EES486 COMPUTER NETWORKS-2

Assignment FINAL

A Secure Chat Application Using RSA and AES

Emir Kaan YERLI, Ege University, emirkaan.yerli@gmail.com, July 2020

INTRODUCTION

The purpose of this project is to create a chat application with the encryption algorithms applied. Since I used the programming language in my thesis, Python was chosen as the dominant language. The RSA algorithm was chosen as the asymmetric algorithm to encrypt the session key, and it was decided that AES is the symmetric algorithm responsible for encrypting and decrypting messages. In the first part, basic information about AES and RSA algorithms and basic information about cryptography will be given. In the second part, the structure and algorithm scheme required for symmetric session chat security are mentioned. In the third section, the codes developed and used for the project are added with their descriptions. In the fourth section, demo screenshots of the chat application are shared.

1. OVERVIEW

1.1. Session Key

A session key is any encryption key used to symmetrically encrypt one communication session only. In other words, it's a temporary key that is only used once, during one stretch of time, for encrypting and decrypting data; future conversations between the two parties would be encrypted with different session keys. A session key would be a password that someone resets every time they log in.

1.2. RSA Algorithm

As asymmetric algorithm to encrypt session key RSA algorithm was chosen. To generate RSA key pair this algorithm had to be used:

i. Chose randomly two large prime numbers p and q
ii. Solve $n = p \cdot q$
iii. Solve Euler function value for n:$\phi(n) = (p-1)\cdot(q-1)$
iv. Chose number e such as $1 < e < \phi(n)$ relatively prime with $\phi(n)$
v. Solve $d=e-1 \bmod \phi(n)$

Public key is defined as number pair (n, e) while private key is defined as pair (n, d)
To encrypt with RSA algorithm message have to be divide in to mi blocks of value not larger than n. **ci=mie mod n**
To decrypt with RSA algorithm every ci block had to be transform like this: **mi=cidmod n**

1.3. Advanced Encryption Standard (AES)

RSA only uses to send and take session key between server and client. After the sharing session keys with this secure method. Server and client can communicate with the new key. For this part AES was decided to be symmetric algorithm responsible for encrypting and decrypting messages. The library of the algorithm was used for coding.

1.4. Symmetric - Asymmetric Encryption

In symmetric encryption, the exact same key is used on both sides of a conversation, for both encrypting and decrypting. In a session that uses symmetric encryption, multiple keys can be used, but a message that is encrypted with one key is decrypted with that same key.

In asymmetric encryption, there are two keys, and data that is encrypted with one key can only be decrypted with the other key – unlike in symmetric encryption, when the same key both encrypts and decrypts. This is also known as public key encryption, because one of the keys is shared publicly.

Acronyms: Ks+: server's public key

- Ks-: servers's private key
- Kc+: client's public key
- Kc-: client's private key
- Ks: sessionKey

2. PROJECT DIAGRAM

1. Selects a random symmetric session key, Ks
2. Server and Client share their public keys without secure algorithms.
3. Client encrypts the session key with Server's public key, (session key+Ks+), Server decrypts with it's private and get the session key.
4. Now they are able to send messages with the AES algoritm. Because they both have the key to encrypt and decrypt.

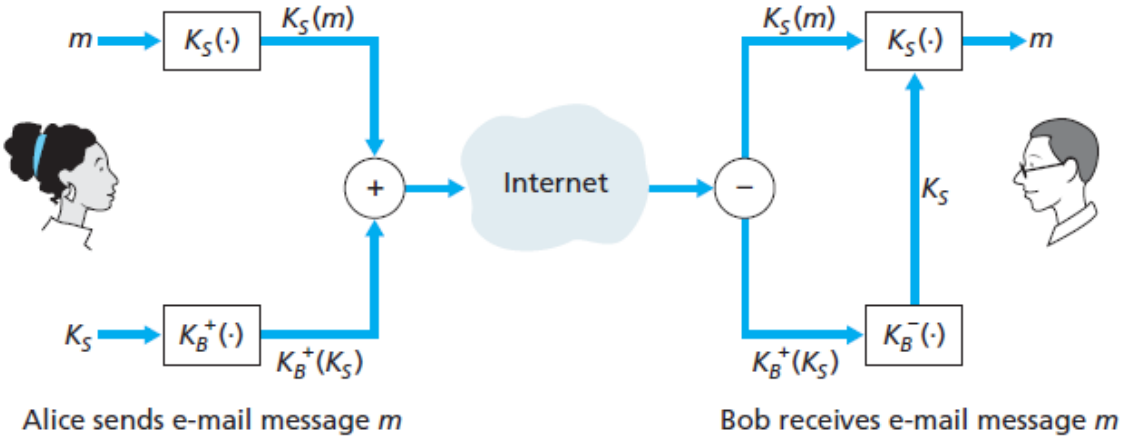


Figure 8.19 ♦ Alice used a symmetric session key, K_S, to send a secret e-mail to Bob

