SecureChat System - Test Report

Course: CS-3002 Information Security

Assignment: Secure Chat System - Testing Evidence

Student: Ahmad Aqeel
Roll No: 22i-1134

Date: November 15, 2025

1. Introduction

This test report documents the comprehensive testing performed on the SecureChat system to verify that all security properties (Confidentiality, Integrity, Authenticity, and Non-Repudiation) are correctly implemented. Each test includes the commands used, expected results, and screenshots as evidence.

2. Test Environment Setup

Before running tests, ensure the following are set up:

1. 1. MySQL Database Container Running:
    - Command: docker ps
    - Expected: securechat-db container should be running

2. 2. Certificates Generated:
    - Commands:

```
python scripts/gen_ca.py --name "FAST-NU Root CA"
python scripts/gen_cert.py --cn server.local --out certs/server
python scripts/gen_cert.py --cn client.local --out certs/client
```

3. 3. Database Initialized:

  - Command: python -m app.storage.db --init

3. Test 1: Wireshark Capture - Confidentiality Verification

3.1 Objective

Verify that all network traffic is encrypted and no plaintext is visible in captured packets.

3.2 Procedure

Step 1: Start Wireshark and select loopback interface

Step 2: Apply filter: tcp.port == 8888

Step 3: Start server in Terminal 1

Step 4: Start client in Terminal 2 and send messages

Step 5: Stop capture and analyze packets

3.3 Commands Used

- Terminal 1 (Server):

```
python -m app.server
```

- Terminal 2 (Client):

```
python -m app.client
```

  - Then: Register/Login and send messages

- Wireshark Filter:

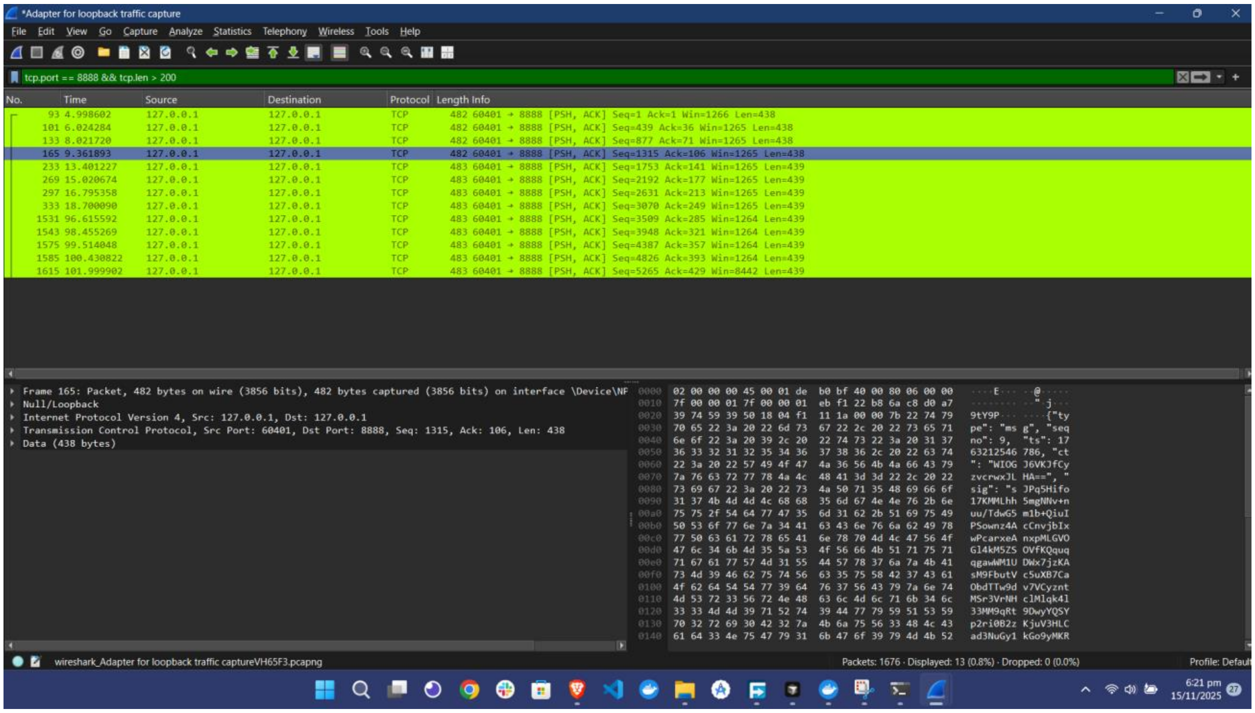
```
tcp.port == 8888
```

3.4 Expected Results

All TCP packets should contain encrypted payloads (base64 encoded data). No readable plaintext should be visible.

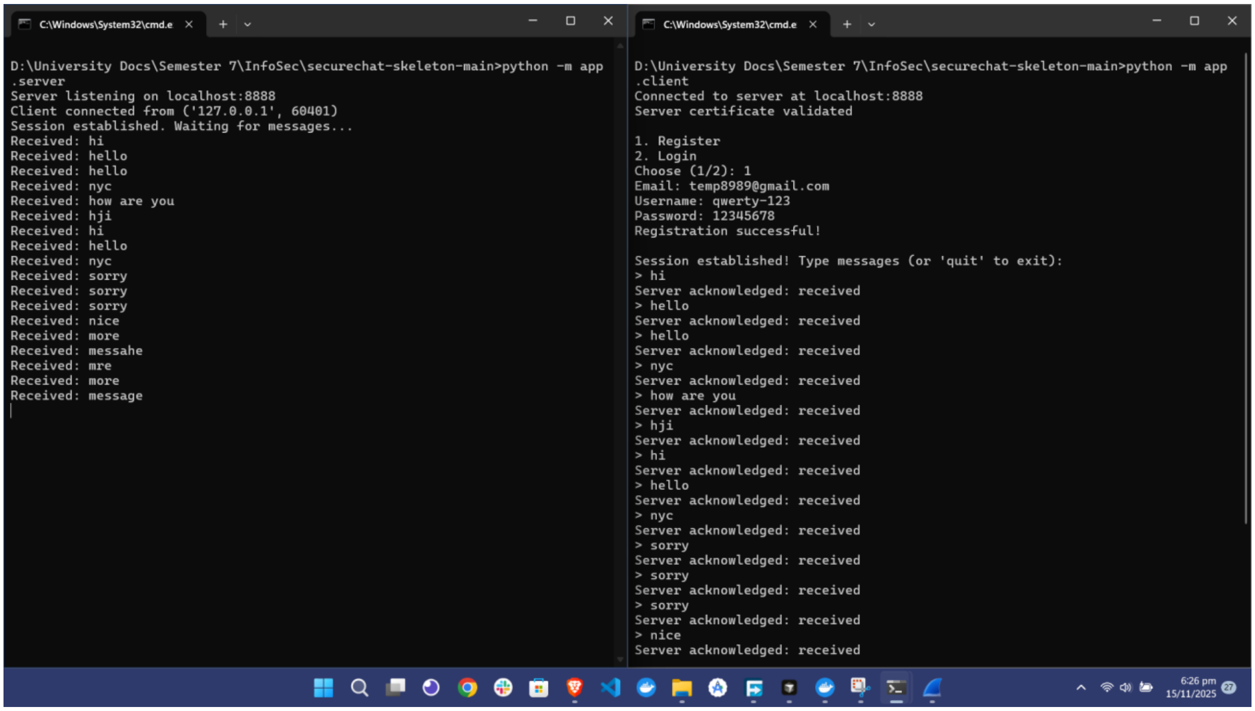
3.5 Evidence

Figure 1: Wireshark Filter Applied (tcp.port == 8888)


