SECURE COMMUNICATION FOR SMART IOT OBJECTS

A PROJECT REPORT

Submitted by

N.ANAND VENKATA SUBBA RAJU-19BCE0264

KOTHA.V.V.M SAI DIVYESH-19BCE0841

DURGA SAI RAKESH-19BCE0127

NADIMPALLI LAKSHMI NARASIMHA RAJU-19BCE2247

KOMMINENI BHARGAV-19BCE0322

For the Course

CSE3009

INTERNET OF THINGS(IOT)

Under the guidance of

Dr. VISHNU SRINIVAS MURTHY Y

VIT®
Vellore Institute of Tec

School of Computer Science and Engineering
Vellore Institute of Technology (VIT), Vellore
Tamil Nadu – 632 014
November 2019

CERTIFICATE

This is to certify that N.ANAND VENKATA SUBBA RAJU-19BCE0264,KOTHA.V.V.M SAI DIVYESH-19BCE0841,DURGA SAI RAKESH-19BCE0127,NADIMPALLI LAKSHMI NARASIMHA RAJU-19BCE2247,KOMMINENI BHARGAV-19BCE0322

3rd year B.Tech, (Computer Science & Engineering) from Vellore Institute of Technology (VIT) has successfully completed his project work in the field of Internet of Things (IoT) on the topic SECURE COMMUNICATION FOR SMART IOT OBJECTS. This is a record of his/her own work carried out during the Fall Semester of the Academic Year 2019-20 under the guidance of Dr. Vishnu Srinivasa Murthy Yarlagadda. He has presented his project in the presence of faculty.

Dr. Vishnu Srinivasa Murthy Y,

Assistant Professor / Guide

Acknowledgement

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

First and foremost, I owe my deep gratitude to our IoT professor Dr. Vishnu Srinivasa Murthy Y, who took keen interest on our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good system. I will also not forget to mention my group participants and also friends for the viable information which they provided to me during the course of this project which ultimately lead to amelioration of this project.

Last but not the least, I would also like to thank VIT University, and lectures with the help of which I was able to grasp knowledge for making this project. Also, I would like to thank my classmates and friends who helped me complete this project in such a short time.

NARASIMHA RAJU
DIVYESH
ANAND
BHARGAV
RAKESH

DECEMBER 2021

INDEX

ABSTRACT

The implementation of our project is to, have secure channels of communication between IoT devices with one another and server or a router. There is a developing number of IoT gadgets and applications and this prompts an expansion in the number and unpredictability of pernicious assaults. It is important to secure IoT systems against pernicious assaults, particularly to keep attackers from getting command over the gadgets. An enormousnumber of security research answers for IoT have been proposed in the most recent years, however the greater part of them are not standardized or interoperable. As the internet of things keeps on growing, the variety and multifaceted nature of IoT applications increments. such networks are defenceless against assaults that intend to take touchy information, assume responsibility for gadgets and disturb administrations. numerous conventions and networking stacks for IoT have been created. some of them are standardized, and give interoperability among gadgets and availability over the web. they have been indicated by normalization bodies for example, IETF and IEEE or by industry coalitions, such likewise rawan coalition and string gathering.

Smart-home IoT systems are growing in popularity thanks to their efficient functionality, be making many menial tasks easy. On the opposite hand, these smart home IoT devices becomes Vulnerable points of our privacy. Privacy of nonpublic data, is usually a topmost priority of E-services. To tackle this pain point, we've used industry standard encryption algorithms (like RSA algorithm) for creating and using Secure channels for communication between IoT devices by Socket Programming

INTRODUCTION

Recently, the concept of the web of Things (IoT) has drawn considerable attention from both industry and academia. In the IoT, countless objects with sensors collect data andsend the info to servers that analyze, manage and use the info so as to construct some forms of smart systems, like smart grid, intelligent transportation systems, healthcaresystems and even smart city. it's critical to determine a secure channel between the sensorsand servers so as to make sure the correctness of collected data. If the collected data is tampered, the results of knowledge analysis is unbelievable, and will even bring serious disaster.

Because IoT security remains an afterthought, cybercriminals generally consider smart devices a "low-hanging fruit" – a target easy to compromise and manipulate. Security (and privacy) advisedly is vital for IoT, and doubtless the sole effective way for a sensible gadget to safeguard its communications is to encrypt them. Unfortunately, it's still hard to reconcile convenience with security when it involves lowresource apparatuses. For that reason alone, many IoT products include either ineffective features that encrypt communications and stored data or none in the slightest degree.

According to a 2020 report by a threat intelligence team called Unit 42, 98% of the 1.2 million IoT devices on corporate networks they analyzed had no capability to encrypt traffic. As a result, 57% of those IoT devices were liable to traffic interception andmanipulation, among other things. the identical report further showed that mixing IoT and IT assets on VLAN is also dangerous, as compromised employee IoT devices could spreadmalware onto corporate networks.

In this project we have used also used Mongo DB connect for authentication purposes. In this project we have also used SHA256 algorithm for hashing of passwords such that the person from DB management also cannot see the password as it is hashed. Three propertiesmake SHA-256 this protected.

1) Tobegin with, it ispracticallydifficult to recreatethe underlying information from thehash value. A bruteforce attack would need to make 2256 endeavors to create the underlying information.
2) Second, having two messages with a similar hash value (called a collision) is amazingly improbable. With 2256 potential hash values (more than the quantity of particles in the known universe), the probability of two being the equivalent is imperceptibly, tiny.
3) At long last, a minor change to the original information adjusts the hash value so much that it's not clear the new hash value is gotten from comparative informationthis is known as the avalanche effect.

   We will be using python language for development of this project. There will be inbuilt libraries such that we can use them and maintenance of this project is veryeasy.

LITERATURE SURVEY

1. **Survey on secure communication protocols for the Internet ofThings**

   The Internet of Things (IoT) is planned as an organization of highly connected devices (things). In today viewpoint, the IoT incorporates differing types of devices, e.g., sensors, actuators, RFID tags, which are totally different as far as size, capacity and usefulness. The principle challenge is that the means by which to control such organization so to work within the standard Internet.

   This paper considered various secure, lightweight and assault safe answers for WSNs and IoT captivated with recognized security necessities and difficulties. We likewise gave a totally unique characterization of existing protocols looking on their key bootstrapping because of deal with build up a secure communicating. These protocols and methods are dissected by various measures so on distinguish the preferences and downsides of every protocol.

2. **A security authorization scheme for smart home Internet of Thingsdevices**

Smart-home IoT arrangements are still during a very beginning phase, security being a basic factor which could affect their adoption rate. one in every of the primary difficulties in planning a software security arrangement for smart-home situations is that the way that various elements don't seem to be under user-control. The authorization messages introduced during this paper are made sure about through the FIDO protocol and openingthe FIDO privatekey requires biometric (fingerprint) authentication on the user-device side.The user-to-device authentication depends on the Android security framework and on the smart-telephone equipment security modules (like fingerprint reader or ARM TrustZone).

3. Secure Communication for Smart IoT Objects

From this paper a reasonable security engineering for IoT with the accompanying goals has been learnt:

1) security suites as of now utilized inside UCNswill keep on being utilized without any alterations on the UCN side,

2) starting security handshakes/systems are taken care of diversely inside the CN so constrained hubs can deal with their unpredictability, accordingly having the option to set up start to finish secure channels, though

3) unconstrained hubs will not see any deviation from their standardtechniques.

4) Securing IoT for Smart Home System

All things considered, they have depicted the plan and usage of a Wi-Fi based IoT savvy home framework that utilizes an entryway to empower secure communication between IoT gadgets, and to likewise permit client to arrange, access and control the framework through easy to use interface running on cell phones, for example, the omnipresent PDA. Communication between gadgets is performed dependent on the User Datagram Protocol. Before information is sent, the message is scrambled through symmetric cryptography like Advanced Encryption Standard utilizing the common key made by the ECDH cycle.

5) The Internet of Things: A survey by Luigi Atzori , Antonio Iera , Giacomo Morabito.

This paper addresses the Internet of Things. Main enabling factor of this promising paradigm is the integration of several technologies and communications solutions. Identification and tracking technologies, wired and wireless sensor and actuator networks, enhanced communication protocols (shared with the Next Generation Internet), and distributed intelligence for smart objects are just the most relevant.

6. **IP Security (IPsec) and Internet Key Exchange (IKE) Document by S. Frankel and S. Krishnan.**

   This document is a snapshot of IPsec- and IKE-related RFCs. It includes a brief description of each RFC, along with background information explaining the motivation and context of IPsec's outgrowths and extensions. It obsoletes RFC 2411,the previous "IP Security Document Roadmap."

7. **Datagram Transport Layer Security Version by E. Rescorla and N. Modadugu:**

This document specifies version 1.2 of the Datagram Transport Layer Security (DTLS) protocol. The DTLS protocol provides communications privacy for datagram protocols. The protocol allows client/server applications to communicate in a way that is designedto prevent eavesdropping, tampering, or message forgery.

8. **"Energy Efficiency in M2M Networks Y. B. Saied, A. Olivereau, and D.Zeghlache.**
   Security requirements for the integration of emerging M2M networks in future internet of things are addressed. The heterogeneous nature of M2M devices raisesnew security challenges that existing proposals could not fulfill. Two entities may not be able to establish a secure end-to-end communication because of the technological gap between them and the resulting inconsistencies in their cryptographic primitives.

9. **Security trends in Internet of things:**

The Internet of things is a network of embedded devices that are uniquely identifiable and have embedded software required to communicate between transient states .The purpose of this study is to explore discrete IOT security challenges pertaining to currently deployed IOT standards and protocols. We have presented a detailed review in this study that focuses on IOT's imminent security aspects , covering identification of risks pertaining to the current IOT system, novel security problems, security projects offered in recent years.

10. **Understanding the unique dynamics of IOT security:**

With so many unsecured internet of things devices flooding enterprise networks, the potential for compromise is increasing dramatically. An IOT related breach can be disaster, both financially and reputation for an organisation of any size. As a result businesses are recognizing that IOT cybersecurity is no longer a luxury but an essential investment in long term operational sustainability.

OVERVIEW OF WORK

Architecture and methodology

Security (and privacy) by design is key for IoT, and probably the only effective way for a smart gadget to protect its communications is to encrypt them. Unfortunately, it is still not easy to reconcile convenience with security when it comes to low-resource apparatuses.For that reason alone, many IoT products come with either ineffective features that encrypt communications and stored data or none at all.

According to a 2020 report by a threat intelligence team called Unit 42, 98% of the 1.2 million IoT devices on corporate networks they analyzed had no capability to encrypt traffic. As a result, 57% of these IoT devices were susceptible to traffic interception and manipulation, among other things. The same report further showed that mixing IoT and IT assets on VLAN may be dangerous, as compromised employee IoT devices could spread malware onto corporate networks.

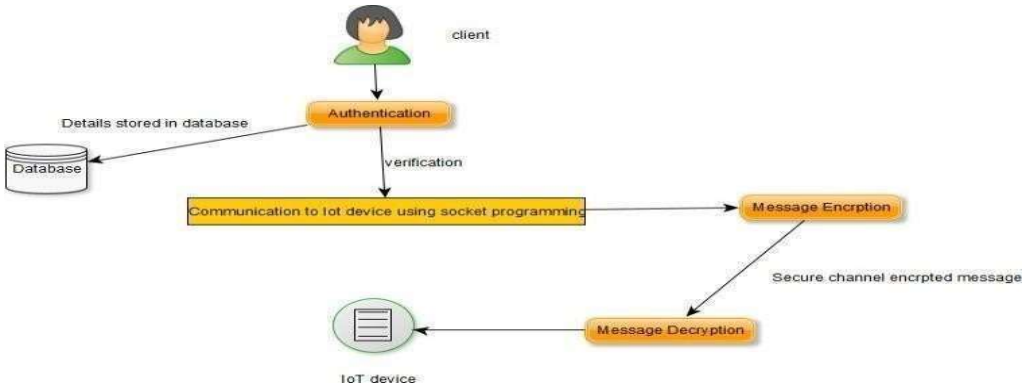
Because of these vulnerabilities and because of the communications being intercepted and manipulated by the hackers the data coming to the IoT device won't be correct and

resulting in malfunction of IoT device and wrong results if a company depending on the IoT device resulting data then those will be in Risk as the data traffic can be manipulated. Hence we came up with the idea of encrypting communication messages between IoT device and client hence it impossible for the hacker to masquerade the traffic.

Advantages

1) it simultaneously achieves confidentiality, integrity, authentication, non-repudiation and anonymity in a logical single step;
2) it is heterogeneous and allows a sensor node in an identity-based cryptography to send a message to a server in a public key infrastructure. These features make our scheme suitable for data transmission in the IoT.

Flow diagram



Software requirements

1)Python programming

2)Mongodb

3) Python inbuilt libraries 4)Socket programming in python

Hardware requirements

There are no specific hardware requirements for this project.

IMPLEMENTATION

Project modules

1) Socket Programming

The IoT device and client will be communicating between each other with **the socket** programming it's a two way broadcasting message can be sent from either side of the devices.

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Creating the socket:

s = socket.socket (socket_family, socket_type, protocol=0)

Server Socket Methods

s.bind()

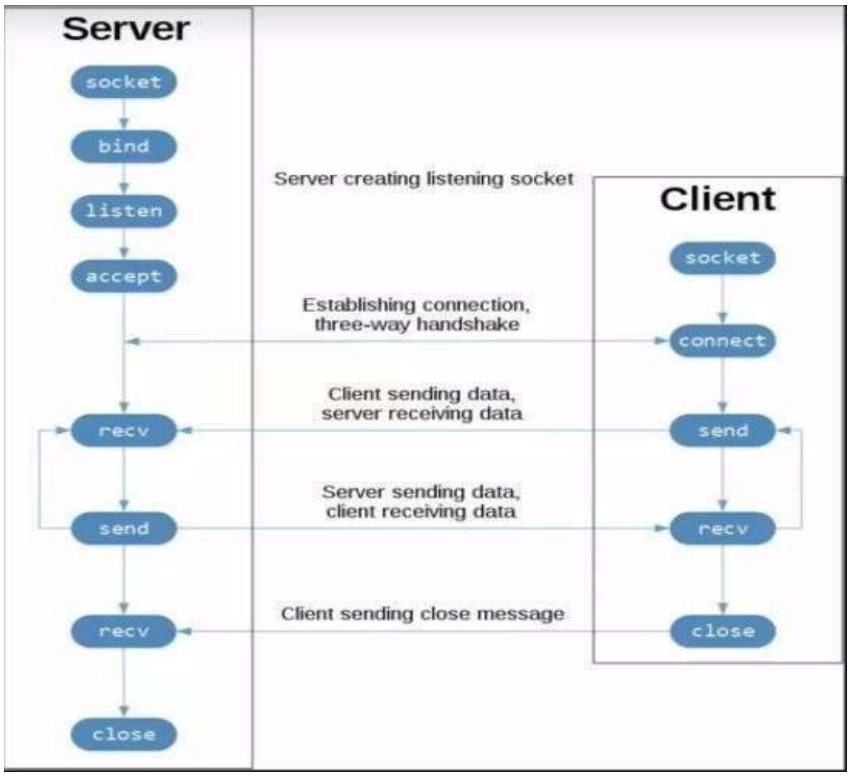
s.listen()

s.accept()

client socket methods

s.connect()

Flow chart



Using socket programming the messages and communication between client and the server Is done

2) Encrypting Iot messages using RSA Algorithm

To prevent the attacks to steal the messages and data we need to encrypt the messages suchthat it should be hard to the hackers to decrypt the traffic .Here In this project we are using RSA algorithm for encrypting the data the algorithm is almost impossible to break. It is Asymmetric key cryptobgraphic algorithm based on public key encryption

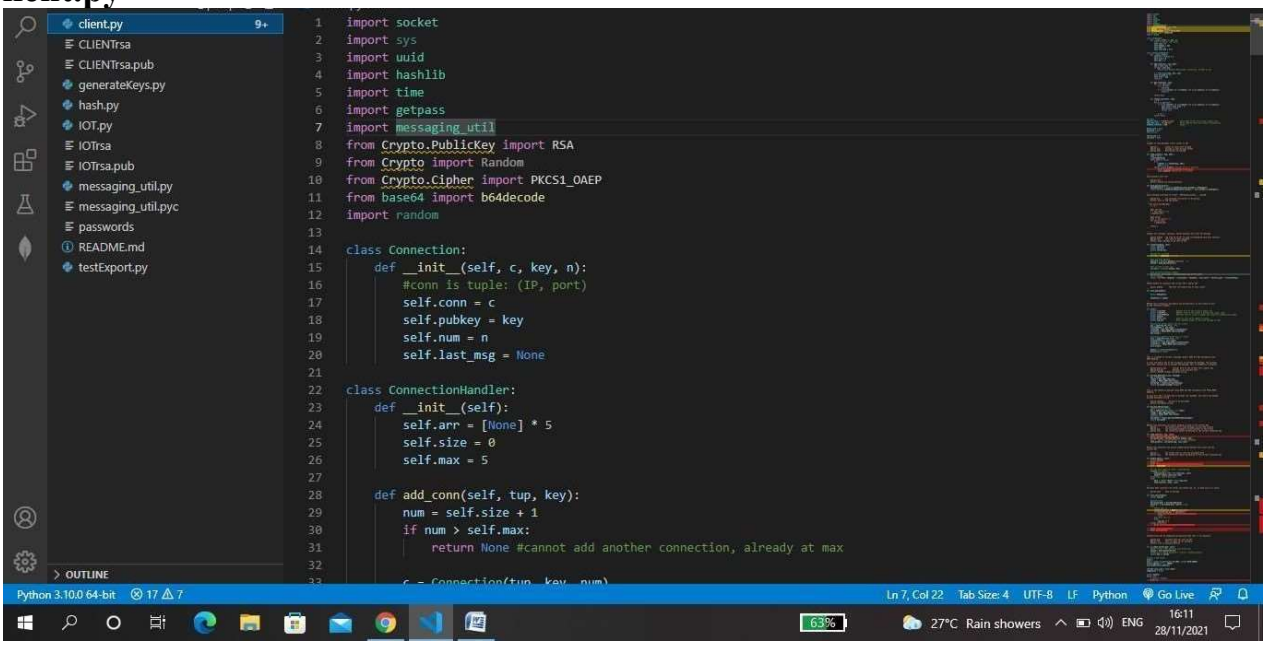
3) User Authentication and stored our details in MongoDB

In our method, we also added another feature for authentication of the user whose details are stored in the database we used MongoDB for storing the details this databaseeasy to use and this is highly scalable. Hence we used MongoDB database. Concerning the security reasons we also hashed the passwords entered by the client and then we stored those in the database.