3. PROJECT CODES

3.1.1 Server Side Codes

✓ Socket Created.

```
LOCALHOST = "127.0.0.1"
PORT = 65534
BUFSIZ = 1024

SERVER = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
SERVER.bind((LOCALHOST, PORT))

SERVER.listen(3)
print("Waiting for connection...")
ACCEPT_THREAD = Thread(target=accept_incoming_connections)
ACCEPT_THREAD.start()
ACCEPT_THREAD.join()
SERVER.close()
```

✓ Sets up handling for incoming clients.

```
def accept_incoming_connections():

    while True:
        client, client_address = SERVER.accept()
        print("%s:%s has connected." % client_address)
        client.send(bytes("Type your name and press enter!", "utf8"))
        Thread(target=handle_client, args=(client,)).start()
```

✓ Handles a single client connection.

```
def handle_client(client):  # Takes client socket as argument.
    # Key pair for each client
    keyPair = RSA.generate(1024)#private key burada üretiliyor
    pubKey = keyPair.publickey()
    pubKeyPEM = pubKey.exportKey()
    print(pubKeyPEM)
    privKeyPEM = keyPair.exportKey()
    print(privKeyPEM)
    name = client.recv(BUFSIZ).decode("utf8")
    # Write public keys of users into a file
    with open(name + "_pub.bin", "w") as f:
        f.write(pubKeyPEM.decode('ascii'))
    # Send private keys of users to them
    client.send(bytes("@" + privKeyPEM.decode('ascii'), "utf-8"))
    message = 'Welcome %s! Type "quit" to exit.' % name
    client.send(bytes(message, "utf-8"))
    clients[client] = name
```

```
    while True:
        msg = client.recv(BUFSIZ)
        if msg != bytes("quit", "utf8"):
            msg = msg.decode()
            print(msg)
            if (msg.startswith('@')):
                parsed_msg = msg.split(":")
                #parsed_msg[0] gönderilen kisi
                #parsed_msg[1] mesajın aes ile sifrelenmis hali
                #parsed_msg[2] aes sifresinin karsı tarafın public keyi ile sifrelenmis hali
                to = parsed_msg[0][1:].lower()
                broadcast(bytes(parsed_msg[1] + ":" + parsed_msg[2], "utf-8"), name + " : ", to)

        else:
            client.send(bytes("quit", "utf-8"))
            client.close()
            del clients[client]
            broadcast(bytes("%s has left the chat.\n" % name, "utf-8"))
            break
```

✓ Message is sent

```
def broadcast(msg, name="", to=""):
    for sock in clients:
        if (to != "" and clients[sock] == to):
            sock.send(bytes(name, "utf-8") + msg)
        elif (to == ""):
            # else:
            sock.send(bytes(name, "utf-8") + msg)
```

3.1.2 Client

✓ **Create random symmetric key(Session key)**

```
key = ''.join(random.choices(string.ascii_uppercase + string.digits, k=10))
print("Random generated key of user : ", key)
aes = AESCipher(key)
```

✓ Connecting the client to the server and encrypting the entered message

```
SERVER = "127.0.0.1"
PORT = 65534
BUFSIZ = 1024

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((SERVER, PORT))

receive_thread = Thread(target=receive)
receive_thread.start()

while True:
    msg = input()
    parsed_msg = msg.split(':')
    if (len(parsed_msg) > 1):
        # Take the recievers public key to encrypt symmetric key
        with open(parsed_msg[0][1:] + "_pub.bin", "r") as pub_file:
            pub_key = RSA.importKey(pub_file.read())
            encryptor = PKCS1_OAEP.new(pub_key)
            encrypted_key = encryptor.encrypt(bytes(key, "utf-8"))

        cipher = aes.encrypt(parsed_msg[1]) + bytes(":", "utf-8") + binascii.hexlify(encrypted_key)
        client.send(bytes(parsed_msg[0] + ':', "utf-8") + cipher)
    else:
        client.send(bytes(msg, "utf-8"))

    if msg == "quit":
        client.close()
        break
```

✓ Decrypting the incoming message

```
def receive():
    """Handles receiving of messages."""
    while True:
        try:
            msg = client.recv(BUFSIZ).decode("utf-8")
            # print(msg)
            if (msg[0] == "@"):
                priv_key = msg[1:]
            elif (':' in msg):
                parsed_msg = msg.split(':')
                private_key = RSA.importKey(priv_key)
                decryptor = PKCS1_OAEP.new(private_key)
                decrypted_key = decryptor.decrypt(binascii.unhexlify(bytes(parsed_msg[2], "utf-8")))
                print("Randomly generated key from the sender : ", decrypted_key.decode("utf-8"))
                new_aes = AESCipher(decrypted_key.decode("utf-8"))
                decrypted = new_aes.decrypt(bytes(parsed_msg[1], "utf-8"))
                print(parsed_msg[0] + ":" + decrypted.decode())
            else:
                print(msg)
        except OSError:  # Possibly client has left the chat.
            break
```

