RollNumber-FullName-TestReport-A02 (Test Report — content)

Title page

- Course / Assignment as above
- Purpose: Validate security properties, provide reproducible evidence (PCAPs, logs, transcripts, receipts).

Test environment

- Host: Kali Linux (local)
- Python: 3.x in virtualenv (cryptography, pycryptodome, pymysql)
- MariaDB (drop-in for MySQL) running local `securechat` DB
- Wireshark capturing on `lo` (loopback) interface
- Project dir: `~/securechat_starter`
- Certificates: `certs/ca.cert.pem`, `certs/server.example.cert.pem`, `certs/client.example.cert.pem`
- Important files produced during tests (examples from runs):
    - `transcripts/client_transcript_<sessionid>.log`
    - `transcripts/client_receipt_<sessionid>.json`
    - `transcripts/server_transcript_<sessionid>.log`
    - `transcripts/server_receipt_<sessionid>.json`

Test 1 — Normal operation (Happy path)

Objective: Show registration/login over encrypted channel and a normal chat.

Steps

Start server (Terminal 1) with:

```
python3 server.py --server-cert certs/server.example.cert.pem
--server-key certs/server.example.key.pem --ca-cert certs/ca.cert.pem
```

1.

Start client (Terminal 2) and register:

```
python3 client.py --mode register --email test@example.com --username
testuser --password mysecret
```

2. Expected: registration succeeds.

Start client again and login:

```
python3 client.py --mode login --email test@example.com --password
mysecret
```

3. Send messages `hiii` and `hellow`, then `/quit`.

Observed server output (copy from server console):

```
listening on 127.0.0.1 9000
client connected ('127.0.0.1', 38130)
received client hello, nonce len 16
client cert verified OK for subject: <Name(CN=client.example)>
derived aes key len 16
auth: login email=test@example.com
Connecting to DB: localhost chatuser securechat
auth phase result: ok - login successful
entering chat loop for ('127.0.0.1', 38130)
signature valid for seqno=1
```

```
[('127.0.0.1', 38130)] seq=1 ts=... msg: hiii
signature valid for seqno=2
[('127.0.0.1', 38130)] seq=2 ts=... msg: hellow
client disconnected ('127.0.0.1', 38130)
SessionReceipt written to transcripts/server_receipt_<sessionid>.json
```

Observed client output (copy from client console):

```
server cert verified OK
Auth result: {'status': 'ok', 'message': 'login successful'}
You can now type messages. Type /quit to exit.
> hiii
sent seq=1
> hellow
sent seq=2
> /quit
Client SessionReceipt written to
transcripts/client_receipt_<sessionid>.json
```

Evidence produced

- `transcripts/server_transcript_<sessionid>.log`
- `transcripts/server_receipt_<sessionid>.json`
- `transcripts/client_transcript_<sessionid>.log`
- `transcripts/client_receipt_<sessionid>.json`
- Capture: save PCAP during run: `normal_session.pcap` (use Wireshark or tshark).

Wireshark notes

- All application payloads are binary JSON; no plaintext credentials were visible on the wire. Use display filter `tcp.port==9000` on interface `lo` to inspect packets.

Test 2 — Invalid certificate rejection

Objective: Server rejects clients whose cert is not signed by the CA.

Test script: `test_invalid_cert_client.py` (uses `certs_bad/badclient.example.cert.pem`)

Steps

1. Generate a fake CA + bad client cert (already created in `certs_bad/`).
2. Run server as in Test 1.
3. Run `python3 test_invalid_cert_client.py`.

Observed server output (expected / recorded):

```
client connected ('127.0.0.1', 12345)
received client hello, nonce len 16
client cert verification FAILED: ...
```

Observed client output (expected / recorded):

```
Server response: {"error":"bad client cert"}
```

Evidence produced

- Server log lines (above) as screenshot.
- PCAP `invalid_cert.pcap` with the initial client hello visible but certificate not trusted.

