

# **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**<sup>®</sup>  
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**

1. Abstract
2. Introduction
3. Literature survey
4. Overview of the work
  - 4.1 Architecture and methodology
  - 4.2 Advantages
  - 4.3 Flow diagram
  - 4.4 Software requirements
  - 4.5 Hardware requirements
5. Implementation
  - 5.1 Project modules
  - 5.2 Execution of project
  - 5.3 Results
6. Conclusion and practical advantages
7. References.

## 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 enormous number 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 and send 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, healthcare systems and even smart city. it's critical to determine a secure channel between the sensors and 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 low resource 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 and manipulation, 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 spread malware onto corporate networks.

In this project we have 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 properties make SHA-256 this protected.

- 1) To begin with, it is practically difficult to recreate the underlying information from the hash value. A brute force attack would need to make 2<sup>256</sup> endeavors to create the underlying information.
- 2) Second, having two messages with a similar hash value (called a collision) is amazingly improbable. With 2<sup>256</sup> 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 information; this 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 very easy.

## **LITERATURE SURVEY**

### **1. Survey on secure communication protocols for the Internet of Things**

The Internet of Things (IoT) is planned as an organization of highly connected devices (things). In today's 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 as to distinguish the preferences and downsides of every protocol.

### **2. A security authorization scheme for smart home Internet of Things devices**

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 opening the FIDO private key 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 UCNs will 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 standard techniques.

### **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 designed to prevent eavesdropping, tampering, or message forgery.

**8. "Energy Efficiency in M2M Networks Y. B. Saied, A. Oliveureau, and D. Zeglache.**

Security requirements for the integration of emerging M2M networks in future internet of things are addressed. The heterogeneous nature of M2M devices raises new 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.

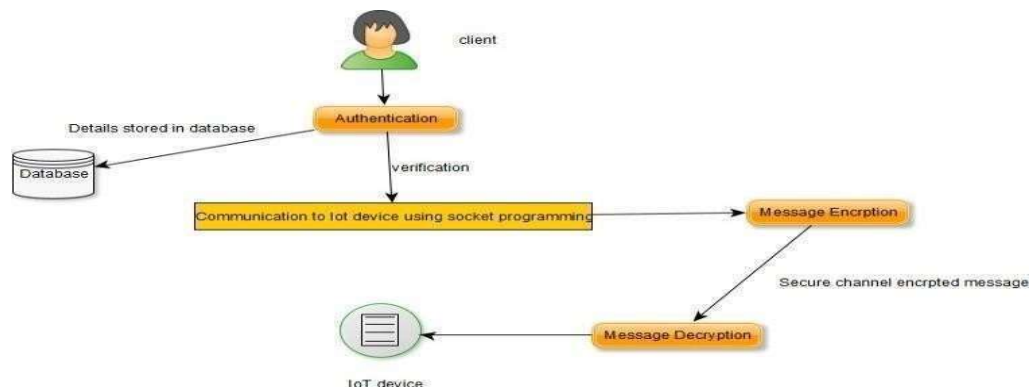
According to a 2020 report by a threat intelligence team called Unit 42, 98% of the 1.2 million IoT devices on corporate networks they analyzed had no capability to encrypt traffic. As a result, 57% of these IoT devices were susceptible to traffic interception and manipulation, among other things. The 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()
```

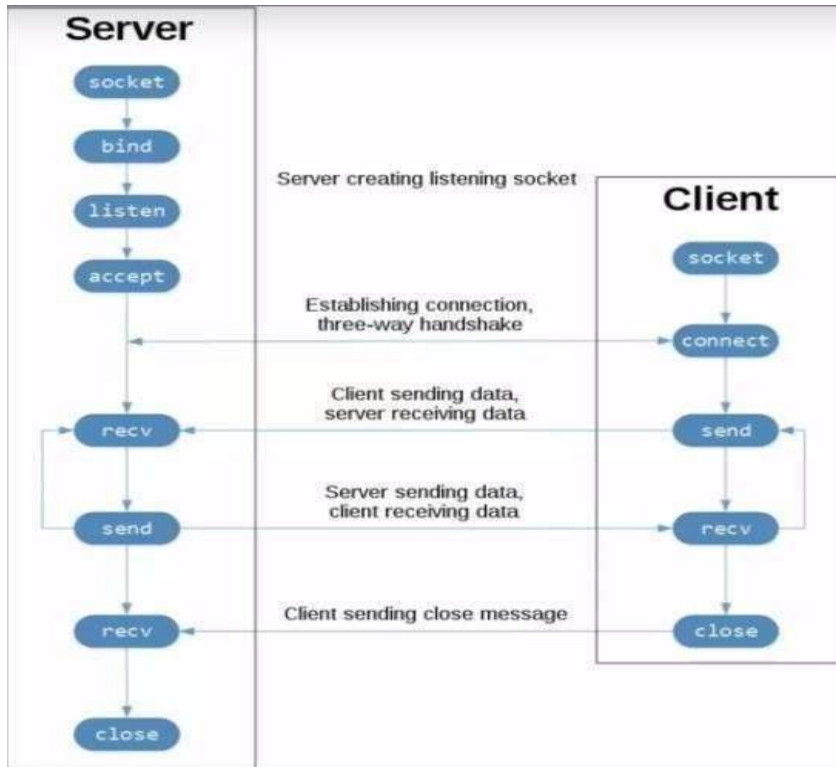
```
s.listen()
```

```
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

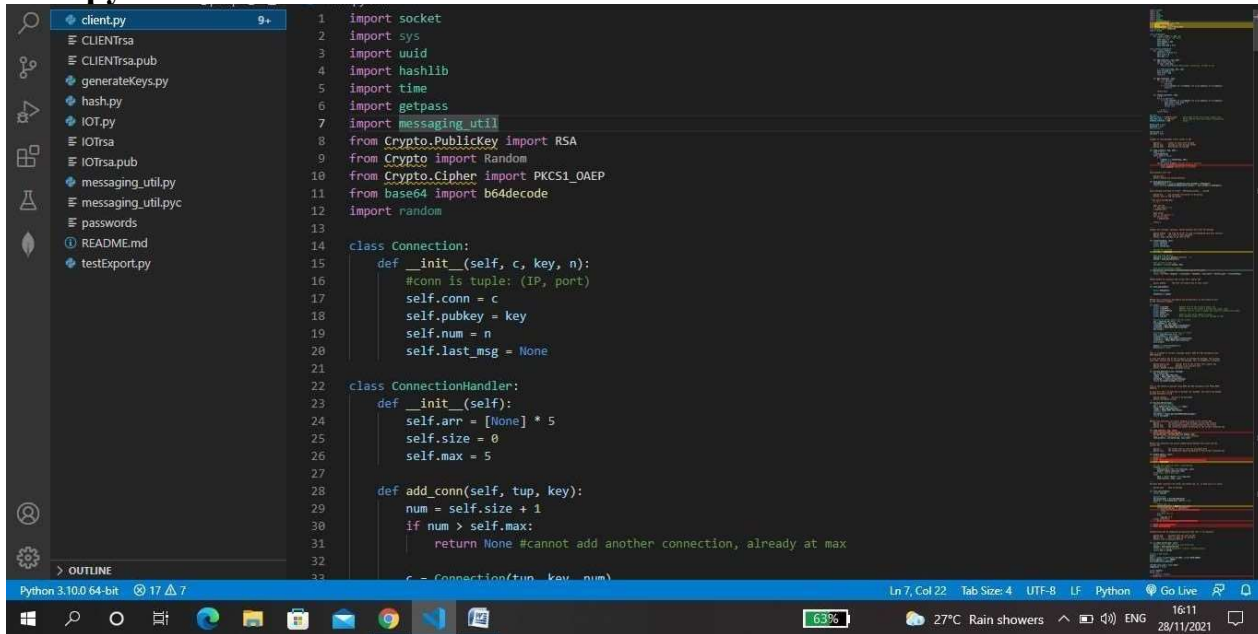
To prevent the attacks to steal the messages and data we need to encrypt the messages such that it should be hard for the hackers to decrypt the traffic. Here in this project we are using the RSA algorithm for encrypting the data; the algorithm is almost impossible to break. It is an asymmetric key cryptographic algorithm based on public key encryption.