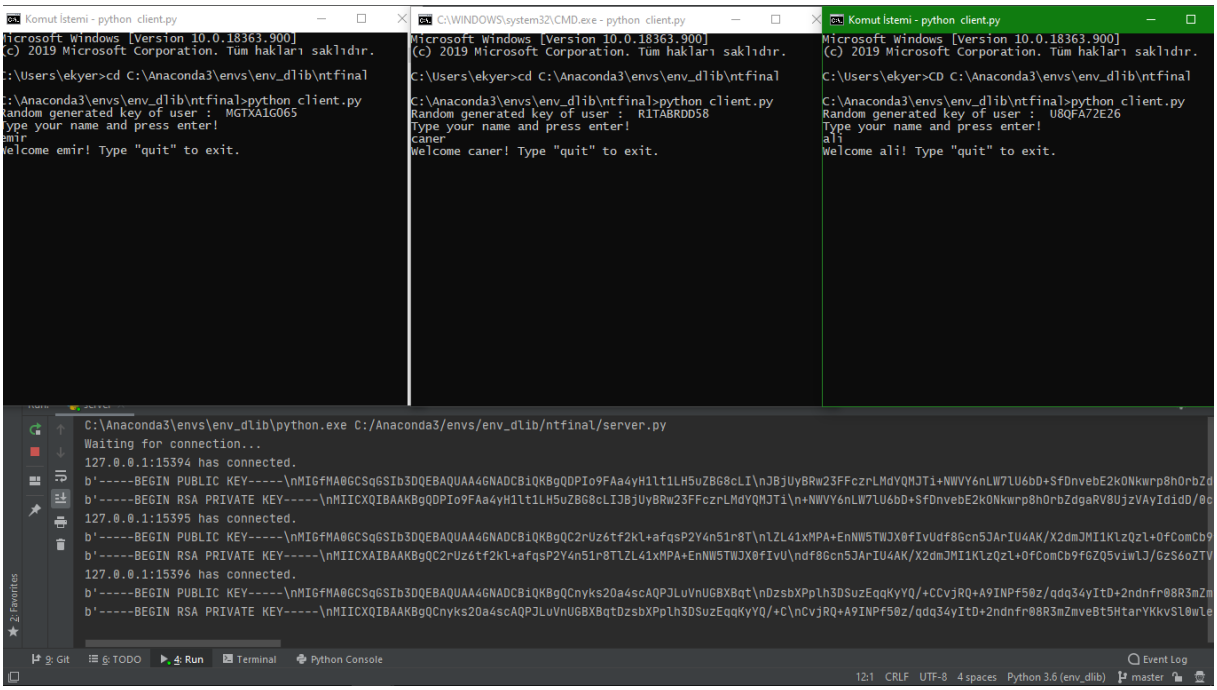
3.1.3 Encryption

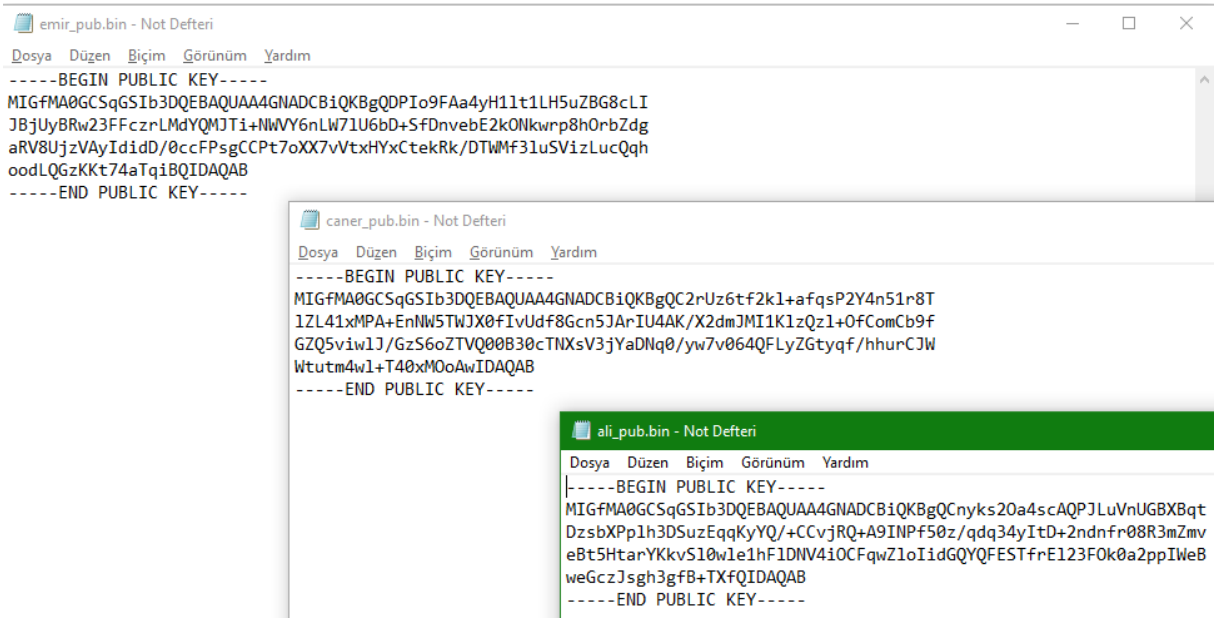
✓ The message is encrypted with aes using session key