Notes for screenshot

- Show server console with the `client cert verification FAILED` line.
- In Wireshark show the relevant packet (client hello) and note there is no plaintext credential data.

Test 3 — Replay attack detection

Objective: Demonstrate replay detection by resending the same message in the same session (same `seqno`).

Test script: `test_replay_attack.py`

Steps

1. Start server.
2. Run `python3 test_replay_attack.py`.

Observed server output (expected / recorded):

```
signature valid for seqno=1
[addr] seq=1 ts=... msg: Replay test message (seq=1)
REPLAY/OUT-OF-ORDER detected: got seqno=1, last_seqno=1
```

Evidence

- Console log snippet above (screenshot).
- PCAP `replay_attack.pcap` capturing two identical app payloads back-to-back for seq=1.

Test 4 — Tampering detection (signature failure)

Objective: Show that modifying ciphertext without re-signing causes signature verification to fail.

Test script: `test_tamper_attack.py` (modifies ciphertext then resends with same signature)

Steps

1. Start server.

2. Run `python3 test_tamper_attack.py`.

Observed server output (expected / recorded):

```
signature valid for seqno=1
[addr] seq=1 ts=... msg: Tamper test message (valid first)
signature verify FAILED: ...
```

Evidence

- Console showing `signature verify FAILED`.
- PCAP `tamper_attack.pcap`.

Non-repudiation verification

Tool: `verify_receipt.py`

Example command (server receipt verified using client cert):

```
python3 verify_receipt.py \
  --transcript transcripts/server_transcript_<sessionid>.log \
  --receipt    transcripts/server_receipt_<sessionid>.json \
  --cert       certs/client.example.cert.pem
```

Observed verifier output (valid case):

```
Transcript file: transcripts/server_transcript_<sessionid>.log
Receipt file   : transcripts/server_receipt_<sessionid>.json
Peer cert      : certs/client.example.cert.pem
Receipt digest : <digest-from-receipt>
Local digest   : <digest-recomputed>
[OK] Transcript hash matches receipt.
[OK] Signature on transcript hash is valid.
[SUCCESS] Receipt is valid for this transcript and certificate.
```

Tamper demo (expected)

If transcript file is modified:

```
[FAIL] Transcript hash mismatch! Transcript has been modified or
receipt is not for this file.
```

-

If receipt signature is modified:

```
[OK] Transcript hash matches receipt.
[FAIL] Signature verification failed: ...
```

-

Test artifacts & locations (attach to submit)

- `transcripts/*` (client/server transcript logs and receipts)
- PCAPs: `normal_session.pcap`, `invalid_cert.pcap`, `replay_attack.pcap`, `tamper_attack.pcap`
- Console log screenshots: server console (happy path), server console (invalid cert), server console (replay), server console (tamper)
- `securechat_schema_dump.sql` (MySQL schema dump)
- `README.md`, `.env.example`, code files & scripts

Reproduction checklist (short instructions to reproduce each test)

1. Ensure `mariadb` is running and the `securechat` DB and `chatuser` exist.

2. `source venv/bin/activate` and set env vars: `export DB_HOST=localhost DB_USER=chatuser DB_PASS='StrongPassword123!' DB_NAME=securechat`.

3. Start server (Terminal 1).

For each test, run the corresponding client or test script in Terminal 2 while capturing packets on `lo` with Wireshark or tshark:

```
sudo tshark -i lo -f "tcp port 9000" -w normal_session.pcap
```

4. Then run the script. Save PCAP and take screenshots of server console and Wireshark view.

5. Verify receipts with `verify_receipt.py` as shown above.

Conclusion

All required security properties (CIANR) have been implemented and tested with reproducible scripts. The evidence produced (PCAPs, transcripts, receipts, console logs) demonstrate that:

- Credentials are never sent in plaintext.
- Invalid certificates are rejected.
- Tampering causes signature verification failure.
- Replayed messages are detected and rejected.
- Both sides can produce verifiable SessionReceipts proving the session contents (and edits break verification).