## 3) User Authentication and stored our details in MongoDB

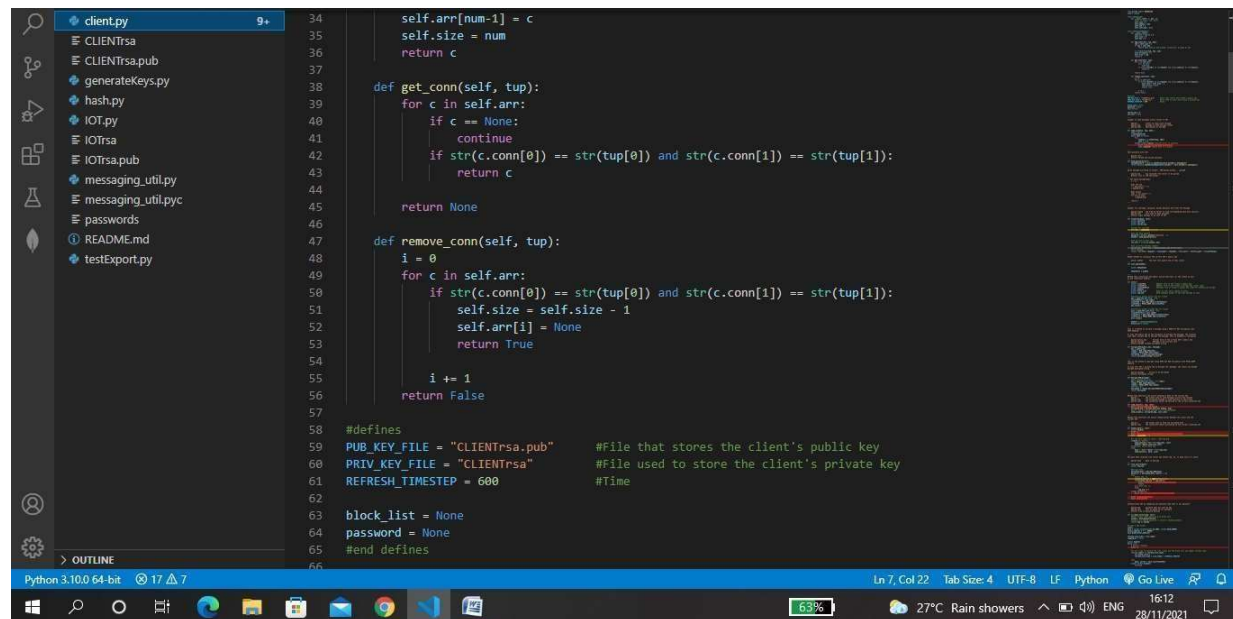
In our method, we also added another feature for authentication of the user whose details are stored in the database. We used MongoDB for storing the details; this database is easy to use and is highly scalable. Hence we used the MongoDB database. Concerning the security reasons, we also hashed the passwords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

## CODE:

### Client code: client.py



```
1 import socket
2 import sys
3 import uuid
4 import hashlib
5 import time
6 import getpass
7 import messaging_util
8 from Crypto.PublicKey import RSA
9 from Crypto import Random
10 from Crypto.Cipher import PKCS1_OAEP
11 from base64 import b64decode
12 import random
13
14 class Connection:
15     def __init__(self, c, key, n):
16         #conn is tuple: (IP, port)
17         self.conn = c
18         self.pubkey = key
19         self.num = n
20         self.last_msg = None
21
22 class ConnectionHandler:
23     def __init__(self):
24         self.arr = [None] * 5
25         self.size = 0
26         self.max = 5
27
28     def add_conn(self, tup, key):
29         num = self.size + 1
30         if num > self.max:
31             return None #cannot add another connection, already at max
32
33         c = Connection(tup, key, num)
```



```
34         self.arr[num-1] = c
35         self.size = num
36         return c
37
38     def get_conn(self, tup):
39         for c in self.arr:
40             if c == None:
41                 continue
42             if str(c.conn[0]) == str(tup[0]) and str(c.conn[1]) == str(tup[1]):
43                 return c
44
45         return None
46
47     def remove_conn(self, tup):
48         i = 0
49         for c in self.arr:
50             if str(c.conn[0]) == str(tup[0]) and str(c.conn[1]) == str(tup[1]):
51                 self.size = self.size - 1
52                 self.arr[i] = None
53                 return True
54             i += 1
55         return False
56
57
58 #defines
59 PUB_KEY_FILE = "CLIENTrsa.pub" #File that stores the client's public key
60 PRIV_KEY_FILE = "CLIENTrsa" #File used to store the client's private key
61 REFRESH_TIMESTEP = 600 #Time
62
63 block_list = None
64 password = None
65 #end defines
66
```

This screenshot shows a Python IDE with a file explorer on the left and a code editor in the center. The file explorer lists several files: client.py, CLIENTrsa, CLIENTrsa.pub, generateKeys.py, hash.py, IOT.py, IOTrsa, IOTrsa.pub, messaging\_util.py, messaging\_util.pyc, passwords, README.md, and testExport.py. The code editor displays the following Python code:

```
67 secret_num = 0
68 iot_salt = None
69 ...
70 wrapper to send messages across socket to IOT
71 ...
72 @param s socket to send stuff through
73 @param msg message to send through socket
74 @param addr destination of message
75 ...
76
77 def send_socket(s, msg, addr) :
78     sent = False
79     s.settimeout(30)
80     while sent == False:
81         try:
82             numSent = s.sendto(msg, addr)
83             sent = True
84         except socket.timeout: #socket.error is subclass
85             print "Timeout, trying again later..."
86             time.sleep(60) #check back in a minute
87     ...
88
89 hash password with salt
90 ...
91 @param salt
92 @return hashed and salted password
93 ...
94 def hash_password(salt):
95     hashedpassword = hashlib.sha256(password.encode()).hexdigest()
96     return hashlib.sha256(hashedpassword.encode() + salt.encode()).hexdigest()
97
98 ...
99 parse message according to format - CMD:param1,param2,...,paramN
```

The status bar at the bottom indicates Python 3.10.0 64-bit, 17 tabs, and the current file is client.py at line 7, column 22. The system tray shows 62% battery, 27°C, rain showers, and the date 28/11/2021.

This screenshot shows the same Python IDE with the code editor displaying the following Python code:

```
100
101 @param msg msg received from server to be parsed
102 @return list of cmd and params
103 ...
104 def parse_message(msg):
105     x = []
106
107     #get the cmd
108     c = msg.split(":",1)
109     x.append(c[0])
110
111     #get params
112     args = c[1].split(",")
113     for arg in args:
114         x.append(arg)
115
116     return x
117
118 ...
119
120 prompts for username, password, hashes password and forms the message
121
122 @param msgnum num used by server to find corresponding hash more securely
123 @param salt salt used to hash password
124 @return login message to be sent to IOT
125 ...
126 def connect(msgnum, salt):
127     global password
128     global iot_salt
129     global secret_num
130
131     #prompt for username
```

The status bar at the bottom indicates Python 3.10.0 64-bit, 17 tabs, and the current file is client.py at line 7, column 22. The system tray shows 62% battery, 27°C, rain showers, and the date 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

132 username = raw_input("username : ")
133
134 #get and hash password
135 password = getpass.getpass("password : ")
136 hashed = hash_password(salt)
137
138 #set up salt to test iot
139 iot_salt = str(uuid.uuid4()).hex)
140
141 #set up diffie hellman numbers
142 secret_num, diffie_pub = messaging_util.get_diffie_nums()
143 #form message
144 return "ACK:PASS:"+msgnum+","+username+","+hashed+","+iot_salt+","+diffie_pub+","+clientPubText
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

```
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

164 global clientPubText #Textual form of client's public key (used for sending and things)
165 global handler
166 global block_list #List of (IP, port) tuples to block
167 global seq_num #The sequence number of the next message to send
168
169 #Initialize global public key for client
170 pub = open(PUB_KEY_FILE, "r")
171 clientPubText = pub.read()
172 clientPub = RSA.importKey(clientPubText)
173 clientPub = PKCS1_OAEP.new(clientPub)
174 pub.close()
175
176 #Initialize global private key for client
177 priv = open(PRIV_KEY_FILE, "r")
178 clientPrivText = priv.read()
179 clientPriv = RSA.importKey(clientPrivText)
180 clientPriv = PKCS1_OAEP.new(clientPriv)
181 priv.close()
182
183 handler = ConnectionHandler()
184 block_list = list()
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```



```
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

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 across the network
@param msg The uncrpyted 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 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

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
...
```

```
261 ***
262 def rcv_secure(data):
263     global seq_num
264
265     #decrypt data
266     decrypted_data = decrypt_RSA(data)
267     param_arr = decrypted_data.rsplit(",",1)
268     try:
269         #check seq. no
270         recieved_seq_num = long(param_arr[1])
271         if(recieved_seq_num != seq_num+1):
272             print "Incorrect sequence number recieved"
273             return
274         #set next seq. no
275         else:
276             seq_num += 2
277     except ValueError:
278         print "Non Integer Sequence Number recieved"
279
280     print "Decrypted data: "
281     print param_arr[0]
282
283 ***
284 authenticates IOT by comparing the password they sent w/ our password
285
286     @param pwd      password that was sent by IOT
287     @param salt      salt to apply to our password
288     @return True if password matches
289 ***
290 def is_legit_server(pwd, salt):
291     #hash (client side) password with given salt
292     hashed = hash_password(salt)
293     #compare our hashed password w/ server's hashed password
294     return pwd == hashed
295
296 #create a UDP socket
297 init()
298 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
299 server_address = ('', 50000)
300 sock.bind(server_address)
301
302 refresh_list_time = time.time()
303 logging_on = False
304
305 global handler
306 while True:
307     # Receive response
308     print ""
309
310     #if it's time to refresh the list, clear out the block list and update refresh time
311     if(time.time() >= refresh_list_time):
312         del block_list[:]
313         refresh_list_time = time.time() + REFRESH_TIMESTEP
314
315     try:
316         data, server = sock.recvfrom(8192)
317     except socket.timeout:
318         continue
319
320     print "Data received from: ", server
321
322     #if server is being blocked, don't read message
323     if(server in block_list):
324         continue
```

```
293 #compare our hashed password w/ server's hashed password
294 return pwd == hashed
295
296 #create a UDP socket
297 init()
298 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
299 server_address = ('', 50000)
300 sock.bind(server_address)
301
302 refresh_list_time = time.time()
303 logging_on = False
304
305 global handler
306 while True:
307     # Receive response
308     print ""
309
310     #if it's time to refresh the list, clear out the block list and update refresh time
311     if(time.time() >= refresh_list_time):
312         del block_list[:]
313         refresh_list_time = time.time() + REFRESH_TIMESTEP
314
315     try:
316         data, server = sock.recvfrom(8192)
317     except socket.timeout:
318         continue
319
320     print "Data received from: ", server
321
322     #if server is being blocked, don't read message
323     if(server in block_list):
324         continue
```