*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port == 8888 && tcp.len > 200

No.   Time        Source     Destination  Protocol Length Info
   93 4.998602    127.0.0.1  127.0.0.1    TCP      482 60401 → 8888 [PSH, ACK] Seq=1 Ack=1 Win=1266 Len=438
  101 6.024284    127.0.0.1  127.0.0.1    TCP      482 60401 → 8888 [PSH, ACK] Seq=439 Ack=36 Win=1265 Len=438
  133 8.021720    127.0.0.1  127.0.0.1    TCP      482 60401 → 8888 [PSH, ACK] Seq=877 Ack=71 Win=1265 Len=438
  165 9.361893    127.0.0.1  127.0.0.1    TCP      482 60401 → 8888 [PSH, ACK] Seq=1315 Ack=106 Win=1265 Len=438
  233 13.401227   127.0.0.1  127.0.0.1    TCP      483 60401 → 8888 [PSH, ACK] Seq=1753 Ack=141 Win=1265 Len=439
  269 15.020674   127.0.0.1  127.0.0.1    TCP      483 60401 → 8888 [PSH, ACK] Seq=2192 Ack=177 Win=1265 Len=439
  297 16.795358   127.0.0.1  127.0.0.1    TCP      483 60401 → 8888 [PSH, ACK] Seq=2631 Ack=213 Win=1265 Len=439
  333 18.700090   127.0.0.1  127.0.0.1    TCP      483 60401 → 8888 [PSH, ACK] Seq=3070 Ack=249 Win=1265 Len=439
 1531 96.615592   127.0.0.1  127.0.0.1    TCP      483 60401 → 8888 [PSH, ACK] Seq=3509 Ack=285 Win=1264 Len=439
 1543 98.455269   127.0.0.1  127.0.0.1    TCP      483 60401 → 8888 [PSH, ACK] Seq=3948 Ack=321 Win=1264 Len=439
 1575 99.514048   127.0.0.1  127.0.0.1    TCP      483 60401 → 8888 [PSH, ACK] Seq=4387 Ack=357 Win=1264 Len=439
 1585 100.430822  127.0.0.1  127.0.0.1    TCP      483 60401 → 8888 [PSH, ACK] Seq=4826 Ack=393 Win=1264 Len=439
 1615 101.999902  127.0.0.1  127.0.0.1    TCP      483 60401 → 8888 [PSH, ACK] Seq=5265 Ack=429 Win=8442 Len=439

▸ Frame 165: Packet, 482 bytes on wire (3856 bits), 482 bytes captured (3856 bits) on interface \Device\NF
▸ Null/Loopback
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 60401, Dst Port: 8888, Seq: 1315, Ack: 106, Len: 438
▸ Data (438 bytes)

0000  02 00 00 00 45 00 01 de  b0 bf 40 00 80 06 00 00   ····E··· ··@·····
0010  7f 00 00 01 7f 00 00 01  eb f1 22 b8 6a c8 d0 a7   ········ ··"·j···
0020  39 74 59 39 50 18 04 f1  11 1a 00 00 7b 22 74 79   9tY9P··· ····{"ty
0030  70 65 22 3a 20 22 6d 73  67 22 2c 20 22 73 65 71   pe": "ms g", "seq
0040  6e 6f 22 3a 20 39 2c 20  22 74 73 22 3a 20 31 37   no": 9,  "ts": 17
0050  36 33 32 31 32 35 34 36  37 38 36 2c 20 22 63 74   63212546 786, "ct
0060  22 3a 20 22 57 49 4f 47  4a 36 56 4b 4a 66 43 79   ": "WIOG J6VKJfCy
0070  7a 76 63 72 77 78 4a 4c  48 41 3d 3d 22 2c 20 22   zvcrwxJL HA==", "
0080  73 69 67 22 3a 20 22 73  4a 50 71 35 48 69 66 6f   sig": "s JPq5Hifo
0090  31 37 4b 4d 4d 4c 68 68  35 6d 67 4e 4e 76 2b 6e   17KMMLhh 5mgNNv+n
00a0  75 75 2f 54 64 77 47 35  6d 31 62 2b 51 69 75 49   uu/TdwG5 m1b+QiuI
00b0  50 53 6f 77 6e 7a 34 41  63 43 6e 76 6a 62 49 78   PSownz4A cCnvjbIx
00c0  77 50 63 61 72 78 65 41  6e 78 70 4d 4c 47 56 4f   wPcarxeA nxpMLGVO
00d0  47 6c 34 6b 4d 35 5a 53  4f 56 66 4b 51 71 75 71   Gl4kM5ZS OVfKQquq
00e0  71 67 61 77 57 4d 31 55  44 57 78 37 6a 7a 4b 41   qgawWM1U DWx7jzKA
00f0  73 4d 39 46 62 75 74 56  63 35 75 58 42 37 43 61   sM9FbutV c5uXB7Ca
0100  4f 62 64 54 54 77 39 64  76 37 56 43 79 7a 6e 74   ObdTTw9d v7VCyznt
0110  4d 53 72 33 56 72 4e 48  63 6c 4d 6c 71 6b 34 6c   MSr3VrNH clMlqk4l
0120  33 33 4d 4d 39 71 52 74  39 44 77 79 59 51 53 59   33MM9qRt 9DwyYQSY
0130  70 32 72 69 30 42 32 7a  4b 6a 75 56 33 48 4c 43   p2ri0B2z KjuV3HLC
0140  61 64 33 4e 75 47 79 31  6b 47 6f 39 79 4d 4b 52   ad3NuGy1 kGo9yMKR

