



دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

عنوان:

گزارش پروژه‌ی پایانی امنیت داده و شبکه

اعضای گروه

علی مهدوی فر - ۹۸۱۰۶۰۷۲

ایمان علیپور - ۹۸۱۰۲۰۲۴

امیرحسین باقری - ۹۸۱۰۵۶۲۱

نام درس

امنیت داده و شبکه

Course Name

Data and Network Security

نیم‌سال دوم ۱۴۰۱-۱۴۰۲

نام استاد درس

دکتر مرتضی امینی

فهرست مطالب

۱	نیازمندی‌های سمت کلاینت
۴	۱-۱ ارتباط ابتدایی کلاینت و سرور
۶	۲-۱ ایجاد حساب کاربری کلاینت
۹	۳-۱ لاگین
۱۱	۴-۱ نمایش کاربران آنلайн
۱۲	۵-۱ ارسال و دریافت پیام
۱۳	۱-۵-۱ ایجاد کلید مادر
۱۶	۲-۵-۱ ارسال پیام و بهره‌گیری از ایده‌ی دابل رچت
۲۰	۶-۱ نگهداری پیام‌ها به صورت امن
۲۵	۷-۱ نگهداری کلیدها به صورت امن
۲۵	۸-۱ ایجاد و مدیریت گروه
۲۶	۱-۸-۱ ارسال پیام در گروه
۲۸	۲-۸-۱ اضافه کردن عضو به گروه
۳۱	۳-۸-۱ حذف فرد از گروه
۳۳	۹-۱ تایید صحبت نشست از طریق کanal امن
۳۵	۱۰-۱ تازه‌سازی کلیدهای نشست
۳۷	۲ نیازمندی‌های سرور
۳۷	۱-۲ ثبت‌نام و احراز هویت کاربران و نگهداری اطلاعات مربوطه به صورت امن
۳۷	۲-۲ ارسال پیام‌ها به مقصد
۳۹	۳-۲ ارائه‌ی لیست کاربران آنلайн
۴۰	۴-۲ ارسال و دریافت پیام‌های مربوط به ساخت کلید نشست
۴۰	۵-۲ نگهداری اطلاعات مربوط به گروه‌ها

۳ نیازمندی‌های امنیتی

۴۴	۱-۳ رمزنگاری انتها به انتها
۴۵	۲-۳ تازگی کلید
۴۵	۳-۳ محربانگی
۴۵	۴-۳ صحت و یکپارچگی
۴۵	۵-۳ حفظ سازگاری
۴۶	۶-۳ احراز اصالت
۴۶	۷-۳ عدم انکار
۴۶	۸-۳ کنترل دسترسی
۴۶	۹-۳ حمله‌ی مرد میانی
۴۶	۱۰-۳ حمله‌ی تکرار
۴۷	۱۱-۳ استفاده از الگوریتم‌های رمزنگاری امن
۴۷	۱۲-۳ محربانگی پیشرو
۴۷	۱۳-۳ محربانگی آینده
۴۷	۴ نکات قابل توجه
۴۷	۵ نحوه‌ی راهاندازی پروژه
	۵۰ مراجع

چکیده

در این پروژه قصد داریم تا یک پیامرسان امن ایجاد کنیم. این پیامرسان امن ارتباط بین کلاینت‌ها را به صورت انتها به انتها رمز خواهد کرد. همچنین دارای محترمانگی پیشرو و محترمانی آینده خواهد بود. همچنین تمامی پیام‌ها امضا و sequence number خواهند داشت تا انواع حملات ممکن بر پروتکل شکست بخورند. روش تبادل کلید پیاده‌سازی شده‌ی ما امکان ایجاد کلید بین یک فرد با یک فرد آفلاین را نیز می‌دهد.

این پیامرسان همچنین امکان ایجاد و مدیریت گروه را به سازنده‌ی گروه می‌دهد. تمامی اعضای گروه امکان ارسال پیام در گروه را دارند و اگر کسی از گروه حذف بشود امکان دریافت و یا رمزگشایی پیام‌های گروه را نخواهد داشت.

واژه‌های کلیدی: Client, Server, End-to-End encryption

۱ نیازمندی‌های سمت کلاینت

پروتکل پیاده‌سازی شده توسط ما در بسیاری از بخش‌ها از پروتکل سیگنال الهام گرفته شده است.
[۱]

در این بخش از ابتدای شروع پروتکل، اقدام به توضیح مراحل پیاده‌سازی آن می‌کنیم.

۱-۱ ارتباط ابتدایی کلاینت و سرور

برای برقراری ارتباط امن بین کلاینت و سرور، ابتدا فرض کردہ‌ایم که کلاینت دارای کلید عمومی سرور می‌باشد. هنگام ارتباط ابتدایی رپرسوکت نوشته‌این که باعث می‌شود کلاینت و سرور با هم هنگام اتصال یک کلید نامتقارن ست کنند. برای ست کردن این کلید، کلاینت ابتدا با کلید عمومی سرور آن را احراز اصالت می‌کند و سپس پارامترهایی تبادل می‌کنند تا یک کلید متقاضان برای جلسه بینشان ست بشود. حال از این پس با استفاده از این کلید متقاضان با هم ارتباط برقرار خواهند کرد. همچنین هر پیام را با دخالت دادن یک شماره‌ی زیادشونده و همچنین یک برچسب زمانی، رمز می‌کنند و در طرف دیگر تازگی پیام‌ها و صحت شماره بررسی می‌شود و در صورت عدم صحبت بسته رد می‌شود.

خروجی این مرحله یک سوکت امن است و برای همین اسم آن را Secure Socket گذاشتیم. پس از این مرحله هیچگونه امکان حمله‌ی تکرار و ارسال پیام از طرف جازدن خود به جای فرد دیگر وجود نخواهد داشت.

```
1 import random
2 import socket
3 import datetime
4
5 import AES
6 import RSA
7 import Resources
8
9
10 class SecureSocket(socket.socket):
11     seq: int = 0
12     aes_key: str = ""
13     raw_socket: socket.socket
```

```

14
15     def send(self, __data: bytes, __flags: int = ...) -> int:
16
16         if self.aes_key == "":
17
17             return self.raw_socket.send(__data)
18
18         else:
19
20             data = (str(self.seq) + Resources.SEP + str(datetime.datetime.now()) +
21
21                 Resources.SEP).encode('ASCII') \
22
22                 + __data
23
23                 self.seq += 1
24
24                 return self.raw_socket.send(AES.encrypt_bytes(data, self.aes_key))
25
25
26         def recv(self, __bufsize: int, __flags: int = ...) -> bytes:
27
27             if self.aes_key == "":
28
28                 return self.raw_socket.recv(__bufsize)
29
29             else:
30
30                 raw_data = self.raw_socket.recv(__bufsize)
31
31
32
32                 if len(raw_data) == 0:
33
33                     return raw_data
34
34
35                 arr = AES.decrypt_bytes(raw_data, self.aes_key).split(Resources.SEP.encode
36
36                 ("ASCII"), maxsplit=3 - 1)
37
37                 seq = int(arr[0])
38
38                 timestamp = arr[1].decode("ASCII")
39
39                 try:
40
40                     assert seq == self.seq
41
41                     Resources.verify_timestamp(timestamp)
42
42                     self.seq += 1
43
43                     return arr[2]
44
44             except (AssertionError, Resources.NotFreshException):
45
45                 print("SecureSocket: Invalid packet received.")
46
46                 print(seq, self.seq)
47
47                 return self.recv(__bufsize)
48
48
49         def establish_client(self, public_key):
50
50             raw_key = str(random.randint(0, 10**6))

```

```

48     key = AES.generate_symmetric_key(raw_key)
49
50     self.send(RSA.encrypt(key, public_key))
51
52     self.aes_key = key
53
54
55
56
57 def establish_server(self, private_key):
58
59     encrypted_key = self.recv(Resources.BUFFER_SIZE)
60
61     self.aes_key = RSA.decrypt(encrypted_key, private_key)
62
63
64
65
66 def wrap_socket(raw_socket) -> SecureSocket:
67
68     secure_socket = SecureSocket()
69
70     secure_socket.raw_socket = raw_socket
71
72     return secure_socket

```

۲-۱ ایجاد حساب کاربری کلاینت

در این بخش و برای ثبت‌نام، ابتدا بررسی می‌کنیم تا یوزر از پیش روی دستگاه وجود نداشته باشد.

```

166 def register_new_user(username, password):
167
168     global client_user
169
170     if os.path.isdir(f"./user/{username}"):
171
172         print("User already exists with this username.")
173
174         return False
175
176     message_type = "register"
177
178     return initialize_user(message_type, username, password)

```

پس از این اقدام به ایجاد کلیدهای یوزر می‌کنیم. برای یک یوزر سه جفت کلید تولید خواهیم کرد. یک جفت کلید RSA که برای امضای پیام‌ها استفاده خواهد شد. دو جفت کلید الجمل که بخش اصلی شروع تبادل کلید پروتکل پیاده‌سازی شده‌ی ما خواهند بود. با ساخت آن‌ها، پیامی برای سرور ارسال می‌کنیم تا ثبت‌نام را انجام بدھیم، در این پیام تمامی کلیدهای عمومی و هش یوزرنیم و پسورد را برای سرور ارسال می‌کنیم. علت ارسال هش یوزرنیم و پسورد این است که از آنجا که یوزرنیم یونیک است، این قبارت هم یونیک خواهد بود و به مهاجم در صورت دسترسی به اطلاعات سرور، دیتایی در مورد شباهت پسورد افراد نمی‌دهد. اگر موفقیت آمیز بود، اقدام به ایجاد یک آبجکت یوزر می‌کنیم.

```

175 def initialize_user(message_type: str, username: str, password: str, old_password_hash
176     = "") -> bool:
177
178     global client_user
179
180     rsa_pr, rsa_pk = RSA.gen_key(username, password)
181     elgamal_pr, elgamal_pk = ElGamal.gen_key(username, password)
182     prekey_pr, prekey_pk = ElGamal.gen_key(username, password, "prekey")
183
184     password_hash = Resources.get_hash(username + password)
185
186     second_field = username if message_type == "register" else old_password_hash
187
188     message = f"{message_type}{Resources.SEP}" \
189             f"{second_field}{Resources.SEP}" \
190             f"{password_hash}{Resources.SEP}" \
191             f"{rsa_pk}{Resources.SEP}" \
192             f"{elgamal_pk}{Resources.SEP}" \
193             f"{prekey_pk}{Resources.SEP}"
194
195     send_to_server(message, sign=(message_type != "register"))
196
197     response = receive_from_server().split(Resources.SEP)
198
199     print(response[2])
200
201     if response[0] == "200":
202
203         client_user = create_user(username, password)
204
205         return True
206
207     return False

```

توابع ایجاد کننده کلید، کلیدهای ایجاد شده را با استفاده از هش یوزرنیم و پسورد به صورت رمزشده روی دستگاه ذخیره می‌کنند. این تابع که مرتبط با این عملیات است را می‌توانید در زیر مشاهده کنید.