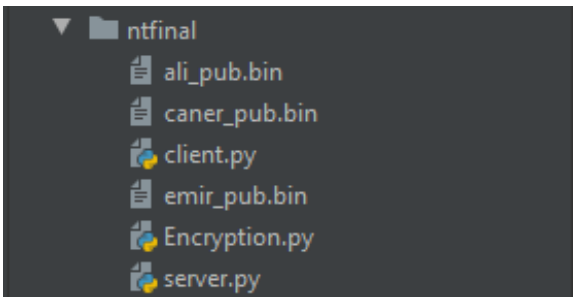
```
class AESCipher:  # AES in Cipher Block Chaining Mode

    def __init__(self, key):
        self.block_size = 16
        self.key = hashlib.sha256(key.encode('utf-8')).digest()

    def encrypt(self, raw):
        raw = pad(raw, self.block_size)
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        print("aes key", self.key)

        return base64.b64encode(iv + cipher.encrypt(raw.encode('utf-8')))

    def decrypt(self, enc):
        enc = base64.b64decode(enc)
        iv = enc[:16]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        print("aes key", self.key)

        return unpad(cipher.decrypt(enc[16:]), self.block_size)

# Padding functions
def pad(s, BS):
    return s + (BS - len(s) % BS) * chr(BS - len(s) % BS)

def unpad(s, BS):
    return s[0:-s[-1]]
```

4. Demonstrate

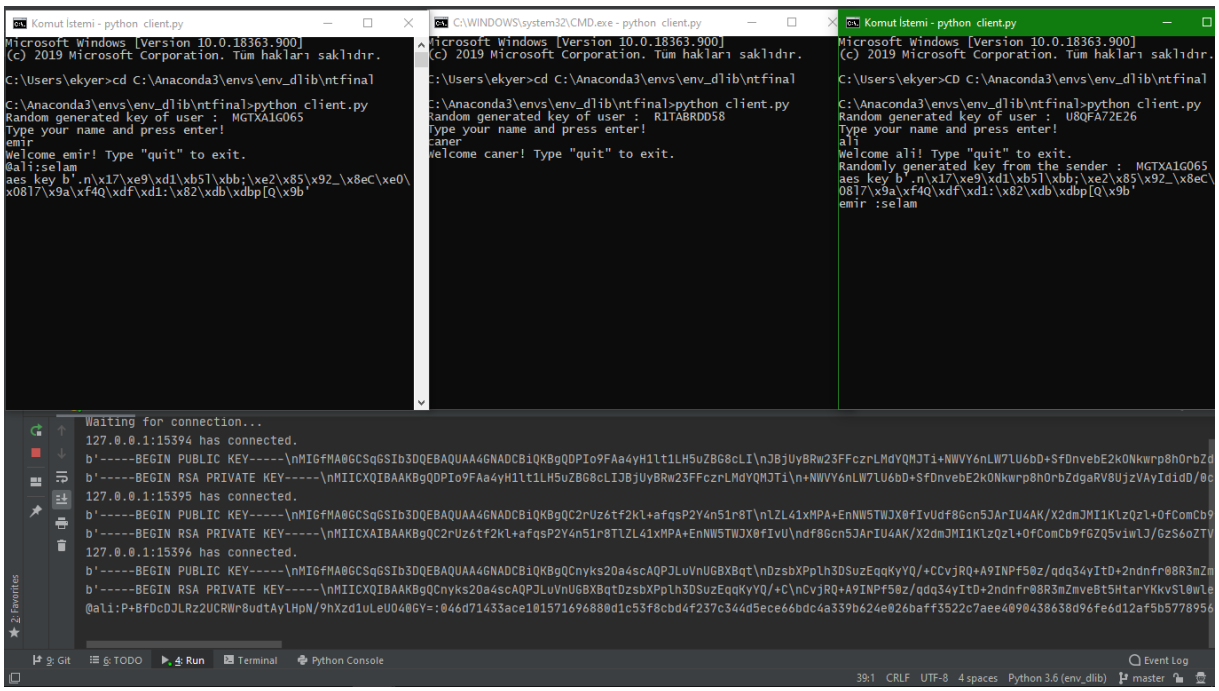
- ✓ First of all, each client gives their names when connecting to the server.
- ✓ These names are assigned randomly generated session keys.
- ✓ Then each is given a public and a private key with RSA.
- ✓ It is saved as a pub file in the public key file folder.