```
325 #if the client is connected to an IOT, decrypt message and send encrypted message
326 if handler.get_conn(server) != None:
327     c = handler.get_conn(server)
328     recv_secure(data)
329     handle_data(sock, c)
330     continue
331
332 #parse the command
333 cmd = messaging_util.parse_message(data)
334
335 #connect message - server asking user to login
336 if(cmd[0] == "CONNECT") :
337     #if we're currently trying to log on, ignore other connect messages
338     if(logging_on):
339         continue
340
341     #give user the option of not connecting to the device - loop to prevent illegal chars
342     while True:
343         c = raw_input("Do you want to connect to "+cmd[3]+"? (Y/N) ")
344         #user tries to log onto IOT
345         if(c == 'Y' or c == 'y') :
346             msg = connect(cmd[2],cmd[1])
347             #changing the port # to the port the server would listen to
348             ackaddr = (server[0], 50001)
349             #print "Sending ", msg, " to ", ackaddr
350             send_socket(sock, msg, ackaddr)
351
352             #user is currently trying to log on -> should be true ;P
353             logging_on = True
354
355             break
356
357
```

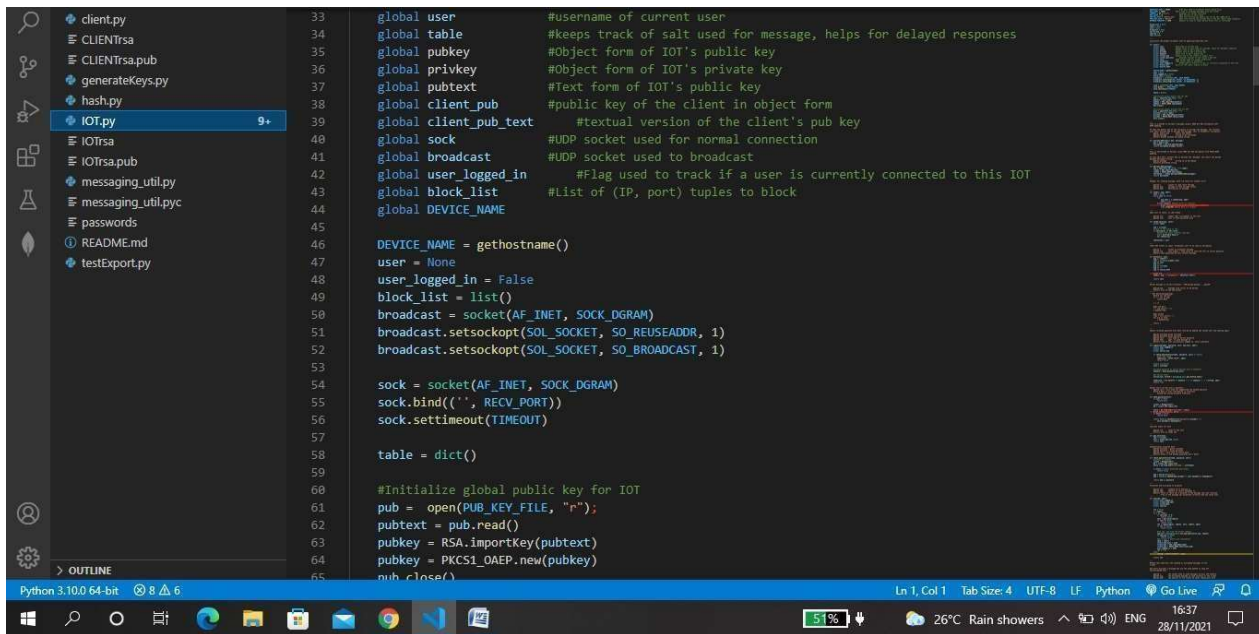
```
358 #user doesn't wanna log onto IOT, block brocasts for now
359 elif(c == 'N' or c=='n') :
360     #put in spam numbers
361     block_list.append(server)
362
363     break
364
365 #ack message / encrypt?
366 elif(cmd[0] == "ACK") :
367     if(cmd[1] == "ENCRYPT") :
368         logging_on = False
369         conn = ackaddr
370
371 #check if there is the correct number of arguments and auth. IOT
372 if(len(cmd) == 5 and is_legit_server(cmd[3], iot_salt)):
373     #set up IOT pub. key
374     init_pub(cmd[2])
375     #get client pub. key
376     #msg = get_pub_msg()
377
378     #generate the starting seq. no.
379     seq_num = messaging_util.set_seq_num(secret_num, cmd[4])
380     if seq_num is None:
381         print "This server is kinda sketchy - bad diffie number"
382         continue
383     #try to add the connection
384     new_conn = handler.add_conn(conn, IOTpubtext)
385     if new_conn == None:
386         print "Unable to add IOT, max connections enabled"
387         continue
388
389     print "Congrats, we logged on."
390     #send socket(sock, msg, ackaddr)
```

```
390
391         handle_data(sock, new_conn) #send something
392
393     else:
394         print "ERROR: There was an error getting the public key from the IOT"
395
396 elif(cmd[0] == "ERROR") :
397     logging_on = False
398     if(cmd[1] == "LOGIN"):
399         print "ERROR : Invalid username/password."
400     if(cmd[1] == "ARGUMENT"):
401         print "ERROR : Bad argument."
402     if(cmd[1] == "NULLPUBKEY"):
403         print "ERROR: Null public key sent."
404
405 else :
406     break
407
408 sock.close()
409
410
411
412
```

## IOT CODE:

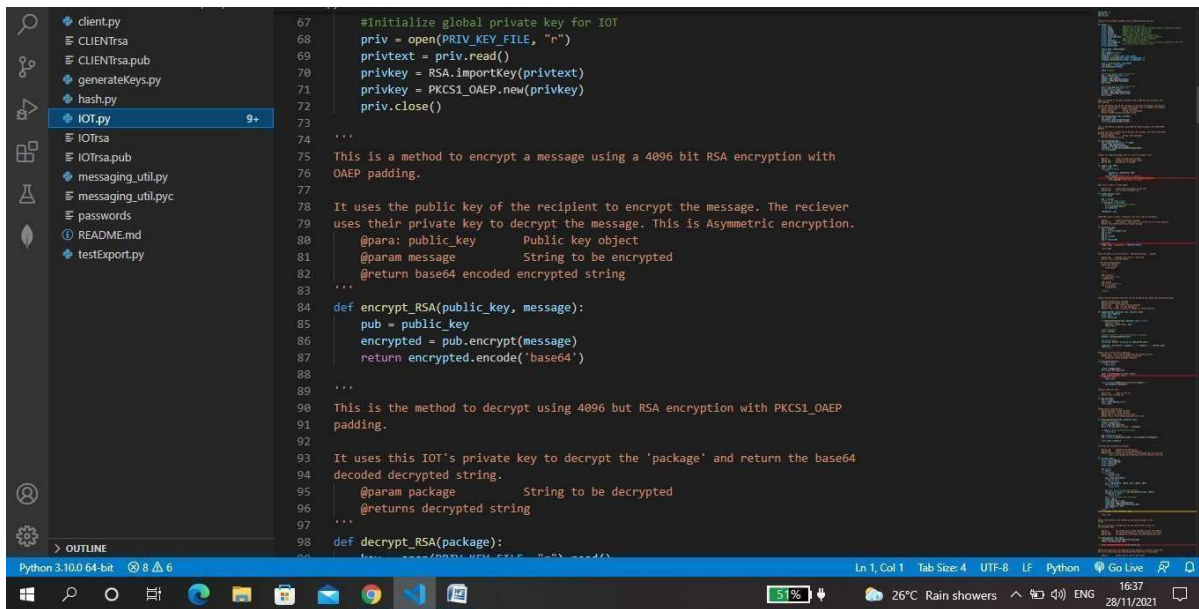
Iot.py:

```
1 import hashlib, uuid
2 import os, sys, time, random
3 import messaging_util
4 from socket import *
5 from Crypto.PublicKey import RSA
6 from Crypto import Random
7 from Crypto.Cipher import PKCS1_OAEP
8 from base64 import b64decode
9 from pymongo import MongoClient
10
11 #DEFINES
12 DEVICE_NAME = "" #Name of the device running the script
13 BROADCAST_PORT = 50000 #The port used to broadcast device being alive
14 RECV_PORT = 50001 #Port used to exchange messages with the client
15 TIMEOUT = 60 #seconds #Timeout for socket recv/send
16 MAX_CACHE = 10 #Max entries in salt table
17 PUB_KEY_FILE = "IOTrsa.pub" #The file storing the public key of the IOT (4096 bits)
18 PRIV_KEY_FILE = "IOTrsa" #The file storing the private key of the IOT (Never send anywhere)
19 REFRESH_TIMESTEP = 3600 #Amount of time it takes before block list is refreshed
20
21 block_list = None
22 table = None
23 sock = None
24 broadcast = None
25 secret_num = 0
26 seq_num = 0
27 #END DEFINES
28
29 ...
30 initialize the global variables such as public/private key info
31 ...
32 def init():
33     global user #username of current user
```



```
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 privkey = PKCS1_OAEP.new(pubkey)
65 priv.close()
```

Python 3.10.0 64-bit 8 6 Ln 1, Col 1 Tab Size: 4 UTF-8 LF Python Go Live 1637 28/11/2021



```
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 receiver
79 uses their private key to decrypt the message. This is Asymmetric encryption.
80 @param: 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 bit 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):
99     priv = open(PRIV_KEY_FILE, "r")
100     privtext = priv.read()
101     privkey = RSA.importKey(privtext)
102     privkey = PKCS1_OAEP.new(privkey)
103     priv.close()
104     package = package.decode('base64')
105     decrypted = privkey.decrypt(package)
```

Python 3.10.0 64-bit 8 6 Ln 1, Col 1 Tab Size: 4 UTF-8 LF Python Go Live 1637 28/11/2021