این موضوع به ما اجازه می‌دهد تا اگر دستگاه یوزر هک شود، کلیدهای خصوصی به صورت رمزشده ذخیره شده باشند و امکان بدست آوردن این کلیدها برای حمله کننده وجود نداشته باشد.

```

12 def save_keys(username: str, password: str, method: str, private_key: str, public_key:
13     str):
14
15     if not os.path.isdir("./user"):
16
17         os.mkdir("./user")
18
19
20         if not os.path.isdir(f"./user/{username}"):
21
22             os.mkdir(f"./user/{username}")

```

```

19     with open(f"./user/{username}/{method}_private.key", 'wb') as content_file:
20         os.chmod(f"./user/{username}/{method}_private.key", 0o600)
21         aes_key = AES.generate_symmetric_key(password)
22         private_key_encrypted = AES.encrypt(private_key, aes_key, AES.default_iv)
23         content_file.write(private_key_encrypted.encode("ASCII"))
24
25     with open(f"./user/{username}/{method}_public.key", 'wb') as content_file:
26         content_file.write(public_key.encode("ASCII"))
27
28     return

```

سرور پس از بررسی اطلاعات ارسالی از طرف کاربر، اگر یوزرنیم آن تکراری نباشد اجازه ایجاد حساب کاربری را به کاربر می‌دهد و جوابی مبنی بر موفقیت در عملیات ثبت‌نام برای ان ارسال می‌کند.

```

73 def register_new_user(client_socket: socket.socket, username, password_hash, rsa_pk,
74     elgamal_pk, prekey_pk):
75
76     if username_exists(username):
77
78         response = f"400{Resources.SEP}" \
79             f"Bad Request{Resources.SEP}" \
80             f"Username already exists"
81
82         response_client(client_socket, response)
83
84         return
85
86
87     new_user = User(username, password_hash, rsa_pk, elgamal_pk, prekey_pk)
88     users.append(new_user)
89     messages[new_user.username] = []
90
91
92     response = f"200{Resources.SEP}" \
93         f"OK{Resources.SEP}" \
94         f"User registered successfully"
95
96     response_client(client_socket, response)
97
98     return new_user

```

از این پس تمامی پیام‌هایی که کاربر ارسال می‌کند امضا خواهد داشت تا احراز اصالت و احراز صحیح شوند و همچنین خاصیت عدم انکار را فراهم کنند.

۳-۱ لاجین

در صورتی که کاربر قصد لاجین داشته باشد، ابتدا پیامی به سرور مبنی بر درخواست لاجین ارسال می‌کند. سرور برای آن یک متغیر تصادفی به اسم سالت ارسال می‌کند و یوزر هش یوزرنیم، پسورد را با سالت ترکیب می‌کند و مجدداً هش می‌گیرد. اینگونه در هر بار لاجین، عبارات متفاوتی بین کلاینت و سرور جابجا می‌شوند.

```
206 def login_user(username, password):
207     global client_user
208     if not os.path.isdir(f"./user/{username}"):
209         print("You don't have the keys for this username")
210         return False
211     try:
212         client_user = create_user(username, password)
213     except Resources.InvalidKeysException:
214         print("Keys are not valid")
215         return False
216     except Resources.WrongPasswordException:
217         print("Wrong password or keys are manipulated")
218         return False
219     load_db(username, Resources.get_hash(username + password))
220     message = f"login{Resources.SEP}" \
221             f"{username}"
222     send_to_server(message, False)
223     response = receive_from_server().split(Resources.SEP)
224
225     if response[0] == "200":
226         salt = response[2]
227         password_hash = Resources.get_hash(username + password)
228         otp = Resources.get_hash(salt + password_hash)
229         message = f"{otp}"
230         send_to_server(message, False)
231         response = receive_from_server().split(Resources.SEP)
232         print(response[2])
233         return response[0] == "200"
234     else:
```

```
235     return False
```

سرور پس از دریافت اطلاعات صحیح و احراز هویت، اصالت و صحت وی به او پیامی بر موفقیت ارسال می‌کند و وضعیت وی را به صورت آنلاین سنت می‌کند.

```
112 def login_user(client_socket: socket.socket, username):
113     for user in users:
114         if username == user.username:
115             salt = random.randint(0, 10 ** 50)
116             response = f"200{Resources.SEP}" \
117                         f"OK{Resources.SEP}" \
118                         f"{salt}"
119             response_client(client_socket, response)
120             otp = receive_from_client(client_socket, None)
121
122             if user.check_password(str(salt), otp):
123                 response = f"200{Resources.SEP}" \
124                         f"OK{Resources.SEP}" \
125                         f"User {user.username} logged in " \
126                         f"successfully"
127                 response_client(client_socket, response)
128                 user.set_online()
129
130             return user
131
132     else:
133         response = f"400{Resources.SEP}" \
134                         f"Bad Request{Resources.SEP}" \
135                         f"Wrong password"
136         response_client(client_socket, response)
137
138     response = f"400{Resources.SEP}" \
139                         f"Bad Request{Resources.SEP}" \
140                         f"User does not exist"
141     response_client(client_socket, response)
142
143     return None
```

۴-۱ نمایش کاربران آنلاین

برای این کار کلاینت پیامی مبنی بر دریافت لیست کاربران به سرور ارسال می‌کند.

```
238 def retrieve_usernames_from_server():
239     message = f"show users list"
240     send_to_server(message, True)
241     response = receive_from_server().split(Resources.SEP)
242     return response[2]
```

سرور هم در پاسخ لیستی از کاربران و وضعیت آنلاین بودن یا نبودن آنها را برای کاربر باز می‌گرداند. شکل ۱

```
154 def show_users_list(client_socket: socket.socket):
155     response = f"200{Resources.SEP}" \
156                 f"OK{Resources.SEP}"
157
158     for user in users:
159         response += f"- {user.username} ({'Online' if user.is_online else 'Offline'})\n"
160
161     response_client(client_socket, response[:-1])
162     return
```

```
Welcome reza.

1: show users list
2: open chat <username>
3: create group <group_name>
4: open group <group_name>
5: renew keys <old password> <new password>
6: logout
> show users list

- iman (Offline)
- ali (Online)
- reza (Online)
Press Enter to continue...■
```

شکل ۱: نمایش لیست کاربران

۵-۱ ارسال و دریافت پیام

در اینجا برای ارسال پیام به یک کاربر دیگر، ابتدا از سرور کلیدهای او را دریافت می‌کنیم.

```
327 def retrieve_keys(username: str):
328     global users
329
330     message = f"retrieve keys{Resources.SEP}" \
331             f"{username}"
332     send_to_server(message, sign=True)
333     response = receive_from_server().split(Resources.SEP, maxsplit=3 - 1)
334
335     if response[0] == "200":
336         rsa_pk, elgamal_pk, prekey_pk = response[2].split(Resources.SEP)
337         users = [user for user in users if user.username != username]
338         user = User(username, "", rsa_pk, int(elgamal_pk), int(prekey_pk))
339         users.append(user)
340
341     return True
```

```

341     else:
342         print(response[2])
343         return False

```

سرور در صورت وجود کاربر، به ما کلیدهای عمومی او را برمی‌گرداند.

```

165 def retrieve_keys(client_socket: socket.socket, username: str):
166     for user in users:
167         if user.username == username:
168             response = f"200{Resources.SEP}" \
169                         f"OK{Resources.SEP}"
170             response += user.rsa_pk + Resources.SEP + str(user.elgamal_pk) + Resources
171                         .SEP + str(user.prekey_pk)
172             response_client(client_socket, response)
173             return
174
175     response = f"404{Resources.SEP}" \
176                         f"Not Found{Resources.SEP}" \
177                         f"User does not exist."
178     response_client(client_socket, response)
179     return

```

با دریافت این کلیدها، پروتکل ایجاد کلید ما شروع می‌شود.