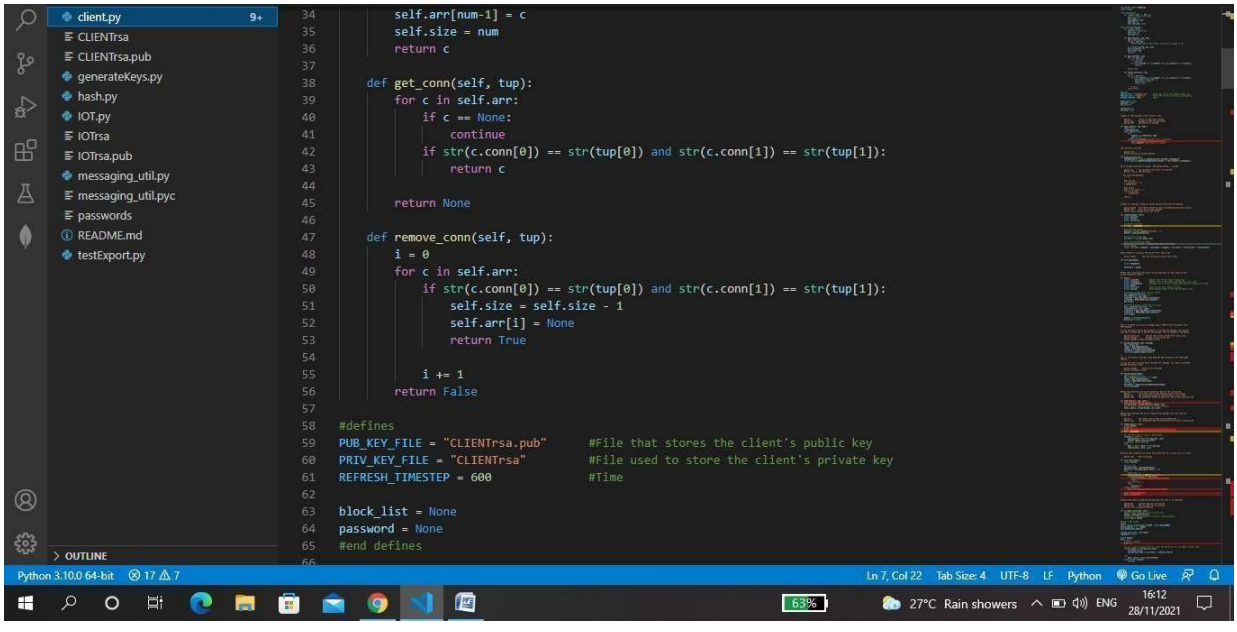
CODE:

Client code:

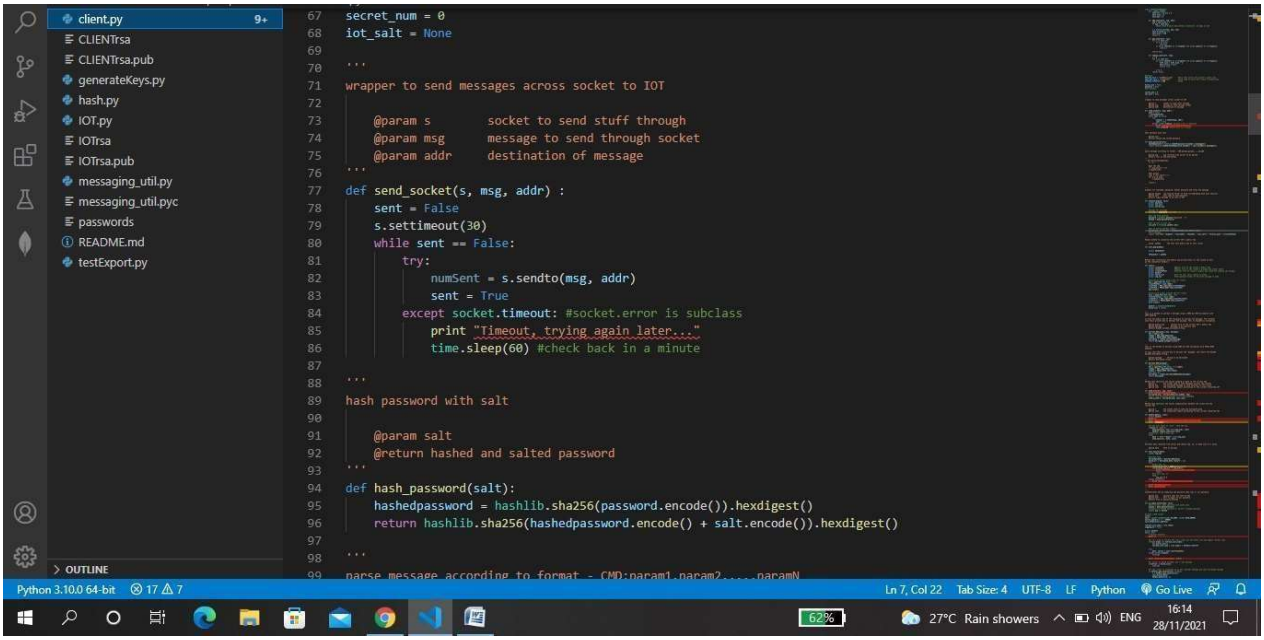
client.py

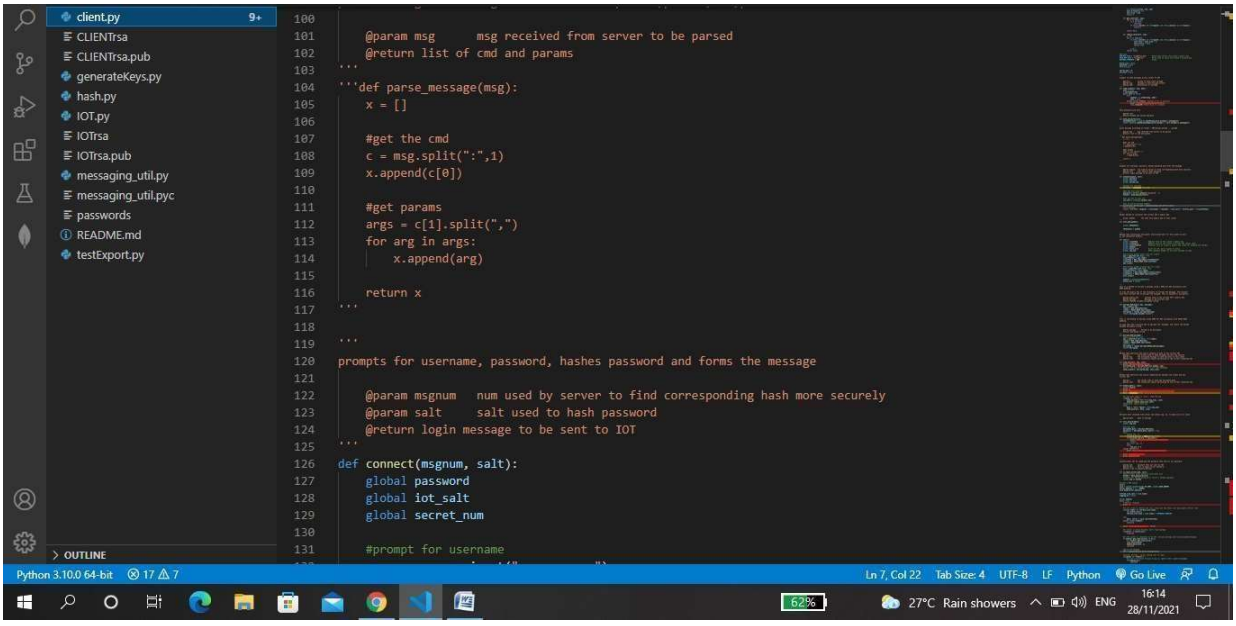


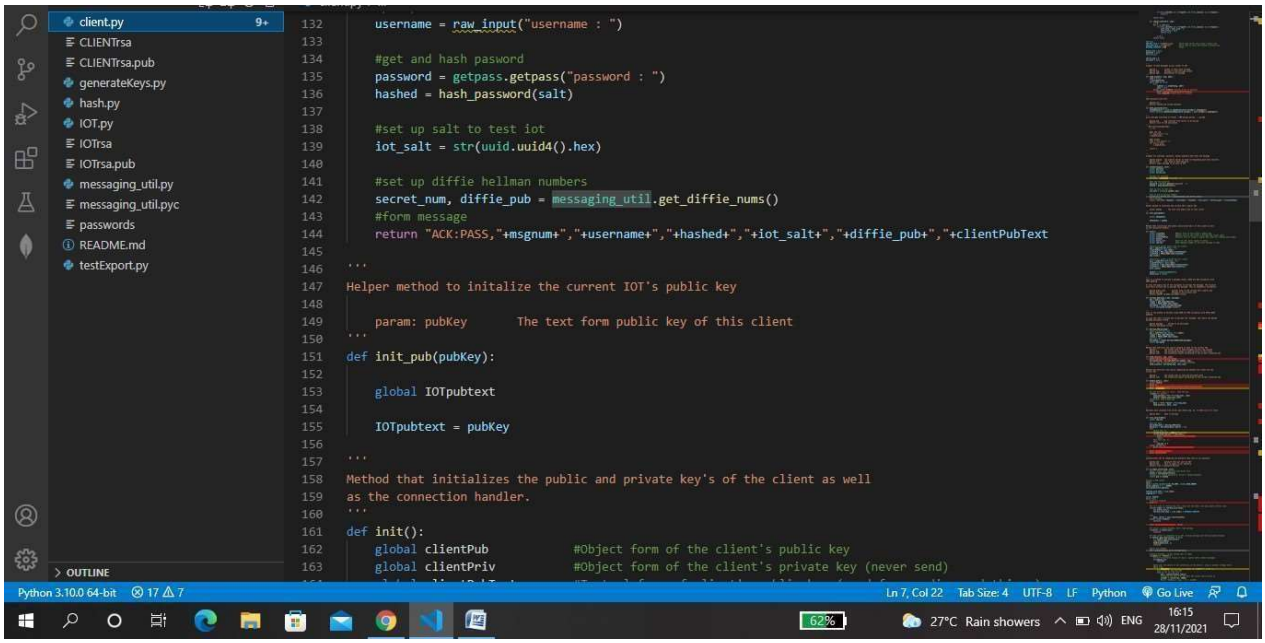
```
import socket
import sys
import uuid
import hashlib
import time
import getpass
import messaging_util
from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import PKCS1_OAEP
from base64 import b64decode
import random

class Connection:
    def __init__(self, c, key, n):
        #conn is tuple: (IP, port)
        self.conn = c
        self.pubkey = key
        self.num = n
        self.last_msg = None

class ConnectionHandler:
    def __init__(self):
        self.arr = [None] * 5
        self.size = 0
        self.max = 5

    def add_conn(self, tup, key):
        num = self.size + 1
        if num > self.max:
            return None #cannot add another connection, already at max
```

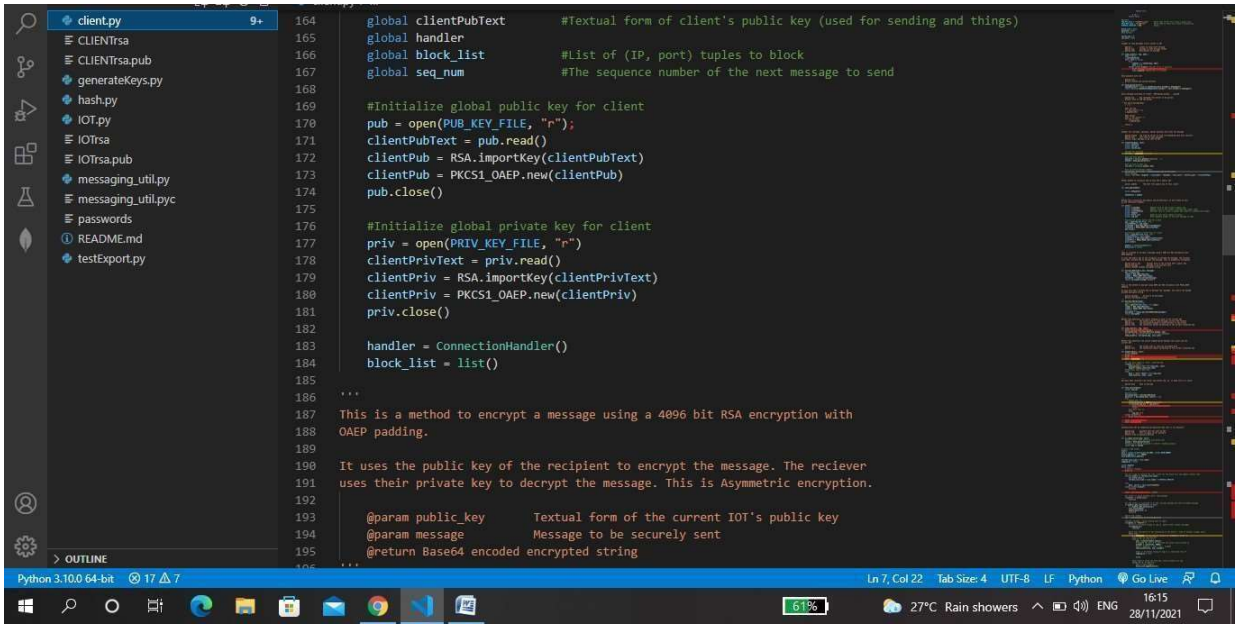


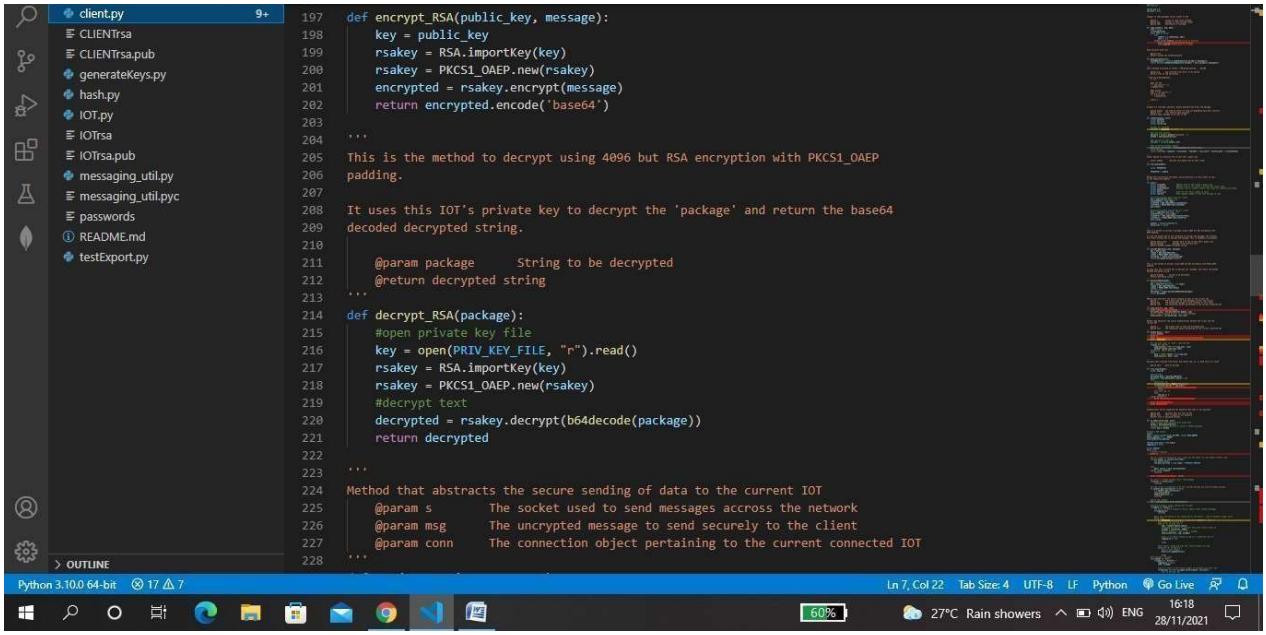
```
        self.arr[num-1] = c
        self.size = num
        return c

    def get_conn(self, tup):
        for c in self.arr:
            if c == None:
                continue
            if str(c.conn[0]) == str(tup[0]) and str(c.conn[1]) == str(tup[1]):
                return c

        return None

    def remove_conn(self, tup):
        i = 0
        for c in self.arr:
            if str(c.conn[0]) == str(tup[0]) and str(c.conn[1]) == str(tup[1]):
                self.size = self.size - 1
                self.arr[i] = None
                return True

            i += 1
        return False

#defines
PUB_KEY_FILE = "CLIENTrsa.pub"      #File that stores the client's public key
PRIV_KEY_FILE = "CLIENTrsa"         #File used to store the client's private key
REFRESH_TIMESTEP = 600              #Time

block_list = None
password = None
#end defines
```

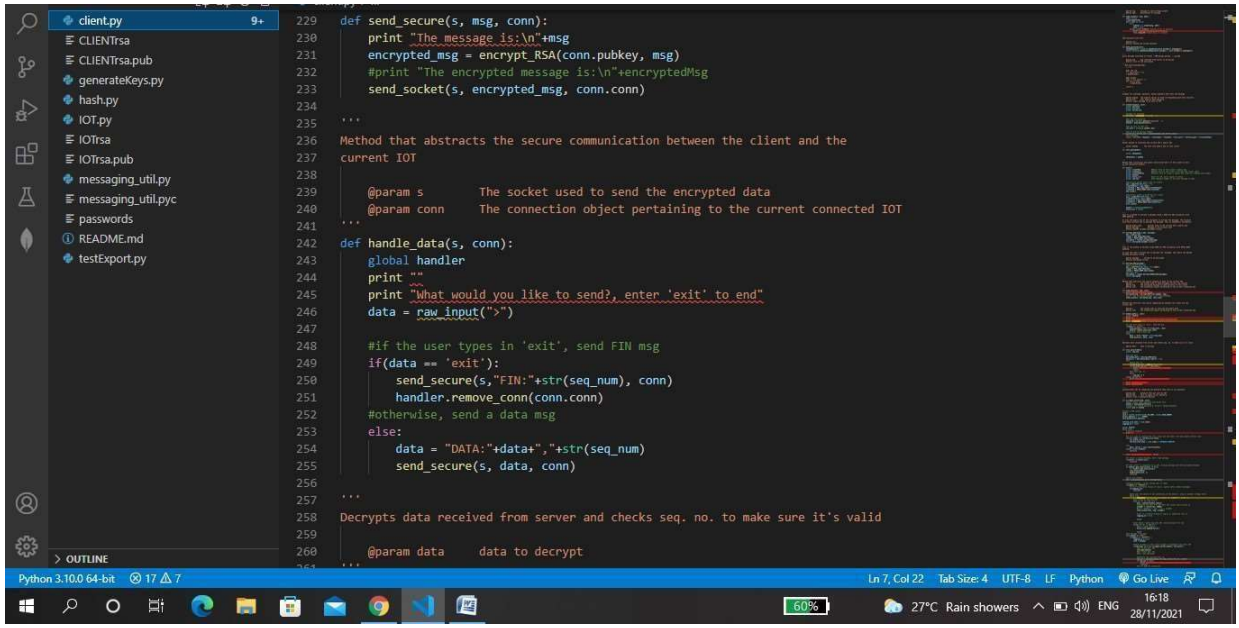

client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
secret_num = 0
iot_salt = None
'''
wrapper to send messages across socket to IOT
@param s socket to send stuff through
@param msg message to send through socket
@param addr destination of message
'''
def send_socket(s, msg, addr) :
sent = False
s.settimeout(30)
while sent == False:
try:
numSent = s.sendto(msg, addr)
sent = True
except socket.timeout: #socket.error is subclass
print "Timeout, trying again later..."
time.sleep(60) #check back in a minute
'''
hash password with salt
@param salt
@return hashed and salted password
'''
def hash_password(salt):
hashedpassword = hashlib.sha256(password.encode()).hexdigest()
return hashlib.sha256(hashedpassword.encode() + salt.encode()).hexdigest()
'''
parse message according to format - CMD:param1,param2.....paramN
Python 3.10.0 64-bit
Ln 7, Col 22
Tab Size: 4
UTF-8
LF
Python
Go Live
62%
27°C Rain showers
ENG
16:14
28/11/2021