```
99 key = open(PRIV_KEY_FILE, "r").read()
100 rsakey = RSA.importKey(key)
101 rsakey = PKCS1_OAEP.new(rsakey)
102 decrypted = rsakey.decrypt(b64decode(package))
103 return decrypted
104
105 """
106 Wrapper for sending message, with try catch for timeout error
107
108 @param s      socket to send stuff through
109 @param msg    message to send through socket
110 @param addr   destination of message
111 """
112 def send(s, msg, addr):
113     sent = False
114     while sent == False:
115         try:
116             num_sent = s.sendto(msg, addr)
117             sent = True
118         except timeout: #socket.error is subclass
119             print "No network connection, trying again later..."
120             time.sleep(60) #check back in a minute
121
122 """
123 adds salt to table, in LRU scheme
124
125 @param num    number that corresponds to the salt
126 @param salt   salt to hash passwords with
127 """
128 def cache_salt(num, salt):
129     global table
130     num = int(num)
```

```
131 num = int(num)
132 #check if the table is full
133 if len(table) == MAX_CACHE :
134     #remove the least recently used key
135     lru = min(table.keys())
136     del table[lru]
137
138 table[num] = salt
139
140 """
141 takes UDP socket as input, broadcasts salt to be used in encryption
142
143 @param s      socket to broadcast through
144 @param num    salt number - used later to find the salt to verify password
145 @return salt generated by this connect message
146 """
147 def brocast(s, num):
148     msg = "CONNECT:"
149     salt = str(uuid.uuid4().hex)
150     msg += salt
151     msg += " "
152     msg += str(num)
153     msg += " "
154     msg += DEVICE_NAME
155
156     print msg
157     send(s, msg, ('<broadcast>', BROADCAST_PORT))
158
159     return salt
160
161 """
162 Parses message in correct protocol - CMD:param1,param2,...,paramN
163 """
```



```
164 @param msg message from server to be parsed
165 @return list of cmd and params
166 ...
167 def parse_message(msg):
168     #check for cmd part
169     if ":" not in msg:
170         return None
171
172     x = []
173
174     #get cmd part
175     c = msg.split(":",1)
176     x.append(c[0])
177
178     #get params
179     args = c[1].split(",")
180     for arg in args:
181         x.append(arg)
182
183     return x
184 ...
185
186 Checks recieved password with pass from db by adding the cached salt and hashing again
187
188 @param username given username
189 @param password given password
190 @param salt salt used to verify password
191 @param addr addr. to send message to
192 @return True if user successfully logged on. False otherwise
193 ...
194
195 def login(username, password, salt, new_salt, addr):
196     global user_logged_in
197     global user
198     global secret_num
199
200     if check_password(username, password, salt) is False:
201         #user not found
202         send(sock, "ERROR:LOGIN", addr)
203         return False
204
205     #login successful
206     user = username
207
208     #re-hash password to ensure Encrypt ack is authentic
209     newhash = hash_password(new_salt)
210
211     #do diffie stuff
212     secret_num, diffyH = messaging_util.get_diffie_nums()
213
214     send(sock, "ACK:ENCRYPT,"+ pubtext + "," + newhash + "," + diffyH, addr)
215     return True
216
217 ...
218 hashes salts current user's password
219 @param salt - salt used for augmenting the hashed password
220 @return None if user information is incorrect
221         hashed and salted password otherwise
222 ...
223 def hash_password(salt):
224     if user is None:
225         return None
226
227     client = MongoClient()
228     db = client.IOT_login_info
```

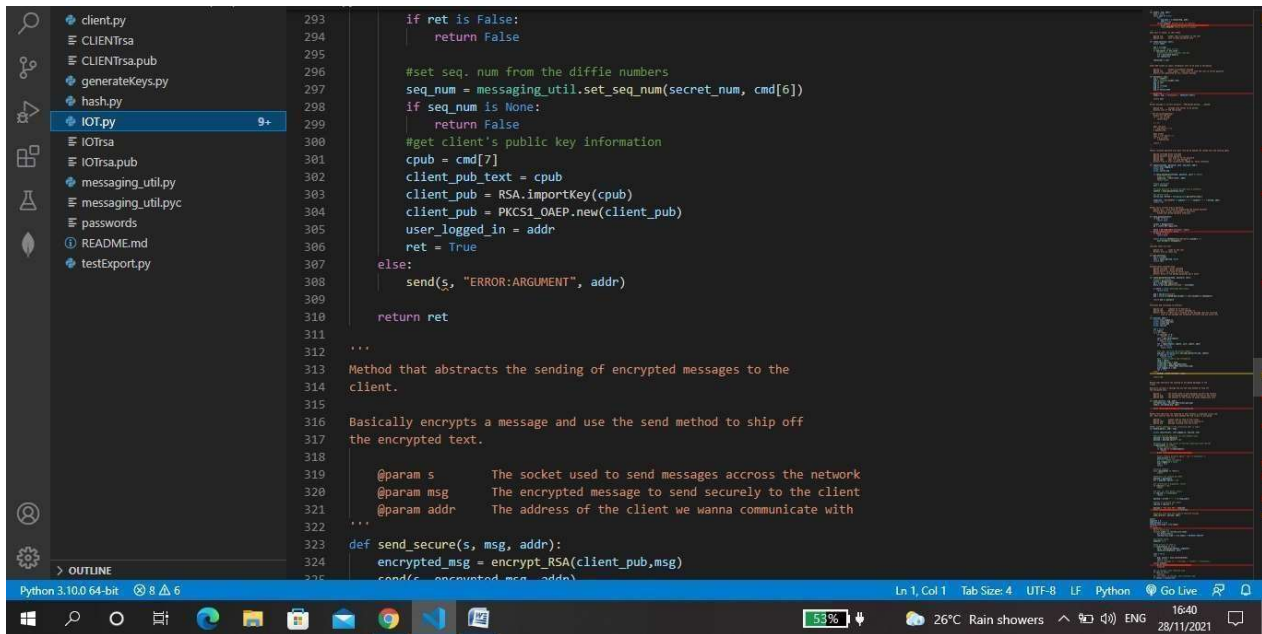
```
196 global user_logged_in
197 global user
198 global secret_num
199
200 if check_password(username, password, salt) is False:
201     #user not found
202     send(sock, "ERROR:LOGIN", addr)
203     return False
204
205 #login successful
206 user = username
207
208 #re-hash password to ensure Encrypt ack is authentic
209 newhash = hash_password(new_salt)
210
211 #do diffie stuff
212 secret_num, diffyH = messaging_util.get_diffie_nums()
213
214 send(sock, "ACK:ENCRYPT,"+ pubtext + "," + newhash + "," + diffyH, addr)
215 return True
216
217 ...
218 hashes salts current user's password
219 @param salt - salt used for augmenting the hashed password
220 @return None if user information is incorrect
221         hashed and salted password otherwise
222 ...
223 def hash_password(salt):
224     if user is None:
225         return None
226
227     client = MongoClient()
228     db = client.IOT_login_info
```

```
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

228 db = client.IOT.login_info
229
230 entry = db.find_one({"username": user})
231 print "ENTRY >>> ", entry
232 if entry is None:
233     return None
234
235 return hashlib.sha256(entry["password"].encode() + \
236     salt.encode()).hexdigest()
237
238 ...
239 searches table for salt
240
241 @param num    index of the salt
242 @return salt at index num
243 ...
244 def get_salt(num):
245     num = int(num)
246     salt = table.get(num, None)
247     return salt
248
249 ...
250 Authenticates received hash
251 @param username - given username
252 @param password - given password
253 @param salt - salt used to offset hash
254 @return False if the hashed passwords don't match
255 ...
256 def check_password(username, password, salt):
257     #connect to mongodb
258     client = MongoClient()
259     db = client.IOT.login_info
260     entry = db.find_one({"username": username})
```

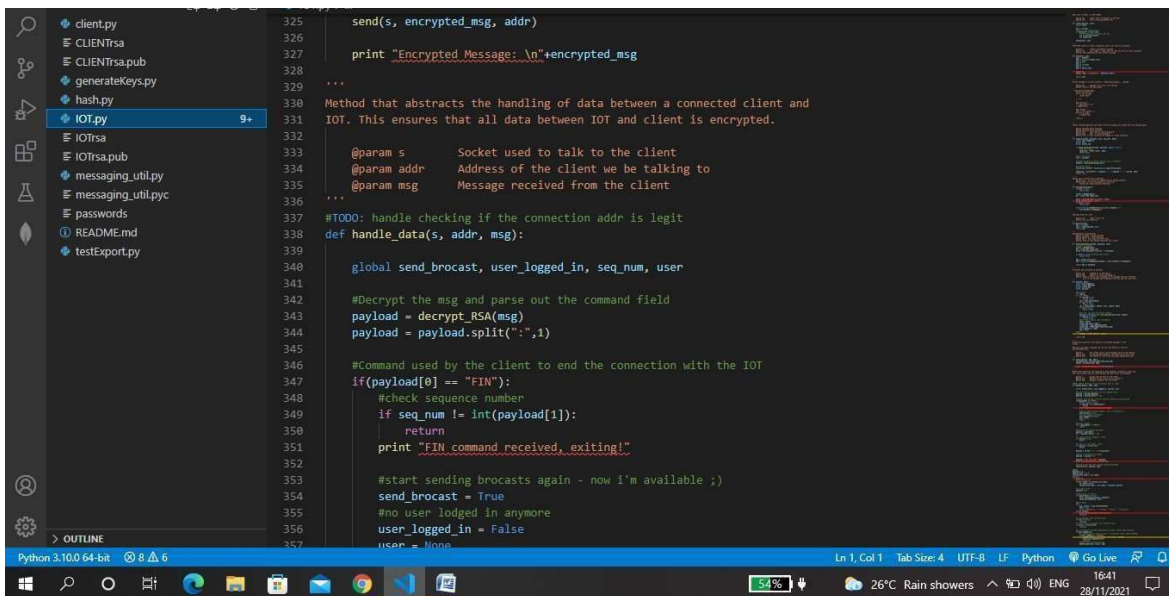
```
client.py
CLIENTrsa
CLIENTrsa.pub
generateKeys.py
hash.py
IOT.py 9+
IOTrsa
IOTrsa.pub
messaging_util.py
messaging_util.pyc
passwords
README.md
testExport.py