۱-۵-۱ ایجاد کلید مادر

ابتدا با الهام از روش X3DH که در سیگنال وجود دارد، با استفاده از کلیدهای دریافتی از سرور، اقدام به ایجاد کلید مادر می‌کنیم. برای این کار سه کلید دیفری هلمن با استفاده از پیامهای دریافتی و جفت کلیدهای خصوصی خودمان و همچنین یک جفت کلید موقتی الجمل دیگر که در لحظه تولید کرده‌ایم را ایجاد می‌کنیم. کلید مادر از اتصال این سه کلید ایجاد می‌شود که ورودی رچت ماست. در اینجا پارامترهای مورد نیاز و کلیدهای عمومی خودمان را نیز برای طرف مقابل ارسال می‌کنیم.

```

265 def x3dh_key_exchange(target_user: User, seq=0) -> bool:
266     print("sending key...")
267
268     ek_pr, ek_pk = ElGamal.gen_key()
269

```

```

270     DH1 = ElGamal.DH_key(target_user.prekey_pk, client_user.elgamal_pr)
271     DH2 = ElGamal.DH_key(target_user.elgamal_pk, ek_pr)
272     DH3 = ElGamal.DH_key(target_user.prekey_pk, ek_pr)
273
274     SK = AES.generate_symmetric_key(str(DH1) + str(DH2) + str(DH3))
275
276     chat = chats[target_user.username]
277     chat.root_key = SK
278     chat.DH_key = ElGamal.DH_key(target_user.prekey_pk, client_user.prekey_pr)
279     chat.our_pr = client_user.prekey_pr
280     chat.their_pk = target_user.prekey_pk
281     chat.their_rsa_pk = target_user.rsa_pk
282
283     new_root_key, message_key = chat.KDF(chat.DH_key, chat.root_key)
284     chat.root_key = new_root_key
285     chat.message_key = message_key
286
287     initial_message = str(client_user.elgamal_pk) \
288                         + Resources.SEP + str(target_user.prekey_pk) \
289                         + Resources.SEP + str(ek_pk) \
290                         + Resources.SEP + client_user.rsa_pk \
291                         + Resources.SEP + str(client_user.prekey_pk)
292
293     print(initial_message)
294
295     message_obj = Message(message_type="x3dh",
296                           source_username=client_user.username,
297                           target_username=target_user.username,
298                           seq=seq,
299                           signature=RSA.sign(str(seq) + initial_message, RSA.
300                           pem_to_private_key(client_user.rsa_pr)),
301                           text=initial_message)
302
303     chats[target_user.username].append_message(message_obj)
304
305     request = str(message_obj)

```

```

305     send_to_server(request, sign=True)
306
307     response = receive_from_server().split(Resources.SEP, maxsplit=3 - 1)
308
309     return response[0] == "200"

```

از اینجا به بعد سرور عموماً پیام‌ها را صرفاً در صف ارسال به کاربر هدف قرار می‌دهد و نقش دیگری ندارد، از این رو تا زمانی که سرور کار خاصی انجام نمی‌دهد کدهای آن را اینجا توضیح نمی‌دهیم.

حال این کلید مادر را به عنوان ورودی به تابع Key derivation function می‌دهیم، مقدار ثابت را نیز برابر با کلید دیفری هلمن حاصل از کلیدهای elgamal_prekey خواهد بود، خروجی این تابع، یک کلید پیام است، حال مجدداً این کلید پیام را به تابع Key derivation function می‌دهیم تا به ما یک کلید پیام جدید و یک کلید نهایی بدهد، کلید پیام برای هر ارسال پیام تا زمانی که پیامی از سمت مقابل دریافت نکرده‌ایم، هر بار یک کلید نهایی و یک کلید پیام جدید تولید می‌کنیم و پیاممان را رمز می‌کنیم. این کار به ما محترمانگی پیشرو را خواهد داد.

در طرف دریافت، گیرنده با دریافت پیام مبنی بر X3DH handshake می‌تواند رچت را برای خودش بازسازی کند و پیام‌های ارسال شده توسط فرد مقابل تا بدان لحظه را رمزگشایی کند.

```

310 def x3dh_extract_key(text: str):
311     print("receiving key...")
312
313     A_elgamal_pk, our_prekey_pk, A_ek_pk, their_rsa_pk, their_prekey_pk = text.split(
314         Resources.SEP)
315
316     A_elgamal_pk = int(A_elgamal_pk)
317     our_prekey_pk = int(our_prekey_pk)
318     A_ek_pk = int(A_ek_pk)
319     their_prekey_pk = int(their_prekey_pk)
320
321     DH1 = ElGamal.DH_key(A_elgamal_pk, client_user.prekey_pr)
322     DH2 = ElGamal.DH_key(A_ek_pk, client_user.elgamal_pr)
323     DH3 = ElGamal.DH_key(A_ek_pk, client_user.prekey_pr)
324
325     SK = AES.generate_symmetric_key(str(DH1) + str(DH2) + str(DH3))
326
327     return SK, their_prekey_pk, their_rsa_pk

```

۲-۵-۱ ارسال پیام و بهره‌گیری از ایده‌ی دابل رچت

هنگام ارسال یک پیام، اگر فرستنده‌ی پیام پیشین ما نباشیم، یک جفت کلید دیفری هلمن تولید می‌کنیم و هنگام ارسال پیام کلید عمومی آن را نیز ارسال می‌کنیم. همچنین کلید مادر را به همراه کلید دیفری هلمن ایجاد شده توسط کلید خصوصی جدید و کلید عمومی فعلی فرد مخاطب به عنوان مقدار ثابت، وارد KDF می‌کنیم تا کلید مادر جدید و کلید پیام جدیدی ایجاد کنیم. این روش همان روش دابل رچت پیاده‌سازی شده در پیام‌رسان سیگنال است.

```
346 def send_message(chat: Chat, message_type, text, target_group=""):  
347     print(f"Sending \'{text}\' to {chat.username}...")  
348     fetch_messages()  
349  
350     if chat.messages[-1].source_username != client_user.username:  
351         private_key, public_key = ElGamal.gen_key()  
352  
353         if send_message_to_server(chat, "dr_pk", str(public_key)):  
354             chat.our_pr = private_key  
355             chat.DH_key = ElGamal.DH_key(chat.their_pk, private_key)  
356             chat.root_key, chat.message_key = chat.KDF(chat.DH_key, chat.root_key)  
357  
358     return send_message_to_server(chat, message_type, text, target_group)
```

طرف مقابل با دریافت پیام، عمل مشابهی را انجام می‌دهد و کلید عمومی ما را استخراج، و مقدار ثابت جدید را محاسبه خواهد کرد. با این کار رچت طرفین همواره سینک خواهد ماند و هر پیام با کلیدی جدید روز خواهد شد که ارتباطی با کلیدهای پیشین و آینده ندارد. اینگونه محرمانگی آینده و پیشرو را تضمین خواهیم کرد. هنگام دریافت، امضای پیام نیز چک خواهد شد و همچنین بررسی می‌کنیم که شماره‌ی Sequence number هم درست باشد، اینگونه پیام احراز اصالت و احراز صحت می‌شود، همچنین محرمانگی آن حفظ می‌شود و امکان انکار آن توسط ارسال کننده وجود نخواهد داشت.

```
414 def receive_message(chat: Chat, message_obj: Message):  
415     new_message_key, the_ultimate_key = chat.KDF(chat.DH_key, chat.message_key)  
416     chat.message_key = new_message_key  
417  
418     # decrypt the message text  
419     message_obj.text = AES.decrypt(message_obj.text, the_ultimate_key)
```

```

420
421     # check the message sign
422
423     user = get_user_by_chat(chat)
424
425     try:
426
427         RSA.verify_signature(str(message_obj.seq) + message_obj.text, message_obj.
428             signature,
429
430             RSA.pem_to_public_key(chat.their_rsa_pk))
431
432         message_obj.source_rsa_pk = chat.their_rsa_pk
433
434     except InvalidSignature:
435
436         return
437
438
439     # update the keys for "dr_pk" control message
440
441     if chat.messages[-1].source_username != message_obj.source_username:
442
443         chat.their_pk = int(message_obj.text)
444
445         chat.DH_key = ElGamal.DH_key(chat.their_pk, chat.our_pr)
446
447         chat.root_key, chat.message_key = chat.KDF(chat.DH_key, chat.root_key)
448
449
450         chat.append_message(message_obj)
451
452     return message_obj

```

تابعی نیز وجود دارد که عملیات دریافت پیامها و پرسس کردن آنها را انجام می‌دهد. این تابع پس از دریافت تمامی پیامها از سرور، بسته به نوع پیام، امضا و شماره‌ی آنها را چک می‌کند و آنها را هندل می‌کند. مثلاً اگر پیام عادی باشد آن را رمزگشایی می‌کند، اگر پیام تبادل X3DH باشد کلیدها را ایجاد می‌کند.

```

441 def fetch_messages():
442
443     request = f"fetch"
444
445     send_to_server(request, sign=True)
446
447     response = receive_from_server().split(Resources.SEP, maxsplit=3 - 1)
448
449
450     # send ACK to server to delete fetched messages
451
452     send_to_server("ack", sign=True)
453
454
455     new_messages = json.loads(response[2])
456
457     new_messages_lists = [message.split(Resources.SEP) for message in new_messages]
458
459     new_messages_lists.sort(key=lambda x: x[3])

```

```

452
453     for message in new_messages:
454
454         message_type, source_username, target_username, target_group, seq, signature,
455         text = message. \
456
456             split(Resources.SEP, maxsplit=7 - 1)
457
457             message_obj = Message(message_type=message_type,
458
458                     source_username=source_username,
459
459                     target_username=target_username,
460
460                     target_group=target_group,
461
461                     seq=seq,
462
462                     signature=signature,
463
463                     text=text)
464
464
465             if message_type not in ["add", "join", "remove"]:
466
466                 retrieve_keys(source_username)
467
467
468                 if message_type == "x3dh":
469
469                     SK, their_prekey_pk, their_rsa_pk = x3dh_extract_key(text)
470
470
471                     if source_username not in chats:
472
472                         chats[source_username] = Chat(source_username)
473
473
474                     chat = chats[source_username]
475
475                     chat.append_message(message_obj)
476
476
477                     chat.root_key = SK
478
478                     chat.DH_key = ElGamal.DH_key(their_prekey_pk, client_user.prekey_pr)
479
479                     chat.our_pr = client_user.prekey_pr
480
480                     chat.their_pk = their_prekey_pk
481
481                     chat.their_rsa_pk = their_rsa_pk
482
482
483                     message_obj.source_rsa_pk = chat.their_rsa_pk
484
484
485                     new_root_key, message_key = chat.KDF(chat.DH_key, chat.root_key)
486                     chat.root_key = new_root_key
486                     chat.message_key = message_key

