Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»
Кафедра информатики

Отчет по лабораторной работе №2

Идентификация и аутентификация пользователей. Протокол Kerberos.

Выполнил:
Брычиков Д.Д.
Проверил:
Чернявский Ю. А.

Минск 2018

Введение

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.

Прежде всего стоит сказать, что при использовании Kerberos нельзя напрямую получить доступ к какому-либо целевому серверу. 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.

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

Одной из наиболее известных криптографических систем с закрытым ключом является DES – Data Encryption Standard. Эта система первой получила статус государственного стандарта в области шифрования данных. Она разработана специалистами фирмы IBM и вступила в действие в США 1977 году. Алгоритм DES по-прежнему широко применяется и заслуживает внимания при изучении блочных шифров с закрытым ключом.

Стандарт DES построен на комбинированном использовании перестановки, замены и гаммирования. Шифруемые данные должны быть представлены в двоичном виде.

Постановка задачи

Создать приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую Алгоритм DES.

В интерфейсе приложения должны быть наглядно представлены:

- Исходные данные протокола (модули, ключи, секретные данные и т.п.);
- 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.

Схема алгоритма

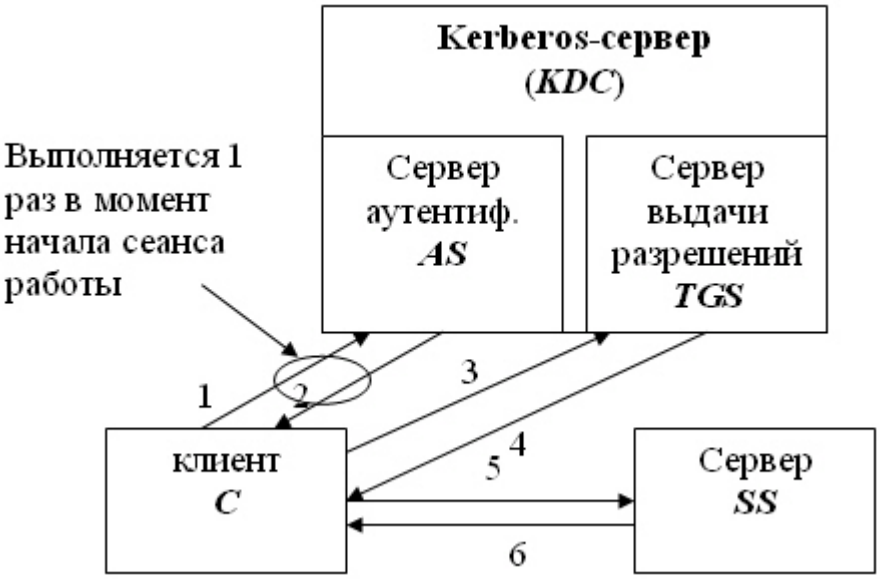

Рис 1. Схема работы протокола Kerberos

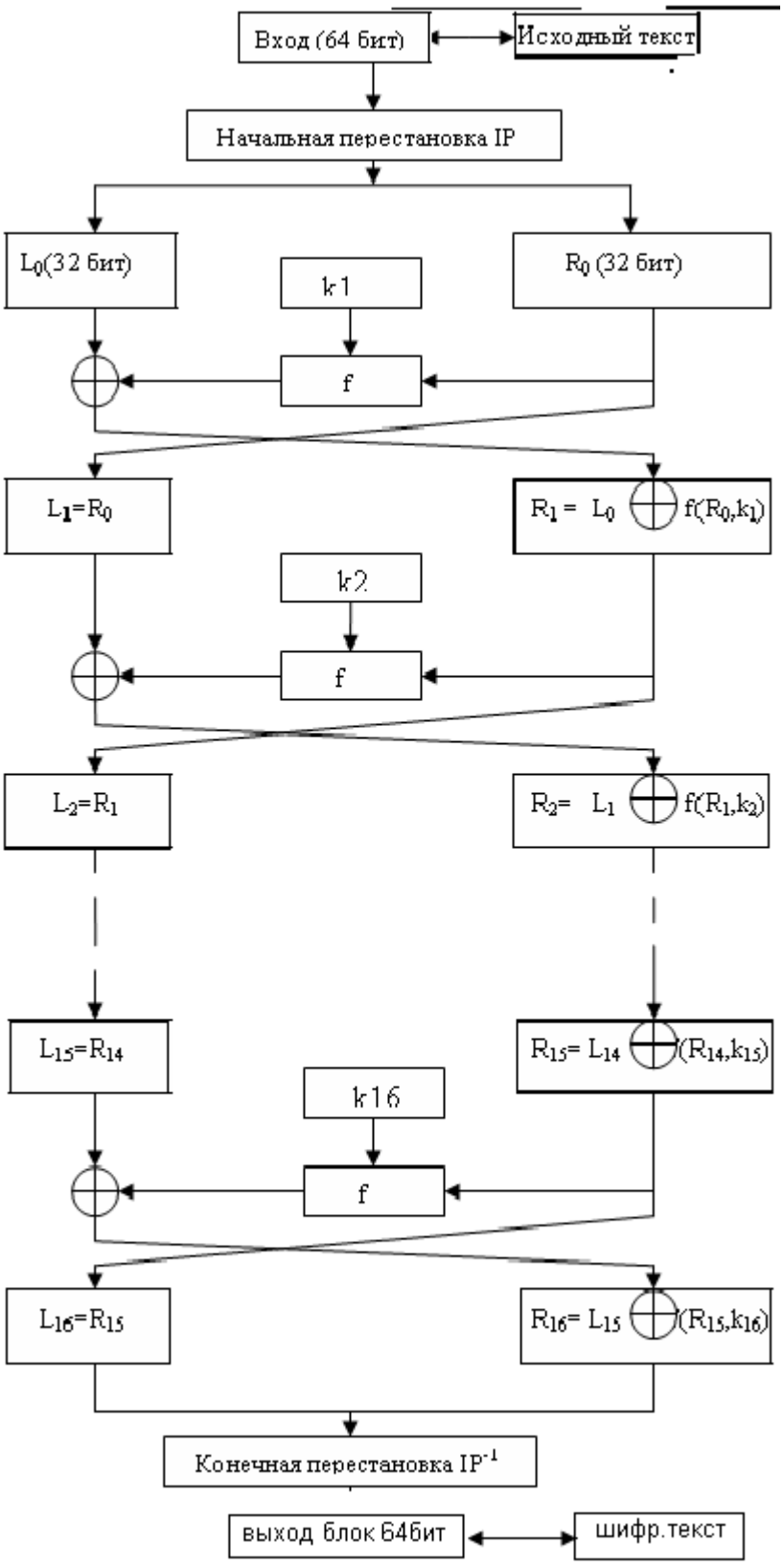

Рис.2 Подробная схема шифрования алгоритма DES

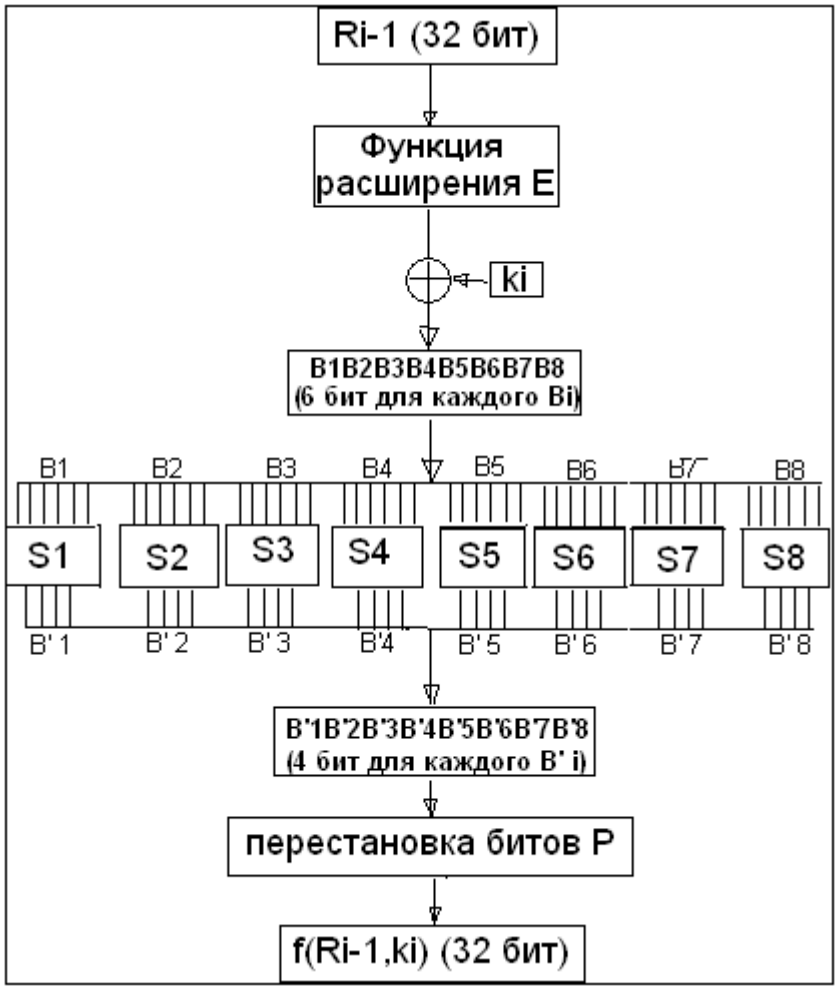

Рис.3 Схема работы функции f

Демонстрация работы программы

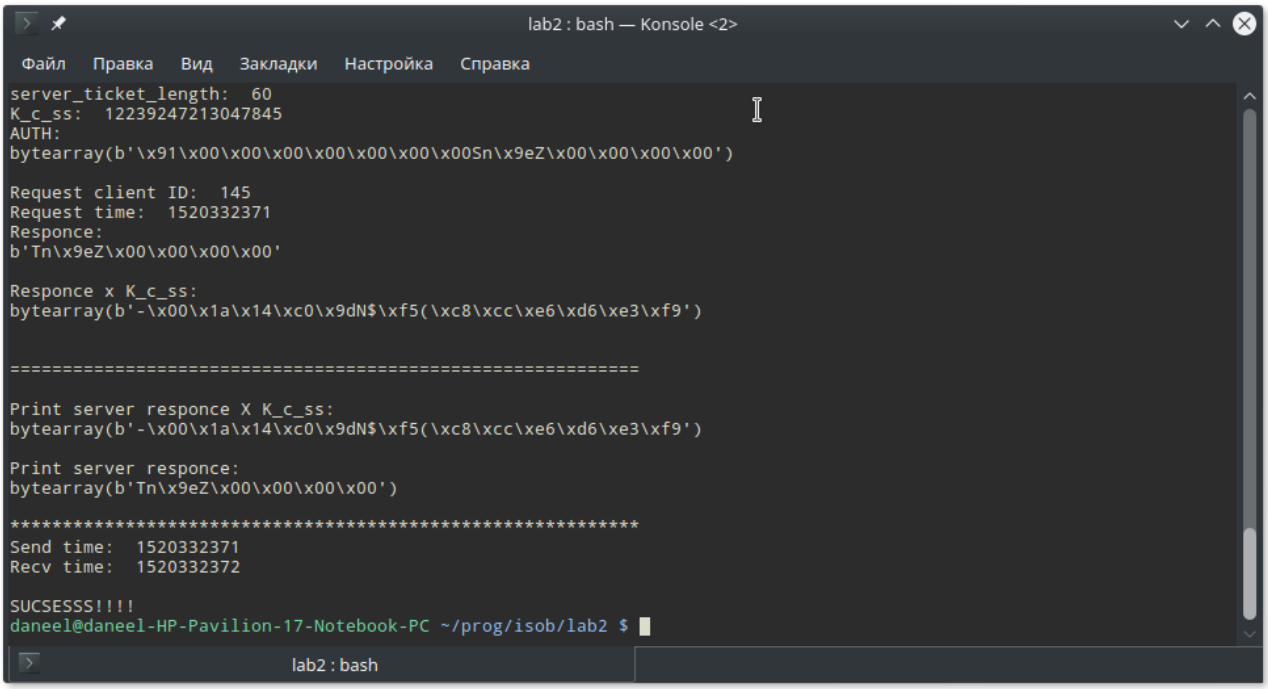

Рис 4. Результат работы первого запуска программы

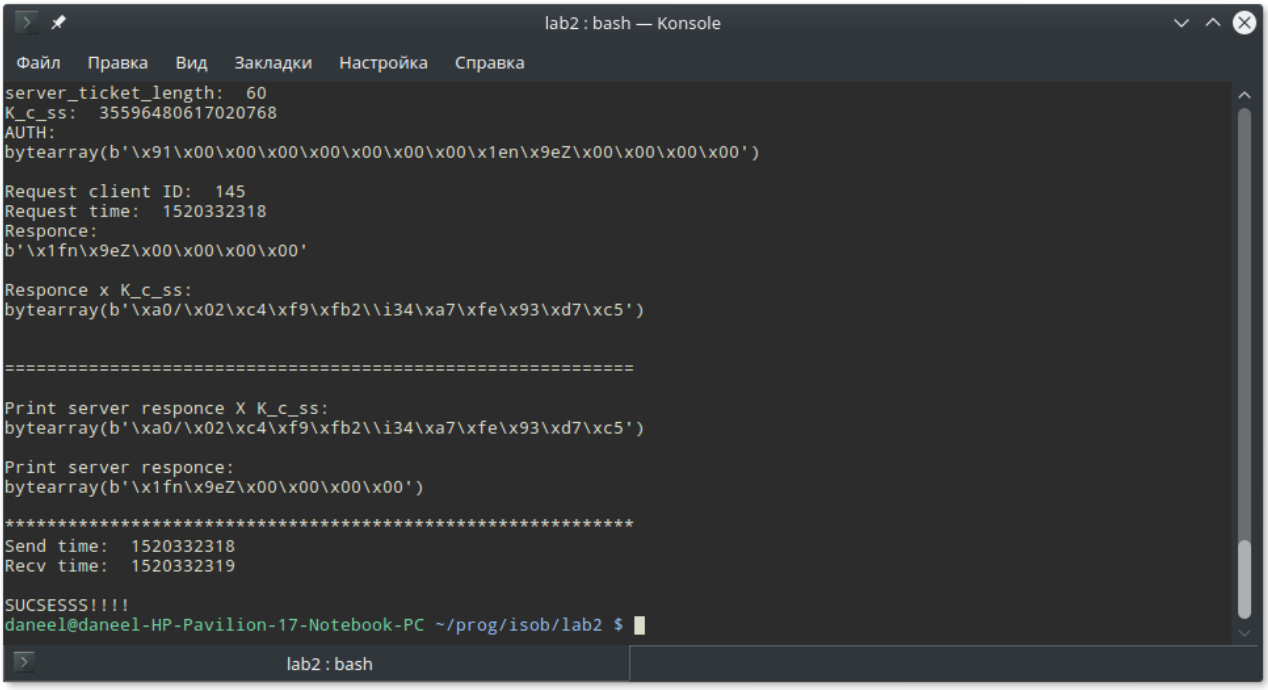

Рис 5. Результат работы второго запуска программы

```
SERVER_ID: 74
SERVER_KEY: 44017009186904142
CLIENT_ID: 89
CLIENT_KEY: 67074212316138624
============================================================
REQUET TO AS SERVER: client_id (89)


==============================REQUEST TO AS(clid=89)==============================
K_c: 67074212316138624
now: 1518542810
GENERATE K_c_tgs key: 40419225602199818

packed TGT(clid, TGS_ID, now, TICKET_LENGHT, C_TGS):
b'Y\x00\x00\x00\x00\x00\x00\x00\x91\x92\x0c\x00\x00\x00\x00\x00\xda\x1f\x83Z\x00\x00\x00\x00<\x00\x0