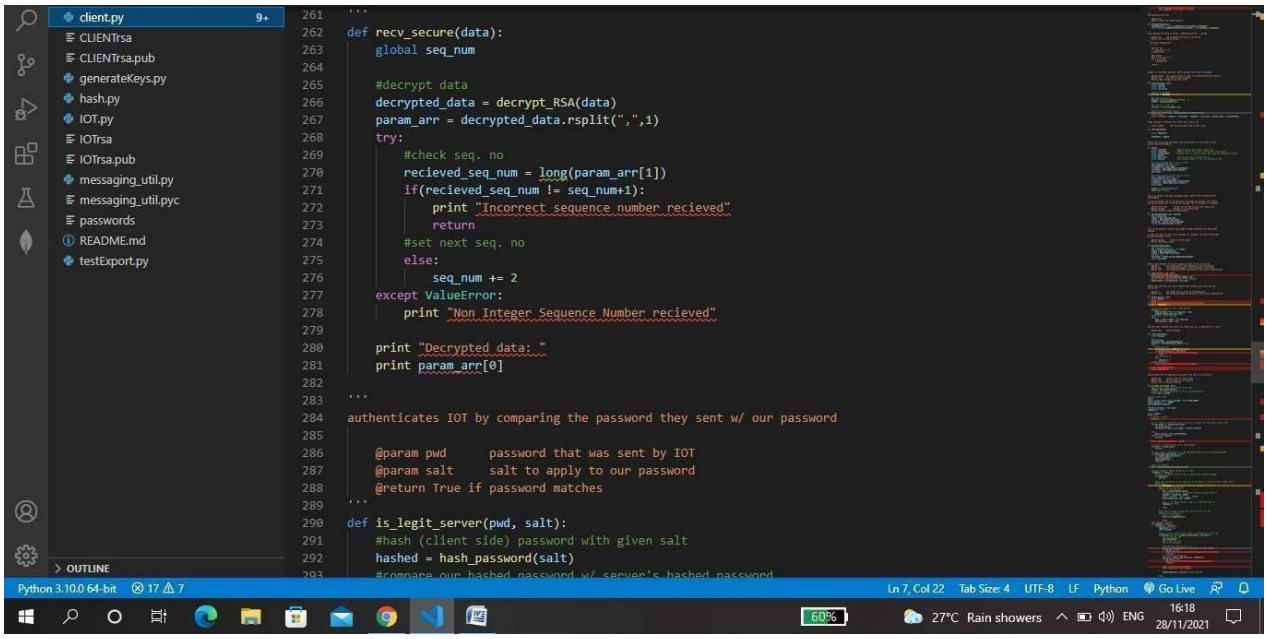

client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
@param msg msg received from server to be parsed
@return list of cmd and params
'''
'''def parse_message(msg):
x = []
#get the cmd
c = msg.split(":",1)
x.append(c[0])
#get params
args = c[1].split(",")
for arg in args:
x.append(arg)
return x
'''
'''
prompts for username, password, hashes password and forms the message
@param msgnum num used by server to find corresponding hash more securely
@param salt salt used to hash password
@return login message to be sent to IOT
'''
def connect(msgnum, salt):
global password
global iot_salt
global secret_num
#prompt for username
Python 3.10.0 64-bit
Ln 7, Col 22
Tab Size: 4
UTF-8
LF
Python
Go Live
62%
27°C Rain showers
ENG
16:14
28/11/2021

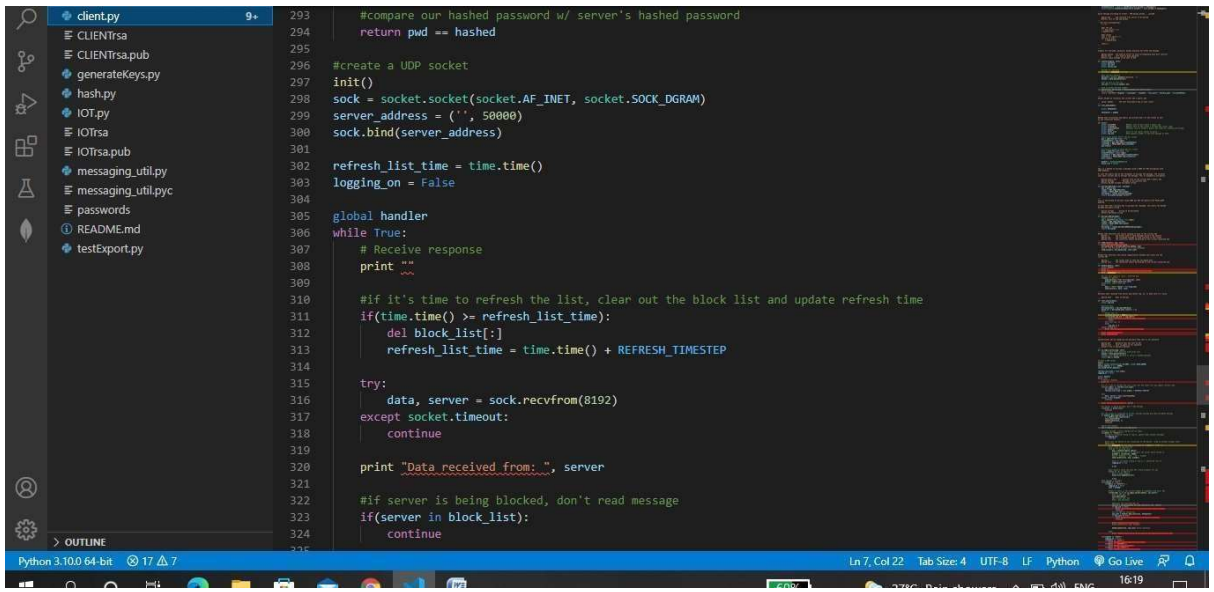

client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
username = raw_input("username : ")
#get and hash pasword
password = getpass.getpass("password : ")
hashed = hash_password(salt)
#set up salt to test iot
iot_salt = str(uuid.uuid4().hex)
#set up diffie hellman numbers
secret_num, diffie_pub = messaging_util.get_diffie_nums()
#form message
return "ACK:PASS,"+msgnum+","+username+","+hashed+","+iot_salt+","+diffie_pub+","+clientPubText
'''
Helper method to initalize the current IOT's public key
param: pubKey The text form public key of this client
'''
def init_pub(pubKey):
global IOTpubtext
IOTpubtext = pubKey
'''
Method that initializes the public and private key's of the client as well
as the connection handler.
'''
def init():
global clientPub #Object form of the client's public key
global clientPriv #Object form of the client's private key (never send)
Python 3.10.0 64-bit ⊗ 17 ⚠ 7
Ln 7, Col 22 Tab Size: 4 UTF-8 LF Python Go Live
62%
27°C Rain showers
ENG
16:15
28/11/2021

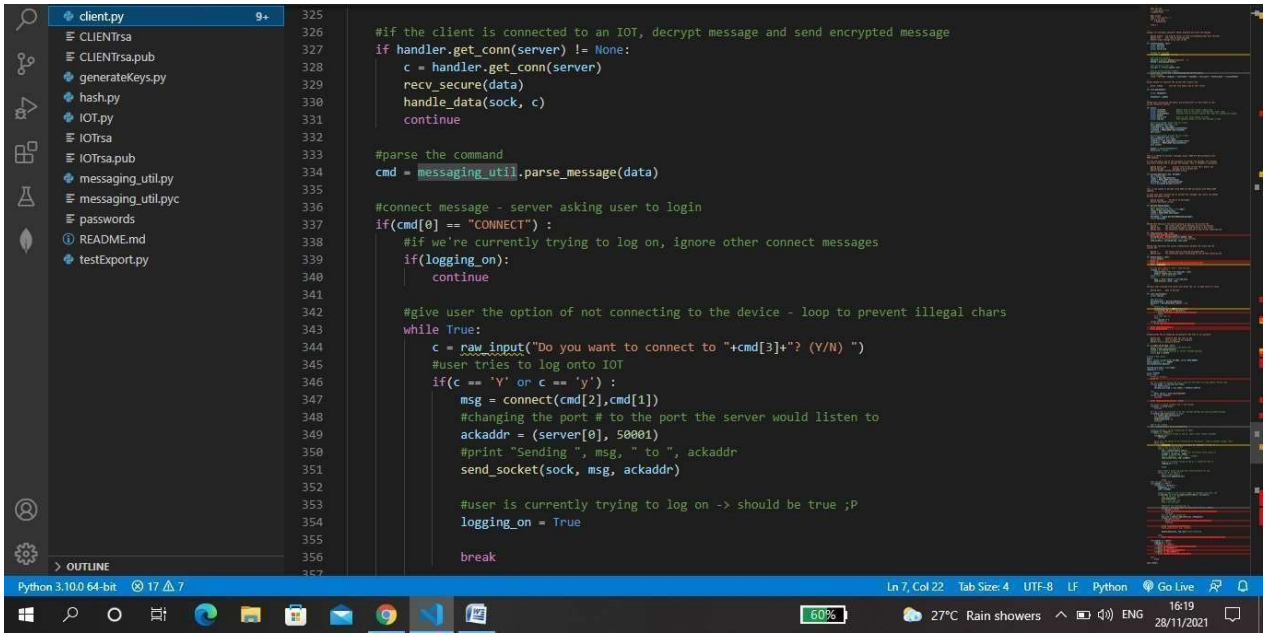

client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
global clientPubText #Textual form of client's public key (used for sending and things)
global handler
global block_list #List of (IP, port) tuples to block
global seq_num #The sequence number of the next message to send
#Initialize global public key for client
pub = open(PUB_KEY_FILE, "r");
clientPubText = pub.read()
clientPub = RSA.importKey(clientPubText)
clientPub = PKCS1_OAEP.new(clientPub)
pub.close()
#Initialize global private key for client
priv = open(PRIV_KEY_FILE, "r")
clientPrivText = priv.read()
clientPriv = RSA.importKey(clientPrivText)
clientPriv = PKCS1_OAEP.new(clientPriv)
priv.close()
handler = ConnectionHandler()
block_list = list()
'''
This is a method to encrypt a message using a 4096 bit RSA encryption with
OAEP padding.
It uses the public key of the recipient to encrypt the message. The reciever
uses their private key to decrypt the message. This is Asymmetric encryption.
@param public_key Textual form of the current IOT's public key
@param message Message to be securely sent
@return Base64 encoded encrypted string
Python 3.10.0 64-bit ⊗ 17 ⚠ 7
Ln 7, Col 22 Tab Size: 4 UTF-8 LF Python Go Live
61%
27°C Rain showers
ENG
16:15
28/11/2021

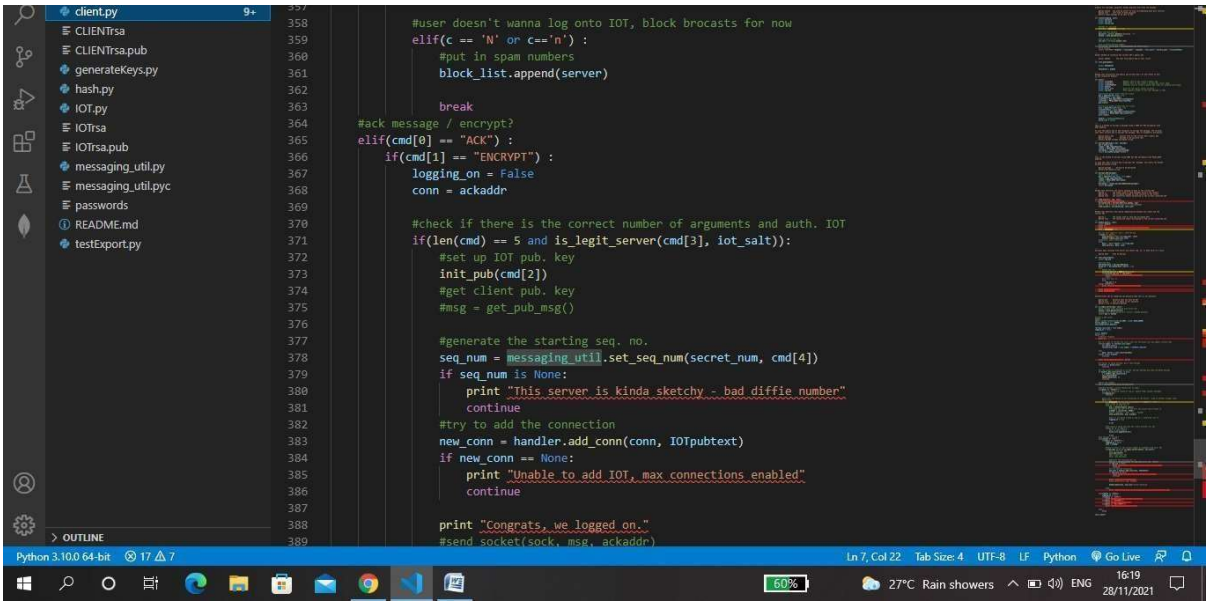

client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
def encrypt_RSA(public_key, message):
key = public_key
rsakey = RSA.importKey(key)
rsakey = PKCS1_OAEP.new(rsakey)
encrypted = rsakey.encrypt(message)
return encrypted.encode('base64')
'''
This is the method to decrypt using 4096 but RSA encryption with PKCS1_OAEP
padding.
It uses this IOT's private key to decrypt the 'package' and return the base64
decoded decrypted string.
@param package String to be decrypted
@return decrypted string
'''
def decrypt_RSA(package):
#open private key file
key = open(PRIV_KEY_FILE, "r").read()
rsakey = RSA.importKey(key)
rsakey = PKCS1_OAEP.new(rsakey)
#decrypt text
decrypted = rsakey.decrypt(b64decode(package))
return decrypted
'''
Method that abstracts the secure sending of data to the current IOT
@param s The socket used to send messages accross the network
@param msg The uncrypted message to send securely to the client
@param conn The connection object pertaining to the current connected IOT
'''
Python 3.10.0 64-bit ⊗ 17 ⚠ 7
Ln 7, Col 22 Tab Size: 4 UTF-8 LF Python Go Live
60%
27°C Rain showers
ENG
16:18
28/11/2021

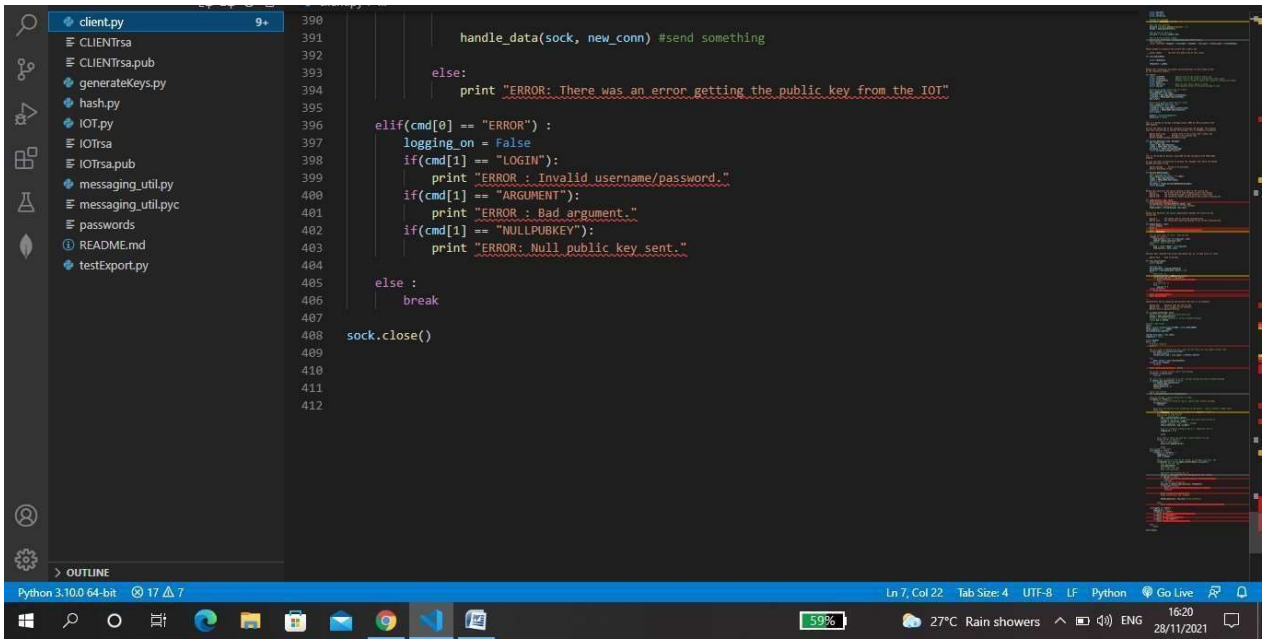

client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
def send_secure(s, msg, conn):
print "The message is:\n"+msg
encrypted_msg = encrypt_RSA(conn.pubkey, msg)
#print "The encrypted message is:\n"+encryptedMsg
send_socket(s, encrypted_msg, conn.conn)
'''
Method that abstracts the secure communication between the client and the
current IOT
@param s The socket used to send the encrypted data
@param conn The connection object pertaining to the current connected IOT
'''
def handle_data(s, conn):
global handler
print ""
print "What would you like to send?, enter 'exit' to end"
data = raw_input(">")
#if the user types in 'exit', send FIN msg
if(data == 'exit'):
send_secure(s,"FIN:"+str(seq_num), conn)
handler.remove_conn(conn.conn)
#otherwise, send a data msg
else:
data = "DATA:"+data+","+str(seq_num)
send_secure(s, data, conn)
'''
Decrypts data received from server and checks seq. no. to make sure it's valid
@param data data to decrypt
'''
Python 3.10.0 64-bit ⊗ 17 ⚠ 7
Ln 7, Col 22 Tab Size: 4 UTF-8 LF Python Go Live
60%
27°C Rain showers
ENG
16:18
28/11/2021


client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
'''
def recv_secure(data):
global seq_num
#decrypt data
decrypted_data = decrypt_RSA(data)
param_arr = decrypted_data.rsplit(",",1)
try:
#check seq. no
recieved_seq_num = long(param_arr[1])
if(recieved_seq_num != seq_num+1):
print "Incorrect sequence number recieved"
return
#set next seq. no
else:
seq_num += 2
except ValueError:
print "Non Integer Sequence Number recieved"
print "Decrypted data: "
print param_arr[0]
'''
authenticates IOT by comparing the password they sent w/ our password
@param pwd password that was sent by IOT
@param salt salt to apply to our password
@return True if password matches
'''
def is_legit_server(pwd, salt):
#hash (client side) password with given salt
hashed = hash_password(salt)
#compare our hashed password w/ server's hashed password
Python 3.10.0 64-bit ⊗ 17 ⚠ 7
Ln 7, Col 22 Tab Size: 4 UTF-8 LF Python Go Live
60%
27°C Rain showers
ENG
16:18
28/11/2021

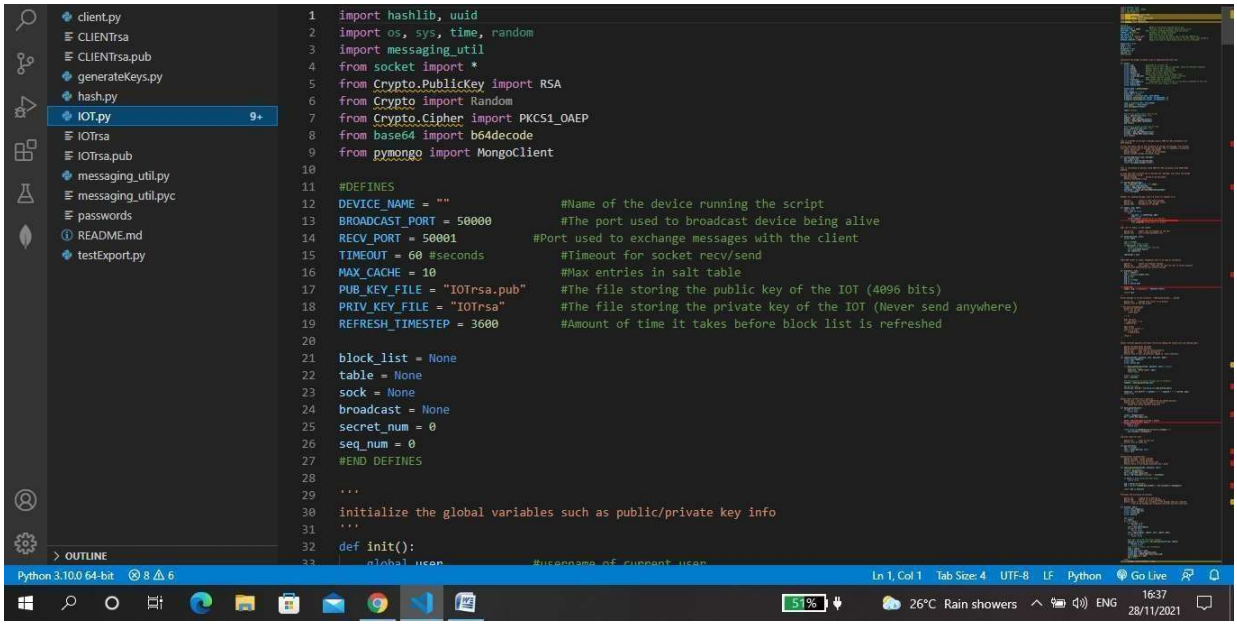

client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
#compare our hashed password w/ server's hashed password
return pwd == hashed
#create a UDP socket
init()
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_address = ('', 50000)
sock.bind(server_address)
refresh_list_time = time.time()
logging_on = False
global handler
while True:
# Receive response
print ""
#if it's time to refresh the list, clear out the block list and update refresh time
if(time.time() >= refresh_list_time):
del block_list[:]
refresh_list_time = time.time() + REFRESH_TIMESTEP
try:
data, server = sock.recvfrom(8192)
except socket.timeout:
continue
print "Data received from: ", server
#if server is being blocked, don't read message
if(server in block_list):
continue
Python 3.10.0 64-bit ⊗ 17 ⚠ 7
Ln 7, Col 22 Tab Size: 4 UTF-8 LF Python Go Live
16:19