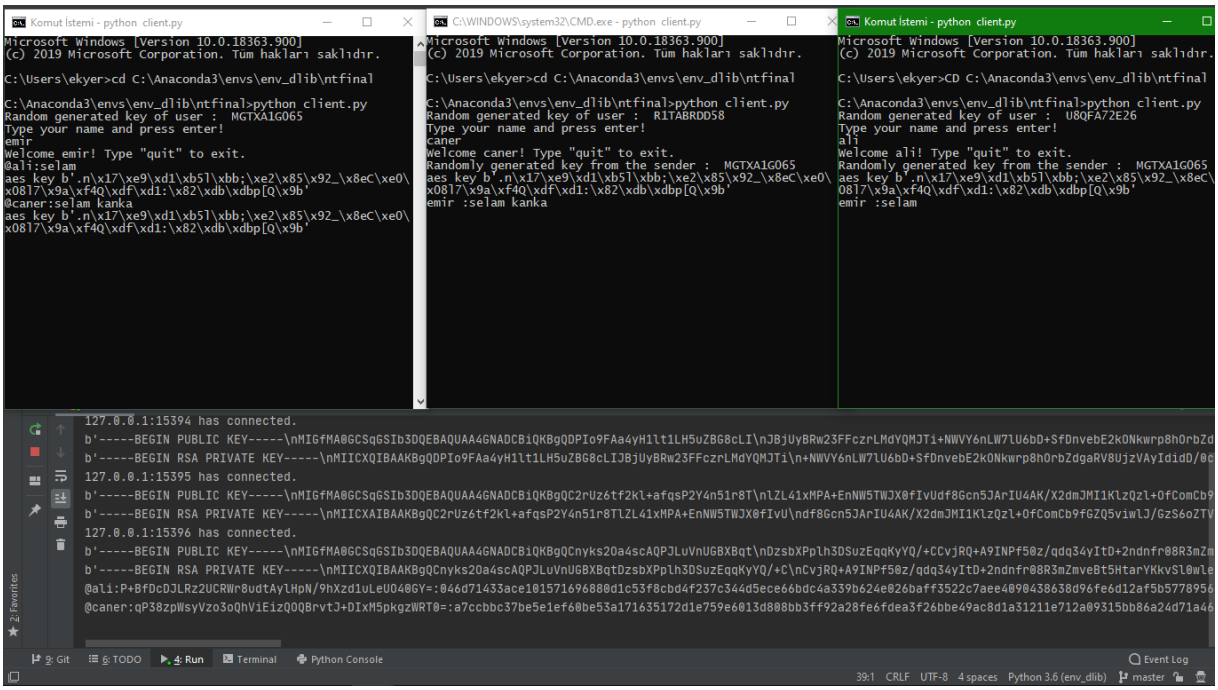

Komut İstemi - python client.py
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. Tüm hakları saklıdır.
C:\Users\ekyer>cd C:\Anaconda3\envs\env_dlib\ntfinal
C:\Anaconda3\envs\env_dlib\ntfinal>python client.py
Random generated key of user : MGTXA1GO65
Type your name and press enter!
emir
Welcome emir! Type "quit" to exit.
C:\WINDOWS\system32\CMD.exe - python client.py
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. Tüm hakları saklıdır.
C:\Users\ekyer>cd C:\Anaconda3\envs\env_dlib\ntfinal
C:\Anaconda3\envs\env_dlib\ntfinal>python client.py
Random generated key of user : R1TABRDD58
Type your name and press enter!
caner
Welcome caner! Type "quit" to exit.
Komut İstemi - python client.py
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. Tüm hakları saklıdır.
C:\Users\ekyer>CD C:\Anaconda3\envs\env_dlib\ntfinal
C:\Anaconda3\envs\env_dlib\ntfinal>python client.py
Random generated key of user : U8QFA72E26
Type your name and press enter!
ali
Welcome ali! Type "quit" to exit.
C:\Anaconda3\envs\env_dlib\python.exe C:/Anaconda3/envs/env_dlib/ntfinal/server.py
Waiting for connection...
127.0.0.1:15394 has connected.
127.0.0.1:15395 has connected.
127.0.0.1:15396 has connected.
9: Git
6: TODO
4: Run
Terminal
Python Console
Event Log
master

