1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский государственный политехнический
3. университет Петра Великого
4. —
5. Институт компьютерных наук и технологий
6. **Кафедра «Информационная безопасность компьютерных систем»**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Криптосистемы на основе задачи дискретного логарифмирования**

по дисциплине «Криптографические методы защиты информации»

Выполнил

студент гр. 43609/3 Князев П.В.

Преподаватель Павленко Е.Ю.

Санкт-Петербург

2019

**Формулировка задания**

1. Получить у преподавателя вариант задания.
2. Разработать в соответствии с вариантом программу (П-1), которая реализует протокол бесключевого шифрования Месси-Омуры

**Результаты работы**

В результате работы на языке python была написана программа, которая реализует протокол бесключевого шифрования Месси-Омуры.

Алгоритм взаимодействия между клиентом и сервером:

1. Клиент генерирует большое простое число P и 3DES ключ K;
2. Клиент генерирует открытый и закрытый показатели E, D в мультипликативной группе поля FP;
3. Клиент зашифровывает K с помошью с показателя Е, получает зашифрованный ключ Ca и посылает P, Ca на сервер;
4. Сервер получает P, Ca. Затем он генерирует свои показатели E, D, используя полученное число P;
5. Сервер зашифровывает Ca своим E, получает Cab и отсылает его обратно клиенту;
6. Клиент получет Cab и расшифровывает его своим D. Полученный результат Cb посылается на сервер.
7. Сервер получает Cab и расшифровывает его своим D, в результате получая исходный ключ K.
8. Теперь клиент и сервер имеют ключ симметричного шифра и могут безопасно взаимодействовать.

Результаты работы программы показаны на рис. 1, 2. Сгенерированные параметры криптосиситемы показаны в таблице 1.

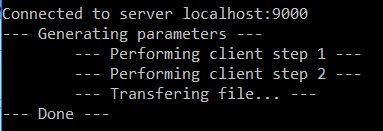
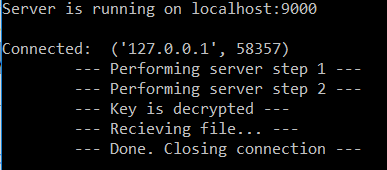
 

Рисунок 1 – Сторона клиента Рисунок 2 – Сторона сервера

Таблица 1 - Сгенерированные параметры криптосистемы:

|  |  |
| --- | --- |
| Параметр | Значение |
| Сообщение *t* | 1837179664448456536694205159276639746923025717357718797142 |
| Открытый показатель клиента *a* | 6236138444318024498510579822822383392721621457099662842449930379  7664068097840662501352737499468504031678329819380910345840905363  4720259061825948691690780498567713831173198175548303332742504552  9863011975027192818768786210198321335586303569621236398959682962  2260583311565101241531718538062309838835160670752059 |
| Открытый показатель сервера b | 10965799621452006941795145464078922664117581327334052651734348747  35019975465688289745780357717557660880106612540465770918802777470  70073107738247525449911727936199043847943753017180719768048460493  60233370034671647001519740315026524586027839240064171277811771505  890924212084510240304013623616756284707564763869 |
| Закрытый  показатель клиента | 30059548858856422575759145230984158042451972388777994452427454074  92907947497983087176659696467857779165453269792816863009437584459  70102623499375813131986664050058862037690991819302184673960719199  95966261917513169728443905226964172627279709278481666043325535217  6850847366977989295890685243695851256801441876229 |
| Закрытый  показатель сервера | 2980003597396198896605864916266152549949823829114064421310977263  8663331427665215720120700342159605059209888629567509515335868621  6434774453862848607273057699409861697561792929605520448820099184  3225657661238539224443191341794789687176701112345779375036664261  56847175196170027507172229952219902146679827380704519 |

Журнал взаимодействия сторон показан на рис. 3, 4, 5.