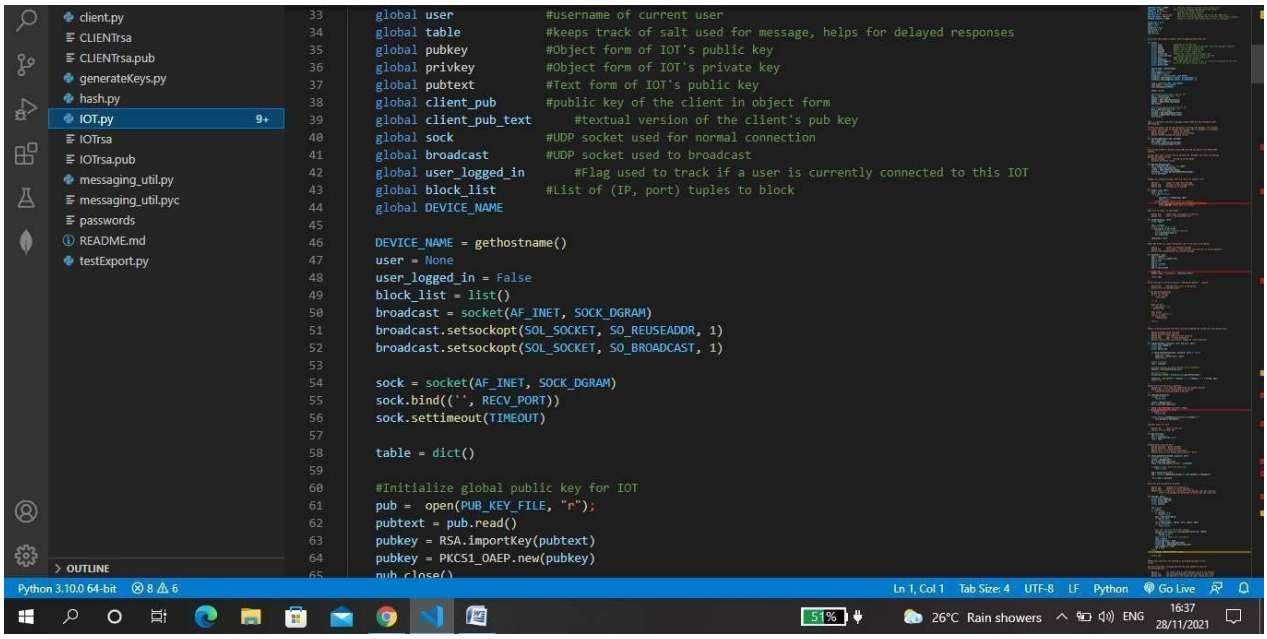

client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
#if the client is connected to an IOT, decrypt message and send encrypted message
if handler.get_conn(server) != None:
c = handler.get_conn(server)
recv_secure(data)
handle_data(sock, c)
continue
#parse the command
cmd = messaging_util.parse_message(data)
#connect message - server asking user to login
if(cmd[0] == "CONNECT") :
#if we're currently trying to log on, ignore other connect messages
if(logging_on):
continue
#give user the option of not connecting to the device - loop to prevent illegal chars
while True:
c = raw_input("Do you want to connect to "+cmd[3]+"? (Y/N) ")
#user tries to log onto IOT
if(c == 'Y' or c == 'y') :
msg = connect(cmd[2],cmd[1])
#changing the port # to the port the server would listen to
ackaddr = (server[0], 50001)
#print "Sending ", msg, " to ", ackaddr
send_socket(sock, msg, ackaddr)
#user is currently trying to log on -> should be true ;P
logging_on = True
break
Python 3.10.0 64-bit ⊗ 17 ⚠ 7
Ln 7, Col 22 Tab Size: 4 UTF-8 LF Python Go Live
60%
27°C Rain showers
ENG
16:19
28/11/2021


client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
#user doesn't wanna log onto IOT, block brocasts for now
elif(c == 'N' or c=='n') :
#put in spam numbers
block_list.append(server)
break
#ack message / encrypt?
elif(cmd[0] == "ACK") :
if(cmd[1] == "ENCRYPT") :
logging_on = False
conn = ackaddr
#check if there is the correct number of arguments and auth. IOT
if(len(cmd) == 5 and is_legit_server(cmd[3], iot_salt)):
#set up IOT pub. key
init_pub(cmd[2])
#get client pub. key
#msg = get_pub_msg()
#generate the starting seq. no.
seq_num = messaging_util.set_seq_num(secret_num, cmd[4])
if seq_num is None:
print "This server is kinda sketchy - bad diffie number"
continue
#try to add the connection
new_conn = handler.add_conn(conn, IOTpubtext)
if new_conn == None:
print "Unable to add IOT, max connections enabled"
continue
print "Congrats, we logged on."
#send_socket(sock, msg, ackaddr)
Python 3.10.0 64-bit ⊗ 17 ⚠ 7
Ln 7, Col 22 Tab Size: 4 UTF-8 LF Python Go Live
60%
27°C Rain showers
ENG
16:19
28/11/2021

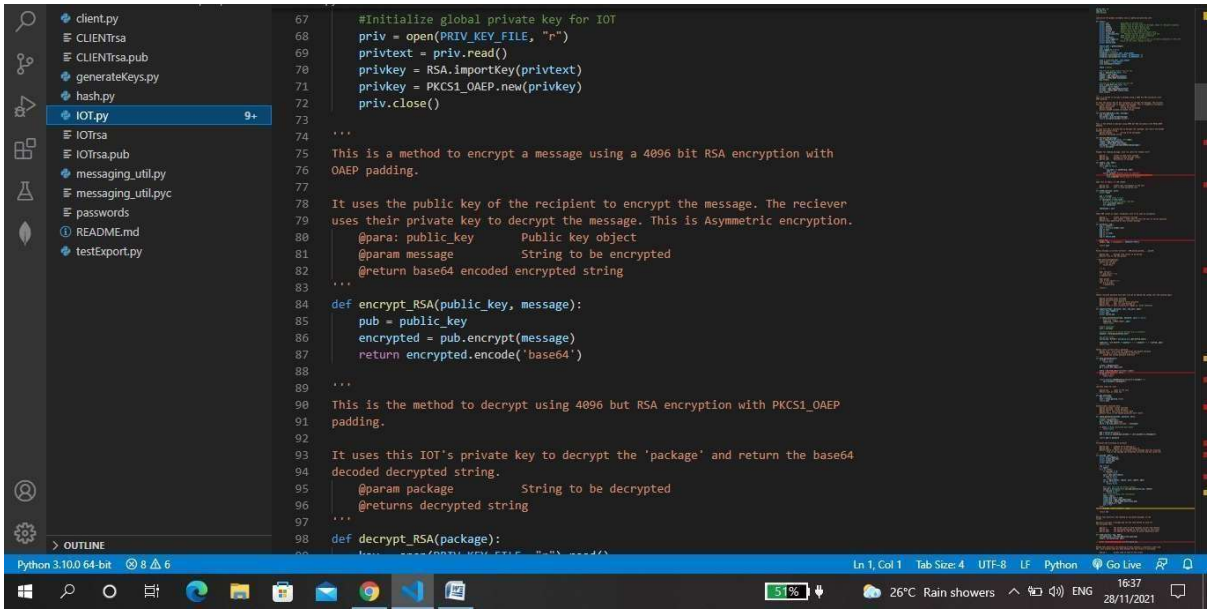


IOT CODE:
Iot.py:

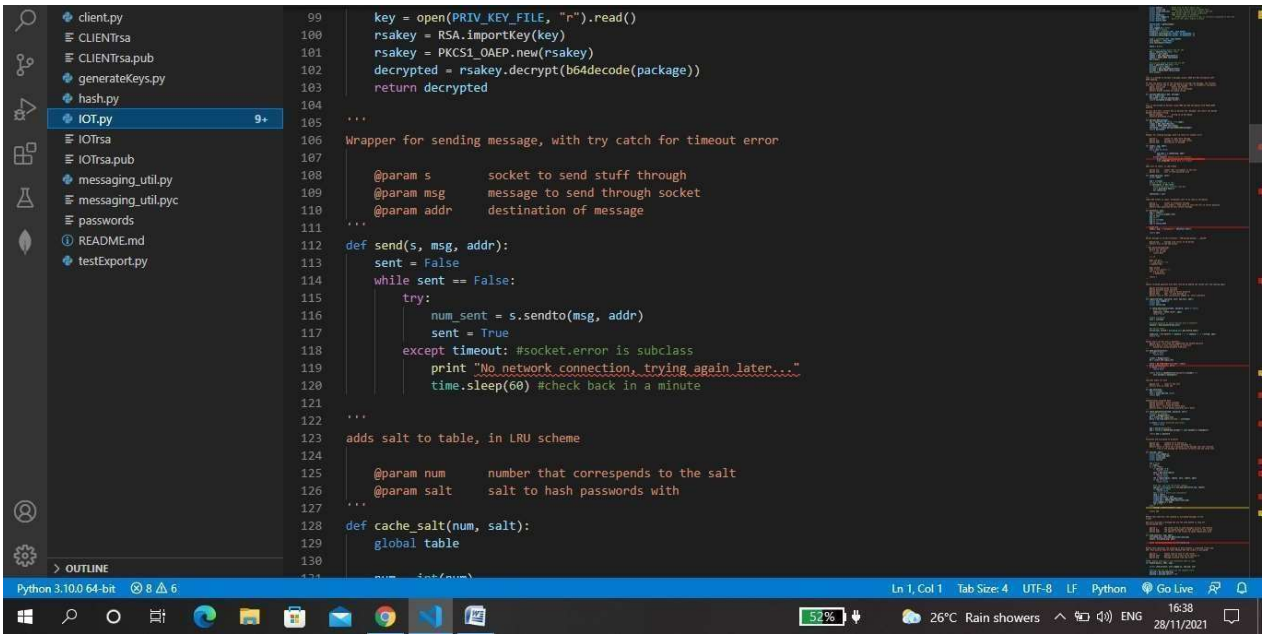


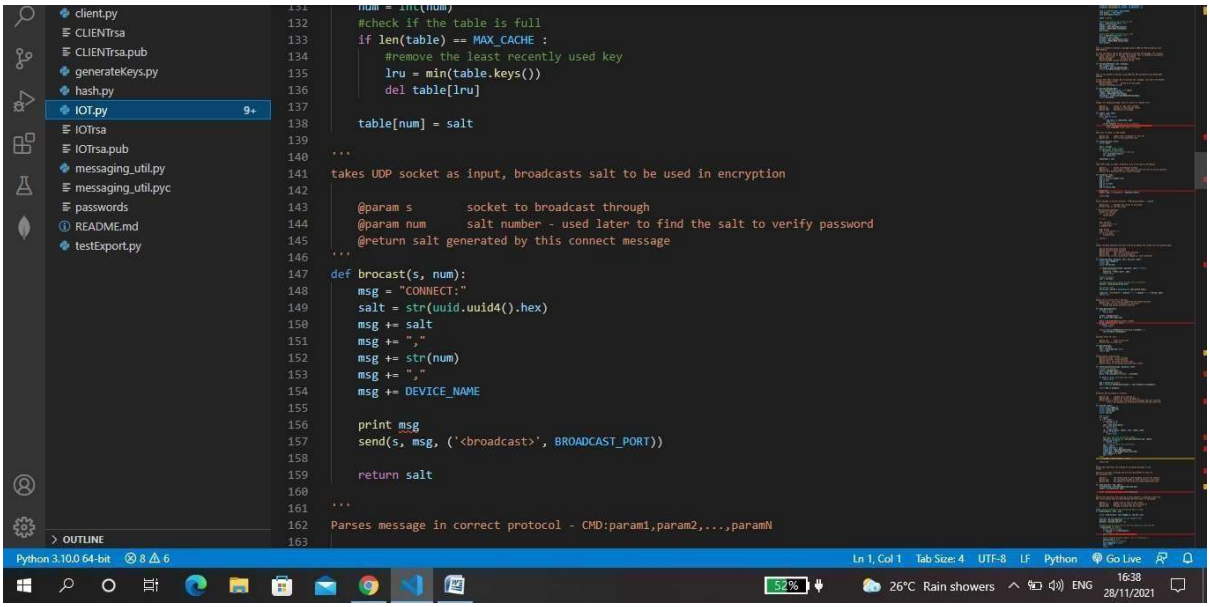


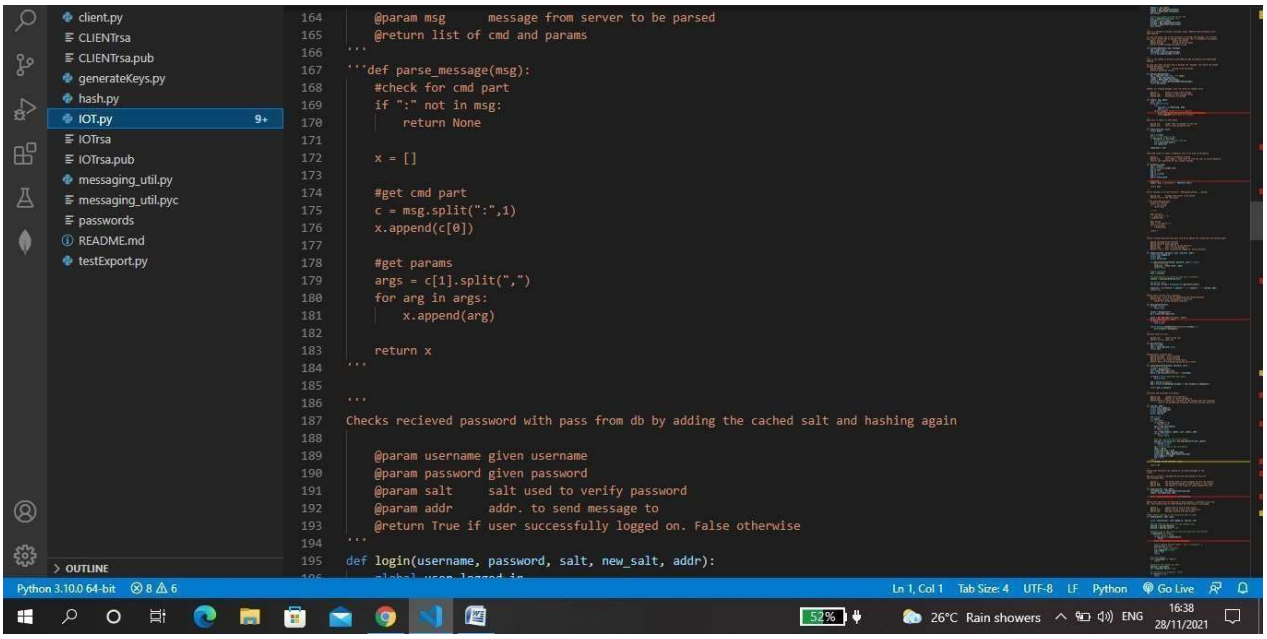

```
33  global user             #username of current user
34  global table            #keeps track of salt used for message, helps for delayed responses
35  global pubkey           #Object form of IOT's public key
36  global privkey          #Object form of IOT's private key
37  global pubtext          #Text form of IOT's public key
38  global client_pub       #public key of the client in object form
39  global client_pub_text      #textual version of the client's pub key
40  global sock             #UDP socket used for normal connection
41  global broadcast        #UDP socket used to broadcast
42  global user_logged_in       #Flag used to track if a user is currently connected to this IOT
43  global block_list       #List of (IP, port) tuples to block
44  global DEVICE_NAME
45
46  DEVICE_NAME = gethostname()
47  user = None
48  user_logged_in = False
49  block_list = list()
50  broadcast = socket(AF_INET, SOCK_DGRAM)
51  broadcast.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
52  broadcast.setsockopt(SOL_SOCKET, SO_BROADCAST, 1)
53
54  sock = socket(AF_INET, SOCK_DGRAM)
55  sock.bind(('', RECV_PORT))
56  sock.settimeout(TIMEOUT)
57
58  table = dict()
59
60  #Initialize global public key for IOT
61  pub =  open(PUB_KEY_FILE, "r");
62  pubtext = pub.read()
63  pubkey = RSA.importKey(pubtext)
64  pubkey = PKCS1_OAEP.new(pubkey)
```

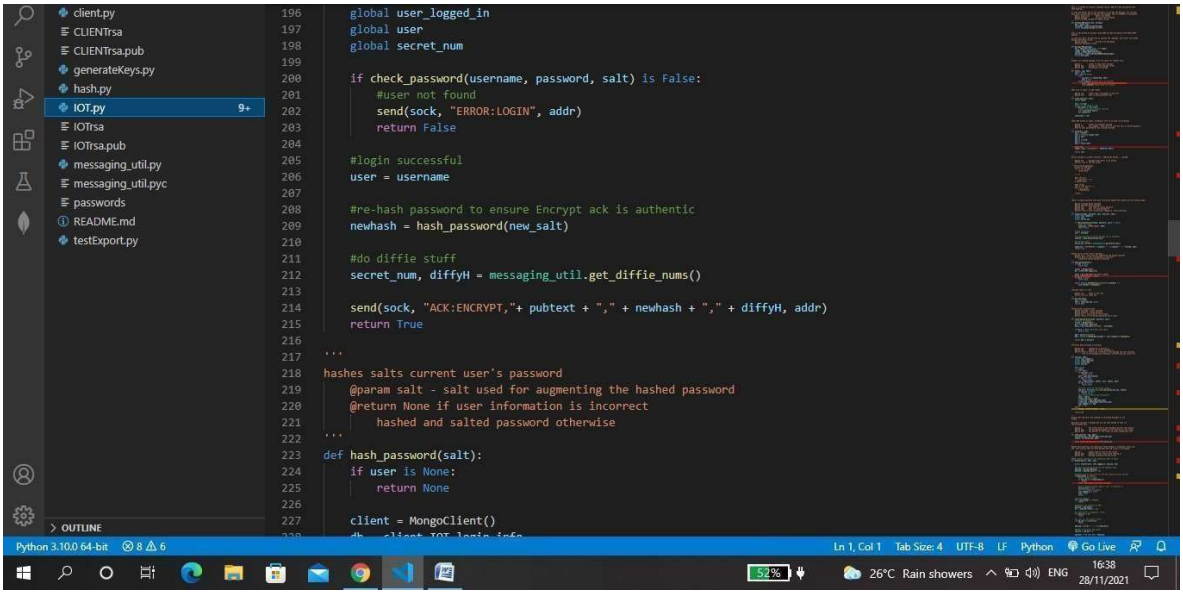


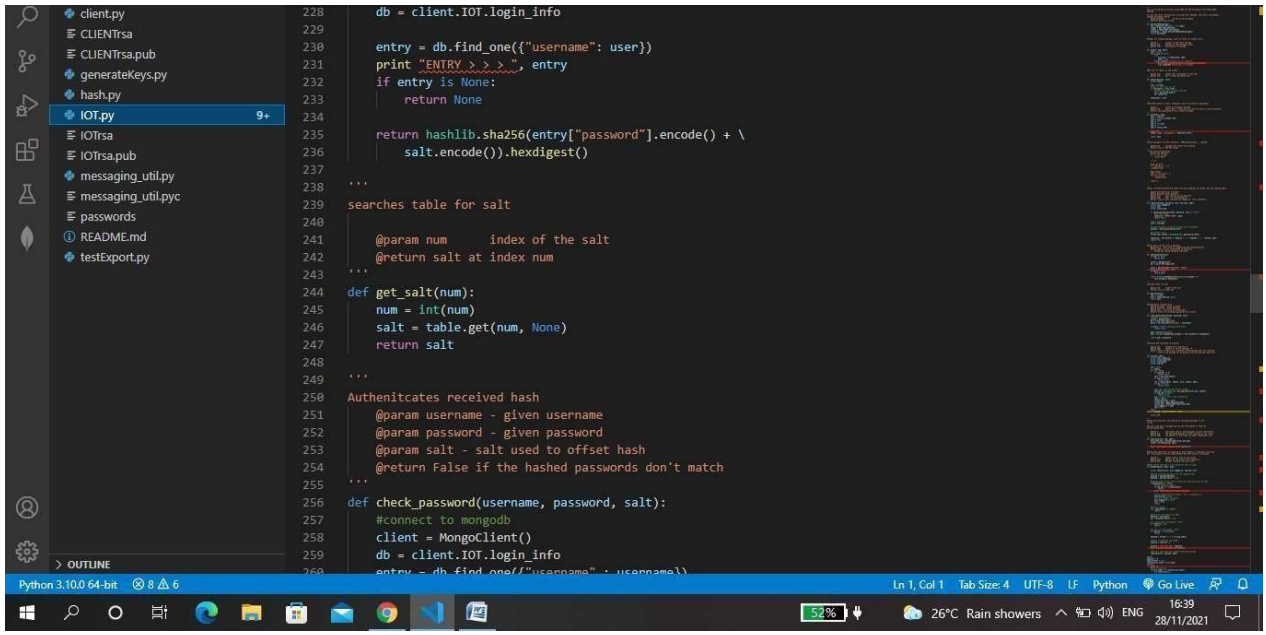

```
67  #Initialize global private key for IOT
68  priv = open(PRIV_KEY_FILE, "r")
69  privtext = priv.read()
70  privkey = RSA.importKey(privtext)
71  privkey = PKCS1_OAEP.new(privkey)
72  priv.close()
73
74  '''
75  This is a method to encrypt a message using a 4096 bit RSA encryption with
76  OAEP padding.
77
78  It uses the public key of the recipient to encrypt the message. The reciever
79  uses their private key to decrypt the message. This is Asymmetric encryption.
80      @para: public_key       Public key object
81      @param message          String to be encrypted
82      @return base64 encoded encrypted string
83  '''
84  def encrypt_RSA(public_key, message):
85      pub = public_key
86      encrypted = pub.encrypt(message)
87      return encrypted.encode('base64')
88
89  '''
90  This is the method to decrypt using 4096 but RSA encryption with PKCS1_OAEP
91  padding.
92
93  It uses this IOT's private key to decrypt the 'package' and return the base64
94  decoded decrypted string.
95      @param package          String to be decrypted
96      @returns decrypted string
97  '''
98  def decrypt_RSA(package):
```