wireshark_Adapter for loopback traffic captureVH65F3.pcapng    Packets: 1676 · Displayed: 13 (0.8%) · Dropped: 0 (0.0%)    Profile: Default
```

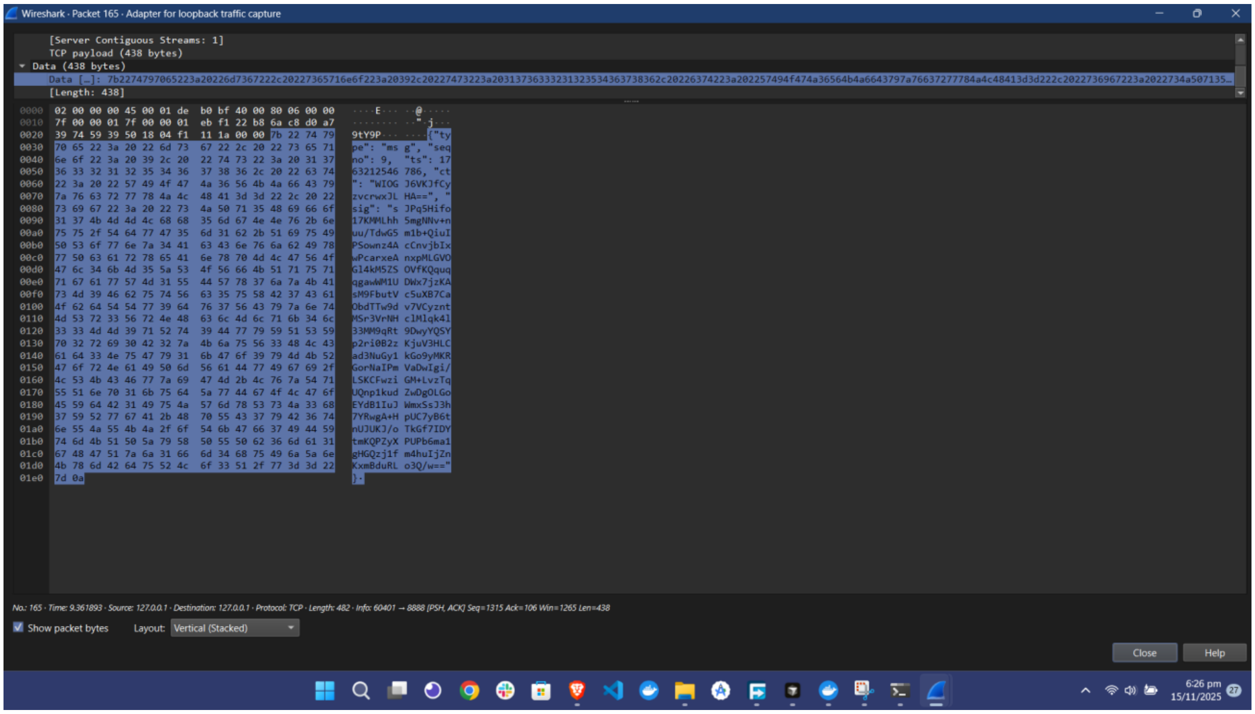
Figure 2: Wireshark Capture - Client to Server Communication



```
D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>python -m app
.server
Server listening on localhost:8888
Client connected from ('127.0.0.1', 60401)
Session established. Waiting for messages...
Received: hi
Received: hello
Received: hello
Received: nyc
Received: how are you
Received: hji
Received: hi
Received: hello
Received: nyc
Received: sorry
Received: sorry
Received: sorry
Received: nice
Received: more
Received: messahe
Received: mre
Received: more
Received: message
```

```
D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>python -m app
.client
Connected to server at localhost:8888
Server certificate validated

1. Register
2. Login
Choose (1/2): 1
Email: temp8989@gmail.com
Username: qwerty-123
Password: 12345678
Registration successful!

Session established! Type messages (or 'quit' to exit):
> hi
Server acknowledged: received
> hello
Server acknowledged: received
> hello
Server acknowledged: received
> nyc
Server acknowledged: received
> how are you
Server acknowledged: received
> hji
Server acknowledged: received
> hi
Server acknowledged: received
> hello
Server acknowledged: received
> nyc
Server acknowledged: received
> sorry
Server acknowledged: received
> sorry
Server acknowledged: received
> sorry
Server acknowledged: received
> nice
Server acknowledged: received
```

Figure 3: Encrypted Payload - No Plaintext Visible



Wireshark · Packet 165 · Adapter for loopback traffic capture

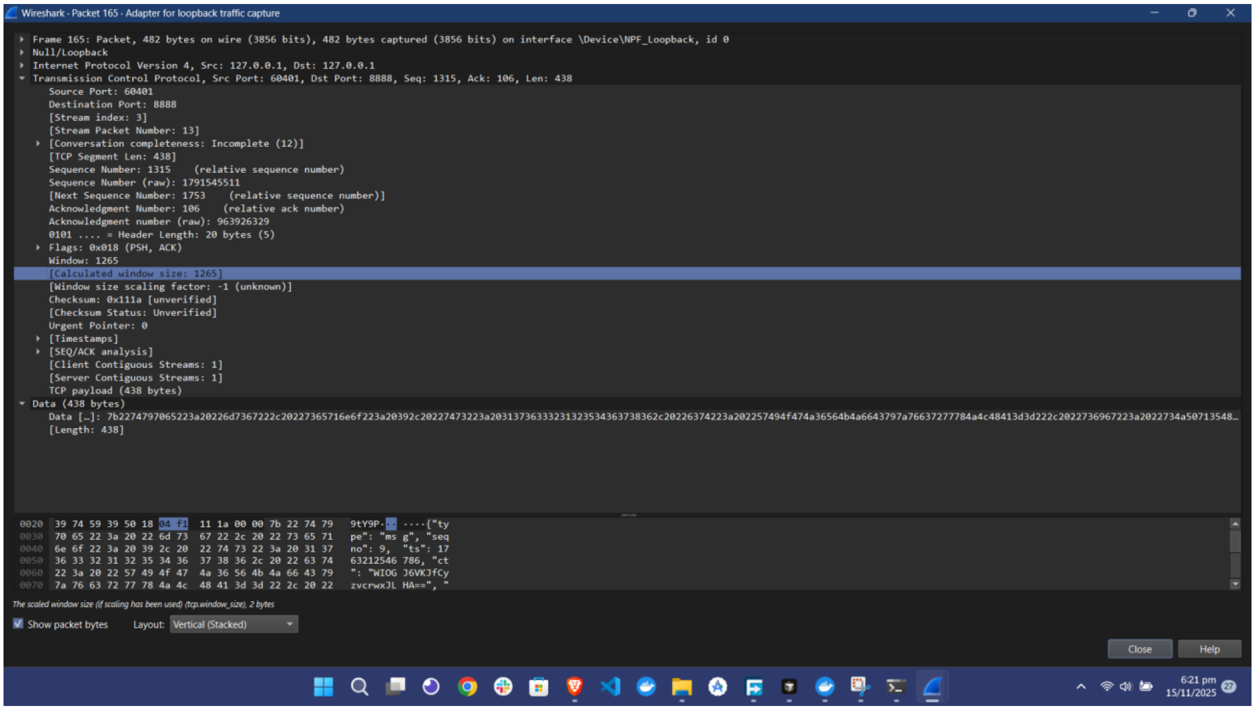
```
        [Server Contiguous Streams: 1]
        TCP payload (438 bytes)
    Data (438 bytes)
        Data […]: 7b2274797065223a20226d7367222c20227365716e6f223a20392c20227473223a20313736333231323534363738362c20226374223a202257494f474a36564b4a6643797a76637277784a4c48413d3d222c2022736967223a2022734a507135…
        [Length: 438]
```