260 entry = db.find_one({"username": username})
261
262 if entry is None: #username dont match
263     return False
264
265 pwd = entry["password"]
266 pwd = (hashlib.sha256(pwd.encode() + salt.encode()).hexdigest()
267
268 return pwd == password
269
270 ...
271 Processes ACK according to protocol
272
273 @param cmd    command we're dealing w/
274 @param addr    address to return messages to
275 @return False if there was a problem in the message that was received
276         True if the message was formatted correctly and has valid info
277 ...
278 def ack(cmd, addr):
279     global user_logged_in
280     global client_pub_text
281     global client_pub
282     global seq_num
283
284     ret = False
285     c = cmd[1]
286     if c == "PASS":
287         if len(cmd) != 8:
288             return False
289         salt = get_salt(cmd[2])
290         if salt == None:
291             return False
292         ret = login(cmd[3], cmd[4], salt, cmd[5], addr)
```



```
293     if ret is False:
294         return False
295
296     #set seq. num from the diffie numbers
297     seq_num = messaging_util.set_seq_num(secret_num, cmd[6])
298     if seq_num is None:
299         return False
300
301     #get client's public key information
302     cpub = cmd[7]
303     client_pub_text = cpub
304     client_pub = RSA.importKey(cpub)
305     client_pub = PKCS1_OAEP.new(client_pub)
306     user_logged_in = addr
307     ret = True
308 else:
309     send(s, "ERROR:ARGUMENT", addr)
310
311 return ret
312
313 ...
314 Method that abstracts the sending of encrypted messages to the
315 client.
316 Basically encrypts a message and use the send method to ship off
317 the encrypted text.
318
319 @param s          The socket used to send messages accross the network
320 @param msg        The encrypted message to send securely to the client
321 @param addr       The address of the client we wanna communicate with
322 ...
323 def send_secure(s, msg, addr):
324     encrypted_msg = encrypt_RSA(client_pub,msg)
325     send(s, encrypted_msg, addr)
```

Python 3.10.0 64-bit 8 6 Ln 1, Col 1 Tab Size: 4 UTF-8 LF Python Go Live 16:40 28/11/2021



```
325     send(s, encrypted_msg, addr)
326
327     print "Encrypted Message: \n"+encrypted_msg
328
329 ...
330 Method that abstracts the handling of data between a connected client and
331 IOT. This ensures that all data between IOT and client is encrypted.
332
333 @param s          Socket used to talk to the client
334 @param addr       Address of the client we be talking to
335 @param msg        Message received from the client
336 ...
337 #TODO: handle checking if the connection addr is legit
338 def handle_data(s, addr, msg):
339
340     global send_broadcast, user_logged_in, seq_num, user
341
342     #Decrypt the msg and parse out the command field
343     payload = decrypt_RSA(msg)
344     payload = payload.split(":",1)
345
346     #Command used by the client to end the connection with the IOT
347     if(payload[0] == "FIN"):
348         #check sequence number
349         if seq_num != int(payload[1]):
350             return
351         print "FIN command received, exiting!"
352
353     #start sending brocasts again - now i'm available ;)
354     send_broadcast = True
355     #no user lodged in anymore
356     user_logged_in = False
357     user = None
```

Python 3.10.0 64-bit 8 6 Ln 1, Col 1 Tab Size: 4 UTF-8 LF Python Go Live 16:41 28/11/2021

```
357     user = None
358     return
359
360     #invalid command
361     elif (payload[0] != "DATA"):
362         return
363
364     #Otherwise the command was DATA
365     payload = payload[1]
366     arr = payload.rsplit(",",1)
367
368     #if insufficient # arguments, return
369     if (len(arr) < 2):
370         return
371
372     #if seq. no. dont match, return
373     if seq_num != int(arr[1]):
374         return
375
376     payload = arr[0] + "," + str(seq_num+1)
377
378     #update to expected seq number
379     seq_num = seq_num + 2
380
381     payload = "You sent IOT: "+payload
382     print "Decrypted Payload: \n"+payload
383
384     #Securely send back the slightly modified message
385     send_secure(s, payload, addr)
386
387
388 init()
389 msgCount = 0
390 send_broadcast = True
```

```
390 refresh_list_time = time.time()
391 while 1:
392     print ""
393     #print block list
394     if(time.time() >= refresh_list_time):
395         del block_list[:]
396         refresh_list_time = time.time() + REFRESH_TIMESTEP
397
398     #increment salt#
399     msgCount += 1
400
401     #send broadcast if need to
402     if send_broadcast == True:
403         salt = brocast(broadcast, msgCount)
404         cache_salt(msgCount, salt)
405
406     recv = False
407     try:
408         msg, server = sock.recvfrom(8192)
409         recv = True
410         #print "message is " + str(msg) + "\nFrom " + str(server)
411     except timeout:
412         print "Socket timeout, trying again!"
413         continue
414
415     #if no message, just continue loop
416     if recv == False:
417         continue
418     #if the sender is blocked, just continue loop
419     if server in block_list:
420         continue
421
422     #if a connection has been established already, handle data securely
```

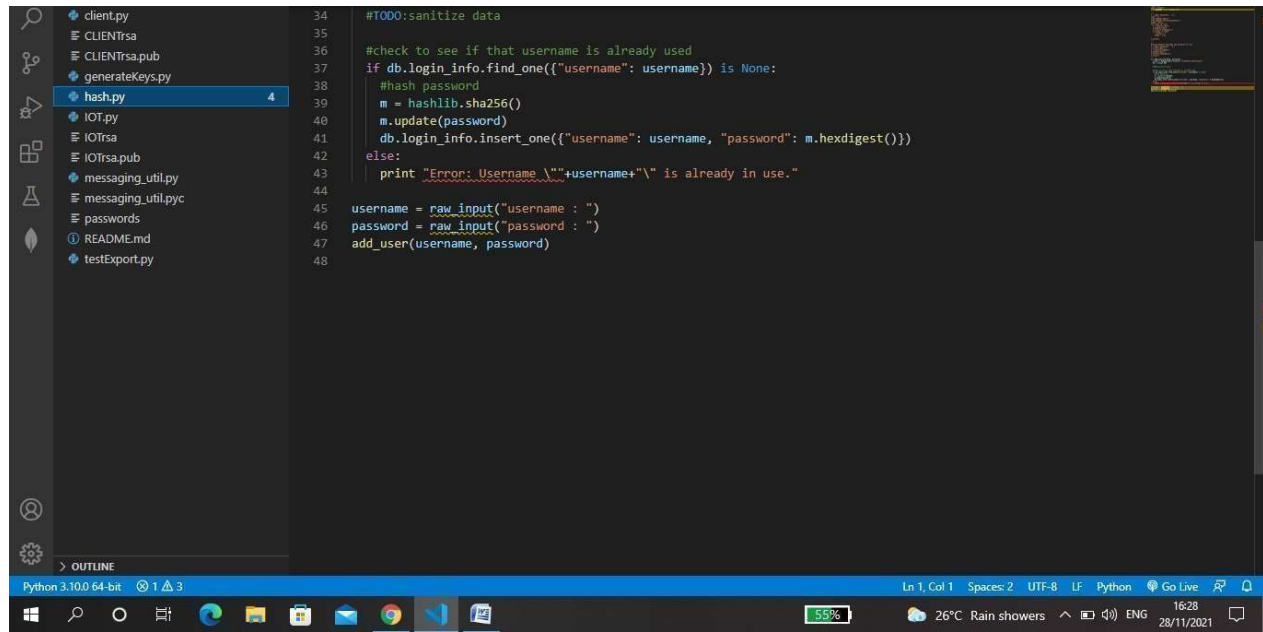