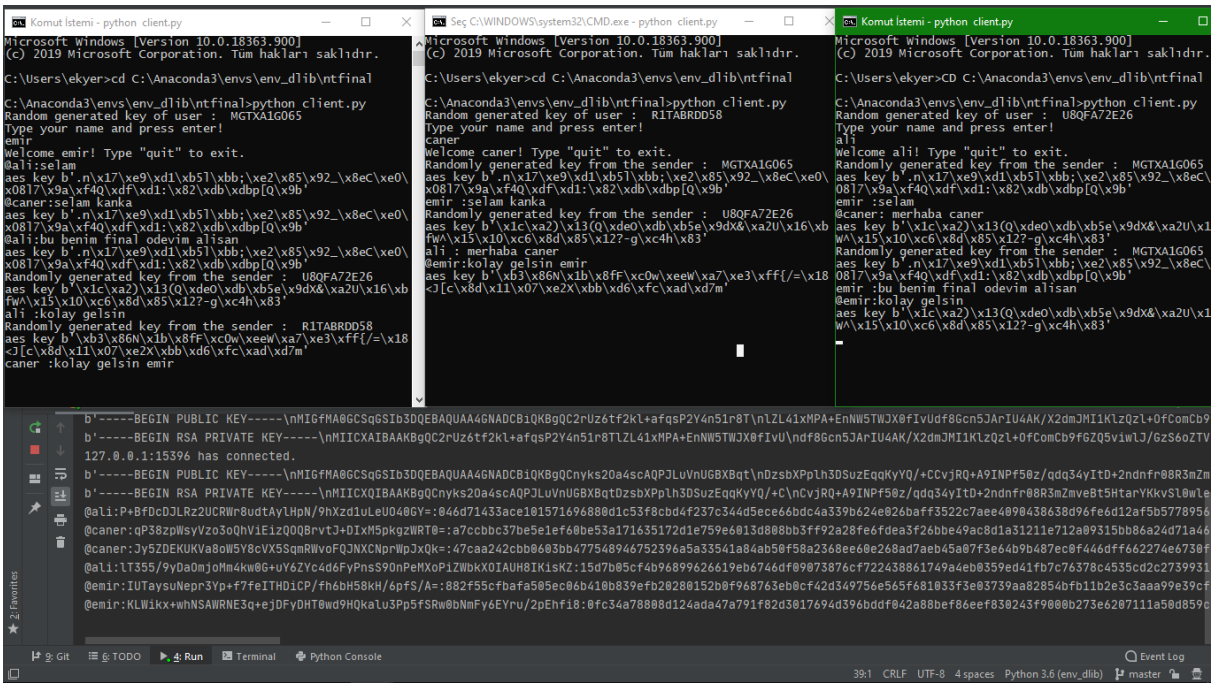

emir_pub.bin - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDPIo9FAa4yH11t1LH5uZBG8cLI
JBjUyBRw23FFczrLMdYQMJTi+NWVY6nLW7lU6bD+SfDnvebE2kONkwrp8hOrbZdg
aRV8UjzVAyIdidD/0ccFPsgCCPt7oXX7vVtxHYxCtekRk/DTWMf3luSVizLucQqh
oodLQGzKKt74aTqiBQIDAQAB
-----END PUBLIC KEY-----
caner_pub.bin - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC2rUz6tf2kl+afqsP2Y4n51r8T
lZL41xMPA+EnNW5TWJX0fIvUdf8Gcn5JArIU4AK/X2dmJMI1KlzQzl+OfComCb9f
GZQ5viwlJ/GzS6oZTVQ00B30cTNXsV3jYaDNq0/yw7v064QFLyZGtyqf/hhurCJW
Wtutm4wl+T40xMOoAwIDAQAB
-----END PUBLIC KEY-----
ali_pub.bin - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCnyks2Oa4scAQPJLuVnUGBXBqt
DzsbXPplh3DSuzEqqKyYQ/+CCvjRQ+A9INPf50z/qdq34yItD+2ndnfr08R3mZmv
eBt5HtarYKkvSl0wle1hFlDNV4iOCFqwZloIidGQYQFESTfrEl23FOk0a2ppIWeB
weGczJsgh3gfB+TXfQIDAQAB
-----END PUBLIC KEY-----