```
0000  02 00 00 00 45 00 01 de  b0 bf 40 00 80 06 00 00   ····E··· ··@·····
0010  7f 00 00 01 7f 00 00 01  eb f1 22 b8 6a c8 d0 a7   ········ ··"·j···
0020  39 74 59 39 50 18 04 f1  11 1a 00 00 7b 22 74 79   9tY9P··· ····{"ty
0030  70 65 22 3a 20 22 6d 73  67 22 2c 20 22 73 65 71   pe": "ms g", "seq
0040  6e 6f 22 3a 20 39 2c 20  22 74 73 22 3a 20 31 37   no": 9,  "ts": 17
0050  36 33 32 31 32 35 34 36  37 38 36 2c 20 22 63 74   63212546 786, "ct
0060  22 3a 20 22 57 49 4f 47  4a 36 56 4b 4a 66 43 79   ": "WIOG J6VKJfCy
0070  7a 76 63 72 77 78 4a 4c  48 41 3d 3d 22 2c 20 22   zvcrwxJL HA==", "
0080  73 69 67 22 3a 20 22 73  4a 50 71 35 48 69 66 6f   sig": "s JPq5Hifo
0090  31 37 4b 4d 4d 4c 68 68  35 6d 67 4e 4e 76 2b 6e   17KMMLhh 5mgNNv+n
00a0  75 75 2f 54 64 77 47 35  6d 31 62 2b 51 69 75 49   uu/TdwG5 m1b+QiuI
00b0  50 53 6f 77 6e 7a 34 41  63 43 6e 76 6a 62 49 78   PSownz4A cCnvjbIx
00c0  77 50 63 61 72 78 65 41  6e 78 70 4d 4c 47 56 4f   wPcarxeA nxpMLGVO
00d0  47 6c 34 6b 4d 35 5a 53  4f 56 66 4b 51 71 75 71   Gl4kM5ZS OVfKQquq
00e0  71 67 61 77 57 4d 31 55  44 57 78 37 6a 7a 4b 41   qgawWM1U DWx7jzKA
00f0  73 4d 39 46 62 75 74 56  63 35 75 58 42 37 43 61   sM9FbutV c5uXB7Ca
0100  4f 62 64 54 54 77 39 64  76 37 56 43 79 7a 6e 74   ObdTTw9d v7VCyznt
0110  4d 53 72 33 56 72 4e 48  63 6c 4d 6c 71 6b 34 6c   MSr3VrNH clMlqk4l
0120  33 33 4d 4d 39 71 52 74  39 44 77 79 59 51 53 59   33MM9qRt 9DwyYQSY
0130  70 32 72 69 30 42 32 7a  4b 6a 75 56 33 48 4c 43   p2ri0B2z KjuV3HLC
0140  61 64 33 4e 75 47 79 31  6b 47 6f 39 79 4d 4b 52   ad3NuGy1 kGo9yMKR
0150  47 6f 72 4e 61 49 50 6d  56 61 44 77 49 67 69 2f   GorNaIPm VaDwIgi/
0160  4c 53 4b 43 46 77 7a 69  47 4d 2b 4c 76 7a 54 71   LSKCFwzi GM+LvzTq
0170  55 51 6e 70 31 6b 75 64  5a 77 44 67 4f 4c 47 6f   UQnp1kud ZwDgOLGo
0180  45 59 64 42 31 49 75 4a  57 6d 78 53 73 4a 33 68   EYdB1IuJ WmxSsJ3h
0190  37 59 52 77 67 41 2b 48  70 55 43 37 79 42 36 74   7YRwgA+H pUC7yB6t
01a0  6e 55 4a 55 4b 4a 2f 6f  54 6b 47 66 37 49 44 59   nUJUKJ/o TkGf7IDY
01b0  74 6d 4b 51 50 5a 79 58  50 55 50 62 36 6d 61 31   tmKQPZyX PUPb6ma1
01c0  67 48 47 51 7a 6a 31 66  6d 34 68 75 49 6a 5a 6e   gHGQzj1f m4huIjZn
01d0  4b 78 6d 42 64 75 52 4c  6f 33 51 2f 77 3d 3d 22   KxmBduRL o3Q/w=="
01e0  7d 0a                                              }·
```

No.: 165 · Time: 9.361893 · Source: 127.0.0.1 · Destination: 127.0.0.1 · Protocol: TCP · Length: 482 · Info: 60401 → 8888 [PSH, ACK] Seq=1315 Ack=106 Win=1265 Len=438

☑ Show packet bytes   Layout: Vertical (Stacked)

Figure 4: Wireshark Packet Detail - Encrypted Data Section



Wireshark · Packet 165 · Adapter for loopback traffic capture

```
    Frame 165: Packet, 482 bytes on wire (3856 bits), 482 bytes captured (3856 bits) on interface \Device\NPF_Loopback, id 0
    Null/Loopback
    Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
    Transmission Control Protocol, Src Port: 60401, Dst Port: 8888, Seq: 1315, Ack: 106, Len: 438
        Source Port: 60401
        Destination Port: 8888
        [Stream index: 3]
        [Stream Packet Number: 13]
        [Conversation completeness: Incomplete (12)]
        [TCP Segment Len: 438]
        Sequence Number: 1315    (relative sequence number)
        Sequence Number (raw): 1791545511
        [Next Sequence Number: 1753    (relative sequence number)]
        Acknowledgment Number: 106    (relative ack number)
        Acknowledgment number (raw): 963926329
        0101 .... = Header Length: 20 bytes (5)
        Flags: 0x018 (PSH, ACK)
        Window: 1265
        [Calculated window size: 1265]
        [Window size scaling factor: -1 (unknown)]
        Checksum: 0x111a [unverified]
        [Checksum Status: Unverified]
        Urgent Pointer: 0
        [Timestamps]
        [SEQ/ACK analysis]
        [Client Contiguous Streams: 1]
        [Server Contiguous Streams: 1]
        TCP payload (438 bytes)
    Data (438 bytes)
        Data […]: 7b2274797065223a20226d7367222c20227365716e6f223a20392c20227473223a20313736333231323534363738362c20226374223a202257494f474a36564b4a6643797a76637277784a4c48413d3d222c2022736967223a2022734a50713548…
        [Length: 438]
```

```
0020  39 74 59 39 50 18 04 f1  11 1a 00 00 7b 22 74 79   9tY9P··· ····{"ty
0030  70 65 22 3a 20 22 6d 73  67 22 2c 20 22 73 65 71   pe": "ms g", "seq
0040  6e 6f 22 3a 20 39 2c 20  22 74 73 22 3a 20 31 37   no": 9,  "ts": 17
0050  36 33 32 31 32 35 34 36  37 38 36 2c 20 22 63 74   63212546 786, "ct
0060  22 3a 20 22 57 49 4f 47  4a 36 56 4b 4a 66 43 79   ": "WIOG J6VKJfCy
0070  7a 76 63 72 77 78 4a 4c  48 41 3d 3d 22 2c 20 22   zvcrwxJL HA==", "
```

The scaled window size (if scaling has been used) (tcp.window_size), 2 bytes

☑ Show packet bytes   Layout: Vertical (Stacked)

3.6 Result

✓PASS: All network traffic is encrypted. No plaintext is visible in captured packets. Confidentiality is verified.

4. Test 2: Invalid Certificate Rejection - Authenticity Verification

4.1 Objective

Verify that the server rejects invalid, self-signed, or untrusted certificates with BAD_CERT error.

4.2 Procedure

Step 1: Generate invalid self-signed certificate

Step 2: Backup valid client certificates

Step 3: Replace valid certificates with invalid ones

Step 4: Attempt to connect to server

Step 5: Restore valid certificates

4.3 Commands Used

- Generate Invalid Certificate:

```
python scripts/gen_invalid_cert.py
```

- Backup Valid Certificates:

```
copy certs\client.crt certs\client.crt.backup
copy certs\client.key certs\client.key.backup
```

- Replace with Invalid Certificate:

```
copy certs\invalid.crt certs\client.crt
copy certs\invalid.key certs\client.key
```

- Start Server:

```
python -m app.server
```

- Attempt Connection:

```
python -m app.client
```

- Restore Valid Certificates:

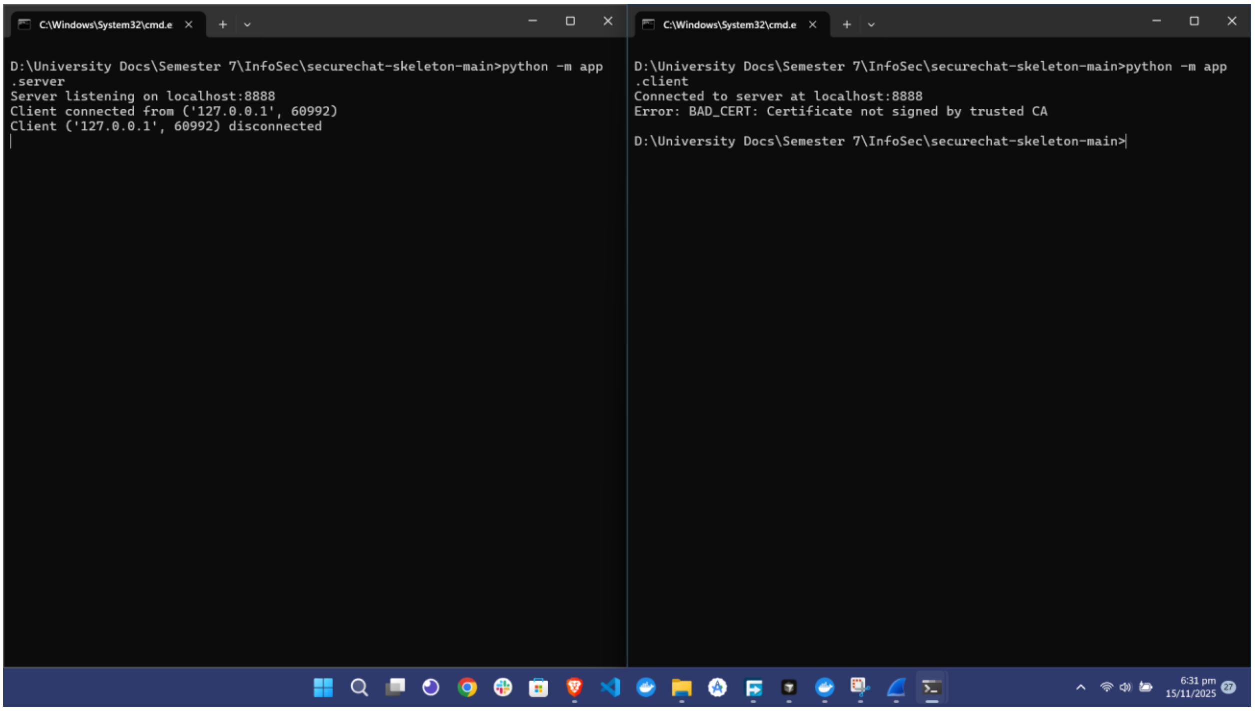
```
copy certs\client.crt.backup certs\client.crt
copy certs\client.key.backup certs\client.key
```

4.4 Expected Results

Connection should be rejected with error: BAD_CERT: Self-signed certificate rejected

4.5 Evidence

Figure 5: Invalid Certificate Rejection (BAD_CERT)



```
D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>python -m app
.server
Server listening on localhost:8888
Client connected from ('127.0.0.1', 60992)
Client ('127.0.0.1', 60992) disconnected
```

```
D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>python -m app
.client
Connected to server at localhost:8888
Error: BAD_CERT: Certificate not signed by trusted CA

D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>
```

4.6 Result

✓PASS: Invalid certificates are rejected. Only certificates signed by the trusted CA are accepted. Authenticity is verified.

5. Test 3: Tamper Detection - Integrity Verification

5.1 Objective

Verify that message tampering is detected through signature verification failure (SIG_FAIL).

5.2 Procedure

Step 1: Start server

Step 2: Run tamper test script

Step 3: Script sends normal message (should succeed)

Step 4: Script sends tampered message (should fail with SIG_FAIL)

5.3 Commands Used

- Terminal 1 (Server):

```
python -m app.server
```

- Terminal 2 (Test Script):

```
python -m tests.tamper_test
```

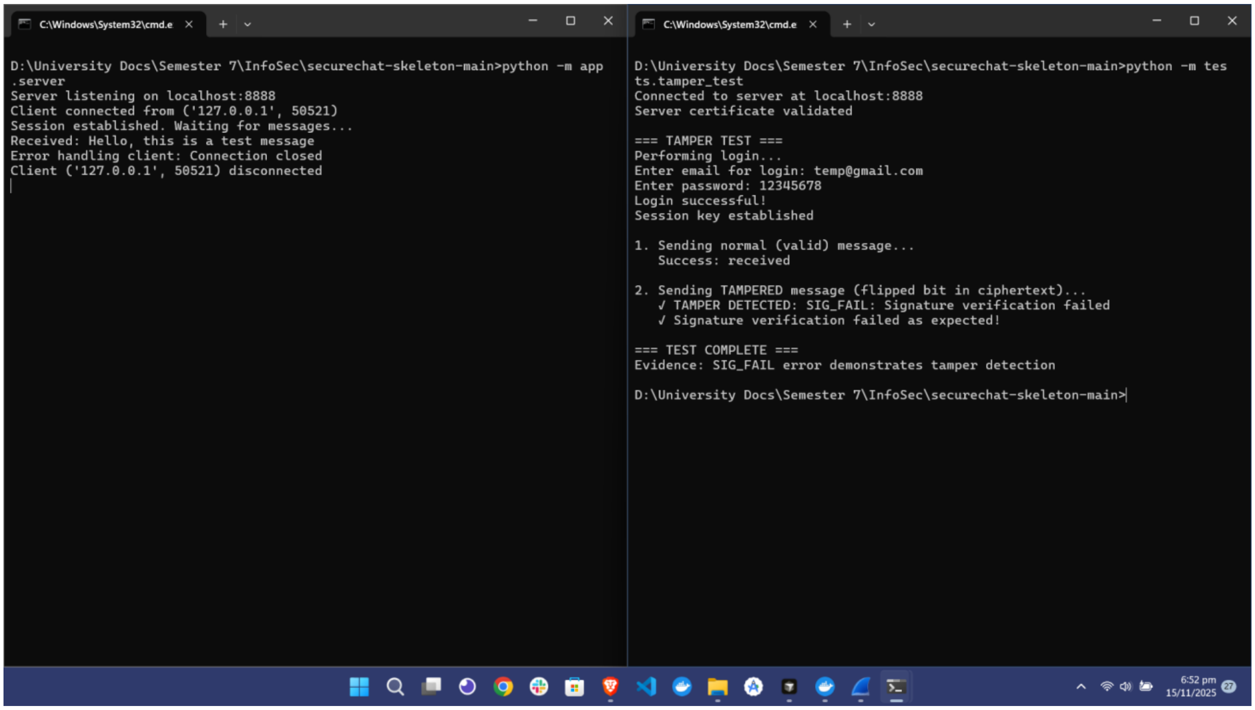
- When prompted, enter:
  - Email: [your registered email]
  - Password: [your password]

5.4 Expected Results

First message should be accepted. Second (tampered) message should be rejected with:
SIG_FAIL: Signature verification failed

5.5 Evidence

Figure 6: Tamper Test - Signature Verification Failure (SIG_FAIL)



```
D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>python -m app
.server
Server listening on localhost:8888
Client connected from ('127.0.0.1', 50521)
Session established. Waiting for messages...
Received: Hello, this is a test message
Error handling client: Connection closed
Client ('127.0.0.1', 50521) disconnected
```

```
D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>python -m tes
ts.tamper_test
Connected to server at localhost:8888
Server certificate validated

=== TAMPER TEST ===
Performing login...
Enter email for login: temp@gmail.com
Enter password: 12345678
Login successful!
Session key established

1. Sending normal (valid) message...
   Success: received

2. Sending TAMPERED message (flipped bit in ciphertext)...
   ✓ TAMPER DETECTED: SIG_FAIL: Signature verification failed
   ✓ Signature verification failed as expected!