0\x00\x00\x00\x00\x00\n\x91\x1dP\x12\x99\x8f\x00'


encrypted TGT x K_as_tgs:
bytearray(b'\x92I\x8c\x19eV\xdc_Ya\xf9P\xe6\xd5\xc1\xce@\x8d\xe6\xf9\x97\xee&\xd2\x9c\xe9\xfd\xd3`\x
det\xd0\x17\xf6\xdc\x16\xd1\x8e\xd4\xf8\xccF\x111\xfc\xfc\xa7\x98')


 Packed responce(TGT_enc, K_c_tgs):
bytearray(b'\x000\x92I\x8c\x19eV\xdc_Ya\xf9P\xe6\xd5\xc1\xce@\x8d\xe6\xf9\x97\xee&\xd2\x9c\xe9\xfd\x
d3`\xdet\xd0\x17\xf6\xdc\x16\xd1\x8e\xd4\xf8\xccF\x111\xfc\xfc\xa7\x98\n\x91\x1dP\x12\x99\x8f\x00')


 Encrypted responce([TGT_enc, K_c_tgs] x K_c):
bytearray(b'\xc0#\xdf\x90B\xef\xc2\x94\x9f\xb7"Z}m\xc7\xbf\x9a*\xcbe/\xd4\xd7\xbb\xd7\xcd\xea\x19\xb
dQ\xf9\xf1\x08\xc0\x1e\xbe\xb1\n^{\x81\xcc\xf5\x06\x93z$&C:\xba\n\x05\xdf\r@\xa8\x1d}\xd7\xe2\x93\xb
1\xf8')

============================================================

as_ticket X K_c:
bytearray(b'\xc0#\xdf\x90B\xef\xc2\x94\x9f\xb7"Z}m\xc7\xbf\x9a*\xcbe/\xd4\xd7\xbb\xd7\xcd\xea\x19\xb
dQ\xf9\xf1\x08\xc0\x1e\xbe\xb1\n^{\x81\xcc\xf5\x06\x93z$&C:\xba\n\x05\xdf\r@\xa8\x1d}\xd7\xe2\x93\xb
1\xf8')

as_ticket:
bytearray(b'\x000\x92I\x8c\x19eV\xdc_Ya\xf9P\xe6\xd5\xc1\xce@\x8d\xe6\xf9\x97\xee&\xd2\x9c\xe9\xfd\x