```

```

487
488     elif message_type == "dr_pk":
489         chat = chats[source_username]
490         receive_message(chat, message_obj)
491
492     elif message_type == "text":
493         chat = chats[source_username]
494         receive_message(chat, message_obj)
495
496     elif message_type == "group_text":
497         chat = chats[source_username]
498         message_obj = receive_message(chat, message_obj)
499         group = groups[message_obj.target_group]
500         group.chat.append_message(message_obj)
501
502     elif message_type == "add":
503         try:
504             RSA.verify_signature(str(seq) + text, message_obj.signature,
505             server_public_key)
506             except InvalidSignature:
507                 continue
508             new_member_username, group_name = text.split(Resources.SEP)
509             groups[group_name].usernames.append(new_member_username)
510             groups[group_name].chat.append_message(message_obj)
511
512     elif message_type == "join":
513         try:
514             RSA.verify_signature(str(seq) + text, message_obj.signature,
515             server_public_key)
516             except InvalidSignature:
517                 continue
518             admin_username, group_name, dumped_usernames = text.split(Resources.SEP)
519             if group_name not in groups:
520                 group = Group(admin_username, group_name)
521                 groups[group.group_name] = group
522                 groups[group_name].usernames = json.loads(dumped_usernames)

```

```

521     groups[group_name].chat.append_message(message_obj)

522

523     elif message_type == "remove":
524
525         try:
526
527             RSA.verify_signature(str(seq) + text, message_obj.signature,
528             server_public_key)
529
530             except InvalidSignature:
531
532                 continue
533
534             removed_member_username, group_name = text.split(Resources.SEP)
535
536             groups[group_name].usernames.remove(removed_member_username)
537
538             groups[group_name].chat.append_message(message_obj)

```

۶-۱ نگهداری پیام‌ها به صورت امن

برای این کار با استفاده از کدهای زیر، هنگام خروج فرد تمامی اطلاعات وی با استفاده از کلید مشتق از یوزرنیم و پسورد او رمز می‌شوند و درون یک فایل ذخیره خواهند شد.

```

35 def dump_chat(chat):
36
37     dumped_chat = {"root_key": chat.root_key,
38
39         "message_key": chat.message_key,
40
41         "DH_key": chat.DH_key,
42
43         "our_pr": chat.our_pr,
44
45         "their_pk": chat.their_pk,
46
47         "their_rsa_pk": chat.their_rsa_pk,
48
49         "seq": chat.seq,
50
51         "username": chat.username,
52
53         "messages": []}
54
55     for message in chat.messages:
56
57         dumped_chat["messages"].append({"source_rsa_pk": message.source_rsa_pk,
58
59             "message_type": message.message_type,
60
61             "source_username": message.source_username,
62
63             "target_username": message.target_username,
64
65             "seq": message.seq, "signature": message.
66
67             signature,
68
69             "text": message.text,
70
71             "target_group": message.target_group}

```

```

53         })
54
55
56
57     def load_chat(dumped_chat):
58
58         username = dumped_chat["username"]
59
60         chat = Chat(username)
61
62         chat.root_key = dumped_chat["root_key"]
63
64         chat.message_key = dumped_chat["message_key"]
65
66         chat.DH_key = dumped_chat["DH_key"]
67
68         chat.our_pr = dumped_chat["our_pr"]
69
70         chat.their_pk = dumped_chat["their_pk"]
71
72         chat.their_rsa_pk = dumped_chat["their_rsa_pk"]
73
74         chat.seq = dumped_chat["seq"]
75
76         chat.messages = []
77
78         for message in dumped_chat["messages"]:
79
80             new_message = Message(message["message_type"], message["source_username"],
81
82                         message["target_username"],
83
84                         message["seq"], message["signature"], message["text"],
85
86                         message["target_group"])
87
88             new_message.source_rsa_pk = message["source_rsa_pk"]
89
90             chat.messages.append(new_message)
91
92
93     return chat
94
95
96
97     def save_to_db():
98
99         db = {"users": [], "chats": [], "groups": []}
100
101        for user in users:
102
103            new_user = {"rsa_pk": user.rsa_pk, "elgamal_pk": user.elgamal_pk, "prekey_pk":
104
105                user.prekey_pk,
106
107                "username": user.username}
108
109            db["users"].append(new_user)
110
111
112        for username in chats:
113
114            chat = chats[username]
115
116            dumped_chat = dump_chat(chat)

```

```

86     db["chats"].append(dumped_chat)
87
88     for group_name in groups:
89         group = groups[group_name]
90         dumped_group = {"admin_username": group.admin_username,
91                         "group_name": group_name,
92                         "usernames": group.usernames}
93
94         new_chat = dump_chat(group.chat)
95         dumped_group["chat"] = new_chat
96
97         db["groups"].append(dumped_group)
98
99
100    with open(f"./user/{client_user.username}/{client_user.username}_db.json", 'wb')
101       as db_file:
102
103        content = json.dumps(db)
104
105        aes_key = AES.generate_symmetric_key(client_user.password_hash)
106
107        db_encrypted = AES.encrypt(content, aes_key, AES.default_iv)
108
109        db_file.write(db_encrypted.encode("ASCII"))
110
111    def dump_chat(chat):
112
113        dumped_chat = {"root_key": chat.root_key,
114
115                        "message_key": chat.message_key,
116
117                        "DH_key": chat.DH_key,
118
119                        "our_pr": chat.our_pr,
120
121                        "their_pk": chat.their_pk,
122
123                        "their_rsa_pk": chat.their_rsa_pk,
124
125                        "seq": chat.seq,
126
127                        "username": chat.username,
128
129                        "messages": []}
130
131        for message in chat.messages:
132
133            dumped_chat["messages"].append({"source_rsa_pk": message.source_rsa_pk,
134
135                                "message_type": message.message_type,
136
137                                "source_username": message.source_username,
138
139                                "target_username": message.target_username,
140
141                                "seq": message.seq, "signature": message.
142
143                                signature,
144
145                                "text": message.text,
146
147                                "target_group": message.target_group}
148
149

```

```

120                               })
121
122     return dumped_chat
123
124 def load_chat(dumped_chat):
125
126     username = dumped_chat["username"]
127
128     chat = Chat(username)
129
130     chat.root_key = dumped_chat["root_key"]
131
132     chat.message_key = dumped_chat["message_key"]
133
134     chat.DH_key = dumped_chat["DH_key"]
135
136     chat.our_pr = dumped_chat["our_pr"]
137
138     chat.their_pk = dumped_chat["their_pk"]
139
140     chat.their_rsa_pk = dumped_chat["their_rsa_pk"]
141
142     chat.seq = dumped_chat["seq"]
143
144     chat.messages = []
145
146     for message in dumped_chat["messages"]:
147
148         new_message = Message(message["message_type"], message["source_username"],
149                               message["target_username"],
150                               message["seq"], message["signature"], message["text"],
151                               message["target_group"])
152
153         new_message.source_rsa_pk = message["source_rsa_pk"]
154
155         chat.messages.append(new_message)
156
157     return chat
158
159
160
161
162
163 def save_to_db():
164
165     db = {"users": [], "chats": [], "groups": []}
166
167     for user in users:
168
169         new_user = {"rsa_pk": user.rsa_pk, "elgamal_pk": user.elgamal_pk, "prekey_pk":
170                     user.prekey_pk,
171                     "username": user.username}
172
173         db["users"].append(new_user)
174
175
176     for username in chats:
177
178         chat = chats[username]
179
180         dumped_chat = dump_chat(chat)

```

```

153     db["chats"].append(dumped_chat)

154

155     for group_name in groups:
156         group = groups[group_name]
157         dumped_group = {"admin_username": group.admin_username,
158                         "group_name": group_name,
159                         "usernames": group.usernames}
160
161         new_chat = dump_chat(group.chat)
162         dumped_group["chat"] = new_chat
163
164         db["groups"].append(dumped_group)

165
166
167     with open(f"./user/{client_user.username}/{client_user.username}_db.imal", 'wb')
168     as db_file:
169
170         content = json.dumps(db)
171
172         aes_key = AES.generate_symmetric_key(client_user.password_hash)
173
174         db_encrypted = AES.encrypt(content, aes_key, AES.default_iv)
175
176         db_file.write(db_encrypted.encode("ASCII"))

```

هنگام ورود نیز با استفاده از کدهای زیر پیامها و کلیدهای فرد رمزگشایی می‌شوند. همچنین تمامی کلیدهای خصوصی که به صورت رمزشده نگهداری شده بودند نیز هنگام ورود رمزگشایی می‌شوند.

```

104 def load_db(username, password_hash):
105
106     global users, chats
107
108
109     with open(f"./user/{username}/{username}_db.imal", 'rb') as db_file:
110
111         db_encrypted = db_file.read().decode("ASCII")
112
113         aes_key = AES.generate_symmetric_key(password_hash)
114
115         content = AES.decrypt(db_encrypted, aes_key, AES.default_iv)
116
117         db = json.loads(content)
118
119
120
121         for dumped_user in db["users"]:
122
123             user = User(dumped_user["username"], dumped_user["rsa_pk"], "",
124                         dumped_user["elgamal_pk"], dumped_user["prekey_pk"])
125
126             users.append(user)
127
128
129         for dumped_chat in db["chats"]:

```

```

119     chat = load_chat(dumped_chat)
120     chats[chat.username] = chat
121
122     for dumped_group in db["groups"]:
123         group = Group(dumped_group["admin_username"], dumped_group["group_name"])
124         group.usernames = dumped_group["usernames"]
125         group.chat = load_chat(dumped_group["chat"])
126         groups[group.group_name] = group

```

۷-۱ نگهداری کلیدها به صورت امن

کدهای مطرح شده در بخش پیشین، این نیازمندی را نیز برطرف می‌کردند. تمامی کلیدها به صورت رمز شده در فایل ذخیره شده و هنگام لاگین ریکاور می‌شوند.