=== TEST COMPLETE ===
Evidence: SIG_FAIL error demonstrates tamper detection

D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>
```

5.6 Result

✓PASS: Message tampering is detected. Any modification to ciphertext invalidates the signature. Integrity is verified.

6. Test 4: Replay Protection Verification

6.1 Objective

Verify that replay attacks are prevented through sequence number checking (REPLAY error).

6.2 Procedure

Step 1: Start server

Step 2: Run replay test script

Step 3: Script sends message with seqno=1 (should succeed)

Step 4: Script sends message with seqno=2 (should succeed)

Step 5: Script resends message with seqno=1 (should fail with REPLAY)

6.3 Commands Used

- Terminal 1 (Server):

```
python -m app.server
```

- Terminal 2 (Test Script):

```
python -m tests.replay_test
```

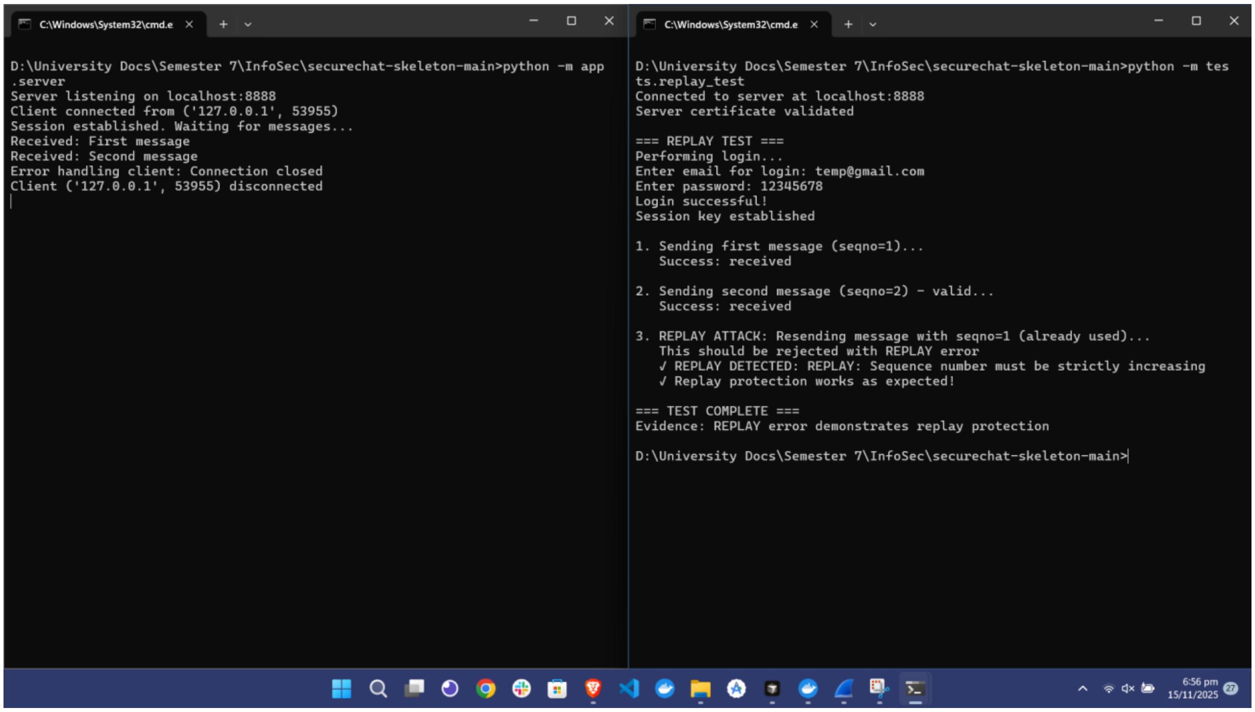
- When prompted, enter:
  - Email: [your registered email]
  - Password: [your password]

6.4 Expected Results

First two messages should be accepted. Third message (replay) should be rejected with: REPLAY: Sequence number must be strictly increasing

6.5 Evidence

Figure 7: Replay Test - Sequence Number Rejection (REPLAY)



```
D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>python -m app
.server
Server listening on localhost:8888
Client connected from ('127.0.0.1', 53955)
Session established. Waiting for messages...
Received: First message
Received: Second message
Error handling client: Connection closed
Client ('127.0.0.1', 53955) disconnected
```

```
D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>python -m tes
ts.replay_test
Connected to server at localhost:8888
Server certificate validated

=== REPLAY TEST ===
Performing login...
Enter email for login: temp@gmail.com
Enter password: 12345678
Login successful!
Session key established

1. Sending first message (seqno=1)...
   Success: received

2. Sending second message (seqno=2) - valid...
   Success: received

3. REPLAY ATTACK: Resending message with seqno=1 (already used)...
   This should be rejected with REPLAY error
   ✓ REPLAY DETECTED: REPLAY: Sequence number must be strictly increasing
   ✓ Replay protection works as expected!

=== TEST COMPLETE ===
Evidence: REPLAY error demonstrates replay protection

D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>
```

6.6 Result

✓PASS: Replay attacks are prevented. Duplicate sequence numbers are rejected. Replay protection is verified.

7. Test 5: Non-Repudiation Verification

7.1 Objective

Verify that transcripts and SessionReceipts can be verified offline, providing cryptographic proof of communication.

7.2 Procedure

Step 1: Run a chat session (register/login, send messages)

Step 2: End session (type quit) - transcript and receipt are automatically saved

Step 3: Run verification script on exported transcript and receipt

Step 4: Verify all message signatures are valid

Step 5: Verify transcript hash matches receipt

Step 6: Verify receipt signature is valid

7.3 Commands Used

- Step 1: Run Chat Session

```
Terminal 1: python -m app.server
Terminal 2: python -m app.client
```

  - Then: Register/Login, send 2-3 messages, type quit

- Step 2: Verify Transcript and Receipt