d3`\xdet\xd0\x17\xf6\xdc\x16\xd1\x8e\xd4\xf8\xccF\x111\xfc\xfc\xa7\x98\n\x91\x1dP\x12\x99\x8f\x00')

K_c_tgs: 40419225602199818

TGT x K_as_tgs:
bytearray(b'\x92I\x8c\x19eV\xdc_Ya\xf9P\xe6\xd5\xc1\xce@\x8d\xe6\xf9\x97\xee&\xd2\x9c\xe9\xfd\xd3`\x
det\xd0\x17\xf6\xdc\x16\xd1\x8e\xd4\xf8\xccF\x111\xfc\xfc\xa7\x98')

now:  1518542810
AUTH: b'Y\x00\x00\x00\x00\x00\x00\xda\x1f\x83Z\x00\x00\x00\x00'

AUTH: b'Y\x00\x00\x00\x00\x00\x00\xda\x1f\x83Z\x00\x00\x00\x00'

REQUEST TO TGS


==============================REQUEST TO TGS==============================
INPUT TGT x K_as_tgs:
bytearray(b'\x92I\x8c\x19eV\xdc_Ya\xf9P\xe6\xd5\xc1\xce@\x8d\xe6\xf9\x97\xee&\xd2\x9c\xe9\xfd\xd3`\x
det\xd0\x17\xf6\xdc\x16\xd1\x8e\xd4\xf8\xccF\x111\xfc\xfc\xa7\x98')

INPUT AUTH x K_c_tgs: bytearray(b'q\xe0"ZI=\xa3$0V\xcc\xc0\xd8(\xe6u\x0c\xc7\xa5\xd0\xf5\xd3\x1a#')

SERVER_ID: 74

decrypted TGT:
bytearray(b'Y\x00\x00\x00\x00\x00\x00\x00\x91\x92\x0c\x00\x00\x00\x00\x00\xda\x1f\x83Z\x00\x00\x00\x
00<\x00\x00\x00\x00\x00\x00\x00\n\x91\x1dP\x12\x99\x8f\x00')

UNPACK
Client ID:  89
tgs_id:  823953
ticket time:  1518542810
ticket time length:  60
K_c_tgs:  40419225602199818
GEnerated K_c_ss:  34808175527828602
```