```
422 #if a connection has been established already, handle data securely
423
424 if(user_logged_in):
425     #if the message is from a client != connected client, ignore message
426     if(user_logged_in != server):
427         send(s, "Bitch, I'm already connected.", server)
428         block_list.append(server)
429         continue
430     #handle the encrypted message
431     handle_data(sock, server, msg)
432
433 #Otherwise, data does not have to be encrypted (and shouldn't be)
434 else:
435     cmd = messaging_util.parse_message(msg)
436     if cmd == None: #bad formatting, blocking
437         block_list.append(server)
438         continue
439
440     if cmd[0] == "ACK":
441         success = ack(cmd, server)
442         if success:
443             send_broadcast = False
444             #break
445
446 print("Closing socket")
447 sock.close()
```

## Hash.py:

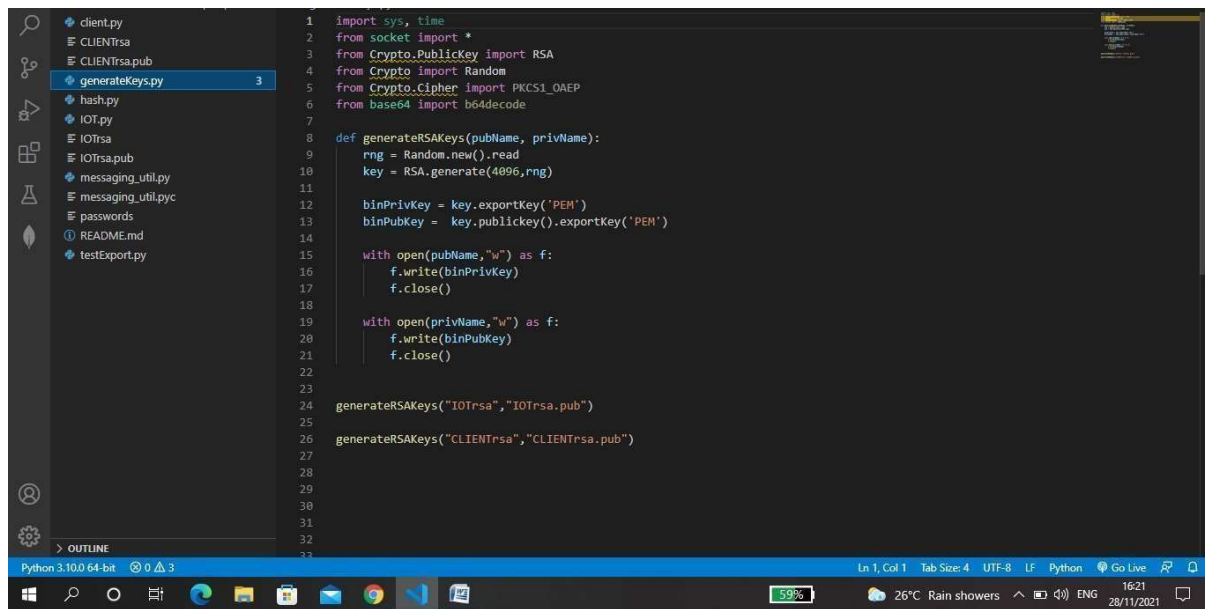
### Hashing and storing in database

```
1 import os
2 import hashlib
3 from pymongo import MongoClient
4
5 ...
6 f = open('passwords', 'w')
7 line = []
8 line.append("admin")
9 line.append("varunsucksweewees")
10 comma = False
11 for field in line:
12     m = hashlib.sha1()
13     m.update(field)
14     f.write(m.hexdigest())
15     if comma == False:
16         f.write(",")
17         comma = True
18
19 f.close()
20 ...
21 ...
22 #writes default username and password to file
23 f = open('passwords', 'w')
24 f.write("admin,")
25 m = hashlib.sha256()
26 m.update("pass")
27 f.write(m.hexdigest())
28 f.close()
29 ...
30 def add_user(username, password):
31     client = MongoClient("mongodb://localhost:27017/isaac")
32     db = client.IOT
33
```



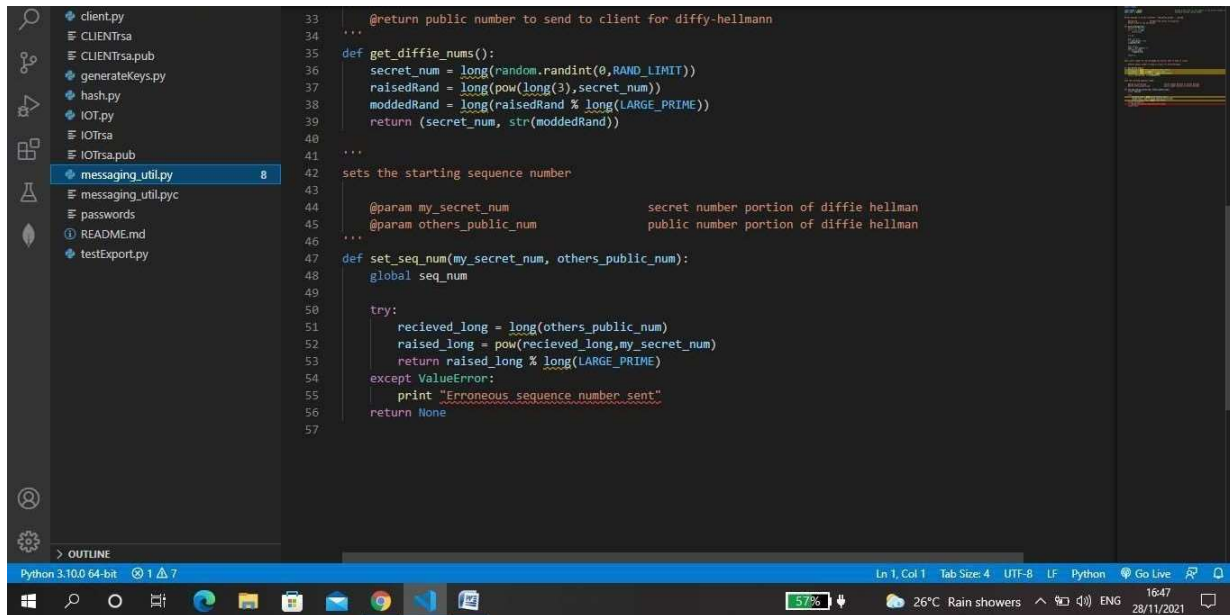
```
34 #TODO:sanitize data
35
36 #check to see if that username is already used
37 if db.login_info.find_one({"username": username}) is None:
38     #hash password
39     m = hashlib.sha256()
40     m.update(password)
41     db.login_info.insert_one({"username": username, "password": m.hexdigest()})
42 else:
43     print "Error: Username \"{}\" is already in use.".format(username)
44
45 username = raw_input("username : ")
46 password = raw_input("password : ")
47 add_user(username, password)
48
```

## Generate keys.py



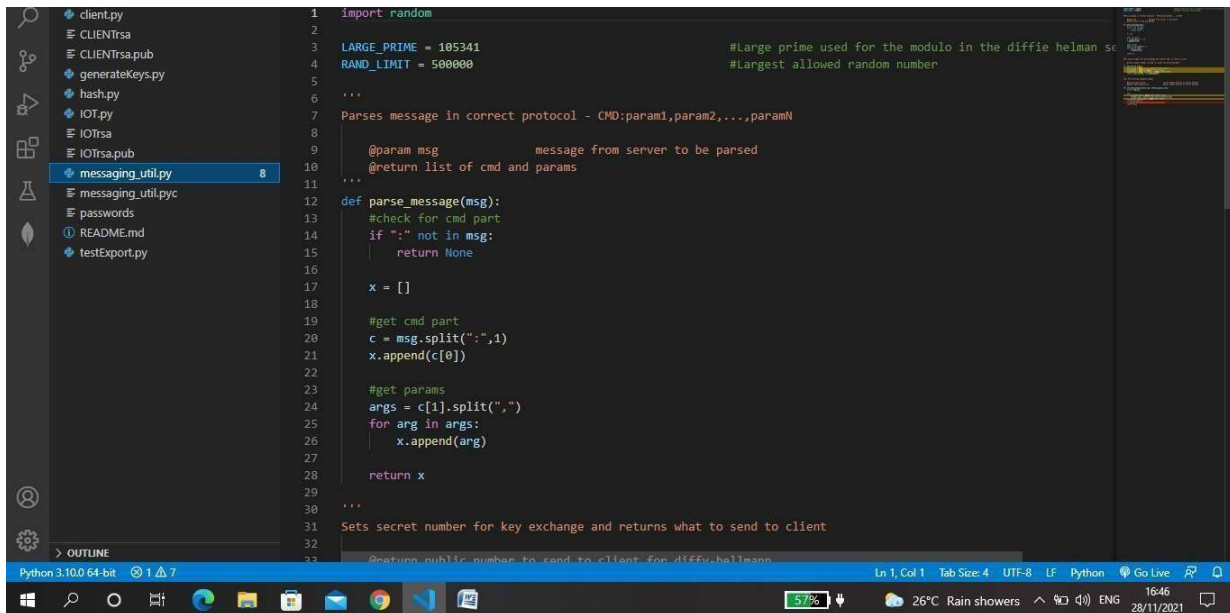
```
1 import sys, time
2 from socket import *
3 from Crypto.PublicKey import RSA
4 from Crypto import Random
5 from Crypto.Cipher import PKCS1_OAEP
6 from base64 import b64decode
7
8 def generateRSAKeys(pubName, privName):
9     rng = Random.new().read
10     key = RSA.generate(4096, rng)
11
12     binPrivKey = key.exportKey('PEM')
13     binPubKey = key.publickey().exportKey('PEM')
14
15     with open(pubName, "w") as f:
16         f.write(binPubKey)
17         f.close()
18
19     with open(privName, "w") as f:
20         f.write(binPrivKey)
21         f.close()
22
23 generateRSAKeys("IOTrsa", "IOTrsa.pub")
24
25 generateRSAKeys("CLIENTrsa", "CLIENTrsa.pub")
26
27
28
29
30
31
32
33
```

## Messaging util.py



This screenshot shows a code editor with the file `messaging_util.py` selected in the left sidebar. The main editor displays the `get_diffie_nums` function, which generates a secret number and a public number for a Diffie-Hellman key exchange. The function includes comments explaining its purpose and parameters.