ntfinal
ali_pub.bin
caner_pub.bin
client.py
emir_pub.bin
Encryption.py
server.py

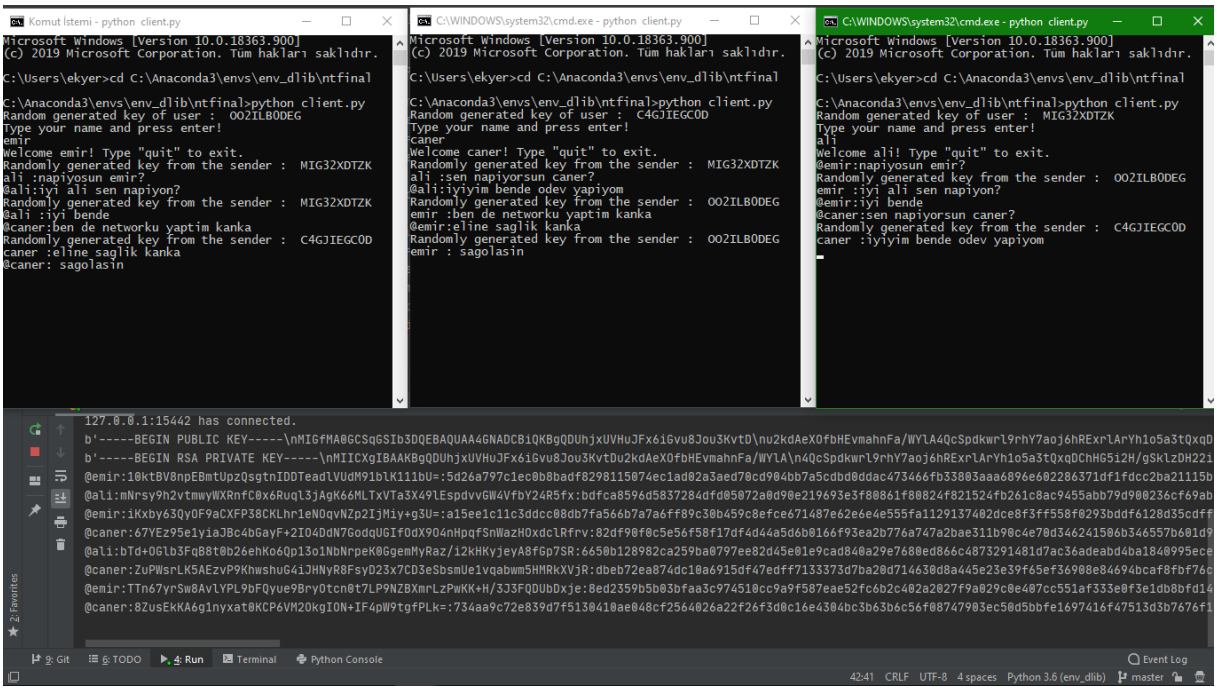
When sending the message, the message is encrypted with AES created using the public key of the person to be sent using session key, and this message is seen in the server. The receiver then deciphers this message using his private key.







Without showing AES keys



5. Codes

Server.py;

```
import socket
from threading import Thread
from Crypto.PublicKey import RSA

clients = {}


def accept_incoming_connections():
    """Sets up handling for incoming clients."""

    while True:
        client, client_address = SERVER.accept()
        print("%s:%s has connected." % client_address)
        client.send(bytes("Type your name and press enter!", "utf8"))
        Thread(target=handle_client, args=(client,)).start()


def handle_client(client):  # Takes client socket as argument.
    """Handles a single client connection."""
    # Key pair for each client
    keyPair = RSA.generate(1024)#private key burada üretiliyor
    pubKey = keyPair.publickey()
    pubKeyPEM = pubKey.exportKey()
    print(pubKeyPEM)
    privKeyPEM = keyPair.exportKey()
    print(privKeyPEM)

    name = client.recv(BUFSIZ).decode("utf8")
    # Write public keys of users into a file
    with open(name + "_pub.bin", "w") as f:
        f.write(pubKeyPEM.decode('ascii'))

    # Send private keys of users to them
    client.send(bytes("@" + privKeyPEM.decode('ascii'), "utf-8"))
    message = 'Welcome %s! Type "quit" to exit.' % name
    client.send(bytes(message, "utf-8"))

    clients[client] = name

    while True:
        msg = client.recv(BUFSIZ)
        if msg != bytes("quit", "utf8"):
            msg = msg.decode()
            print(msg)
            if (msg.startswith('@')):
                parsed_msg = msg.split(":")
                #parsed_msg[0] gönderilen kişi
                #parsed_msg[1] mesajın aes ile şifrelenmiş hali
                #parsed_msg[2] aes şifresinin karşı tarafın public keyi ile şifrelenmiş hali
                to = parsed_msg[0][1:].lower()
                broadcast(bytes(parsed_msg[1] + ":" + parsed_msg[2], "utf-8"), name + " : ", to)
```

```
    else:
        client.send(bytes("quit", "utf-8"))
        client.close()
        del clients[client]
        broadcast(bytes("%s has left the chat.\n" % name, "utf-8"))
        break


def broadcast(msg, name="", to=""):
    for sock in clients:
        if (to != "" and clients[sock] == to):
            sock.send(bytes(name, "utf-8") + msg)
        elif (to == ""):
            # else:
            sock.send(bytes(name, "utf-8") + msg)


LOCALHOST = "127.0.0.1"
PORT = 65534
BUFSIZ = 1024

SERVER = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
SERVER.bind((LOCALHOST, PORT))

SERVER.listen(3)
print("Waiting for connection...")
ACCEPT_THREAD = Thread(target=accept_incoming_connections)
ACCEPT_THREAD.start()
ACCEPT_THREAD.join()
SERVER.close()
```