packed TGS(clid, now):

b'Y\x00\x00\x00\x00\x00\x00\x00J\x00\x00\x00\x00\x00\x00\x00\xda\x1f\x83Z\x00\x00\x00\x00<\x00\x00\x
00\x00\x00\x00\x00\x00z\x9c\x9d\xe9\xd9\xa9{\x00'

Encrypted TGS x K_ss_tgs :

bytearray(b"\xfc\x08b\x00\xa8\xdf-\xc9\x8e\xd8\x93(\x196\x96BNb,\x11Q\xab3\xd3\xa0\xeey\'Jr@\x1b\xea
\xf1\x1d\xb6\x8d!\xf1\x8ai6t?\xcc1\xb0\xa7")

Packed responce(TGS x K_ss_tgs, K_c_ss):

bytearray(b"\x000\xfc\x08b\x00\xa8\xdf-\xc9\x8e\xd8\x93(\x196\x96BNb,\x11Q\xab3\xd3\xa0\xeey\'Jr@\x1
b\xea\xf1\x1d\xb6\x8d!\xf1\x8ai6t?\xcc1\xb0\xa7z\x9c\x9d\xe9\xd9\xa9{\x00")

Responce x K_c_tgs:

bytearray(b'k\x7f&\xf8\xec\xea\xea\x8b\xac\xac\xfe\x17\xb1\xc5:d\x84\x12\x19\xf9\xa7\xafTt`\xa8\xac\
x83Yo\x96E$c,~"\x98\xd1\xae5\x88\xb6*b/=\xf9\xe2\xd4\\\xf2tlY\x85\xfd\xe9\xebwd\x0e_\xf4')
==

Ticket x K_c_tgs:

bytearray(b'k\x7f&\xf8\xec\xea\xea\x8b\xac\xac\xfe\x17\xb1\xc5:d\x84\x12\x19\xf9\xa7\xafTt`\xa8\xac\
x83Yo\x96E$c,~"\x98\xd1\xae5\x88\xb6*b/=\xf9\xe2\xd4\\\xf2tlY\x85\xfd\xe9\xebwd\x0e_\xf4')

Ticket:

bytearray(b"\x000\xfc\x08b\x00\xa8\xdf-\xc9\x8e\xd8\x93(\x196\x96BNb,\x11Q\xab3\xd3\xa0\xeey\'Jr@\x1
b\xea\xf1\x1d\xb6\x8d!\xf1\x8ai6t?\xcc1\xb0\xa7z\x9c\x9d\xe9\xd9\xa9{\x00")

TGS_enc:
bytearray(b"\xfc\x08b\x00\xa8\xdf-\xc9\x8e\xd8\x93(\x196\x96BNb,\x11Q\xab3\xd3\xa0\xeey\'Jr@\x1b\xea
\xf1\x1d\xb6\x8d!\xf1\x8ai6t?\xcc1\xb0\xa7")

K_c_ss: 34808175527828602
AUTH2:
b'Y\x00\x00\x00\x00\x00\x00\x00\xda\x1f\x83Z\x00\x00\x00\x00'

AUTH2 x K_c_ss:
bytearray(b'\xc4\x93c\xc6/\x90\xa6\xc9p\xaf\xd7?\xbbx\xc6\x1a<o=\x11\xea\x12\x03\xc5')

REQUEST TO SERVER
==============================REQUEST TO SERVER==============================
Input TGS x K_ss_tgs:
bytearray(b"\xfc\x08b\x00\xa8\xdf-\xc9\x8e\xd8\x93(\x196\x96BNb,\x11Q\xab3\xd3\xa0\xeey\'Jr@\x1b\xea
\xf1\x1d\xb6\x8d!\xf1\x8ai6t?\xcc1\xb0\xa7")

Input AUTH x K_c_ss:
bytearray(b'\xc4\x93c\xc6/\x90\xa6\xc9p\xaf\xd7?\xbbx\xc6\x1a<o=\x11\xea\x12\x03\xc5')

TGS:
bytearray(b'Y\x00\x00\x00\x00\x00\x00\x00J\x00\x00\x00\x00\x00\x00\x00\xda\x1f\x83Z\x00\x00\x00\x00<
\x00\x00\x00\x00\x00\x00\x00z\x9c\x9d\xe9\xd9\xa9{\x00')