```
33 @return public number to send to client for diffie-hellmann
34 ...
35 def get_diffie_nums():
36     secret_num = long(random.randint(0,RAND_LIMIT))
37     raisedRand = long(pow(long(3),secret_num))
38     moddedRand = long(raisedRand % long(LARGE_PRIME))
39     return (secret_num, str(moddedRand))
40
41 ...
42 sets the starting sequence number
43
44 @param my_secret_num      secret number portion of diffie hellman
45 @param others_public_num  public number portion of diffie hellman
46 ...
47 def set_seq_num(my_secret_num, others_public_num):
48     global seq_num
49
50     try:
51         recieved_long = long(others_public_num)
52         raised_long = pow(recieved_long,my_secret_num)
53         return raised_long % long(LARGE_PRIME)
54     except ValueError:
55         print "Erroneous sequence number sent"
56     return None
57
```



This screenshot shows the same code editor with the file `messaging_util.py` selected. The main editor displays the `parse_message` function, which parses a message received from a server into a command and parameters. The function includes comments explaining its purpose and parameters.

```
1 import random
2
3 LARGE_PRIME = 105341          #Large prime used for the modulo in the diffie hellman sc
4 RAND_LIMIT = 500000          #Largest allowed random number
5
6 ...
7 Parses message in correct protocol - CMD:param1,param2,...,paramN
8
9 @param msg      message from server to be parsed
10 @return list of cmd and params
11 ...
12 def parse_message(msg):
13     #check for cmd part
14     if ":" not in msg:
15         return None
16
17     x = []
18
19     #get cmd part
20     c = msg.split(":",1)
21     x.append(c[0])
22
23     #get params
24     args = c[1].split(",")
25     for arg in args:
26         x.append(arg)
27
28     return x
29
30 ...
31 Sets secret number for key exchange and returns what to send to client
32
33 @return public number to send to client for diffie-hellmann

```

## EXUCUTION RESULTS:

```
client.py 9+ X
client.py > ...
1 import socket
2 import sys
3 import uuid
4 import hashlib
5 import time
6 import getpass
7 import messaging_util

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\lenovo\Desktop\iot project> python hash.py
username : iotproject
password : iotproject
PS C:\Users\lenovo\Desktop\iot project>
```

```
client.py 9+ X
client.py > ...
1 import socket
2 import sys
3 import uuid
4 import hashlib
5 import time
6 import getpass
7 import messaging_util

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

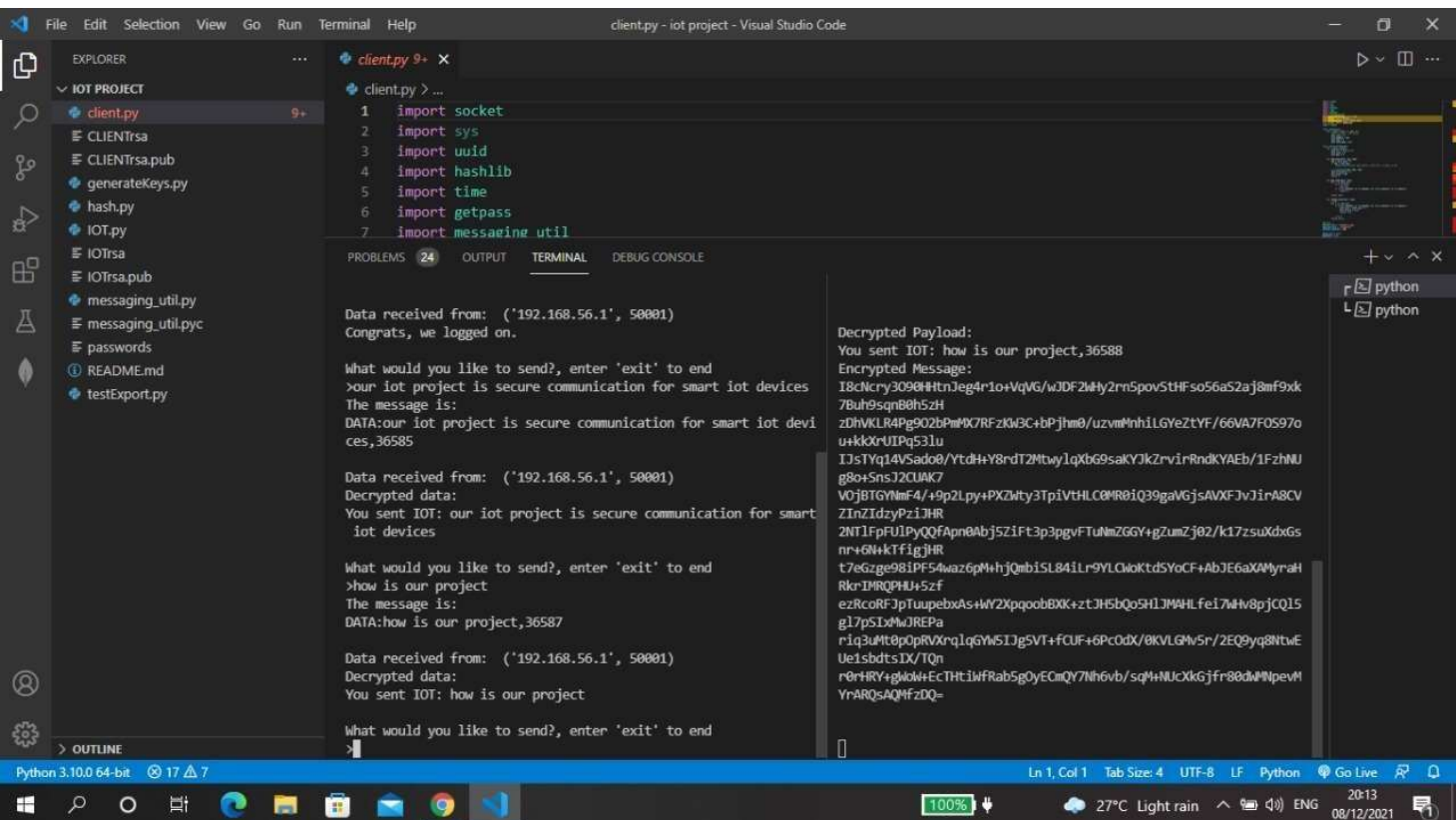
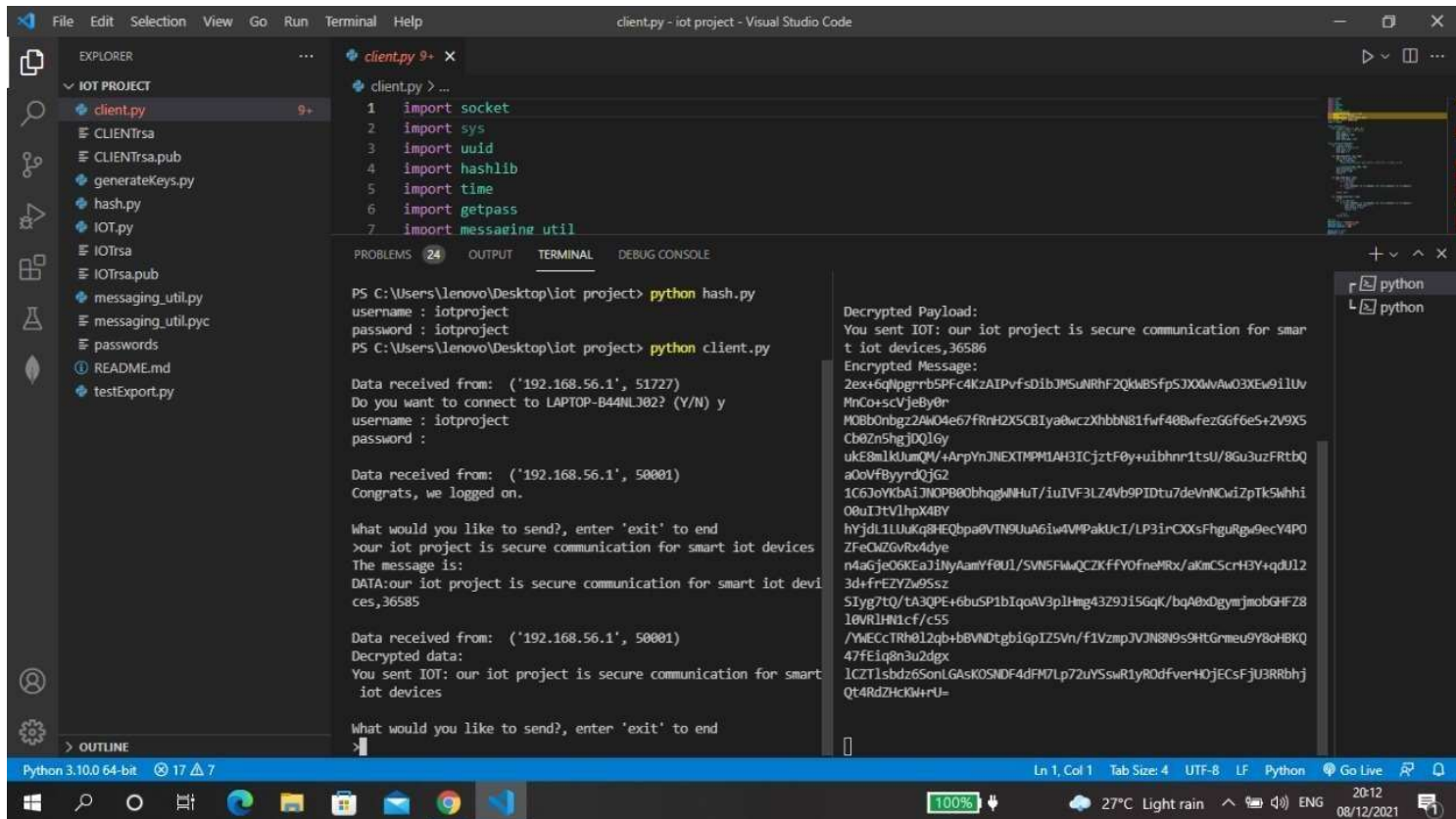
PS C:\Users\lenovo\Desktop\iot project> python hash.py
username : iotproject
password : iotproject
PS C:\Users\lenovo\Desktop\iot project> python client.py

Data received from: ('192.168.56.1', 51727)
Do you want to connect to LAPTOP-B44NL382? (Y/N) y
username : iotproject
password :

Data received from: ('192.168.56.1', 58001)
Congrats, we logged on.