```
python -m tests.verify_transcript --transcript transcripts/client_localhost_8888.export.txt --
cert certs/client.crt --expected-cn client.local
```

  - Note: Receipt file is auto-detected from transcripts/ directory

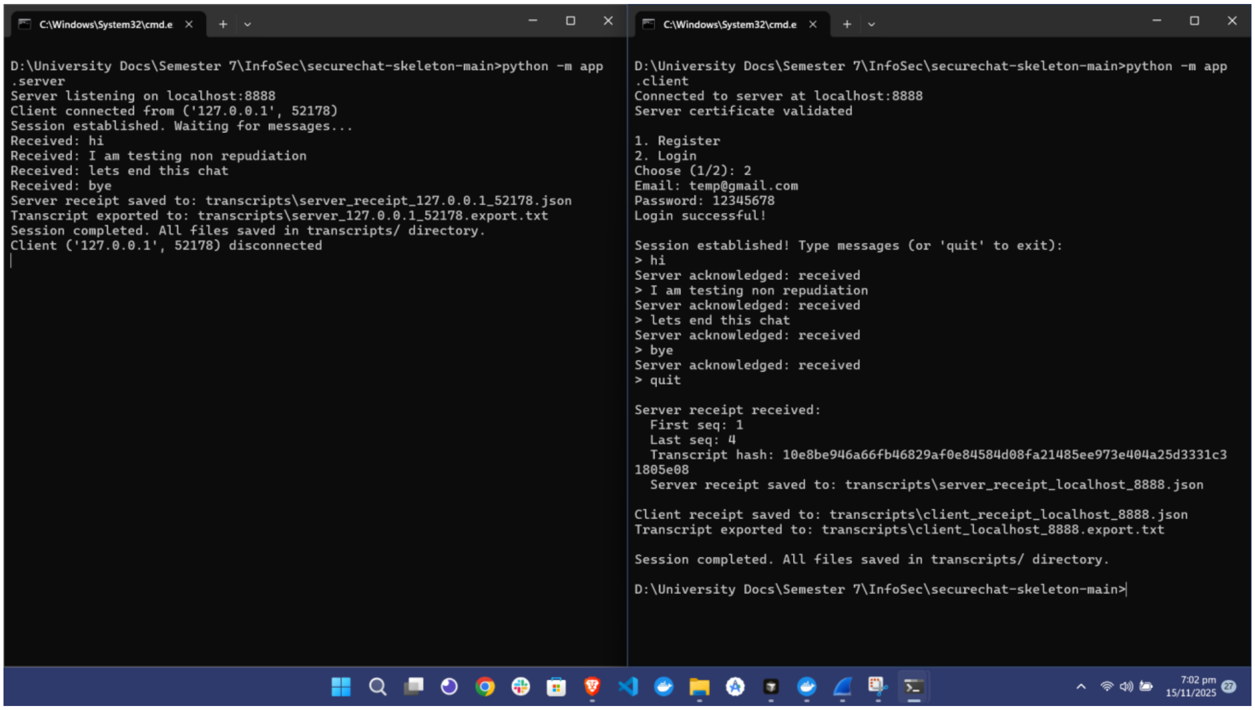
7.4 Expected Results

Verification should show:

- ✓All message signatures are valid
- ✓Receipt hash matches computed transcript hash
- ✓Receipt signature is valid
- ✓ALL CHECKS PASSED

7.5 Evidence

Figure 8: Non-Repudiation Verification - Transcript and Receipt Validation



```
D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>python -m app
.server
Server listening on localhost:8888
Client connected from ('127.0.0.1', 52178)
Session established. Waiting for messages...
Received: hi
Received: I am testing non repudiation
Received: lets end this chat
Received: bye
Server receipt saved to: transcripts\server_receipt_127.0.0.1_52178.json
Transcript exported to: transcripts\server_127.0.0.1_52178.export.txt
Session completed. All files saved in transcripts/ directory.
Client ('127.0.0.1', 52178) disconnected
```

```
D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>python -m app
.client
Connected to server at localhost:8888
Server certificate validated

1. Register
2. Login
Choose (1/2): 2
Email: temp@gmail.com
Password: 12345678
Login successful!

Session established! Type messages (or 'quit' to exit):
> hi
Server acknowledged: received
> I am testing non repudiation
Server acknowledged: received
> lets end this chat
Server acknowledged: received
> bye
Server acknowledged: received
> quit

Server receipt received:
  First seq: 1
  Last seq: 4
  Transcript hash: 10e8be946a66fb46829af0e84584d08fa21485ee973e404a25d3331c3
1805e08
  Server receipt saved to: transcripts\server_receipt_localhost_8888.json

Client receipt saved to: transcripts\client_receipt_localhost_8888.json
Transcript exported to: transcripts\client_localhost_8888.export.txt

Session completed. All files saved in transcripts/ directory.