۸-۱ ایجاد و مدیریت گروه

در این بخش ابتدا کلاینت پیامی مبنی بر ایجاد گروه برای سرور ارسال می‌کند.

```

615 def create_group(group_name: str):
616     request = f"create{Resources.SEP}{group_name}"
617     send_to_server(request, sign=True)
618     response = receive_from_server().split(Resources.SEP, maxsplit=3 - 1)
619     if response[0] == "200":
620         group = Group(client_user.username, group_name)
621         groups[group_name] = group
622         print(response[2])
623     return True
624
625     print(response[2])

```

سرور با دریافت این پیام و بررسی امکان ایجاد گروه با یوزرنیم ارسالی، جواب کلاینت را می‌دهد. سپس کلاینت در صورت جواب مثبت سرور، آبجکت آن را ایجاد می‌کند. سرور نیز یک آبجکت گروه ایجاد می‌کند و اسم آن و ادمین آن را مشخص می‌کند. ایجاد کننده را نیز در لیست اعضای گروه قرار می‌دهد.

```

234 def create_group(client_socket: socket.socket, user: User, group_name: str):
235     if username_exists(group_name):
236         response = f"400{Resources.SEP}" \
237             f"Bad Request{Resources.SEP}" \
238             f"Username already exists"
239         response_client(client_socket, response)
240         return
241
242     group = Group(user.username, group_name)
243     groups[group_name] = group
244     response = f"200{Resources.SEP}" \
245             f"OK{Resources.SEP}" \
246             f"Group created successfully"
247     response_client(client_socket, response)

```

۱-۸-۱ ارسال پیام در گروه

هنگام ارسال پیام، کاربر ابتدا پیام‌ها را از سرور دریافت می‌کند تا اعضای فعلی گروه و همچنین پیام‌های آن را بدست آورد. پس از این برای ارسال پیام، هر فرد پیامی را به روش ارسال پیام خصوصی به افراد، البته با تایپ متفاوت، در گروه ارسال می‌کند، یعنی برای تمامی اعضای فعلی گروه پیام را مستقلاً ارسال می‌کند.

```

361 def send_group_message(group: Group, message_type, text):
362     print(f"Sending \"{text}\" to {group.group_name}...")
363     fetch_messages()
364
365     message_obj = Message(message_type="group_text",
366                           source_username=client_user.username,
367                           target_username=client_user.username,
368                           seq=0,
369                           signature=rsa.sign(str(0) + text, RSA.pem_to_private_key(
370                               client_user.rsa_pr)),
371                           text=text,
372                           target_group=group.group_name)

```

```

373     # save our own message to chat of group
374     group.chat.append_message(message_obj)
375
376     # send the message to other members of group
377     for username in group.usernames:
378         if username != client_user.username:
379             open_chat(username)
380             send_message(chats[username], message_type, text, group.group_name)

```

در سمت دریافت، هر فرد پیام‌ها را دریافت می‌کند و اگر تایپ آن از جنس پیام‌های گروه باشد، آن را در لیست پیام‌های گروه قرار می‌دهد. با این روش تمامی نیازمندی‌های محترمانگی پیش رو، محترمانگی اینده، حفظ محترمانگی پیام، احراز اصالت و صحبت و عدم انکار را مشابه‌ای برای پیام‌های ارسالی در گروه نیز داریم.

از سمتی سرور هنگامی که همچین پیامی ارسال می‌شود، ابتدا بررسی می‌کند که این فرد عضو گروه باشد و در این صورت پیام آن را برای افراد ارسال می‌کند، وگرنه به او پیغام خطا نشان می‌دهد. اینگونه کنترل دسترسی در سمت ارسال کننده‌ی پیام را خواهیم داشت.

```

181 def save_message(client_socket: socket.socket, message: str):
182     _type, source_username, target_username, target_group, seq, signature, text =
183     message. \
184         split(Resources.SEP, maxsplit=7 - 1)
185     message_obj = Message(message_type=_type,
186                           source_username=source_username,
187                           target_username=target_username,
188                           target_group=target_group,
189                           seq=seq,
190                           signature=signature,
191                           text=text)
192
193     for user in users:
194         if user.username == target_username:
195             if message_obj.target_group != "":
196                 if message_obj.target_group not in groups:
197                     response = f"404{Resources.SEP}" \

```

```

198                     f"Group does not exist."
199
200             response_client(client_socket, response)
201
202         elif message_obj.source_username not in groups[message_obj.
203             target_group].usernames:
204
205             response = f"403{Resources.SEP}" \
206                 f"Forbidden{Resources.SEP}" \
207                     f"You are not a member of this group."
208
209             response_client(client_socket, response)
210
211         return
212
213
214     messages[user.username].append(message_obj)
215
216
217     response = f"200{Resources.SEP}" \
218                 f"OK{Resources.SEP}" \
219                     f"Message saved."
220
221     response_client(client_socket, response)
222
223     return
224
225
226     response = f"404{Resources.SEP}" \
227                 f"Not Found{Resources.SEP}" \
228                     f"User does not exist."
229
230     response_client(client_socket, response)
231
232     return

```

۱-۸-۲ اضافه کردن عضو به گروه

برای این کار کلاینت یک پیام مبنی بر اضافه کردن فرد به گروه ارسال می‌کند و در صورت تایید آن توسط سرور، فرد را به یوزرهای درون گروه خودش در آبجکت گروه مورد نظر اضافه می‌کند که این در تابع fetch messages اتفاق می‌افتد.

```

653 def add_member_to_group(group: Group, new_member_username: str):
654
655     fetch_messages()
656
657     message = f"add{Resources.SEP}{group.group_name}{Resources.SEP}{"new_member_username}"
658
659     send_to_server(message, True)

```

```

657     response = receive_from_server().split(Resources.SEP)
658
659     print(response[2])

```

در سمت سرور، ابتدا سرور بررسی می‌کند که فرد ادمین گروه باشد، سپس بررسی می‌کند که فرد وجود داشته باشد و اگر همگی این موضوعات درست بودند، پیامی به تمامی اعضای اعم از ادمین گروه مبنی بر اضافه شدن فردی به گروه می‌فرستد تا همگی فرد جدید را به لیست اعضای خودشان در گروه اضافه کنند.

```

250 def add_member_to_group(client_socket: socket.socket, user: User, group_name,
251     new_member_username):
252     # check if group exists
253     if group_name not in groups:
254         response = f"404{Resources.SEP}" \
255                 f"Not Found{Resources.SEP}" \
256                 f"Group does not exist."
257     response_client(client_socket, response)
258
259     return
260
261
262     group = groups[group_name]
263
264     # check if sender of add request is member of the group
265     if user.username not in group.usernames:
266         response = f"403{Resources.SEP}" \
267                 f"Not Forbidden{Resources.SEP}" \
268                 f"You are not a member of this group."
269     response_client(client_socket, response)
270
271     return
272
273
274     # check if sender of add request is admin of the group
275     if user.username != group.admin_username:
276         response = f"403{Resources.SEP}" \
277                 f"Not Forbidden{Resources.SEP}" \
278                 f"You are not the admin of this group."
279     response_client(client_socket, response)
280
281     return

```

```

276
277     # check if requested user exists
278
279     if new_member_username not in [user.username for user in users]:
280
281         response = f"404{Resources.SEP}" \
282             f"Not Found{Resources.SEP}" \
283             f"User does not exist."
284
285         response_client(client_socket, response)
286
287         return
288
289
290     # check if requested user is already in the group
291
292     if new_member_username in group.usernames:
293
294         response = f"400{Resources.SEP}" \
295             f"Not Not Found{Resources.SEP}" \
296             f"User is already a member of this group."
297
298         response_client(client_socket, response)
299
300         return
301
302
303     for old_member_username in group.usernames:
304
305         # if old_member_username != user.username:
306
307         add_text = new_member_username + Resources.SEP + group.group_name
308
309         message_obj = Message(message_type="add",
310
311                         source_username="",
312
313                         target_username=old_member_username,
314
315                         seq=0,
316
317                         signature=RSA.sign(str(0) + add_text, server_private_key
318
319 ),
320
321                         text=add_text)
322
323         messages[old_member_username].append(message_obj)
324
325
326         group.usernames.append(new_member_username)
327
328
329         join_text = group.admin_username + Resources.SEP + group.group_name + Resources.
330
331             SEP + json.dumps(group.usernames)
332
333         message_obj = Message(message_type="join",
334
335                         source_username="",
336
337                         target_username=new_member_username,

```

```

310         seq=0,
311         signature=RSA.sign(str(0) + join_text, server_private_key),
312         text=join_text)
313     messages[new_member_username].append(message_obj)
314
315     response = f"200{Resources.SEP}" \
316             f"OK{Resources.SEP}" \
317             f"User added to group successfully."
318     response_client(client_socket, response)
319     return

```

۳-۸-۱ حذف فرد از گروه

با توجه به نحوه پیاده‌سازی گروه، این کار نیز مشابه عملیات اضافه کردن صورت می‌گیرد. کلاینت گروه پیامی مبنی بر حذف فرد از گروه به سرور ارسال می‌کند.

```

1 def remove_member_from_group(group: Group, member_username: str):
2     fetch_messages()
3
4     message = f"remove{Resources.SEP}{group.group_name}{Resources.SEP}{member_username}"
5
6     send_to_server(message, True)
7     response = receive_from_server().split(Resources.SEP)
8
9     print(response[2])

```

از سوی دیگر سرور پس از بررسی ادمین بودن شخص و حضور شخص گفته شده در گروه، پیامی مبنی بر حذف افراد به همگی اعضاء ارسال می‌کند.

```

322 def remove_member_from_group(client_socket: socket.socket, user: User, group_name,
323                             to_be_removed_username):
324
325     # check if group exists
326
327     if group_name not in groups:
328
329         response = f"404{Resources.SEP}" \
330                 f"Not Found{Resources.SEP}" \
331                 f"Group does not exist."
332
333         response_client(client_socket, response)
334
335         return

```

```

330
331     group = groups[group_name]
332
333     # check if sender of remove request is member of the group
334     if user.username not in group.usernames:
335         response = f"403{Resources.SEP}" \
336             f"Not Forbidden{Resources.SEP}" \
337             f"You are not a member of this group."
338
339     response_client(client_socket, response)
340
341     # check if sender of remove request is admin of the group
342     if user.username != group.admin_username:
343         response = f"403{Resources.SEP}" \
344             f"Not Forbidden{Resources.SEP}" \
345             f"You are not the admin of this group."
346
347     response_client(client_socket, response)
348
349     # check if requested user exists
350     if to_be_removed_username not in [user.username for user in users]:
351         response = f"404{Resources.SEP}" \
352             f"Not Found{Resources.SEP}" \
353             f"User does not exist."
354
355     response_client(client_socket, response)
356
357     # check if requested user is already in the group
358     if to_be_removed_username not in group.usernames:
359         response = f"404{Resources.SEP}" \
360             f"Not Found{Resources.SEP}" \
361             f"User is not a member of this group."
362
363     response_client(client_socket, response)
364
365     for member_username in group.usernames:

```

```

366     # if member_username != user.username:
367
368     remove_text = to_be_removed_username + Resources.SEP + group.group_name
369
370     message_obj = Message(message_type="remove",
371                           source_username="",
372                           target_username=member_username,
373                           seq=0,
374                           signature=RSA.sign(str(0) + remove_text,
375                           server_private_key),
376                           text=remove_text)
377
378     messages[member_username].append(message_obj)
379
380     group.usernames.remove(to_be_removed_username)
381
382     response = f"200{Resources.SEP}" \
383                 f"OK{Resources.SEP}" \
384                 f"User removed from group successfully."
385
386     response_client(client_socket, response)
387
388     return

```

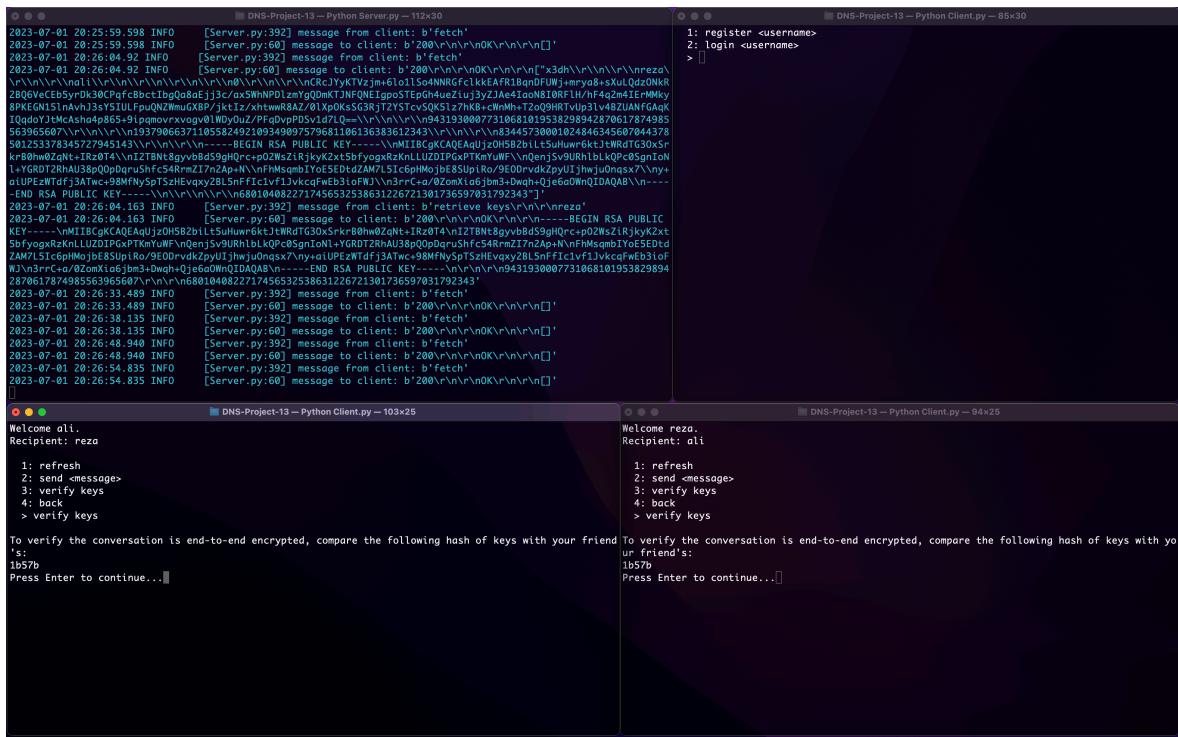
۹-۱ تایید صحت نشست از طریق کانال امن

برای این کار در چت دو طرفه، چند کاراکتر اول کلید فعلی پیام را نشان می‌دهیم، اگر این متغیر برای هر دو طرف یکسان باشد، نشان از صحت نشست دارد. شکل ۲

```

582 def verify_keys(chat: Chat):
583
584     fetch_messages()
585
586     print("To verify the conversation is end-to-end encrypted, compare the following
587          hash of keys with your friend's:")
588
589     print(Resources.get_hash(chat.message_key)[:5])

```



شکل ۲: احراز صحت نشست از طریق کانال امن در چت دوطرفه

برای احراز صحت نشست در گروه لازم است تا هر کس کلیدهایش را با سایرین بررسی کند، کد زیر نیز همیان امکان را فراهم می‌کند. شکل ۳

```

582 def verify_group_keys(group: Group):
583     fetch_messages()
584     print(
585         "To verify the conversation is end-to-end encrypted, compare the following
586         hashes of keys with your friends':")
587     for username in group.usernames:
588         if username != client_user.username:
589             try:
590                 print(username + ": " + Resources.get_hash(chats[username].message_key
591 )[:5])
592             except KeyError:
593                 print(username + ": " + "none")

```

شکل ۳: احراز صحت نشست از طریق کانال امن در گروه

۱-۱ تازه‌سازی کلیدهای نشست

برای این کار کلاینت به سرور اطلاع می‌دهد که قصد این کار را دارد، همچنین پسورد قدیمی و جدیدش را نیز برای سرور ارسال می‌کند. در صورت موفقیت کلاینت به تمامی افرادی که به آنها چت دارد، پیامی مبنی بر handshake X3DH ارسال می‌کند تا رچت‌ها را پارامترهای جدید سازگار شوند و از نو ایجاد شوند، همچنین کلیدهای عمومی جدید را نیز برای آنها ارسال می‌کند تا افراد بتوانند در صورت نیاز امضاهای پیام‌های میانی را بررسی کنند، چرا که سرور دیگر کلید عمومی‌های پیشین فردا نخواهد داشت.

```
720 def renew_keys(old_password, new_password):
721     # fetch messages received from other users based on our old keys
722     fetch_messages()
723
724     if Resources.get_hash(client_user.username + old_password) != client_user.
725         password_hash:
726         print("Wrong password.")
727
728     return
```

```

727     old_password_hash = Resources.get_hash(client_user.username + old_password)
728
729     if initialize_user("renew", client_user.username, new_password, old_password_hash):
730         :
731
732     for target_user in users:
733         x3dh_key_exchange(target_user)

```

در سمت سرور نیز پس از بررسی صحت کلاینت و درست بودن پسورد آن، اطلاعات جدید می‌شوند.

```

92 def renew_keys(client_socket: socket.socket, user: User, old_password_hash,
93                 new_password_hash, rsa_pk, elgamal_pk,
94                 prekey_pk):
95
96     if user.password_hash != old_password_hash:
97
98         response = f"400{Resources.SEP}" \
99                     f"Bad Request{Resources.SEP}" \
100                    f"Wrong password"
101
102     response_client(client_socket, response)
103
104
105     user.password_hash = new_password_hash
106     user.rsa_pk = rsa_pk
107     user.elgamal_pk = elgamal_pk
108     user.prekey_pk = prekey_pk
109
110
111     response = f"200{Resources.SEP}" \
112                     f"OK{Resources.SEP}" \
113                     f"Keys and password renewed successfully"
114
115     response_client(client_socket, response)
116
117     return

```

تمامی افرادی که این فرد با آنها چت داشته پیامی از جنس X3DH دریافت می‌کنند (در تابع `fetch messages` این موضوع هندل شده است که پیشتر مشاهده کردید) و کلیدهای میانی را ایجاد می‌کنند تا پیام‌ها را رمزگشایی کنند و رچت‌هایشان را با این فرد هماهنگ کنند.

۲ نیازمندی‌های سرور

۱-۲ ثبت‌نام و احراز هویت کاربران و نگهداری اطلاعات مربوطه به صورت امن

بخش‌های مرتبط با ثبت‌نام و احراز هویت کاربران را پیشتر توضیح دادیم. برای نگهداری اطلاعات مربوطه به صورت امن اینگونه عمل کردیم که سرور هش یوزرنیم و پسورد کاربران را نگه می‌دارد و در صورت دسترسی مهاجم به مخزن اطلاعاتی آن، مهاجم نخواهد توانست به رمز افراد دسترسی پیدا کند.

```
73 def register_new_user(client_socket: socket.socket, username, password_hash, rsa_pk,
74   elgamal_pk, prekey_pk):
75
76     if username_exists(username):
77
78         response = f"400{Resources.SEP}" \
79             f"Bad Request{Resources.SEP}" \
80             f"Username already exists"
81
82         response_client(client_socket, response)
83
84         return
85
86
87     new_user = User(username, password_hash, rsa_pk, elgamal_pk, prekey_pk)
88     users.append(new_user)
89     messages[new_user.username] = []
90
91
92     response = f"200{Resources.SEP}" \
93         f"OK{Resources.SEP}" \
94         f"User registered successfully"
95
96     response_client(client_socket, response)
97
98     return new_user
```

۲-۲ ارسال پیام‌ها به مقصد

برای این منظور، سرور پیام‌ها را در لیست مربوط به هر فرد قرار می‌دهد و هنگام fetch کردن پیام‌ها توسط کاربر، تمامی این پیام‌ها برای وی ارسال خواهند شد.

```
181 def save_message(client_socket: socket.socket, message: str):
```

```

182     _type, source_username, target_username, target_group, seq, signature, text =
183     message. \
184
185     split(Resources.SEP, maxsplit=7 - 1)
186
187     message_obj = Message(message_type=_type,
188                           source_username=source_username,
189                           target_username=target_username,
190                           target_group=target_group,
191                           seq=seq,
192                           signature=signature,
193                           text=text)
194
195
196     for user in users:
197
198         if user.username == target_username:
199
200             if message_obj.target_group != "":
201
202                 if message_obj.target_group not in groups:
203
204                     response = f"404{Resources.SEP}" \
205                               f"Not Found{Resources.SEP}" \
206                               f"Group does not exist."
207
208                     response_client(client_socket, response)
209
210                     return
211
212             elif message_obj.source_username not in groups[message_obj.
213                                         target_group].usernames:
214
215                 response = f"403{Resources.SEP}" \
216                               f"Forbidden{Resources.SEP}" \
217                               f"You are not a member of this group."
218
219                 response_client(client_socket, response)
220
221                 return
222
223
224             messages[user.username].append(message_obj)
225
226
227             response = f"200{Resources.SEP}" \
228                               f"OK{Resources.SEP}" \
229                               f"Message saved."
230
231             response_client(client_socket, response)
232
233             return
234
235