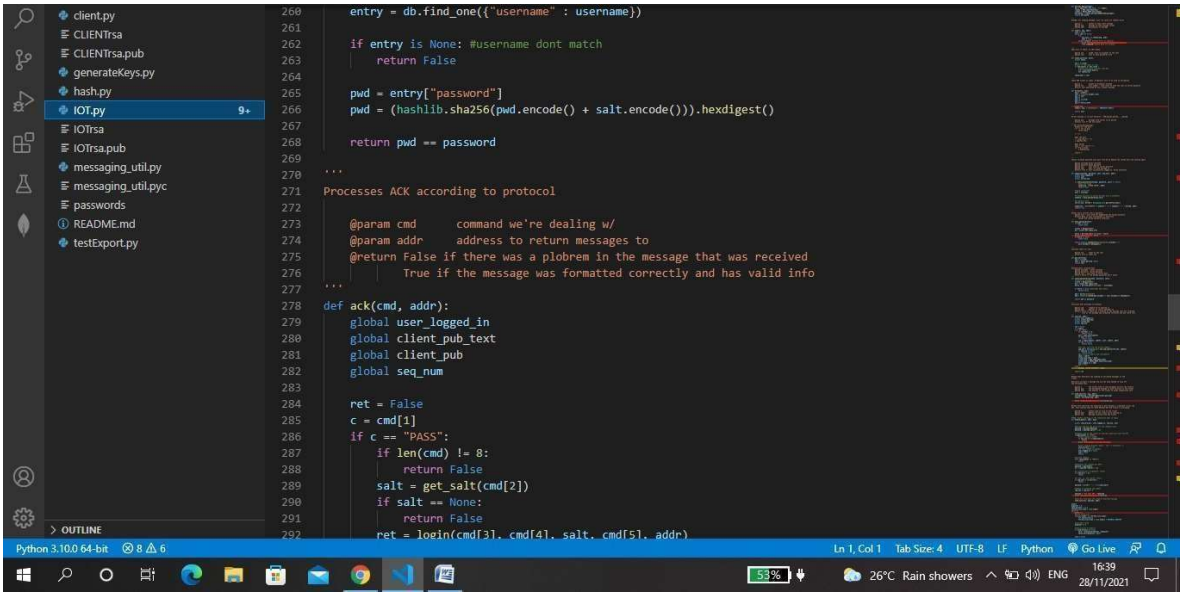

client.py
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py 9+
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
key = open(PRIV_KEY_FILE, "r").read()
rsakey = RSA.importKey(key)
rsakey = PKCS1_OAEP.new(rsakey)
decrypted = rsakey.decrypt(b64decode(package))
return decrypted
'''
Wrapper for sending message, with try catch for timeout error
@param s socket to send stuff through
@param msg message to send through socket
@param addr destination of message
'''
def send(s, msg, addr):
sent = False
while sent == False:
try:
num_sent = s.sendto(msg, addr)
sent = True
except timeout: #socket.error is subclass
print "No network connection, trying again later..."
time.sleep(60) #check back in a minute
'''
adds salt to table, in LRU scheme
@param num number that corresponds to the salt
@param salt salt to hash passwords with
'''
def cache_salt(num, salt):
global table
Python 3.10.0 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
LF
Python
Go Live
52%
26°C Rain showers
ENG
16:38
28/11/2021

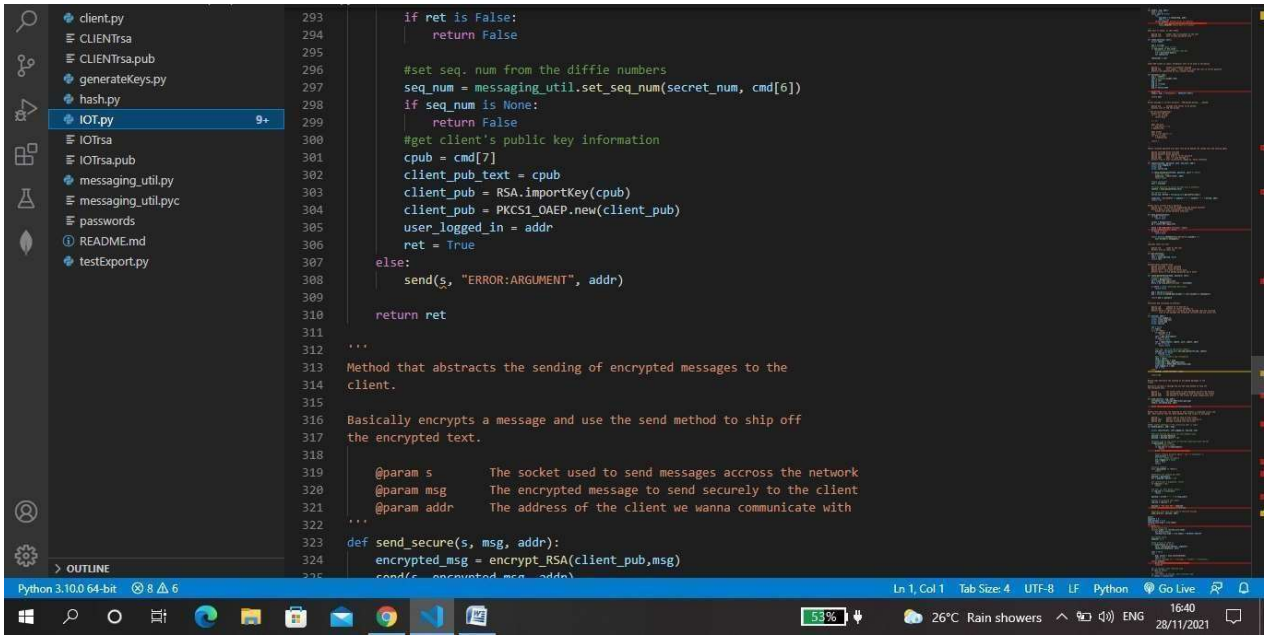

client.py
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py 9+
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
#check if the table is full
if len(table) == MAX_CACHE :
#remove the least recently used key
lru = min(table.keys())
del table[lru]
table[num] = salt
'''
takes UDP socket as input, broadcasts salt to be used in encryption
@param s socket to broadcast through
@param num salt number - used later to find the salt to verify password
@return salt generated by this connect message
'''
def brocast(s, num):
msg = "CONNECT:"
salt = str(uuid.uuid4().hex)
msg += salt
msg += ","
msg += str(num)
msg += ","
msg += DEVICE_NAME
print msg
send(s, msg, ('<broadcast>', BROADCAST_PORT))
return salt
'''
Parses message in correct protocol - CMD:param1,param2,...,paramN
Python 3.10.0 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
LF
Python
Go Live
52%
26°C Rain showers
ENG
16:38
28/11/2021

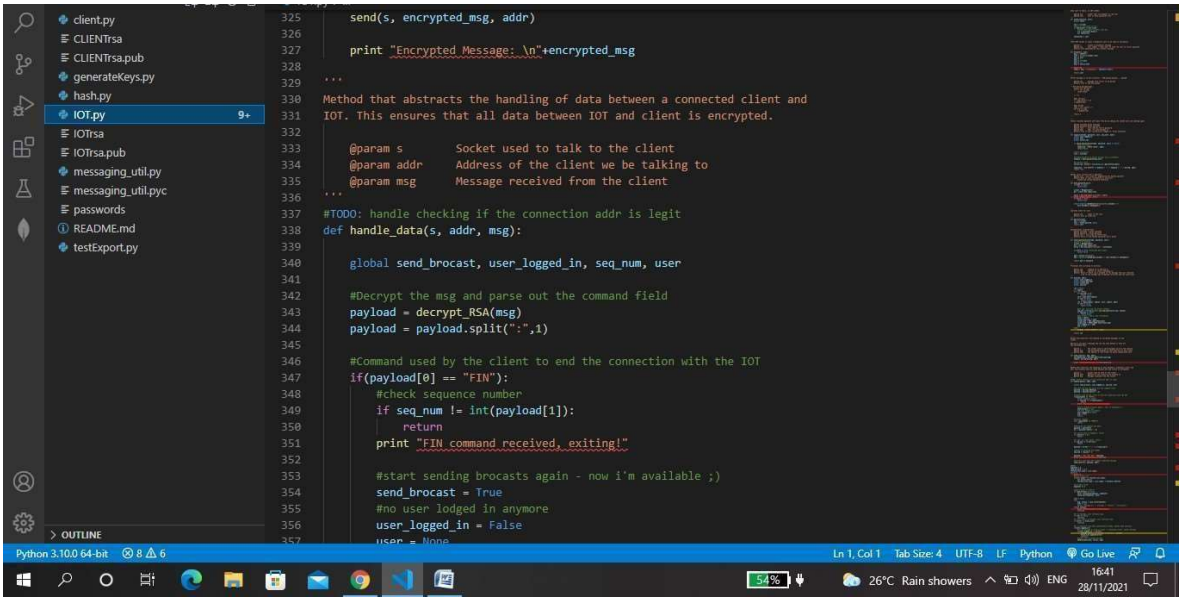

client.py
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py 9+
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
@param msg message from server to be parsed
@return list of cmd and params
'''
'''def parse_message(msg):
#check for cmd part
if ":" not in msg:
return None
x = []
#get cmd part
c = msg.split(":",1)
x.append(c[0])
#get params
args = c[1].split(",")
for arg in args:
x.append(arg)
return x
'''
'''
Checks recieved password with pass from db by adding the cached salt and hashing again
@param username given username
@param password given password
@param salt salt used to verify password
@param addr addr. to send message to
@return True if user successfully logged on. False otherwise
'''
def login(username, password, salt, new_salt, addr):
Python 3.10.0 64-bit
Ln 1, Col 1 Tab Size: 4 UTF-8 LF Python Go Live
52%
26°C Rain showers
ENG
16:38
28/11/2021

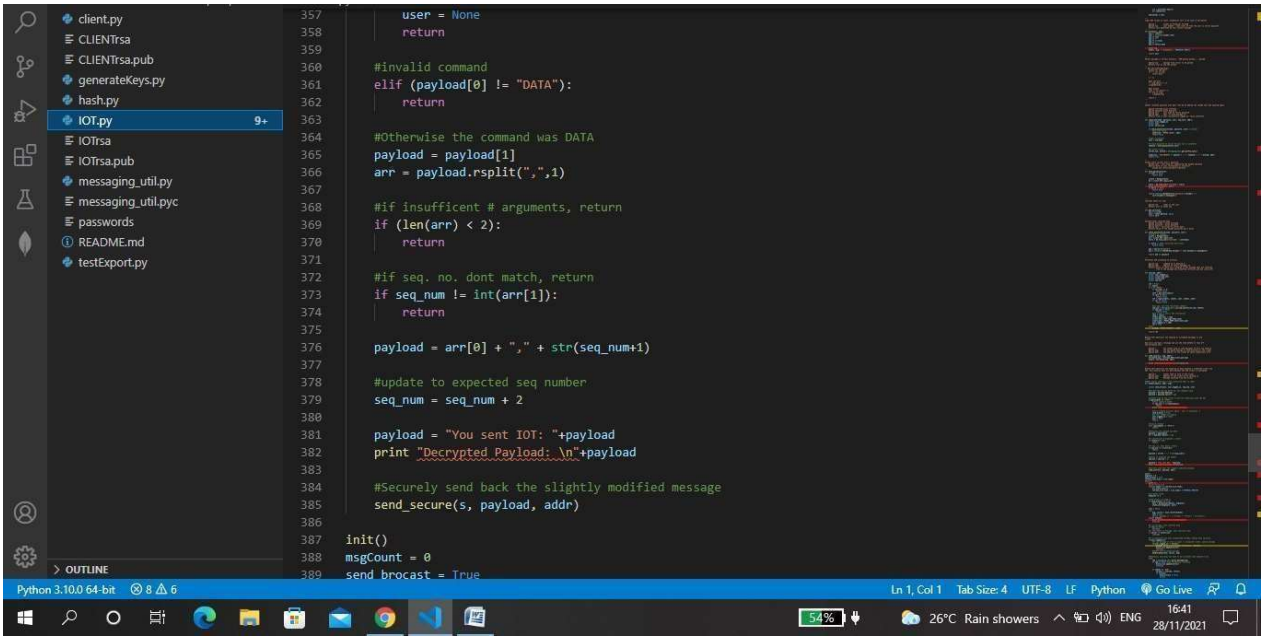

client.py
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py 9+
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
global user_logged_in
global user
global secret_num
if check_password(username, password, salt) is False:
#user not found
send(sock, "ERROR:LOGIN", addr)
return False
#login successful
user = username
#re-hash password to ensure Encrypt ack is authentic
newhash = hash_password(new_salt)
#do diffie stuff
secret_num, diffyH = messaging_util.get_diffie_nums()
send(sock, "ACK:ENCRYPT,"+ pubtext + "," + newhash + "," + diffyH, addr)
return True
'''
hashes salts current user's password
@param salt - salt used for augmenting the hashed password
@return None if user information is incorrect
hashed and salted password otherwise
'''
def hash_password(salt):
if user is None:
return None
client = MongoClient()
Python 3.10.0 64-bit
Ln 1, Col 1 Tab Size: 4 UTF-8 LF Python Go Live
52%
26°C Rain showers
ENG
16:38
28/11/2021

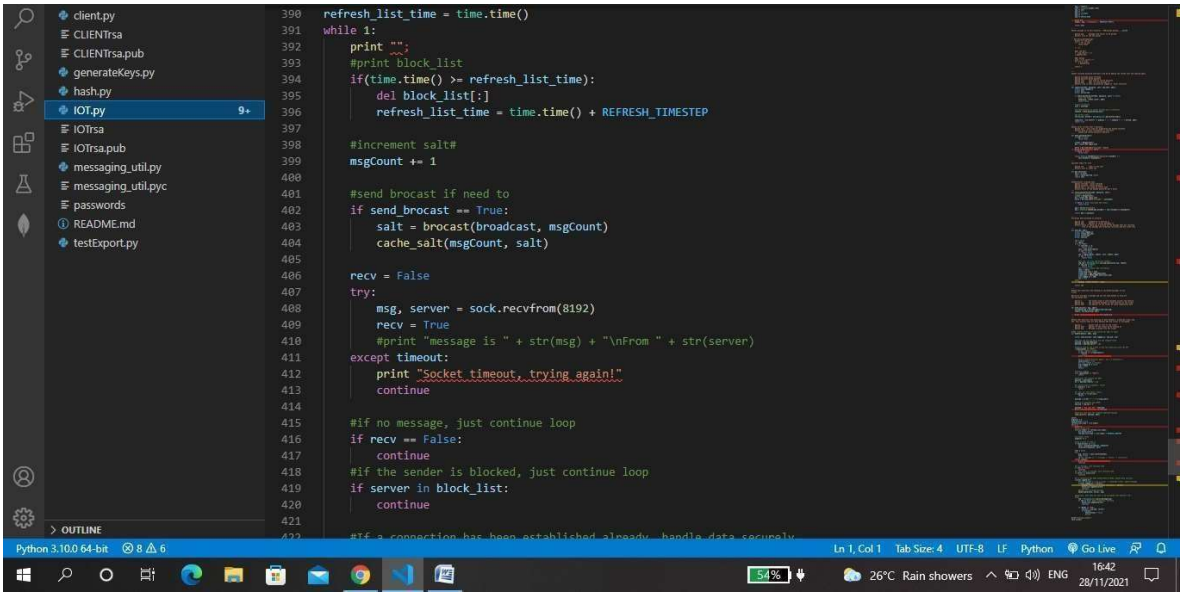

client.py
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py 9+
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
db = client.IOT.login_info
entry = db.find_one({"username": user})
print "ENTRY > > > ", entry
if entry is None:
return None
return hashlib.sha256(entry["password"].encode() + \
salt.encode()).hexdigest()
'''
searches table for salt
@param num index of the salt
@return salt at index num
'''
def get_salt(num):
num = int(num)
salt = table.get(num, None)
return salt
'''
Authenitcates received hash
@param username - given username
@param password - given password
@param salt - salt used to offset hash
@return False if the hashed passwords don't match
'''
def check_password(username, password, salt):
#connect to mongodb
client = MongoClient()
db = client.IOT.login_info
Python 3.10.0 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
LF
Python
Go Live
52%
26°C Rain showers
ENG
16:39
28/11/2021