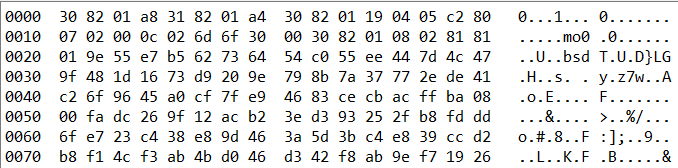


Рисунок 3 – Отправка клиентом

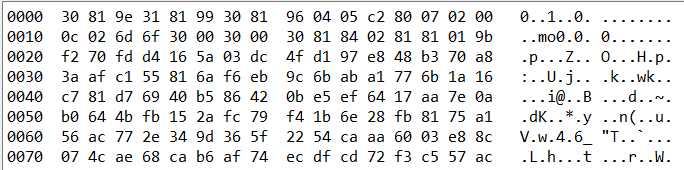


Рисунок 4 – Отправка сервером

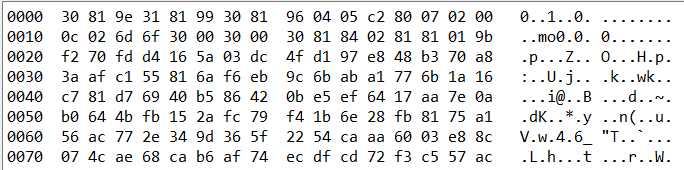


Рисунок 5 – Отправка клиентом

**Ответы на контрольные вопросы**

1. **Покажите, что задача Месси-Омуры сводится к задаче дискретного логарифмирования.**

Пусть атакующему известны , , . Тогда для того, чтобы получить ему необходимо найти . Зная , атакующий может легко найти следующим образом: . Таким образом, задача Месси-Омуры сводится к задаче дискретного логарифмирования.

1. **Подвержен ли протокол шифрования Эль-Гамаля к атаке на основе подобранных шифртекстов?**

Пусть атакующему известен шифртекст и . Тогда если на вход дешифратору он подает , то имеем следующее:

Таким образом, атакующий преобразовал в , следовательно он может преобразовать входное сообщение в некоторое сообщение на выходе, тем самым, он совершает подделку исходного сообщения.

1. **Перечислите задачи, положенные в основу схемы подписи Эль-Гамаля. Почему важно, чтобы период генератора случайных чисел, используемого в схеме подписи, был достаточно большим?**

В основу схемы подписи Эль-Гамаля положены следующие задачи:

* задача дискретного логарифмирования;
* задача предсказания случайного числа ;
* число не должно повторятся в течение времени жизни ключа .

Если период генератора случайных будет мал, то злоумышленник может отследить повторяющийся случайный показатель , а зная его, исходное сообщение и подпись, он может легко найти секретный ключ:

**Выводы**

В результате выполнения лабораторной работы были изучены схемы шифрования и подписи Эль-Гамаля, бесключевое шифрование Месси-Омуры, а также задачи, положенные в основу данных схем.

*Приложение*

Main.py

import cryptutils

import asnapi

import socket

import sys

import os

RECV\_BUF = 1024

PRIME\_BITS = 1024

DES3\_KEY\_BITS = 192

def assert\_rcv(data):

if not data:

raise Exception('data is None')

def transfer\_file(sock, filename, length, key\_int):

key = cryptutils.int\_to\_bytes(key\_int)

if length % RECV\_BUF == 0:

chunks = length // RECV\_BUF

else:

chunks = length // RECV\_BUF + 1

with open(filename, 'rb') as f:

for \_ in range(chunks):

data = f.read(RECV\_BUF)

sock.send(cryptutils.des\_encrypt(data, key))

def client(ip\_addr, port, filename):

sock = socket.socket()

sock.settimeout(10)

sock.connect((ip\_addr, port))

print('\nConnected to server %s:%s' % (ip\_addr, port))

# client generates prime and des key

# then client sends key to server via Massey - Omura protocol

print('--- Generating parameters ---')

key\_int = cryptutils.getrandbits(DES3\_KEY\_BITS)

prime = cryptutils.get\_big\_prime(PRIME\_BITS)

client = cryptutils.MasseyOmuraProtocol(prime)

# step 1: client encrypts key with its own e

print('\t--- Performing client step 1 ---')

encrypted\_key = client.encrypt\_msg(key\_int)

asn1\_blob = asnapi.export\_client\_msg1(prime, prime, encrypted\_key)

sock.send(asn1\_blob)

# step 2: client recieves its data back from server

# but now it's encrypted with server's e,

# so it's time to remove client's lock

print('\t--- Performing client step 2 ---')

asn1\_blob = sock.recv(RECV\_BUF)

assert\_rcv(asn1\_blob)

parsed\_asn1 = asnapi.import\_server\_msg1(asn1\_blob)

unlocked\_data = client.decrypt\_msg(parsed\_asn1['data'])