```

```

216     response = f"404{Resources.SEP}" \
217         f"Not Found{Resources.SEP}" \
218         f"User does not exist."
219     response_client(client_socket, response)
220     return

```

پیام‌های گروه هم مشابها در همین تابع برای فرد درخواست کننده ذخیره و ارسال خواهد شد.

```

223 def fetch_messages(client_socket: socket.socket, user: User):
224     response = f"200{Resources.SEP}" \
225         f"OK{Resources.SEP}" \
226         f"{json.dumps([str(message) for message in messages[user.username]])}"
227     response_client(client_socket, response)
228
229     buffer = receive_from_client(client_socket, user)
230     if buffer == "ack":
231         messages[user.username] = []

```

۳-۲ ارائه‌ی لیست کاربر آنلاین

برای این کار سرور لیستی از کاربران و وضعیت آن‌ها را برای کلاینت ارسال می‌کند.

```

154 def show_users_list(client_socket: socket.socket):
155     response = f"200{Resources.SEP}" \
156         f"OK{Resources.SEP}"
157
158     for user in users:
159         response += f"- {user.username} ({'Online' if user.is_online else 'Offline'})\n"
160
161     response_client(client_socket, response[:-1])
162     return

```

۴-۲ ارسال و دریافت پیام‌های مربوط به ساخت کلید نشست

سرور تمامی پیام‌ها را در یک لیست قرار می‌دهد و این بخش مشابه با ارسال پیام‌ها به مقصد می‌باشد، همچنین تمامی پیام‌های کنترلی گرووه و پیام‌های مرتبط با تغییر کلیدها نیز به همان شکل هندل شده‌اند.

۵-۲ نگهداری اطلاعات مربوط به گروه‌ها

همانطور که پیشتر هم گفته شد، سرور تمامی اطلاعات مربوط به گروه‌ها اعم از اعضای گروه، اسم آن و سازنده‌ی آن را نگهداری می‌کند.

```
234 def create_group(client_socket: socket.socket, user: User, group_name: str):
235     if username_exists(group_name):
236         response = f"400{Resources.SEP}" \
237                     f"Bad Request{Resources.SEP}" \
238                     f"Username already exists"
239         response_client(client_socket, response)
240         return
241
242     group = Group(user.username, group_name)
243     groups[group_name] = group
244     response = f"200{Resources.SEP}" \
245                     f"OK{Resources.SEP}" \
246                     f"Group created successfully"
247     response_client(client_socket, response)
248
249
250 def add_member_to_group(client_socket: socket.socket, user: User, group_name,
251                         new_member_username):
252     # check if group exists
253     if group_name not in groups:
254         response = f"404{Resources.SEP}" \
255                     f"Not Found{Resources.SEP}" \
256                     f"Group does not exist."
257         response_client(client_socket, response)
```

```

257     return
258
259     group = groups[group_name]
260
261     # check if sender of add request is member of the group
262     if user.username not in group.usernames:
263         response = f"403{Resources.SEP}" \
264             f"Not Forbidden{Resources.SEP}" \
265             f"You are not a member of this group."
266         response_client(client_socket, response)
267     return
268
269     # check if sender of add request is admin of the group
270     if user.username != group.admin_username:
271         response = f"403{Resources.SEP}" \
272             f"Not Forbidden{Resources.SEP}" \
273             f"You are not the admin of this group."
274         response_client(client_socket, response)
275     return
276
277     # check if requested user exists
278     if new_member_username not in [user.username for user in users]:
279         response = f"404{Resources.SEP}" \
280             f"Not Found{Resources.SEP}" \
281             f"User does not exist."
282         response_client(client_socket, response)
283     return
284
285     # check if requested user is already in the group
286     if new_member_username in group.usernames:
287         response = f"400{Resources.SEP}" \
288             f"Not Not Found{Resources.SEP}" \
289             f"User is already a member of this group."
290         response_client(client_socket, response)
291     return
292

```

```
293     for old_member_username in group.usernames:
294
295         # if old_member_username != user.username:
296
297         add_text = new_member_username + Resources.SEP + group.group_name
298
299         message_obj = Message(message_type="add",
300
301             source_username="",
302             target_username=old_member_username,
303             seq=0,
304             signature=RSA.sign(str(0) + add_text, server_private_key
305         ),
306
307             text=add_text)
308
309         messages[old_member_username].append(message_obj)
310
311
312         group.usernames.append(new_member_username)
313
314
315         join_text = group.admin_username + Resources.SEP + group.group_name + Resources.
316 SEP + json.dumps(group.usernames)
317
318         message_obj = Message(message_type="join",
319
320             source_username="",
321             target_username=new_member_username,
322             seq=0,
323             signature=RSA.sign(str(0) + join_text, server_private_key),
324             text=join_text)
325
326         messages[new_member_username].append(message_obj)
327
328
329         response = f"200{Resources.SEP}" \
330             f"OK{Resources.SEP}" \
331             f"User added to group successfully."
332
333         response_client(client_socket, response)
334
335         return
336
337
338
339
340
341
342 def remove_member_from_group(client_socket: socket.socket, user: User, group_name,
343 to_be_removed_username):
344
345     # check if group exists
346
347     if group_name not in groups:
348
349         response = f"404{Resources.SEP}" \
```

```

326             f"Not Found{Resources.SEP}" \
327             f"Group does not exist."
328         response_client(client_socket, response)
329     return
330
331     group = groups[group_name]
332
333     # check if sender of remove request is member of the group
334     if user.username not in group.usernames:
335         response = f"403{Resources.SEP}" \
336             f"Not Forbidden{Resources.SEP}" \
337             f"You are not a member of this group."
338         response_client(client_socket, response)
339     return
340
341     # check if sender of remove request is admin of the group
342     if user.username != group.admin_username:
343         response = f"403{Resources.SEP}" \
344             f"Not Forbidden{Resources.SEP}" \
345             f"You are not the admin of this group."
346         response_client(client_socket, response)
347     return
348
349     # check if requested user exists
350     if to_be_removed_username not in [user.username for user in users]:
351         response = f"404{Resources.SEP}" \
352             f"Not Found{Resources.SEP}" \
353             f"User does not exist."
354         response_client(client_socket, response)
355     return
356
357     # check if requested user is already in the group
358     if to_be_removed_username not in group.usernames:
359         response = f"404{Resources.SEP}" \
360             f"Not Found{Resources.SEP}" \
361             f"User is not a member of this group."

```

```

362     response_client(client_socket, response)
363
364
365     for member_username in group.usernames:
366
366         # if member_username != user.username:
367
367             remove_text = to_be_removed_username + Resources.SEP + group.group_name
368
368             message_obj = Message(message_type="remove",
369
369                 source_username="",
370
370                 target_username=member_username,
371
371                 seq=0,
372
372                 signature=RSA.sign(str(0) + remove_text,
373
373                     server_private_key),
374
374                     text=remove_text)
375
375             messages[member_username].append(message_obj)
376
376             group.usernames.remove(to_be_removed_username)
377
377
378             response = f"200{Resources.SEP}" \
379
379                 f"OK{Resources.SEP}" \
380
380                 f"User removed from group successfully."
381
381             response_client(client_socket, response)
382
382             return

```

۳ نیازمندی‌های امنیتی

در این بخش علت اینکه پروتکل ما هریک از نیازمندی‌های امنیتی را برآورده می‌کند را بیان می‌کنیم.

۱-۳ رمزگاری انتهای به انتهای

از آنجا که بین دوطرف ارتباط، کلیدهایی تبادل می‌شود که نیاز به داده‌های خصوصی یک طرف ارتباط دارد، پروتکل ما این نیازمندی را برآورده می‌کند. برای رمزگشایی پیام‌های ارتباط، نیاز است اطلاعات خصوصی یکی از طرفین را مهاجم بدست بیاورد. همچنین سرور پیام‌ها را به صورت رمز شده می‌بیند و هیچگاه اطلاعات مربوط با ساخت کلید را به صورت کامل بدست نمی‌آورد پس این

نیازمندی توسط پروتکل ما برآورده شده است.

۲-۳ تازگی کلید

اولاً توجه کنید که ما مستقل از آنلاین بودن فرد هدف کلید مادر را تبادل می‌کنیم. پس تا وقتی از طرف مقابل پیامی دریافت نکرده باشیم امکان بررسی تازگی نیست. از طرفی حین پاسخ دادن به فرد مقابل، پارامتر جدیدی ایجاد می‌کنیم که آن را خودمان داریم، از طرفی فرد مقابل پس از کدگشایی پیام ما متوجه می‌شود ما این پارامتر را ایجاد کرده ایم که توانسته‌ایم به او پاسخ بدھیم و پیام پیشینش را رمزگشایی کنیم. بنابراین با هر پاسخ گرفتن، دریافت کننده از تازگی کلید می‌تواند اطمینان کسب کند. همچنین از اینکه رچت‌های طرفین با هم سینک است و طرفین می‌توانند پیام‌های یکدیگر را کدگشایی کنند نیز می‌توانند متوجه تازگی کلید بشوند.

از طرفی توجه کنید که تمامی پیام‌ها امضای مرتبط با Sequence number دارند، پس تکرار آن‌ها ممکن نیست، بنابراین مطمئن هستیم کلید سنت شده کلیدی تازه خواهد بود چرا که Sequence number آن با امضای آن تطابق دارد.

۳-۳ محربانگی

واضحاً تمامی پیام‌ها به صورت انتهای رمز شده‌اند، پس پروتکل ما محربانگی را حفظ می‌کند.