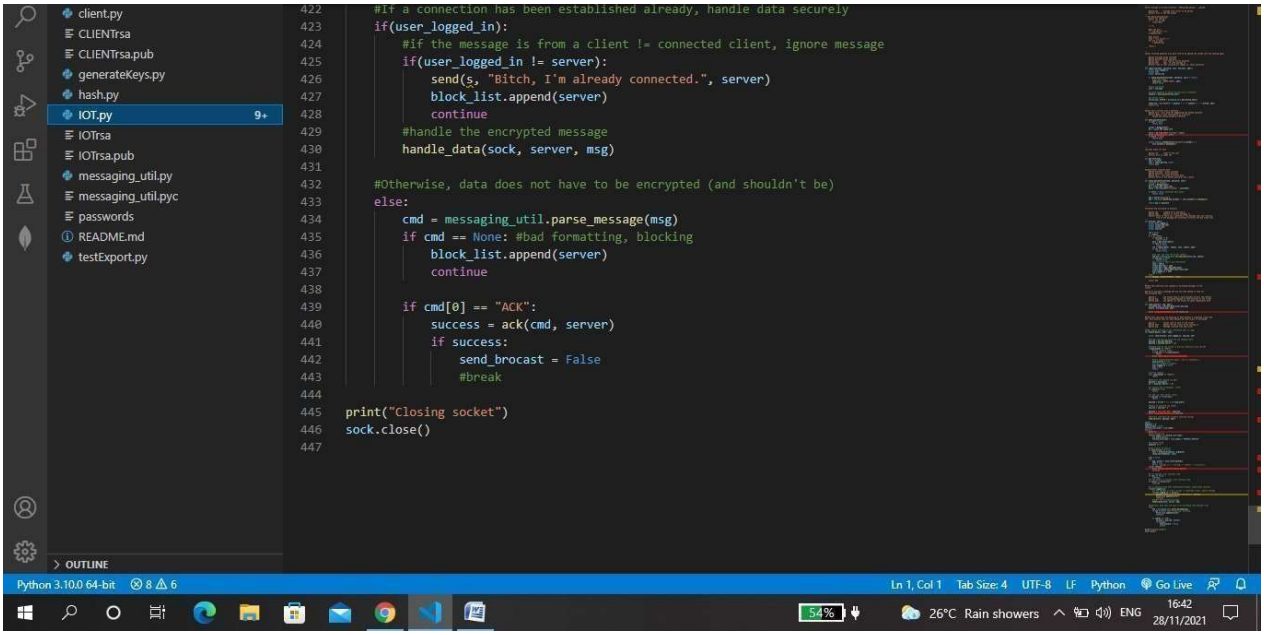

client.py
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py 9+
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
entry = db.find_one({"username" : username})
if entry is None: #username dont match
return False
pwd = entry["password"]
pwd = (hashlib.sha256(pwd.encode() + salt.encode())).hexdigest()
return pwd == password
'''
Processes ACK according to protocol
@param cmd command we're dealing w/
@param addr address to return messages to
@return False if there was a plobrem in the message that was received
True if the message was formatted correctly and has valid info
'''
def ack(cmd, addr):
global user_logged_in
global client_pub_text
global client_pub
global seq_num
ret = False
c = cmd[1]
if c == "PASS":
if len(cmd) != 8:
return False
salt = get_salt(cmd[2])
if salt == None:
return False
ret = login(cmd[3], cmd[4], salt, cmd[5], addr)
Python 3.10.0 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
LF
Python
Go Live
53%
26°C Rain showers
ENG
16:39
28/11/2021


client.py
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py 9+
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
if ret is False:
return False
#set seq. num from the diffie numbers
seq_num = messaging_util.set_seq_num(secret_num, cmd[6])
if seq_num is None:
return False
#get client's public key information
cpub = cmd[7]
client_pub_text = cpub
client_pub = RSA.importKey(cpub)
client_pub = PKCS1_OAEP.new(client_pub)
user_logged_in = addr
ret = True
else:
send(s, "ERROR:ARGUMENT", addr)
return ret
'''
Method that abstracts the sending of encrypted messages to the
client.
Basically encrypts a message and use the send method to ship off
the encrypted text.
@param s The socket used to send messages accross the network
@param msg The encrypted message to send securely to the client
@param addr The address of the client we wanna communicate with
'''
def send_secure(s, msg, addr):
encrypted_msg = encrypt_RSA(client_pub,msg)
Python 3.10.0 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
LF
Python
Go Live
53%
26°C Rain showers
ENG
16:40
28/11/2021


client.py
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py 9+
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
send(s, encrypted_msg, addr)
print "Encrypted Message: \n"+encrypted_msg
'''
Method that abstracts the handling of data between a connected client and
IOT. This ensures that all data between IOT and client is encrypted.
@param s Socket used to talk to the client
@param addr Address of the client we be talking to
@param msg Message received from the client
'''
#TODO: handle checking if the connection addr is legit
def handle_data(s, addr, msg):
global send_brocast, user_logged_in, seq_num, user
#Decrypt the msg and parse out the command field
payload = decrypt_RSA(msg)
payload = payload.split(":",1)
#Command used by the client to end the connection with the IOT
if(payload[0] == "FIN"):
#check sequence number
if seq_num != int(payload[1]):
return
print "FIN command received, exiting!"
#start sending brocasts again - now i'm available ;)
send_brocast = True
#no user lodged in anymore
user_logged_in = False
Python 3.10.0 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
LF
Python
Go Live
54%
26°C Rain showers
ENG
16:41
28/11/2021

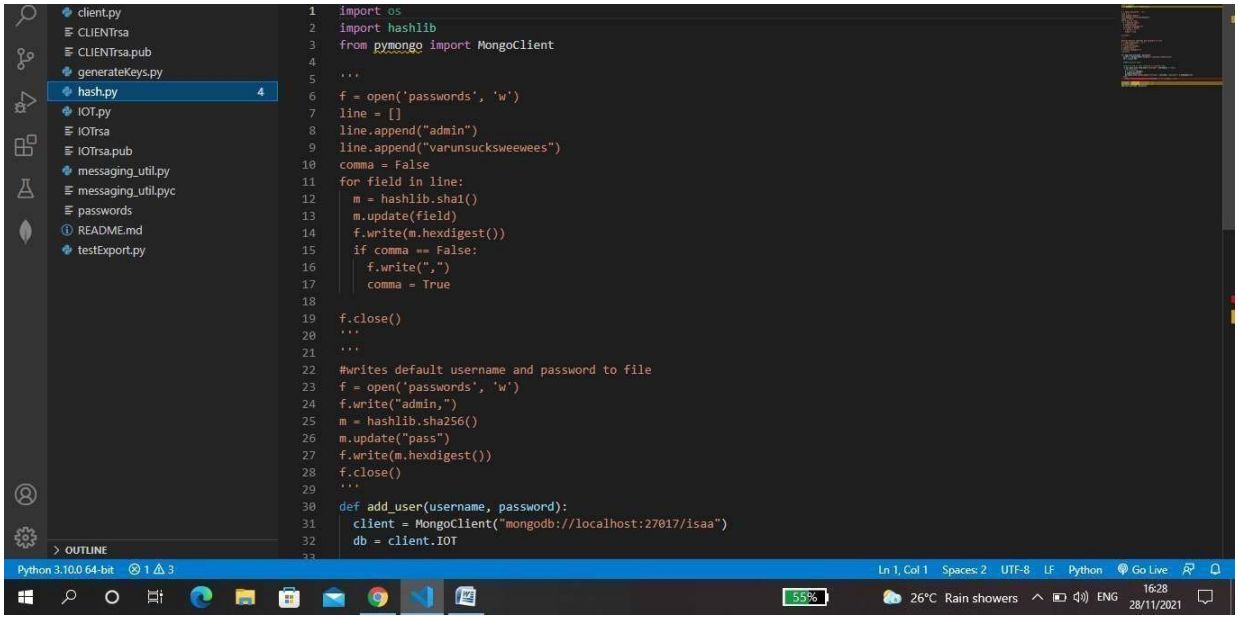

client.py
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py 9+
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
user = None
return
#invalid command
elif (payload[0] != "DATA"):
return
#Otherwise the command was DATA
payload = payload[1]
arr = payload.rsplit(",",1)
#if insufficent # arguments, return
if (len(arr) < 2):
return
#if seq. no. dont match, return
if seq_num != int(arr[1]):
return
payload = arr[0] + "," + str(seq_num+1)
#update to expected seq number
seq_num = seq_num + 2
payload = "You sent IOT: "+payload
print "Decrypted Payload: \n"+payload
#Securely send back the slightly modified message
send_secure(s, payload, addr)
init()
msgCount = 0
send_brocast = True
Python 3.10.0 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
LF
Python
Go Live
54%
26°C Rain showers
ENG
16:41
28/11/2021

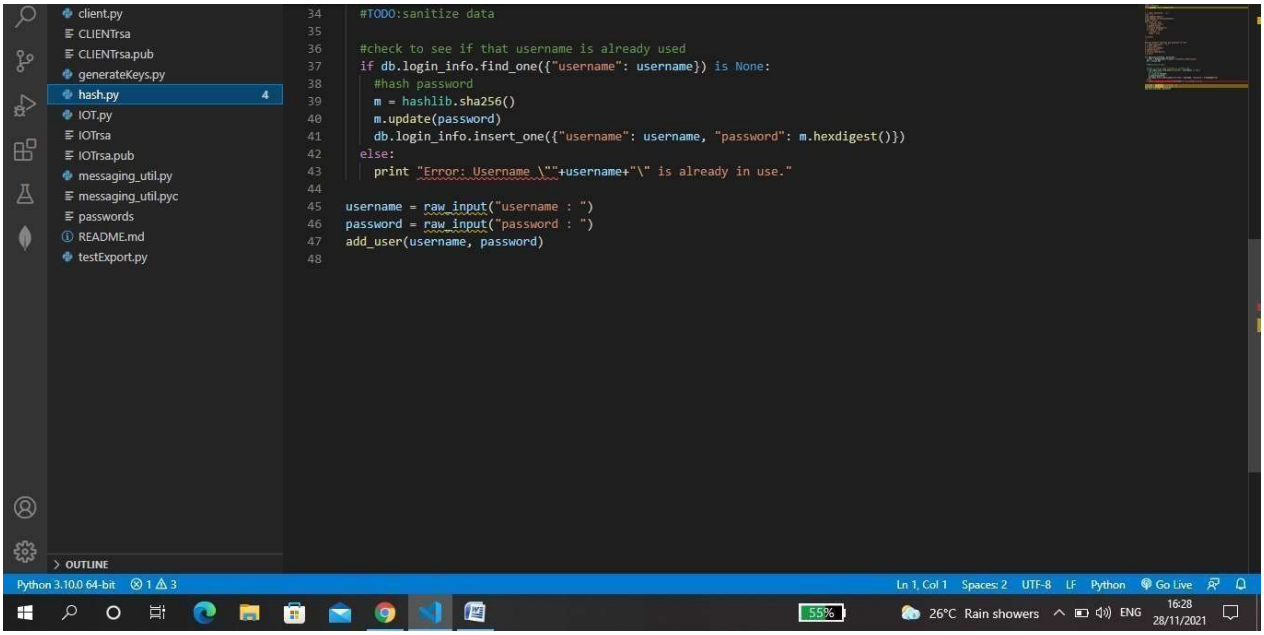

client.py
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py 9+
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
OUTLINE
refresh_list_time = time.time()
while 1:
print "";
#print block_list
if(time.time() >= refresh_list_time):
del block_list[:]
refresh_list_time = time.time() + REFRESH_TIMESTEP
#increment salt#
msgCount += 1
#send brocast if need to
if send_brocast == True:
salt = brocast(broadcast, msgCount)
cache_salt(msgCount, salt)
recv = False
try:
msg, server = sock.recvfrom(8192)
recv = True
#print "message is " + str(msg) + "\nFrom " + str(server)
except timeout:
print "Socket timeout, trying again!"
continue
#if no message, just continue loop
if recv == False:
continue
#if the sender is blocked, just continue loop
if server in block_list:
continue
Python 3.10.0 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
LF
Python
Go Live
54%
26°C Rain showers
ENG
16:42
28/11/2021



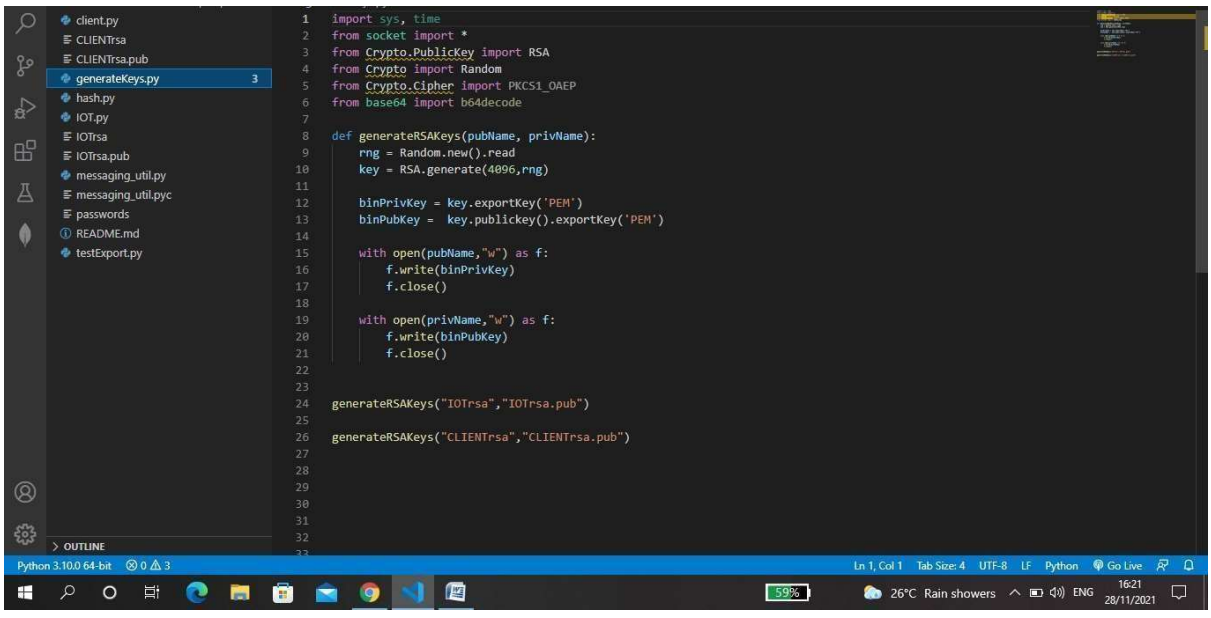
Hash.py:

Hashing and storing in database

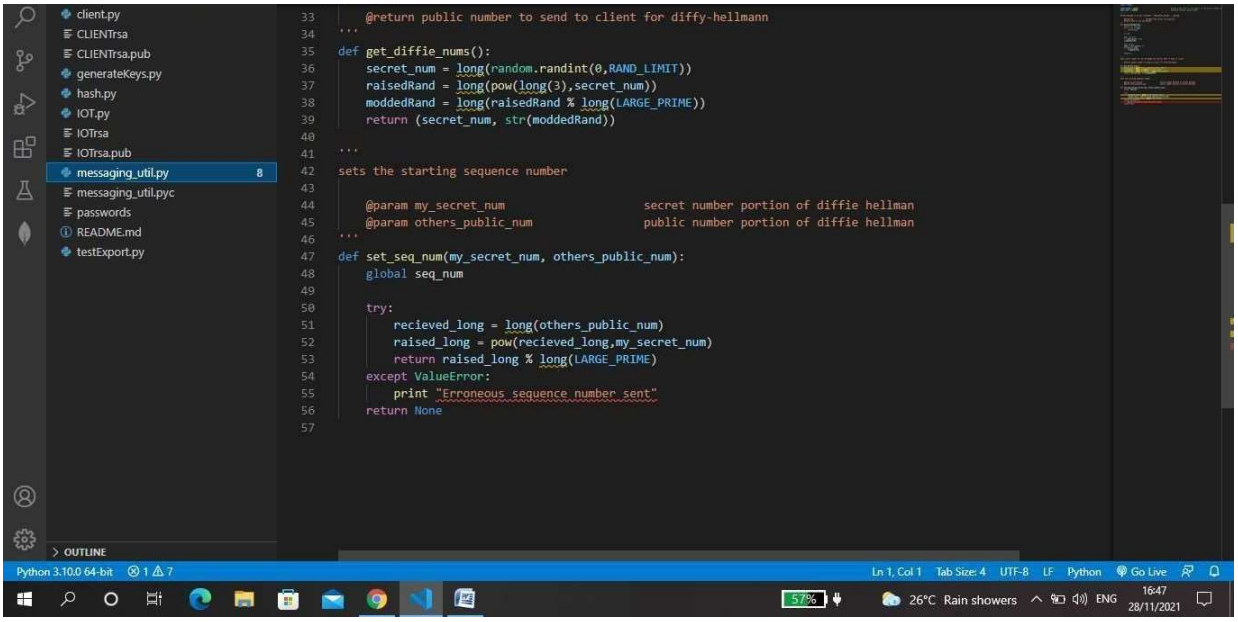




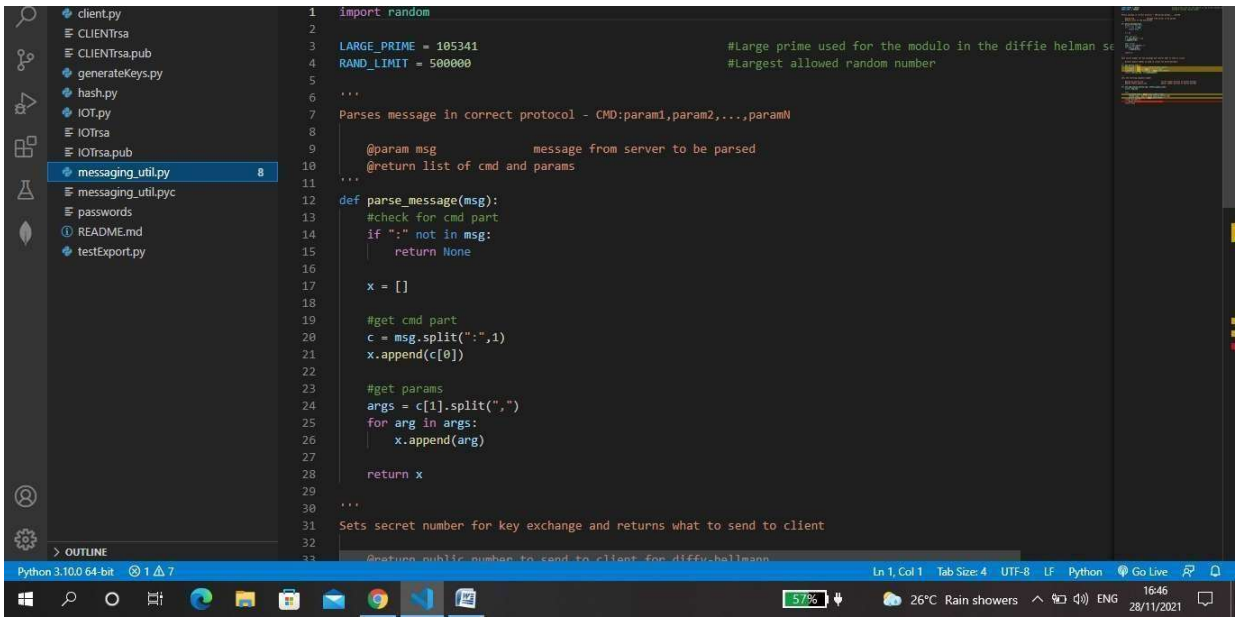
Generate keys.py



Messaging util.py

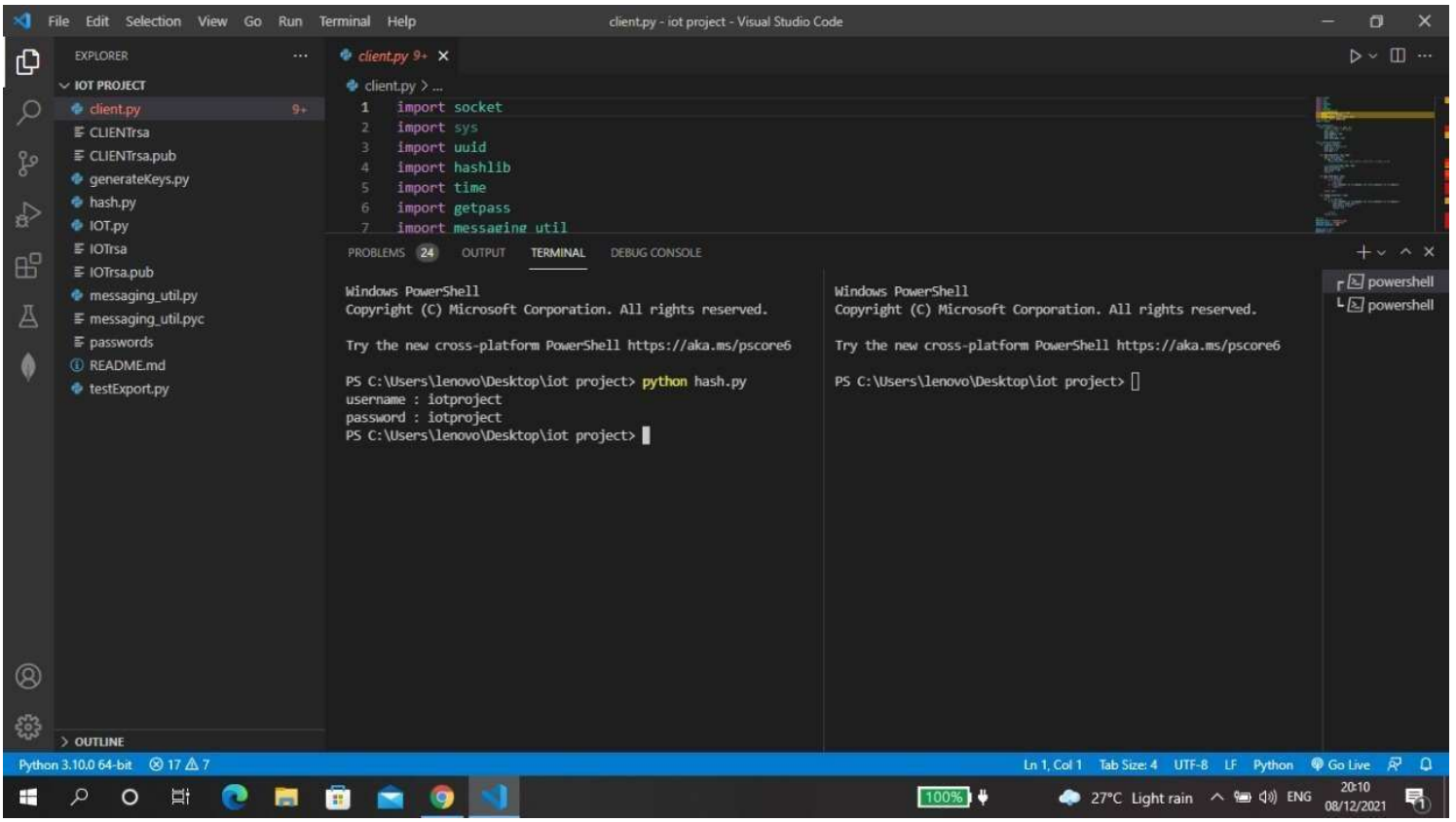


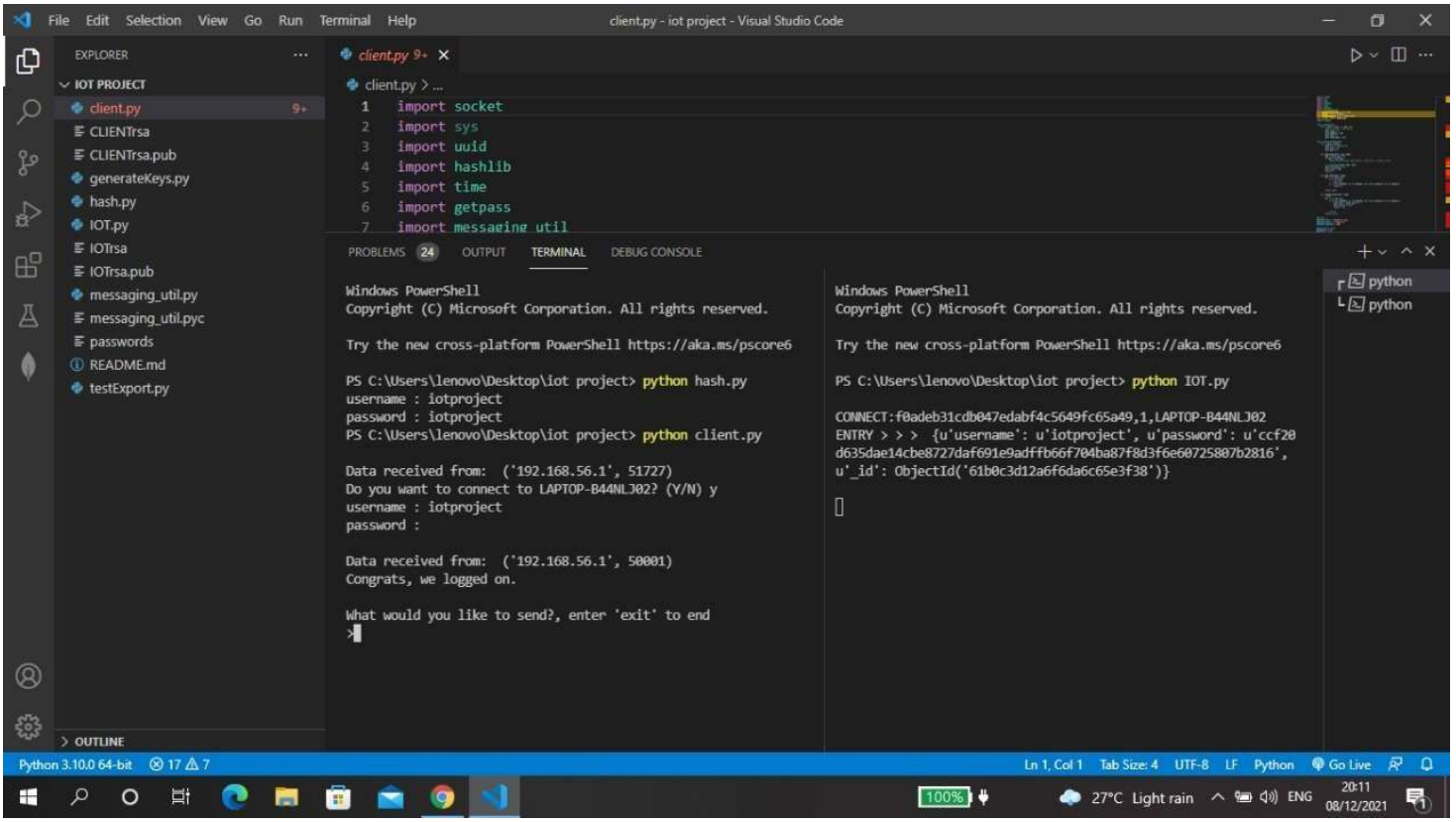
```
    @return public number to send to client for diffy-hellmann
'''
def get_diffie_nums():
    secret_num = long(random.randint(0,RAND_LIMIT))
    raisedRand = long(pow(long(3),secret_num))
    moddedRand = long(raisedRand % long(LARGE_PRIME))
    return (secret_num, str(moddedRand))

'''
sets the starting sequence number

    @param my_secret_num                secret number portion of diffie hellman
    @param others_public_num            public number portion of diffie hellman
'''
def set_seq_num(my_secret_num, others_public_num):
    global seq_num

    try:
        recieved_long = long(others_public_num)
        raised_long = pow(recieved_long,my_secret_num)
        return raised_long % long(LARGE_PRIME)
    except ValueError:
        print "Erroneous sequence number sent"
    return None
```

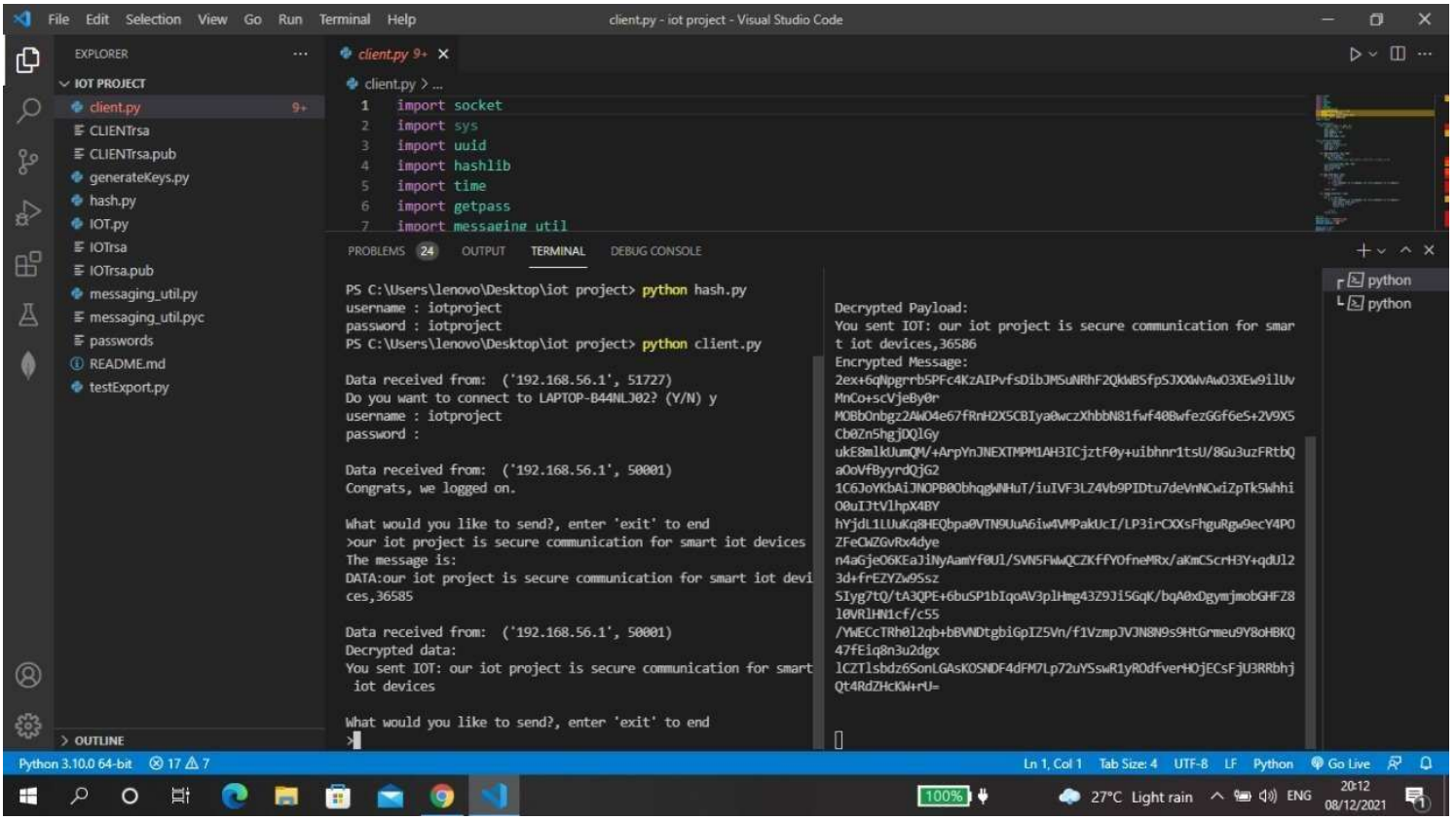


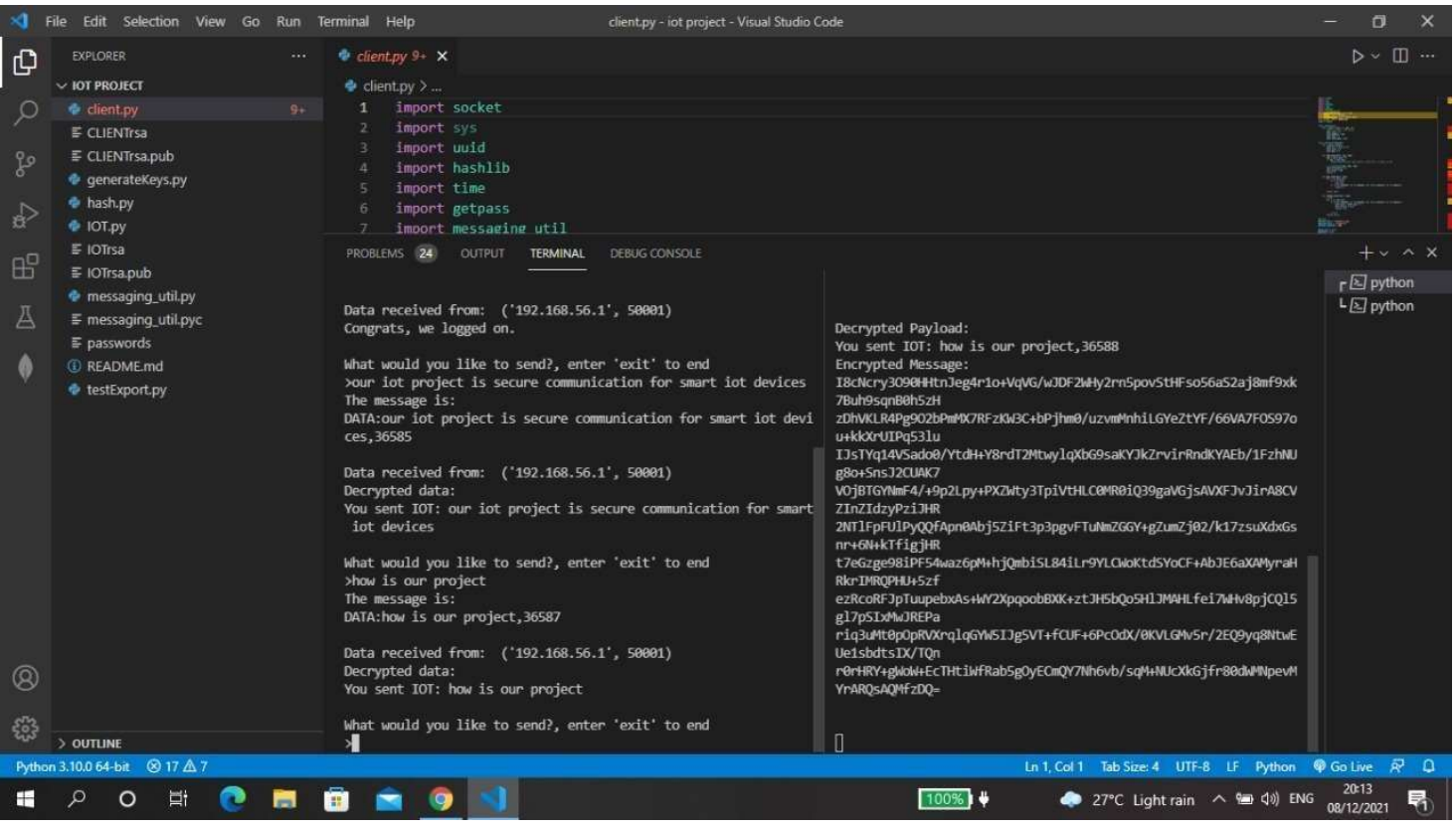
```
import random

LARGE_PRIME = 105341                            #Large prime used for the modulo in the diffie helman se
RAND_LIMIT = 500000                             #Largest allowed random number

'''
Parses message in correct protocol - CMD:param1,param2,...,paramN

    @param msg              message from server to be parsed
    @return list of cmd and params
'''
def parse_message(msg):
    #check for cmd part
    if ":" not in msg:
        return None

    x = []

    #get cmd part
    c = msg.split(":",1)
    x.append(c[0])

    #get params
    args = c[1].split(",")
    for arg in args:
        x.append(arg)

    return x

'''
Sets secret number for key exchange and returns what to send to client
```

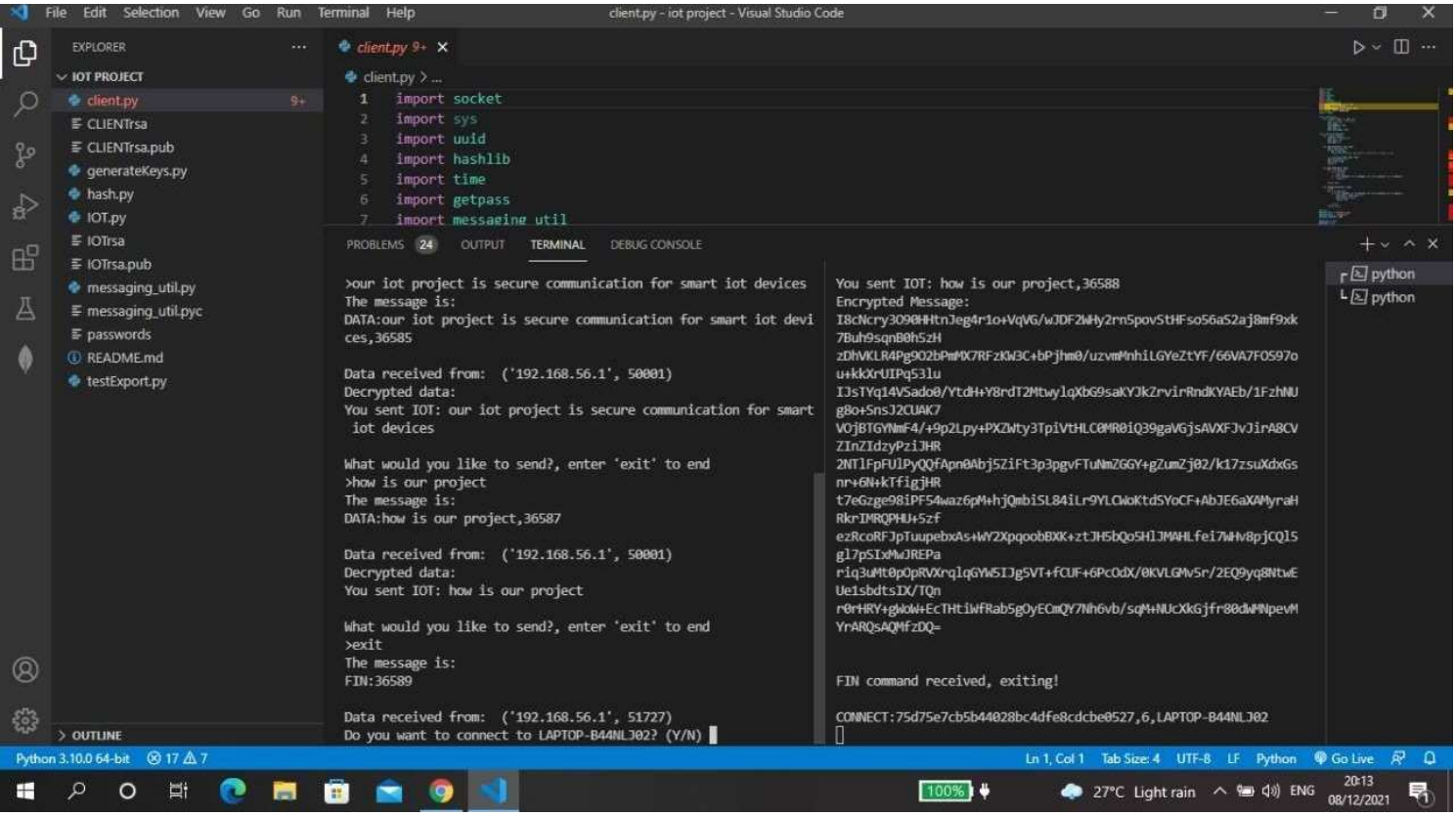
EXUCUTION RESULTS:

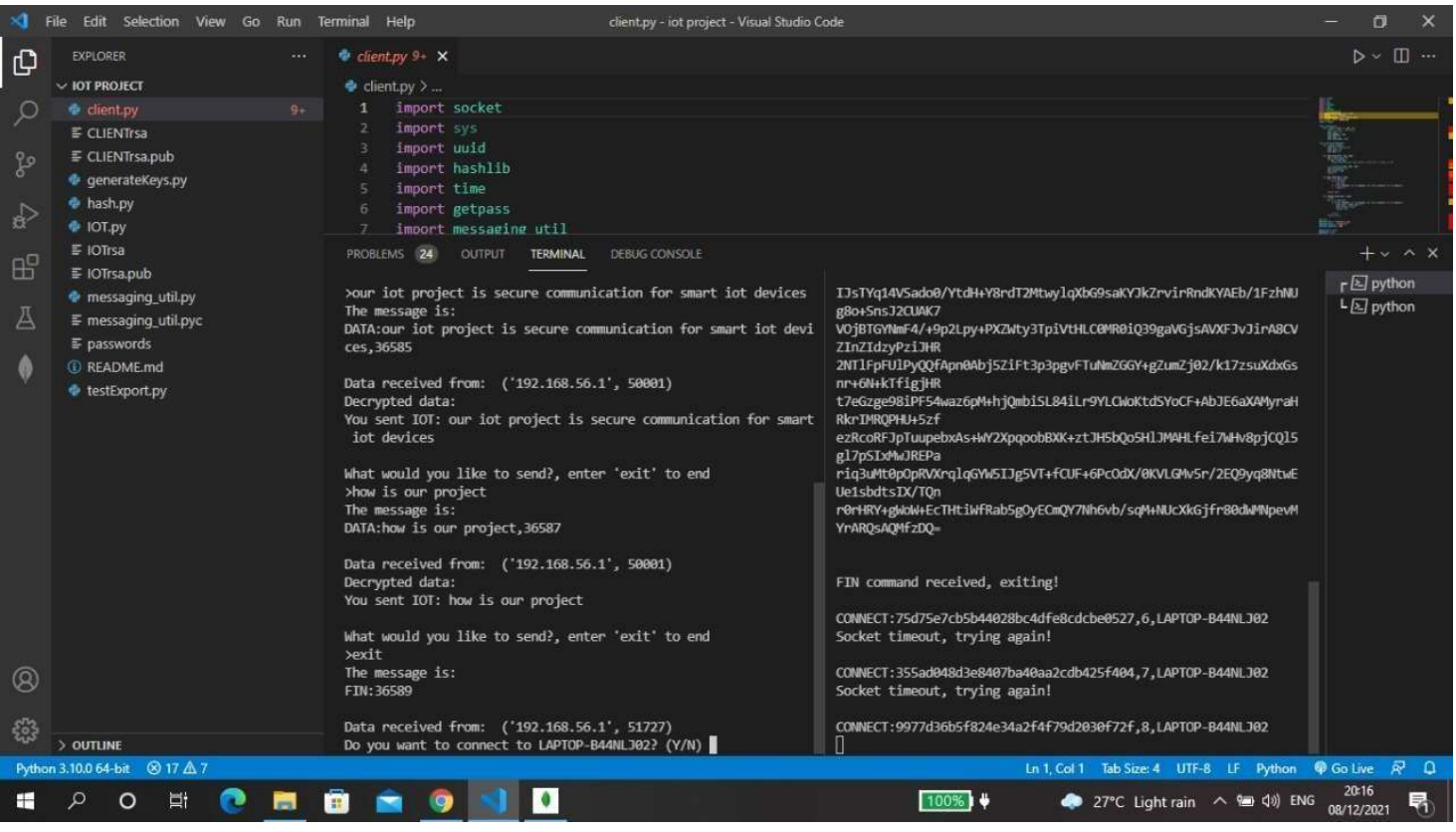




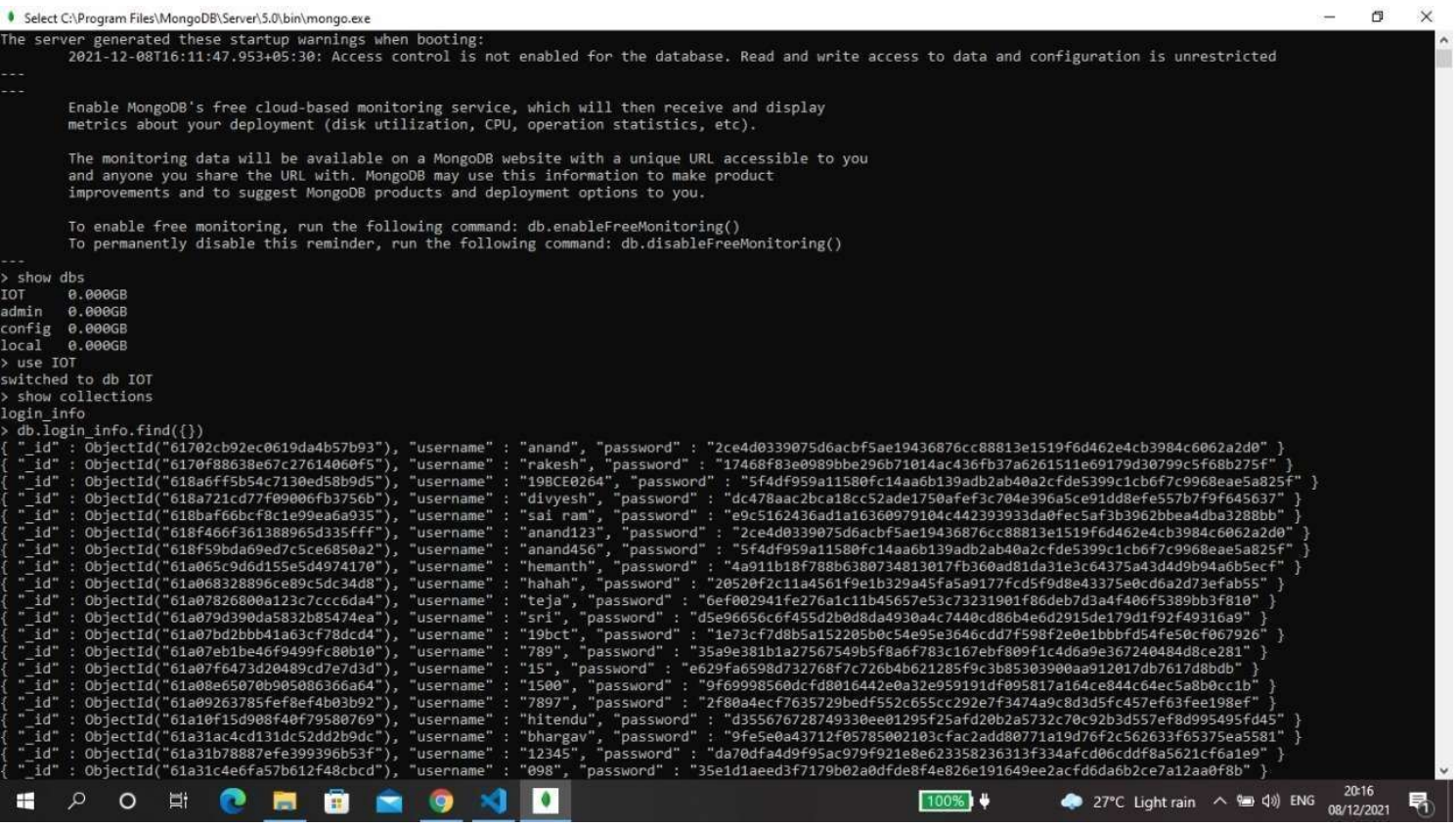

File Edit Selection View Go Run Terminal Help
client.py - iot project - Visual Studio Code
EXPLORER
IOT PROJECT
client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
client.py 9+
client.py > ...
1 import socket
2 import sys
3 import uuid
4 import hashlib
5 import time
6 import getpass
7 import messaging_util
PROBLEMS 24
OUTPUT
TERMINAL
DEBUG CONSOLE
PS C:\Users\lenovo\Desktop\iot project> python hash.py
username : iotproject
password : iotproject
PS C:\Users\lenovo\Desktop\iot project> python client.py
Data received from: ('192.168.56.1', 51727)
Do you want to connect to LAPTOP-B44NLJ02? (Y/N) y
username : iotproject
password :
Data received from: ('192.168.56.1', 50001)
Congrats, we logged on.
What would you like to send?, enter 'exit' to end
>our iot project is secure communication for smart iot devices
The message is:
DATA:our iot project is secure communication for smart iot devi
ces,36585
Data received from: ('192.168.56.1', 50001)
Decrypted data:
You sent IOT: our iot project is secure communication for smart
iot devices
What would you like to send?, enter 'exit' to end
>
Decrypted Payload:
You sent IOT: our iot project is secure communication for smar
t iot devices,36586
Encrypted Message:
2ex+6qNpgrrb5PFc4KzAIPvfsDibJMSuNRhF2QkWBSfpSJX0WvAwO3XEw9ilUv
MnCo+scVjeBy0r
MOBbOnbgz2AWO4e67fRnH2X5CBIya0wczXhbbN81fwf40BwfezGGf6eS+2V9X5
Cb0Zn5hgjDQlGy
ukE8mlkUumQM/+ArpYnJNEXTMPM1AH3ICjztF0y+uibhnr1tsU/8Gu3uzFRtbQ
aOoVfByyrdQjG2
1C6JoYKbAiJNOPB0ObhqgWNHuT/iuIVF3LZ4Vb9PIDtu7deVnNCwiZpTk5Whhi
O0uIJtVlhpX4BY
hYjdL1LUuKq8HEQbpa0VTN9UuA6iw4VMPakUcI/LP3irCXXsFhguRgw9ecY4P0
ZFeCWZGvRx4dye
n4aGjeO6KEaJiNyAamYf0Ul/SVN5FWwQCZKfFYOfneMRx/aKmCScrH3Y+qdUl2
3d+frEZYZw9Ssz
SIyg7tQ/tA3QPE+6buSP1bIqoAV3plHmg43Z9Ji5GqK/bqA0xDgymjmobGHFZ8
l0VRlHN1cf/c55
/YwECcTRh0l2qb+bBVNDtgbiGpIZ5Vn/f1VzmpJVJN8N9s9HtGrmeu9Y8oHBKQ
47fEiq8n3u2dgx
lCZTlsbdz6SonLGAsKOSNDF4dFM7Lp72uYSswR1yROdfverHOjECsFjU3RRbhj
Qt4RdZHcKW+rU=
Python 3.10.0 64-bit ⊗ 17 ⚠ 7
Ln 1, Col 1 Tab Size: 4 UTF-8 LF Python Go Live
100%
27°C Light rain
ENG
20:12
08/12/2021


File Edit Selection View Go Run Terminal Help
client.py - iot project - Visual Studio Code
EXPLORER
IOT PROJECT
client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
client.py 9+
client.py > ...
1 import socket
2 import sys
3 import uuid
4 import hashlib
5 import time
6 import getpass
7 import messaging_util
PROBLEMS 24
OUTPUT
TERMINAL
DEBUG CONSOLE
Data received from: ('192.168.56.1', 50001)
Congrats, we logged on.
What would you like to send?, enter 'exit' to end
>our iot project is secure communication for smart iot devices
The message is:
DATA:our iot project is secure communication for smart iot devi
ces,36585
Data received from: ('192.168.56.1', 50001)
Decrypted data:
You sent IOT: our iot project is secure communication for smart
iot devices
What would you like to send?, enter 'exit' to end
>how is our project
The message is:
DATA:how is our project,36587
Data received from: ('192.168.56.1', 50001)
Decrypted data:
You sent IOT: how is our project
What would you like to send?, enter 'exit' to end
>
Decrypted Payload:
You sent IOT: how is our project,36588
Encrypted Message:
I8cNcry3O90HHtnJeg4r1o+VqVG/wJDF2WHy2rn5pov5tHFso56a52aj8mf9xk
7Buh9sqnB0h5zH
zDhVKLR4Pg9O2bPmMX7RFzKW3C+bPjhm0/uzvmMnhiLGYeZtYF/66VA7FOS97o
u+kkXrUIPq53lu
IJsTYq14VSado0/YtdH+Y8rdT2MtwylqXbG9saKYJkZrvirRndKYAEb/1FzhNU
g8o+SnsJ2CUAK7
VOjBTGYNmF4/+9p2Lpy+PXZWty3TpiVtHLC0MR0iQ39gaVGjsAVXFJvJirA8CV
ZInZIdzyPziJHR
2NTlFpFUlPyQQfApn0Abj5ZiFt3p3pgvFTuNmZGGY+gZumZj02/k17zsuXdxGs
nr+6N+kTfigjHR
t7eGzge98iPF54waz6pM+hjQmbiSL84iLr9YLCWoKtdSYoCF+AbJE6aXAMyraH
RkrIMRQPHU+5zf
ezRcoRFJpTuupebxAs+WY2XpqoobBXK+ztJH5bQo5HlJMAHLfei7WHv8pjCQl5
gl7pSIxMwJREPa
riq3uMt0pOpRVXrqlqGYW5IJg5VT+fCUF+6PcOdX/0KVLGMv5r/2EQ9yq8NtwE
Ue1sbdtsIX/TQn
r0rHRY+gWoW+EcTHtiWfRab5gOyECmQY7Nh6vb/sqM+NUcXkGjfr80dWMNpevM
YrARQsAQMfzDQ=
Python 3.10.0 64-bit ⊗ 17 ⚠ 7
Ln 1, Col 1 Tab Size: 4 UTF-8 LF Python Go Live
100%
27°C Light rain
ENG
20:13
08/12/2021


File Edit Selection View Go Run Terminal Help
client.py - iot project - Visual Studio Code
EXPLORER
IOT PROJECT
client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
client.py 9+
client.py > ...
1 import socket
2 import sys
3 import uuid
4 import hashlib
5 import time
6 import getpass
7 import messaging_util
PROBLEMS 24 OUTPUT TERMINAL DEBUG CONSOLE
>our iot project is secure communication for smart iot devices
The message is:
DATA:our iot project is secure communication for smart iot devi
ces,36585
Data received from: ('192.168.56.1', 50001)
Decrypted data:
You sent IOT: our iot project is secure communication for smart
iot devices
What would you like to send?, enter 'exit' to end
>how is our project
The message is:
DATA:how is our project,36587
Data received from: ('192.168.56.1', 50001)
Decrypted data:
You sent IOT: how is our project
What would you like to send?, enter 'exit' to end
>exit
The message is:
FIN:36589
Data received from: ('192.168.56.1', 51727)
Do you want to connect to LAPTOP-B44NLJ02? (Y/N)
You sent IOT: how is our project,36588
Encrypted Message:
I8cNcry3O90HHtnJeg4r1o+VqVG/wJDF2WHy2rn5povStHFso56aS2aj8mf9xk
7Buh9sqnB0h5zH
zDhVKLR4Pg9O2bPmMX7RFzKW3C+bPjhm0/uzvmMnhiLGYeZtYF/66VA7FOS97o
u+kkXrUIPq53lu
IJsTYq14VSado0/YtdH+Y8rdT2MtwylqXbG9saKYJkZrvirRndKYAEb/1FzhNU
g8o+SnsJ2CUAK7
VOjBTGYNmF4/+9p2Lpy+PXZWty3TpiVtHLC0MR0iQ39gaVGjsAVXFJvJirA8CV
ZInZIdzyPziJHR
2NTlFpFUlPyQQfApn0Abj5ZiFt3p3pgvFTuNmZGGY+gZumZj02/k17zsuXdxGs
nr+6N+kTfigjHR
t7eGzge98iPF54waz6pM+hjQmbiSL84iLr9YLCWoKtdSYoCF+AbJE6aXAMyraH
RkrIMRQPHU+5zf
ezRcoRFJpTuupebxAs+WY2XpqoobBXK+ztJH5bQo5HlJMAHLfei7WHv8pjCQl5
gl7pSIxMwJREPa
riq3uMt0pOpRVXrqlqGYW5IJg5VT+fCUF+6PcOdX/0KVLGMv5r/2EQ9yq8NtwE
Ue1sbdtsIX/TQn
r0rHRY+gWoW+EcTHtiWfRab5gOyECmQY7Nh6vb/sqM+NUcXkGjfr80dWMNpevM
YrARQsAQMfzDQ=
FIN command received, exiting!
CONNECT:75d75e7cb5b44028bc4dfe8cdcbe0527,6,LAPTOP-B44NLJ02
python
python
OUTLINE
Python 3.10.0 64-bit 17 7
Ln 1, Col 1 Tab Size: 4 UTF-8 LF Python Go Live
100%
27°C Light rain
ENG
20:13
08/12/2021


File Edit Selection View Go Run Terminal Help
client.py - iot project - Visual Studio Code
EXPLORER
IOT PROJECT
client.py 9+
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py
client.py 9+
client.py > ...
1 import socket
2 import sys
3 import uuid
4 import hashlib
5 import time
6 import getpass
7 import messaging_util
PROBLEMS 24 OUTPUT TERMINAL DEBUG CONSOLE
>our iot project is secure communication for smart iot devices
The message is:
DATA:our iot project is secure communication for smart iot devi
ces,36585
Data received from: ('192.168.56.1', 50001)
Decrypted data:
You sent IOT: our iot project is secure communication for smart
iot devices
What would you like to send?, enter 'exit' to end
>how is our project
The message is:
DATA:how is our project,36587
Data received from: ('192.168.56.1', 50001)
Decrypted data:
You sent IOT: how is our project
What would you like to send?, enter 'exit' to end
>exit
The message is:
FIN:36589
Data received from: ('192.168.56.1', 51727)
Do you want to connect to LAPTOP-B44NLJ02? (Y/N)
IJsTYq14VSado0/YtdH+Y8rdT2MtwylqXbG9saKYJkZrvirRndKYAEb/1FzhNU
g8o+SnsJ2CUAK7
VOjBTGYNmF4/+9p2Lpy+PXZWty3TpiVtHLC0MR0iQ39gaVGjsAVXFJvJirA8CV
ZInZIdzyPziJHR
2NTlFpFUlPyQQfApn0Abj5ZiFt3p3pgvFTuNmZGGY+gZumZj02/k17zsuXdxGs
nr+6N+kTfigjHR
t7eGzge98iPF54waz6pM+hjQmbiSL84iLr9YLCWoKtdSYoCF+AbJE6aXAMyraH
RkrIMRQPHU+5zf
ezRcoRFJpTuupebxAs+WY2XpqoobBXK+ztJH5bQo5HlJMAHLfei7WHv8pjCQl5
gl7pSIxMwJREPa
riq3uMt0pOpRVXrqlqGYW5IJg5VT+fCUF+6PcOdX/0KVLGMv5r/2EQ9yq8NtwE
Ue1sbdtsIX/TQn
r0rHRY+gWoW+EcTHtiWfRab5gOyECmQY7Nh6vb/sqM+NUcXkGjfr80dWMNpevM
YrARQsAQMfzDQ=
FIN command received, exiting!
CONNECT:75d75e7cb5b44028bc4dfe8cdcbe0527,6,LAPTOP-B44NLJ02
Socket timeout, trying again!
CONNECT:355ad048d3e8407ba40aa2cdb425f404,7,LAPTOP-B44NLJ02
Socket timeout, trying again!
CONNECT:9977d36b5f824e34a2f4f79d2030f72f,8,LAPTOP-B44NLJ02
python
python
OUTLINE
Python 3.10.0 64-bit 17 7
Ln 1, Col 1 Tab Size: 4 UTF-8 LF Python Go Live
100%
27°C Light rain
ENG
20:16
08/12/2021

Mongo db for authentication



Conclusion

IOT devices need a good level of authentication and authorization. In this project we have implemented a secure channel communication protocol between IOT Device and a Client. We have used multiple concepts to complete this project : Socket Programing, Cryptography. The implementation can even be easily integrated into modern IoT applications since we have used MongoDB as a Database for authentication purposes which is quite popular in IoT device developer communities. In this implementation commands from Client device to IoT Device are encrypted by industry standard RSA algorithm.

.

Practical Advantages of our project

Our project was made such a way that it is resistant to the following attacks:

1)Eavesdropping Attack
2) Man in the middle attack
3)Arp Spoofing attacks

And resistant to all attacks that can intercept the network traffic

REFERENCES

1. Nguyen, K. T., Laurent, M., & Oualha, N. (2015). Survey on secure communication protocols for the Internet of Things. *Ad Hoc Networks*, *32*, 17-31.
2. Chifor, B. C., Bica, I., Patriciu, V. V., & Pop, F. (2018). A security authorization scheme for smart home Internet of Things devices. *Future Generation Computer Systems*, *86*, 740749.
3. Bonetto, R., Bui, N., Lakkundi, V., Olivereau, A., Serbanati, A., & Rossi, M. (2012, June). Secure communication for smart IoT objects: Protocol stacks, use cases and practical examples. In *2012 IEEE international symposium on aworld of wireless, mobile and multimedia networks (WoWMoM)* (pp. 1-7). IEEE.
4. Santoso, F. K., & Vun, N. C. (2015, June). Securing IoT for smart homesystem. In *2015 International Symposium on Consumer Electronics (ISCE)* (pp. 1-2). IEEE.