D:\University Docs\Semester 7\InfoSec\securechat-skeleton-main>
```

7.6 Result

✓PASS: Transcripts and SessionReceipts can be verified offline. All signatures are valid. Non-repudiation is verified.

8. Test 6: Database Schema and User Storage Verification

8.1 Objective

Verify that user credentials are stored securely with salted SHA-256 password hashes.

8.2 Procedure

Step 1: Access Adminer web interface

Step 2: Login to MySQL database

Step 3: View users table structure and data

8.3 Commands/Steps Used

4. 1. Open browser: http://localhost:8081
5. 2. Login credentials:

    - System: MySQL
    - Server: db
    - Username: user
    - Password: 12345678
    - Database: securechat

6. 3. Select users table
7. 4. View table structure and data

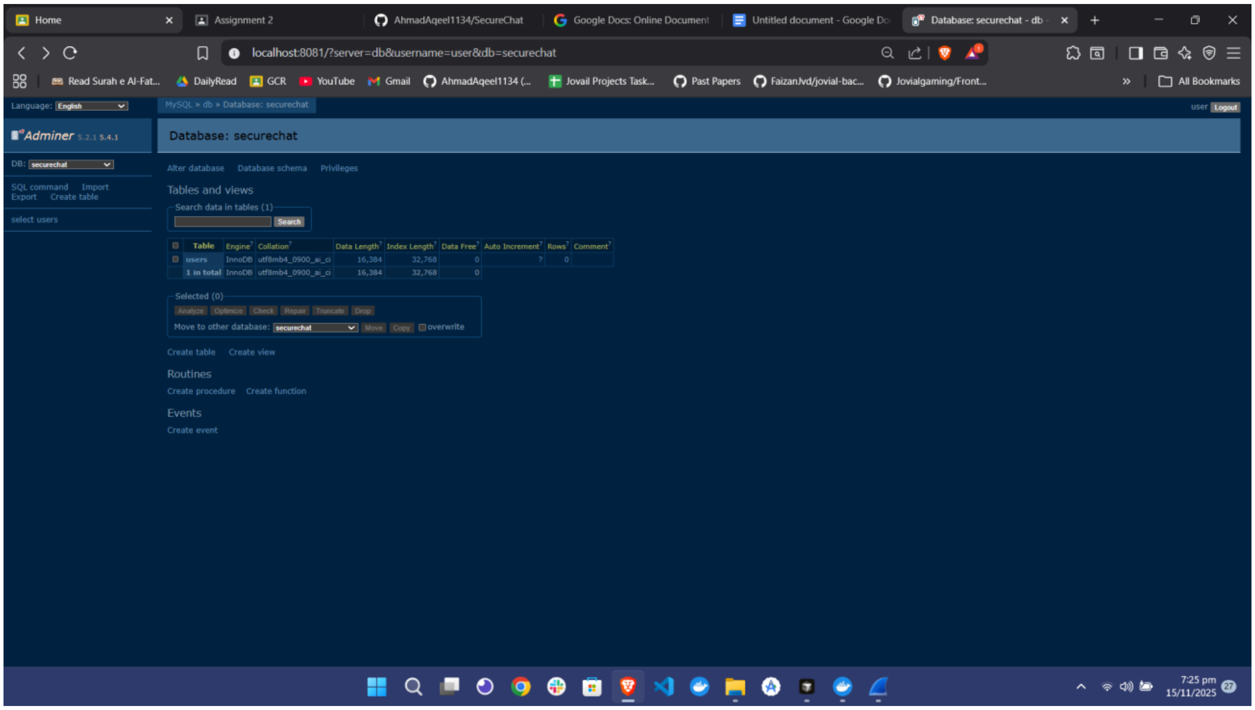
8.4 Expected Results

Users table should show:

- • email (VARCHAR, PRIMARY KEY)
- • username (VARCHAR, UNIQUE)
- • salt (VARBINARY(16))
- • pwd_hash (CHAR(64)) - SHA-256 hex string
- • No plaintext passwords stored

8.5 Evidence

Figure 9: Database Schema in Adminer



localhost:8081/?server=db&username=user&db=securechat

MySQL » db » Database: securechat

Database: securechat

Alter database   Database schema   Privileges

Tables and views

Search data in tables (1)

Table	Engine	Collation	Data Length	Index Length	Data Free	Auto Increment	Rows	Comment
users	InnoDB	utf8mb4_0900_ai_ci	16,384	32,768	0	?	0	
1 in total	InnoDB	utf8mb4_0900_ai_ci	16,384	32,768	0			

Selected (0)

Move to other database: securechat   □ overwrite

Create table   Create view

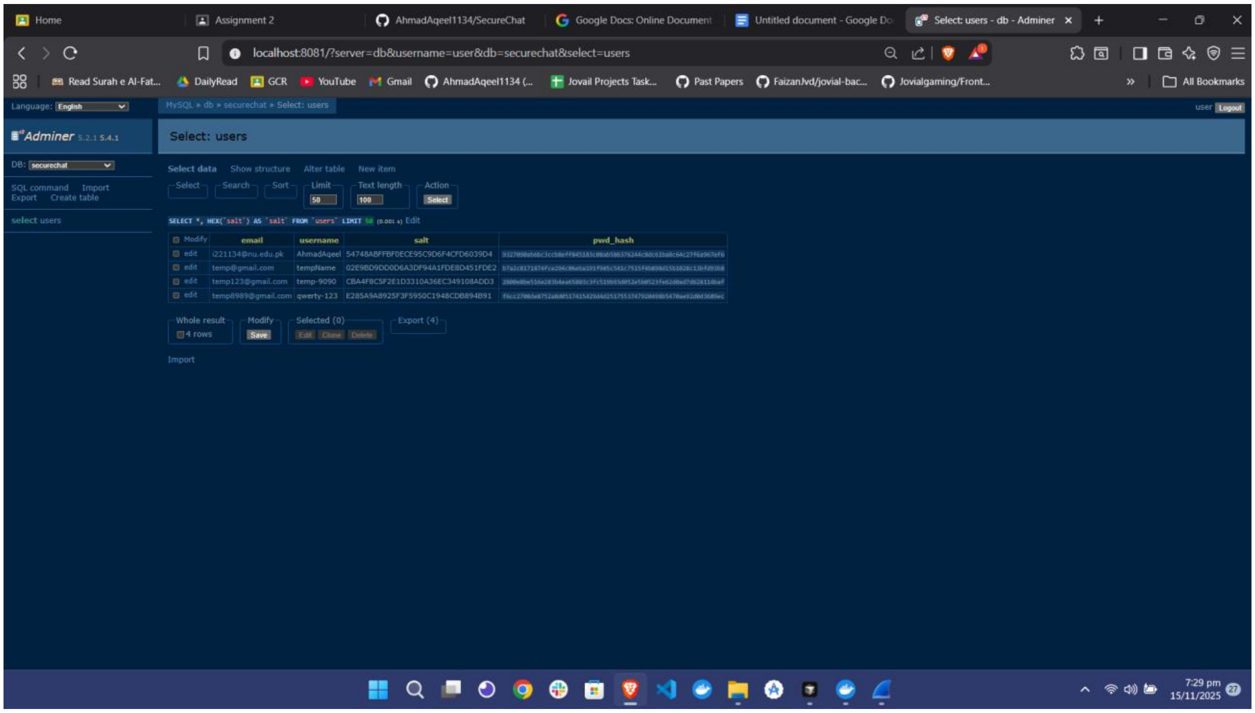
Routines

Create procedure   Create function

Events

Create event

Figure 10: Users Table with Sample Records



localhost:8081/?server=db&username=user&db=securechat&select=users

MySQL » db » securechat » Select: users

Select: users

Select data   Show structure   Alter table   New item

Limit: 50   Text length: 100

```
SELECT *, HEX(`salt`) AS `salt` FROM `users` LIMIT 50 (0.001 s) Edit
```

Modify	email	username	salt	pwd_hash
edit	i221134@nu.edu.pk	AhmadAqeel	54748ABFFBF0ECE95C9D6F4CFD6039D4	9327898ab6bc3ccb8eff845183c88ab58637624 4c8dc61ba8c64c27f6a967ef6
edit	temp@gmail.com	tempName	02E9BD9DD0D6A3DF94A1FDE8D451FDE2	b7a2c8171874fca294c86eba191f985c541c7515f4b898d15b1828c13bfd9308
edit	temp123@gmail.com	temp-9090	CBA4F8C5F2E1D3310A36EC349108ADD3	2880e8be516a283b4ea65883c3fc519b93d852e5b8523fe62d8ad7d628114baf
edit	temp8989@gmail.com	qwerty-123	E285A9A8925F3F5950C1948CDB894B91	f6cc2708b6e8752a8d8517415429d4d251755374792889 8b5478ae92d8d3689ec

Whole result: □ 4 rows   Modify: Save   Selected (0)   Export (4)

Import

8.6 Result

✓PASS: User credentials are stored securely with salted SHA-256 hashes. No plaintext passwords are stored.

9. Test Summary

Test Case	Security Property	Result	Evidence Figure
Wireshark Capture	Confidentiality	PASS	Figures 1-4
Invalid Certificate	Authenticity	PASS	Figure 5
Tamper Detection	Integrity	PASS	Figure 6
Replay Protection	Replay Protection	PASS	Figure 7
Non-Repudiation	Non-Repudiation	PASS	Figure 8
Database Verification	Secure Storage	PASS	Figures 9-10

10. Conclusion

All tests have been successfully completed and verified. The SecureChat system correctly implements all required security properties:

- • Confidentiality: Verified through Wireshark capture showing encrypted payloads only
- • Integrity: Verified through tamper detection test showing signature verification failure
- • Authenticity: Verified through invalid certificate rejection test
- • Non-Repudiation: Verified through offline transcript and receipt verification
- • Replay Protection: Verified through sequence number checking
- • Secure Storage: Verified through database inspection showing salted password hashes

All test evidence has been captured through screenshots and command outputs. The system meets all assignment requirements and demonstrates proper implementation of cryptographic protocols at the application layer.