۴-۳ صحت و یکپارچگی

از اینکه امضای پیام با متن آن سازگاری دارد می‌توان به برطرف کردن این نیازمندی پی‌برد. هرکس امضای پیامی را که دریافت کرده را بررسی می‌کند و از آنجا که امضای آن با پیام تطابق دارد می‌تواند به صحت و یکپارچگی آن پی‌برد.

۵-۳ حفظ سازگاری

به علت وجود Sequence number افزایشی در پروتکل ما، این نیازمندی برطرف شده است و دریافت کننده از Sequence number پیام دریافتی می‌تواند ترتیب آن را بفهمد.

۶-۳ احراز اصالت

با بررسی امضای پیام که در آن Sequence number حضور دارد، امضای پیام‌های یکسان متفاوت خواهد بود و احراز اصالت پیام‌ها را می‌توان انجام داد. بنابراین هیچ مهاجمی نخواهد توانست از طرف یک کارخواه، پیامی ارسال کند.

۷-۳ عدم انکار

تمامی پیام‌های ارسالی پس از لگین، دارای امضا می‌باشند. پس نیازمندی عدم انکار توسط پروتکل ما برطرف شده است.

۸-۳ کنترل دسترسی

در تمامی پیام‌های کنترلی مرتبط با گروه، سرور بررسی می‌کند که درخواست دهنده ادمین گروه باشد، پس این نیازمندی برطرف شده است. از طرفی حین ارسال پیام به اعضای گروه، سرور بررسی می‌کند که این فرد عضو آن گروه باشد، وگرنه پیام‌های آن را انتقال نمی‌دهد.

۹-۳ حمله‌ی مرد میانی

هنگام ارتباط کلاینت با سرور، صحت و اصالت سرور با استفاده از کلید عمومی آن بررسی می‌شود، پس از ارتباط با سرور و از آنجا که سرور قابل اعتماد است، هیچگاه امکان حمله‌ی مرد میانی وجود نخواهد داشت. همچنین تمامی پیام‌های کلاینت به سرور هم امضا دارند و در هر پیام احراز اصالت دو طرفه صورت می‌گیرد. پس از احراز اصالت سرور توسط کلاینت هم همانطور که گفته شد، کلیدهای متقارنی ست می‌شوند که برای همان ارتباط هستند.

۱۰-۳ حمله‌ی تکرار

برای این حمله در پروتکل ما بخش‌های زیادی وجود دارند. تمامی پیام‌های رد و بدل شده بین کلاینت و سرور استمپ زمانی و یک Sequence number دارند. از طرفی تمامی پیام‌های رد و بدل شده بین کلاینت‌ها هم Sequence number فزاینده و امضایی مرتبط با این Sequence number دارند، بنابراین امکان هیچ نوع حمله‌ی تکرار در پروتکل ما وجود ندارد.

۱۱-۳ استفاده از الگوریتم‌های رمزنگاری امن

تمامی رمزنگاری‌های متقارن با استفاده از AES، تمامی توابع هشا استفاده تابع SHA256 و توابع رمزنگاری نامتقارن هم RSA می‌باشند. همچنین ایجاد اعداد تصادفی هم به صورت امن صورت گرفته‌اند.

۱۲-۳ محربانگی پیشرو

در پروتکل ما کلیدهای رمزنگاری آینده از رد شدن کلیدهای رمزنگاری پیشین از KDF ایجاد شده‌اند. این موضوع محربانگی پیشرو را تضمین می‌کند چرا که KDF یک تابع یک طرفه است و از کلیدهای آینده نمی‌توان کلیدهای گذشته را بازیابی کرد.

۱۳-۳ محربانگی آینده

در پروتکل ما روش Double ratchet پیاده‌سازی شده است. این موضوع به ما این امکان را می‌دهد که حتی اگر یک کلید نشست لوبود، با تبادلهای دیفی‌هلمن مکرر، کلیدهای نشست بعدی قابل ایجاد نباشند و عمل Self Healing که از سمت مقابل می‌آید، امکان ایجاد کلیدهای آینده از بین می‌رود.

۴ نکات قابل توجه

در پروتکل ما امکان تبادل کلید با فرد آفلاین وجود دارد. علاوه بر این، امکان حذف افراد از گروه نیز در پیاده‌سازی ما وجود دارد. همچنین امکان اضافه کردن افراد آفلاین به گروه و حذف افراد آفلاین از گروه نیز وجود دارد. امنیت پروتکل ما در زمینه‌ی گروه، مشابه امنیت پروتکل ما در چت دوطرفه است و تمامی ویژگی‌های امنیتی را دارد.

۵ نحوه راه‌اندازی پروژه

پس از نصب نیازمندی‌ها، برای ران کردن کد سرور به شکل زیر عمل کنید.

```
1 cd <path_to_project>
2 python3 Server.py
```

برای راهاندازی کلاینت هم به صورت زیر عمل کنید.

```
1 cd <path_to_project>
2 python3 Client.py
```

در منوها تمامی کامندها و نحوه استفاده از آنها نمایش داده می‌شوند، برای مثال در شکل ۴ ایمان در منوی گروه است می‌تواند کامندهای زیر را اجرا کند.

```
1 refresh
2 send Hello world
3 verify keys
4 add reza
5 remove ali
6 list members
7 back
```

علی که در منوی چت با ایمان است می‌تواند کامندهای زیر را اجرا کند.

```
1 refresh
2 send hey iman
3 verify keys
4 back
```

و رضا که در منوی اصلی است می‌تواند کامندهای زیر را اجرا کند.

```
1 show users list
2 open chat iman
3 create group dns2
4 open group dns
5 renew keys
6 logout
```

اگر رضا در وضعیت پیش از ورود باشد هم می‌تواند کامندهای زیر را اجرا کند.

```
1 register reza2
2 login reza
```

```

DNS-Project-13 — Python Server.py — 112x30
2870617874985563965607\vn\vn\n680104082271745653253631226721301736597031792343
2023-07-01 20:30:17.299 INFO [Server.py:392] message from client: b'retrieve keys\nOK\nnrezo'
2023-07-01 20:30:17.299 INFO [Server.py:60] message to client: b'200\nOK\nnrezo\n-----BEGIN RSA PUBLIC
KEY-----\nMIIBIjCgKCAQEAqjzOH5B2b1LtSuhu6rkxt1wRdtG30xSrkB0hw0ZQh+IzR0t4v1zTBNt8gvb8d59gHQrc-p02Wz1RjkyKzt
SbfyogxRzNlLUZD1P0xpTKyuWF\nnOenjSy9URhLb1kQPe05gn1n0n+GRDT2RRAU38q0OpgruShFc54KrmZ1nAp+Nnfmhsqmb1y0ESEDd
ZM7L51c6p0MojeBSUp1Ro/9e0DrvrkzpyU1jhjuOnssx7ny+aiUPezMdjfj3ATwc-98MFNySpSzHEvqxy2BL5nfFic1v1JvkkcfWeB3loF
WJN3rrCce/ZonXtd6jb3-Dwh-Qjeed0MnIDAQABn----END RSA PUBLIC KEY-----\n\nv9431930007731.068101953829894
2870617874985563965607\vn\vn\n680104082271745653253631226721301736597031792343
2023-07-01 20:30:17.299 INFO [Server.py:392] message from client: b'group_text\nOK\nn\n
2023-07-01 20:30:17.299 INFO [Server.py:60] message to client: b'200\nOK\nn\n
2023-07-01 20:30:22.707 INFO [Server.py:392] message from client: b'group_text\nOK\nn\n
2023-07-01 20:30:22.707 INFO [Server.py:60] message to client: b'200\nOK\nn\n
2023-07-01 20:30:22.843 INFO [Server.py:392] message from client: b'retrieve keys\nOK\nnrezo'
2023-07-01 20:30:22.843 INFO [Server.py:60] message to client: b'200\nOK\nnrezo\n-----BEGIN RSA PUBLIC
KEY-----\nMIIBIjCgKCAQEAqjzOH5B2b1LtSuhu6rkxt1wRdtG30xSrkB0hw0ZQh+IzR0t4v1zTBNt8gvb8d59gHQrc-p02Wz1RjkyKzt
SbfyogxRzNlLUZD1P0xpTKyuWF\nnOenjSy9URhLb1kQPe05gn1n0n+GRDT2RRAU38q0OpgruShFc54KrmZ1nAp+Nnfmhsqmb1y0ESEDd
ZM7L51c6p0MojeBSUp1Ro/9e0DrvrkzpyU1jhjuOnssx7ny+aiUPezMdjfj3ATwc-98MFNySpSzHEvqxy2BL5nfFic1v1JvkkcfWeB3loF
WJN3rrCce/ZonXtd6jb3-Dwh-Qjeed0MnIDAQABn----END RSA PUBLIC KEY-----\n\nv9431930007731.068101953829894
2870617874985563965607\vn\vn\n680104082271745653253631226721301736597031792343
2023-07-01 20:30:27.163 INFO [Server.py:392] message from client: b'fetch'
2023-07-01 20:30:27.163 INFO [Server.py:60] message to client: b'200\nOK\nn\n
2023-07-01 20:30:38.832 INFO [Server.py:392] message from client: b'fetch'
2023-07-01 20:30:38.832 INFO [Server.py:60] message to client: b'200\nOK\nn\n
2023-07-01 21:32:37.883 INFO [Server.py:392] message from client: b'fetch'
2023-07-01 21:32:37.883 INFO [Server.py:60] message to client: b'200\nOK\nn\n
2023-07-01 21:32:48.99 INFO [Server.py:392] message from client: b'fetch'
2023-07-01 21:32:48.100 INFO [Server.py:60] message to client: b'200\nOK\nn\n

```

شکل ۴: نمایش تمامی منوها

درنهایت برای مشاهده‌ی ریپوی گیتهاب ما، می‌توانید به [۲] مراجعه کنید.

References

- [1] "Signal documentation," <https://www.signal.org/docs/>. Accessed: 2023-07-01.
- [2] "Dns-project-group-13 1401-1402-2," <https://github.com/AlipourIm/DNS-Project-13>. Accessed: 2023-07-01.