UNPACK
clid: 89
server id: 74
ticket time: 1518542810
server_ticket_length: 60
K_c_ss: 34808175527828602
AUTH:
bytearray(b'Y\x00\x00\x00\x00\x00\x00\xda\x1f\x83Z\x00\x00\x00\x00')

Request client ID: 89
Request time: 1518542810
Responce:
b'\xdb\x1f\x83Z\x00\x00\x00\x00'

Responce x K_c_ss:
bytearray(b'\x1b\xff\xb3\xd8\x94\xbd\xd5\x85\x1a7s*G4\x85L')

==

Print server responce X K_c_ss:
bytearray(b'\x1b\xff\xb3\xd8\x94\xbd\xd5\x85\x1a7s*G4\x85L')

```
Print server responce:
bytearray(b'\xdb\x1f\x83Z\x00\x00\x00\x00')

**********************************************************
Send time:  1518542810
Recv time:  1518542811

SUCSESSS!!!!
```

Листинг 1. Полный результат третьего запуска программы

Вывод

В результате проделанной работы был реализован протокол аутентификация Kerberos при помощи симметричного алгоритма шифрования DES.

Kerberos учитывает тот факт что обмен информации может происходить в незащищенной среде а передаваемые пакеты могут быть перехвачены и модифицированы. Для обмена информацией клиент и сервер, должны использовать доверенную третью сторону. Подразумевается, что регистрация на серверах Kerberos происходит через защищенную среду , потому что в в этот момент сервер выдает клиенту или серверу их персональный ключ. В дальнейшем все передачи могут происходить в незащищенной среде.

Протокол Kerberos устроен так, что не требует больших на грузок на сервера. Информацию между ними передает клиент, однако в зашифрованном виде.

Протокол шифрования DES ранее являлся очень популярным, особенно в США. Современные знания о криптографии не выявили скрытых слабостей этого алгоритма. Протокол не требует сложных математических вычислений и может быть реализован очень эффективно. Однако в настоящее время протокол считается устаревшим, поскольку с ростом вычислительных возможностей компьютеров, взлом протокола стал возможен.

Тем не менее DES может быть использован для создания краткосрочных защищенных соединений.