What would you like to send?, enter 'exit' to end
>
```





```
client.py 9+ X
client.py > ...
1 import socket
2 import sys
3 import uuid
4 import hashlib
5 import time
6 import getpass
7 import message_util

>our iot project is secure communication for smart iot devices
The message is:
DATA:our iot project is secure communication for smart iot devices,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:
I8chcry3090H#HnJeg4r1o+VqVG/wJDF2Mhy2rn5povStHF:so56a52aj8mf9xk
7Buh9sqnB0h5Zl
zDhVKLR4Pg902bPmMX7RFzKW3C+bPjhm0/uzvmMnhILGYeZtYF/66VA7F0597o
u+kk0uUIPq531u
IJsTYqI4VSado8/YtdH+Y8rdT2Mtwy1qXbG9saKYJkZrvirRndKYAEB/1FznhU
g8o+SnsJ2CUAK7
VojBTGYmF4/+9p2Lpy+PXZwty3TpIvHLC0MR0iQ39gaVjsAVXFJvJ1rA8CV
ZInZIdzyPziJHR
2NT1FpFUIPyQQFAPn0Abj5ZiFt3p3pgvFTuNmZGGY+gZumZj02/k17zsuXdxGs
nr+6N+kTfigjHR
t7eGzge981PF54waz6pM+hjQmb1SL84iLr9YLQwKtdSYoCF+AbJE6aXAMyrah
RkrIMRQPHu+5zf
ezRcoRFJpTuupebXAs+WY2XpooobBXX+ztJH5bQoSH1JMAHLfei7W+h8pJCQ15
g17pSIxWJREPA
riq3uMt0p0PrVXrq1qGYW5IJgSVT+fCUF+6PCOdX/8KVLGMv5r/2EQ9yq8NtwE
Ue1sbdtsIX/TQn
r0rHRY+gk4W+EcTHTiWFrab5gOyECmQY7Nhbvb/sqM+NUCXkGjfr80dWNPevM
YrARQsAQMFzDQ=

FIN command received, exiting!

CONNECT:75d75e7cb5b44028bc4dfe8cdcb0527,6,LAPTOP-B44NLJ02
```

```
client.py 9+ X
client.py > ...
1 import socket
2 import sys
3 import uuid
4 import hashlib
5 import time
6 import getpass
7 import message_util

>our iot project is secure communication for smart iot devices
The message is:
DATA:our iot project is secure communication for smart iot devices,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)

IJsTYqI4VSado8/YtdH+Y8rdT2Mtwy1qXbG9saKYJkZrvirRndKYAEB/1FznhU
g8o+SnsJ2CUAK7
VojBTGYmF4/+9p2Lpy+PXZwty3TpIvHLC0MR0iQ39gaVjsAVXFJvJ1rA8CV
ZInZIdzyPziJHR
2NT1FpFUIPyQQFAPn0Abj5ZiFt3p3pgvFTuNmZGGY+gZumZj02/k17zsuXdxGs
nr+6N+kTfigjHR
t7eGzge981PF54waz6pM+hjQmb1SL84iLr9YLQwKtdSYoCF+AbJE6aXAMyrah
RkrIMRQPHu+5zf
ezRcoRFJpTuupebXAs+WY2XpooobBXX+ztJH5bQoSH1JMAHLfei7W+h8pJCQ15
g17pSIxWJREPA
riq3uMt0p0PrVXrq1qGYW5IJgSVT+fCUF+6PCOdX/8KVLGMv5r/2EQ9yq8NtwE
Ue1sbdtsIX/TQn
r0rHRY+gk4W+EcTHTiWFrab5gOyECmQY7Nhbvb/sqM+NUCXkGjfr80dWNPevM
YrARQsAQMFzDQ=

FIN command received, exiting!

CONNECT:75d75e7cb5b44028bc4dfe8cdcb0527,6,LAPTOP-B44NLJ02
Socket timeout, trying again!

CONNECT:355ad048d3e8407ba40aa2c0b425f404,7,LAPTOP-B44NLJ02
Socket timeout, trying again!

CONNECT:9977d36b5f824e34a2f4f79d2030f72f,8,LAPTOP-B44NLJ02
```

# Mongo db for authentication

```
Select C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe
The server generated these startup warnings when booting:
2021-12-08T16:11:47.953+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted

Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

> show dbs
IOT      0.000GB
admin    0.000GB
config   0.000GB
local    0.000GB
> use IOT
switched to db IOT
> show collections
login_info
> db.login_info.find({})
{ "_id" : ObjectId("61702cb92ec0619da4b57b93"), "username" : "anand", "password" : "2ce4d0339075d6acbf5ae19436876cc88813e1519f6d462e4cb3984c6062a2d0" }
{ "_id" : ObjectId("6170f88638e67c27614060f5"), "username" : "rakesh", "password" : "17468f83e0989bbe296b71014ac436fb37a6261511e69179d30799c5f68b275f" }
{ "_id" : ObjectId("618a6ff5b54c7130ad58b9d5"), "username" : "198CE0264", "password" : "5f4df959a11580fc14aa6b139adb2ab40a2cfd5399c1cb6f7c9968eae5a825f" }
{ "_id" : ObjectId("618a721cd77f0906fb3756b"), "username" : "divyesh", "password" : "dc478aac2bca18c52ade1750afef3c704e396a5ce91dd8ef5e57b7f9f645637" }
{ "_id" : ObjectId("618baf66bcf8c1e99ea6a935"), "username" : "sai ram", "password" : "e9c5162436ad1a16360979104c442393933da0fec5af3b3962bbea4dba3288bb" }
{ "_id" : ObjectId("618f466f361388965d335fff"), "username" : "anand123", "password" : "2ce4d0339075d6acbf5ae19436876cc88813e1519f6d462e4cb3984c6062a2d0" }
{ "_id" : ObjectId("618f59bda69ed7c5ce6850a2"), "username" : "anand456", "password" : "5f4df959a11580fc14aa6b139adb2ab40a2cfd5399c1cb6f7c9968eae5a825f" }
{ "_id" : ObjectId("61a065c9d6d15e5d4974170"), "username" : "hemanth", "password" : "4a911b18f788b6380734813017fb360ad81da31e3c64375a43d4d9b94a6b5ecf" }
{ "_id" : ObjectId("61a068328896ce89c5dc34d8"), "username" : "hahah", "password" : "20520f2c11a4561f9e1b329a45fa5a9177fcd5f9d8e43375e0cd6a2d73efab55" }
{ "_id" : ObjectId("61a07826800a123c7ccc6da4"), "username" : "teja", "password" : "6ef002941fe276a1c11b45657e53c73231901f86deb7d3a4f406f5389bb3f810" }
{ "_id" : ObjectId("61a079d390da5832b85474ea"), "username" : "sri", "password" : "d5e96656c6f455d2b0d8da4930a4c7440cd86b4e6d2915de179d1f92f49316a9" }
{ "_id" : ObjectId("61a07bd2bb41a63cf78dcd4"), "username" : "19bct", "password" : "1e73cf7d8b5a152205b0c54e95e3646cdd7f598f2e0e1bbbfdd54fe50cf067926" }
{ "_id" : ObjectId("61a07eb1be46f9499fc88b10"), "username" : "789", "password" : "35a9e381b1a27567549b5f8a6f783c167ebf809f1c4d6a9e367240484d8ce281" }
{ "_id" : ObjectId("61a07f6473d20489cd7e7d3d"), "username" : "15", "password" : "e629fa6598d732768f7c726b4b621285f9c3b85303900aa912017db7617d8bdb" }
{ "_id" : ObjectId("61a08a65070b905086366a64"), "username" : "1500", "password" : "9f69998560dcfd8016442e0a32e059191df095817a164ce844c64ec5a80cc1b" }
{ "_id" : ObjectId("61a09263785fe8ef4b03b92"), "username" : "7897", "password" : "2f80a4ecf7635729bedf552c655cc292e7f3474a9c8d3d5fc457ef63fee198ef" }
{ "_id" : ObjectId("61a10f15d908f40f79580769"), "username" : "hitendu", "password" : "d355676728749330ee01295f25afd20b2a5732c70c92b3d557ef8d995495fd45" }
{ "_id" : ObjectId("61a31ac4cd131dc52dd2b9dc"), "username" : "bhargav", "password" : "9fe5e0a43712f05785002103cfac2add80771a19d76f2c562633f65375ea5581" }
{ "_id" : ObjectId("61a31b78887efe399396b53f"), "username" : "12345", "password" : "da70dfa4d9f95ac979f921e8e623358236313f334afcd06cdd8a5621cf6a1e9" }
{ "_id" : ObjectId("61a31c466fa57b612f48cbcd"), "username" : "098", "password" : "35e1d1aedd3f7179b02a0dfde8f4e826e191649ee2acfd6da6b2ce7a12aa0f8b" }
```

## 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 a world 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.