clien.py

```
import socket
from threading import Thread
from Encryption import AESCipher
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import binascii
import string
import random

# Create random symmetric key
key = ''.join(random.choices(string.ascii_uppercase + string.digits, k=10))
print("Random generated key of user : ", key)
aes = AESCipher(key)
#priv_key = ''


def receive():
    """Handles receiving of messages."""
    while True:
        try:
            msg = client.recv(BUFSIZ).decode("utf-8")
            # print(msg)
            if (msg[0] == "@"):
                priv_key = msg[1:]
            elif (':' in msg):
```

```
            parsed_msg = msg.split(':')
            private_key = RSA.importKey(priv_key)
            decryptor = PKCS1_OAEP.new(private_key)
            decrypted_key = decryptor.decrypt(binascii.unhexlify(bytes(parsed_msg[2], "utf-8")))
            print("Randomly generated key from the sender : ", decrypted_key.decode("utf-8"))
            new_aes = AESCipher(decrypted_key.decode("utf-8"))
            decrypted = new_aes.decrypt(bytes(parsed_msg[1], "utf-8"))
            print(parsed_msg[0] + ":" + decrypted.decode())
        else:
            print(msg)
    except OSError:  # Possibly client has left the chat.
        break


SERVER = "127.0.0.1"
PORT = 65534
BUFSIZ = 1024

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((SERVER, PORT))

receive_thread = Thread(target=receive)
receive_thread.start()

while True:
    msg = input()
    parsed_msg = msg.split(':')
    if (len(parsed_msg) > 1):
        # Take the recievers public key to encrypt symmetric key
        with open(parsed_msg[0][1:] + "_pub.bin", "r") as pub_file:
            pub_key = RSA.importKey(pub_file.read())
            encryptor = PKCS1_OAEP.new(pub_key)
            encrypted_key = encryptor.encrypt(bytes(key, "utf-8"))

        cipher = aes.encrypt(parsed_msg[1]) + bytes(":", "utf-8") + binascii.hexlify(encrypted_key)
        client.send(bytes(parsed_msg[0] + ':', "utf-8") + cipher)
    else:
        client.send(bytes(msg, "utf-8"))

    if msg == "quit":
        client.close()
        break
```

Encryption.py

```
from Cryptodome.Cipher import AES
import hashlib
import base64
from Cryptodome import Random


class AESCipher:  # AES in Cipher Block Chaining Mode

    def __init__(self, key):
        self.block_size = 16
        self.key = hashlib.sha256(key.encode('utf-8')).digest()

    def encrypt(self, raw):
```

```
        raw = pad(raw, self.block_size)
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        #print("aes key", self.key)

        return base64.b64encode(iv + cipher.encrypt(raw.encode('utf-8')))

    def decrypt(self, enc):
        enc = base64.b64decode(enc)
        iv = enc[:16]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        #print("aes key", self.key)

        return unpad(cipher.decrypt(enc[16:]), self.block_size)


# Padding functions
def pad(s, BS):
    return s + (BS - len(s) % BS) * chr(BS - len(s) % BS)


def unpad(s, BS):
    return s[0:-s[-1]]
```

[1] “Encrypted chat with Python, M2Crypto”, https://shanetully.com/2013/05/encrypted-chat-with-python-m2crypto-and-ncurses/, Accessed July 2020.

[2]”Asymmetric-Key Encryption Nedir ve Nerelerde Kullanılır?” https://medium.com/@gokhansengun/ asymmetric-key-encryption-nedir-ve-nerelerde-kullan%C4%B1l%C4%B1r-e658ff7a9acb/, Accessed July 2020

[3]”RSA VS AES ENCRYPTION - A PRIMER” , https://info.townsendsecurity.com/rsa-vs-aes-encryption-a-primer/, Accessed july 2020

[4]“Data Hiding System”, https://github.com/kerimembel/Data-Hiding-System/, Accessed July 2020