# step 3: send file attributes

if '.' in filename:

filename\_srv = filename[:filename.rfind('.')] + '.srv.'

filename\_srv += filename.split('.')[-1]

else:

filename\_srv = filename + '.srv'

length = os.path.getsize(filename)

asn1\_blob = asnapi.export\_client\_msg2(

unlocked\_data, length, filename\_srv.encode())

sock.send(asn1\_blob)

# step 4: transmit file

# recv to synchronize client and server

if sock.recv(RECV\_BUF).decode() != 'Ok':

raise Exception('Error transfering file')

print('\t--- Transfering file... ---')

transfer\_file(sock, filename, length, key\_int)

print('--- Done ---')

def recieve\_file(conn, filename, length, key):

open(filename, 'w').close()

if length % RECV\_BUF == 0:

chunks = length // RECV\_BUF

else:

chunks = length // RECV\_BUF + 1

with open(filename, 'ab') as f:

for \_ in range(chunks - 1):

f.write(cryptutils.des\_decrypt(

conn.recv(RECV\_BUF), key))

last\_chunk\_n = length % RECV\_BUF or RECV\_BUF

f.write(cryptutils.des\_decrypt(

conn.recv(RECV\_BUF), key)[:last\_chunk\_n])

def server(ip\_addr, port):

sock = socket.socket()

sock.bind((ip\_addr, port))

sock.listen(1)

print('Server is running on %s:%s' % (ip\_addr, str(port)))

while True:

conn, addr = sock.accept()

print('\nConnected: ', addr)

try:

asn1\_blob = conn.recv(RECV\_BUF)

assert\_rcv(asn1\_blob)

parsed\_asn1 = asnapi.import\_client\_msg1(asn1\_blob)

data = parsed\_asn1['data']

prime = parsed\_asn1['prime']

# step 1: server encrypts client's data with its own e

print('\t--- Performing server step 1 ---')

server = cryptutils.MasseyOmuraProtocol(prime)

encrypted\_data = server.encrypt\_msg(data)

asn1\_blob = asnapi.export\_server\_msg1(encrypted\_data)

conn.send(asn1\_blob)

# step 2: server decrypts client's data with its own d

print('\t--- Performing server step 2 ---')

asn1\_blob = conn.recv(RECV\_BUF)

assert\_rcv(asn1\_blob)

parsed\_asn1 = asnapi.import\_client\_msg2(asn1\_blob)

data = parsed\_asn1['data']

length = parsed\_asn1['length']

filename = parsed\_asn1['name']

# step 3: decrypt main message with decrypted\_key

decrypted\_key = server.decrypt\_msg(data)

key = cryptutils.int\_to\_bytes(decrypted\_key)

print('\t--- Key is decrypted ---')

# step 4: recieve file

# synchronize with client

print('\t--- Recieving file... ---')

conn.send('Ok'.encode())

recieve\_file(conn, filename, length, key)

print('\t--- Done. Closing connection ---')

except Exception as e:

print('Error occured: %s. Closing connection\n' % str(e))

finally:

conn.close()

def usage():

print('Usage:')

print('\tClient mode: main.py client <ip> <port> <filename>')

print('\tServer mode: main.py server <ip> <port>')

if \_\_name\_\_ == '\_\_main\_\_':

if len(sys.argv) == 4 or len(sys.argv) == 5:

mode = sys.argv[1]

if mode == 'client':

filename = sys.argv[4]

client(sys.argv[2], int(sys.argv[3]), filename)

elif mode == 'server':

server(sys.argv[2], int(sys.argv[3]))

else:

usage()

else:

usage()

#for i in range(10000):

#

# print(i)

# prime = cryptutils.get\_big\_prime(PRIME\_BITS)

#

# server = cryptutils.MasseyOmuraProtocol(prime)

# client = cryptutils.MasseyOmuraProtocol(prime)

#

# data\_int = cryptutils.getrandbits(192)

# encrypted\_data\_cli = client.encrypt\_msg(data\_int)

# encrypted\_data\_srv = server.encrypt\_msg(encrypted\_data\_cli)

# decrypted\_data\_cli = client.decrypt\_msg(encrypted\_data\_srv)

# decrypted\_data\_srv = server.decrypt\_msg(decrypted\_data\_cli)

#

# assert data\_int == decrypted\_data\_srv

#

#assert False