24daef'

K\_tgs\_ss = '3030353861396231'

if \_\_name\_\_ == "\_\_main\_\_":

while True:

try:

data = sock.recvfrom(1024)[0].decode()

if data[:2] == '5|':

print('\n-= 5 =-')

TGS\_encode, AUT\_encode = data[2:].split('|')

print(' TGS(e):', TGS\_encode)

TGS = des\_decode(TGS\_encode, K\_tgs\_ss, True)

print(' TGS(d):', TGS, '\n')

TGS = TGS.split('|')

print(' AUT(e):', AUT\_encode)

AUT = des\_decode(AUT\_encode, TGS[4], True)

print(' AUT(d):', AUT, '\n')

AUT = AUT.split('|')[1:]

if TGS[0] == AUT[0] and time\_compare(TGS[2], AUT[1], TGS[3]):

print('\*Подлинность CLIENT подтверждена!\*')

print('Ключ для защиты сеанса связи Kc\_ss:', TGS[4])

print('\n-= 6 =-')

res = AUT[1] + '1'

print(' res(d):', res)

res\_encode = des\_encode(res, TGS[4], True)

print(' res(e):', res\_encode)

sock.sendto(res\_encode.encode(), client\_addr)

else:

print('\*Истекло время ключа или неправильный идентификатор\*')

print('\n')

pass

except socket.error:

pass