Исходный код

```
import des
import random
from struct import *
import time
import tgs

client_keys = {}

AS_TGS = 16204593401359085
TGS_ID = tgs.TGS_ID
TICKET_LENGHT = 60 #sec

def register():
    clid = random.randrange(0,256,1)
    key = des.get_random_key()
    client_keys[clid] = key
    return clid, key


def get_ticket(clid):
    print("\n\n" + "="*30 + format("REQUEST TO AS(clid={})").format(clid) + "="*30)
    if clid not in client_keys:
        print("\nERROR :FAILD TO FOUND CLIENT")
        exit(1)

    clkey = client_keys[clid]
    print("K_c: {}".format(clkey))
    tm = int(time.time())
    print("now: {}".format(tm))
    C_TGS = des.get_random_key()
    print("GENERATE K_c_tgs key: {}".format(C_TGS))
    TGT = pack("QQQQQ", clid, TGS_ID, tm, TICKET_LENGHT, C_TGS)
    print("\npacked TGT(clid, TGS_ID, now, TICKET_LENGHT, C_TGS): {}\n".format(TGT))
    TGT_enc = des.encrypt(TGT, AS_TGS)
    print("\nencrypted TGT x K_as_tgs: {}\n".format(TGT_enc))

    responce = bytearray()
    responce.append(len(TGT_enc) // 256)
    responce.append(len(TGT_enc) % 256)
    responce = responce + TGT_enc + pack("Q", C_TGS)
    print("\n Packed responce(TGT_enc, K_c_tgs):  {}\n".format(responce))

    res = des.encrypt(responce, clkey)
    print("\n Encrypted responce([TGT_enc, K_c_tgs] x K_c):  {} \n".format(res))
    print("="*60)
    print()
    return res
#!/usr/bin/env python3


import as_server
import tgs
import server
import des
from struct import *
import time


my_id, my_key = as_server.register()
print("CLIENT_ID: {}".format(my_id))
print("CLIENT_KEY: {}".format(my_key))


print("="*60)
print("REQUET TO AS SERVER: client_id ({})".format(my_id))
ticket_enc = as_server.get_ticket(my_id)
print("as_ticket X K_c: {}\n".format(ticket_enc))

ticket = des.decrypt(ticket_enc, my_key)
print("as_ticket: {}\n".format(ticket))

TGT_enc_len = ticket[0] * 256 + ticket[1]
TGT_enc = ticket[2: 2 + TGT_enc_len]
```

```
C_TGS, = unpack("Q", ticket[2 + TGT_enc_len :])

print("K_c_tgs: {}\n".format(C_TGS))
print("TGT x K_as_tgs: {}\n".format(TGT_enc))

tm = int(time.time())
print("now: ", tm)

auth = pack("QQ", my_id, tm)
print("AUTH: {}\n".format(auth))

auth_enc = des.encrypt(auth, C_TGS)
print("AUTH: {}\n".format(auth))

print("REQUEST TO TGS")
ticket_enc = tgs.get_ticket(TGT_enc, auth_enc, server.get_id())
print("Ticket x K_c_tgs:\n {}\n".format(ticket_enc))

ticket = des.decrypt(ticket_enc, C_TGS)
print("Ticket:\n {}\n".format(ticket))

TGS_enc_len = ticket[0] * 256 + ticket[1]
TGS_enc = ticket[2: 2 + TGS_enc_len]
K_c_ss, = unpack("Q", ticket[2 + TGS_enc_len :])

print("TGS_enc:\n{}\n".format(TGS_enc))
print("K_c_ss: {}".format(K_c_ss))

request_time = int(time.time())
auth2 = pack("QQ", my_id, request_time)
auth2_enc = des.encrypt(auth2, K_c_ss)
print("AUTH2:\n{}\n".format(auth2))
print("AUTH2 x K_c_ss:\n{}\n".format(auth2_enc))

print("REQUEST TO SERVER")
responce_enc = server.request(TGS_enc, auth2_enc)

responce = des.decrypt(responce_enc, K_c_ss)
resp_time, = unpack("Q", responce)

print("Print server responce X K_c_ss: \n{}\n".format(responce_enc))
print("Print server responce: \n{}\n".format(responce))


print("*"*60)
print("Send time: ", request_time )
print("Recv time: ", resp_time  )

if resp_time != request_time + 1:
    print("ERROR")
    exit(1)


print("\nSUCSESSS!!!!")
IP = [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]

IP_n = [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]


E = [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]

S1 = [[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]]

S2 = [[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]]

S3 = [[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]]
```

```
S4 = [[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]]

S5 = [[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]]

S6 = [[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]]

S7 = [[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]]

S8 = [[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]]

S = [S1, S2, S3, S4, S5, S6, S7, S8]

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]

C0 = [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]

D0 = [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]

K = [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]


SH = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
import des_const as dc
import random


def binaryze(b, cnt=0):
    res = []
    while b:
        res.append(b & 1)
        b >>= 1

    for _ in range(cnt - len(res)):
        res.append(0)

    return reversed(res)


def debinaryze(seq):
    b = 0
    for bit in seq:
        b = (b << 1) + bit
    return b


def keys_gen(key):
    byts = [[] for i in range(8)]
    for i, bit in enumerate(key):
        byts[i // 7].append(bit)
    for byt in byts:
        ch = 0
        for bit in byt:
            ch = ch ^ bit
        byt.append(ch ^ 1)

    nk = []
    for byt in byts:
        nk = nk + byt

    c0 = [nk[pos-1] for pos in dc.C0]
    d0 = [nk[pos-1] for pos in dc.D0]

    res = []
    ci, di = c0, d0
```

```
    for shift in dc.SH:
        ci = ci[shift:] + ci[:shift]
        di = di[shift:] + di[:shift]
        full_key = ci + di
        res.append([full_key[pos-1] for pos in dc.K])

    return res

def f(d, key):
    ext_d = [d[pos-1] for pos in dc.E]
    add_d = [ext_d[i] ^ key[i] for i in range(len(ext_d))]

    s_res = []
    for block_id in range(8):
        block_start = block_id * 6
        a = add_d[block_start] * 2 + add_d[block_start+5]
        b = 0
        sh = 8
        for pos in range(block_start+1, block_start+5):
            b += sh * add_d[pos]
            sh >>= 1

        s_res += binaryze(dc.S[block_id][a][b], 4)

    return [s_res[pos-1] for pos in dc.P]


def encrypt(mesg, key):
    mesg_add = bytearray([len(mesg) // 256, len(mesg) % 256])
    while (len(mesg_add) + len(mesg)) % 8:
        mesg_add.append(random.randrange(0,256,1))


    keys = keys_gen(binaryze(key, 56))
    mesg_ext = mesg_add + mesg

    result = bytearray()
    for bs in range(0, len(mesg_ext), 8):
        d = []
        for i in range(bs, bs+8):
            d.extend(binaryze(mesg_ext[i], 8))

        d_ip = [d[pos-1] for pos in dc.IP]
        dl, dr = d_ip[:32], d_ip[32:]
        for roundn in range(16):
            new_dl = dr
            f_res = f(dr, keys[roundn])
            new_dr = [f_res[pos] ^ dl[pos] for pos in range(len(f_res))]
            dl, dr = new_dl, new_dr

        new_d = dl + dr
        d_res = [new_d[pos-1] for pos in dc.IP_n]
        for i in range(0, 64, 8):
            result.append(debinaryze(d_res[i:i+8]))

    return result


def decrypt(mesg, key):
    keys = keys_gen(binaryze(key, 56))

    result = bytearray()
    for bs in range(0, len(mesg), 8):
        d = []
        for i in range(bs, bs+8):
            d.extend(binaryze(mesg[i], 8))

        d_ip = [d[pos-1] for pos in dc.IP]
        dl, dr = d_ip[:32], d_ip[32:]
        for roundn in range(15, -1, -1):
            new_dr = dl
            f_res = f(dl, keys[roundn])
            new_dl = [f_res[pos] ^ dr[pos] for pos in range(len(f_res))]
            dl, dr = new_dl, new_dr

        new_d = dl + dr
        d_res = [new_d[pos-1] for pos in dc.IP_n]
        for i in range(0, 64, 8):
```

```
            result.append(debinaryze(d_res[i:i+8]))

    ln = result[0] * 256 + result[1]
    exceed = len(result) - ln
    return result[exceed:]


def get_random_key():
    return random.randrange(0,1<<56,1)
import tgs
import time
import des
from struct import *


my_id, my_key = tgs.register()
print("SERVER_ID: {}".format(my_id))
print("SERVER_KEY: {}".format(my_key))


def get_id():
    return my_id


def request(TGS_enc, AUTH_enc):
    print("="*30 +"REQUEST TO SERVER"+ "="*30)
    print("Input TGS x K_ss_tgs: \n{}\n".format(TGS_enc))
    print("Input AUTH x K_c_ss: \n{}\n".format(AUTH_enc))

    TGS = des.decrypt(TGS_enc, my_key)
    print("TGS: \n{}\n".format(TGS))
    clid, server_id, ticket_time, server_ticket_length, K_c_ss = unpack("QQQQQ", TGS)

    print("UNPACK")
    print("clid: ", clid)
    print("server id: ", server_id)
    print("ticket time: ", ticket_time)
    print("server_ticket_length: ", server_ticket_length)
    print("K_c_ss: ", K_c_ss)

    now = int(time.time())
    if server_id != my_id or now - ticket_time > server_ticket_length:
        print("ERROR")
        exit(1)

    auth = des.decrypt(AUTH_enc, K_c_ss)
    print("AUTH: \n{}\n".format(auth))
    clid2, tm2 = unpack("QQ", auth)

    print("Request client ID: ", clid2)
    print("Request time: ", tm2)

    if clid != clid2 or now - tm2 > server_ticket_length:
        print("ERROR")
        exit(1)

    responce = pack("Q", tm2+1)

    print("Responce: \n{}\n".format(responce))
    res = des.encrypt(responce, K_c_ss)
    print("Responce x K_c_ss: \n{}\n".format(res))
    print()
    print("="*60)
    print()
    return res
import des
import random
from struct import *
import time

server_keys = {}


AS_TGS = 16204593401359085
TGS_ID = 823953
SERVER_TICKET_LENGHT = 60 #sec
```

```
def register():
    serv_id = random.randrange(0,256,1)
    key = des.get_random_key()
    server_keys[serv_id] = key
    return serv_id, key


def get_ticket(TGT_enc, AUTH_enc, server_id):
    print("\n\n" + "="*30 + format("REQUEST TO TGS") + "="*30)
    print("INPUT TGT x K_as_tgs: {}\n".format(TGT_enc))
    print("INPUT AUTH x K_c_tgs: {}\n".format(AUTH_enc))
    print("SERVER_ID: {}".format(server_id))

    TGT = des.decrypt(TGT_enc, AS_TGS)
    print("\ndecrypted TGT: {}\n".format(TGT))
    clid, tgs_id, tm1, ticket_length, C_TGS = unpack("QQQQQ", TGT)

    print("UNPACK")
    print("Client ID: ", clid)
    print("tgs_id: ", tgs_id)
    print("ticket time: ", tm1)
    print("ticket time length: ", ticket_length)
    print("K_c_tgs: ", C_TGS)

    if server_id not in server_keys:
        print("ERROR: INVALID SERVER")
        exit(1)

    now = int(time.time())
    if now - tm1 > ticket_length or tgs_id != TGS_ID:
        print("ERROR: INVALID TIME")
        exit(1)

    AUTH = des.decrypt(AUTH_enc, C_TGS)
    clid2, tm2 = unpack("QQ", AUTH)

    if now - tm2 > ticket_length or clid != clid2:
        print("ERROR: INVALID TIME")
        exit(1)

    K_c_ss = des.get_random_key()
    print("GEnerated K_c_ss: ", K_c_ss)

    TGS = pack("QQQQQ", clid, server_id, now, SERVER_TICKET_LENGHT, K_c_ss)
    print("\npacked TGS: \n", TGS)
    TGS_enc = des.encrypt(TGS, server_keys[server_id])
    print("\nEncrypted TGS: \n", TGS_enc)

    responce = bytearray()
    responce.append(len(TGS_enc) // 256)
    responce.append(len(TGS_enc) % 256)
    responce = responce + TGS_enc + pack("Q", K_c_ss)

    print("\nPacked responce( TGS x K_ss_tgs, K_c_ss): \n", responce)

    res = des.encrypt(responce, C_TGS)
    print("\nResponce x K_c_tgs: \n", res)
    print("="*60)
    print()
    return res
```