M Abdurrahman Khan

22i-1148 H

TESTS

Running Tests

All test scripts are located in the `tests/` directory.

Test 1: Tamper Test (SIG_FAIL)

Tests that message tampering is detected via signature verification failure.

Terminal 1:

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

Terminal 2:

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

When prompted:

- Enter your registered email
- Enter your password

Expected Output:

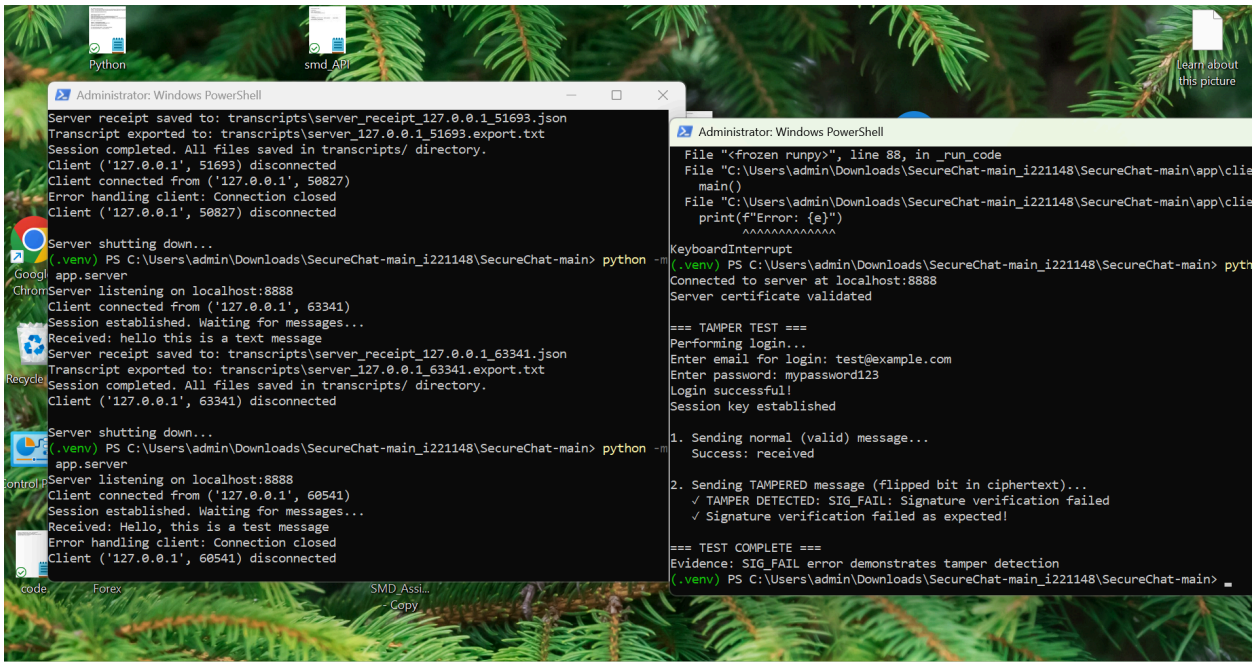
```
=== TAMPER TEST ===
Performing login...
Login successful!
Session key established

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!
```

Evidence: Screenshot showing `SIG_FAIL` error.



Test 2: Replay Test (REPLAY)

Tests that replay attacks are prevented via sequence number checking.

Terminal 1:

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

Terminal 2:

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

When prompted:

- Enter your registered email
- Enter your password

Expected Output:

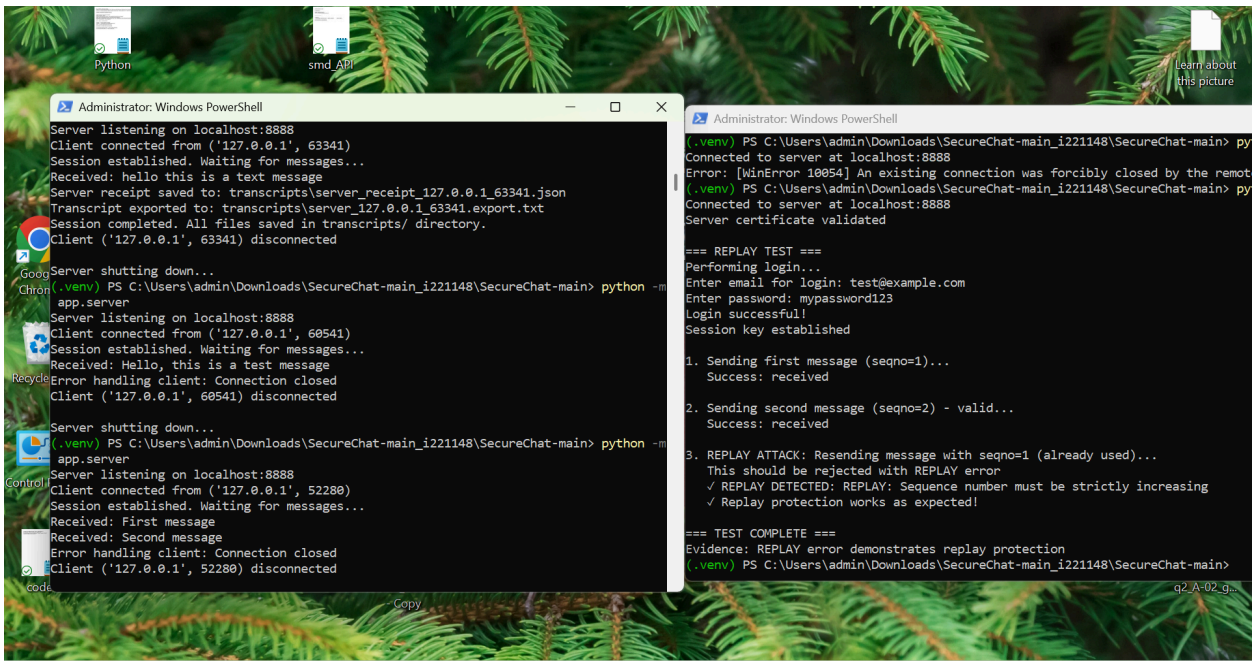
```
=== REPLAY TEST ===
Performing login...
Login successful!
Session key established

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)...
   ✓ REPLAY DETECTED: REPLAY: Sequence number 1 already seen
   ✓ Replay protection works as expected!
```

Evidence: Screenshot showing `REPLAY` error.



Test 3: Invalid Certificate Test (BAD_CERT)

Tests that invalid/self-signed certificates are rejected.

Step 1: Generate Invalid Certificate

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

Step 2: Backup Valid Certificates

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

Step 3: Replace with Invalid Certificate

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

Step 4: Start Server

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

Step 5: Try to Connect

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

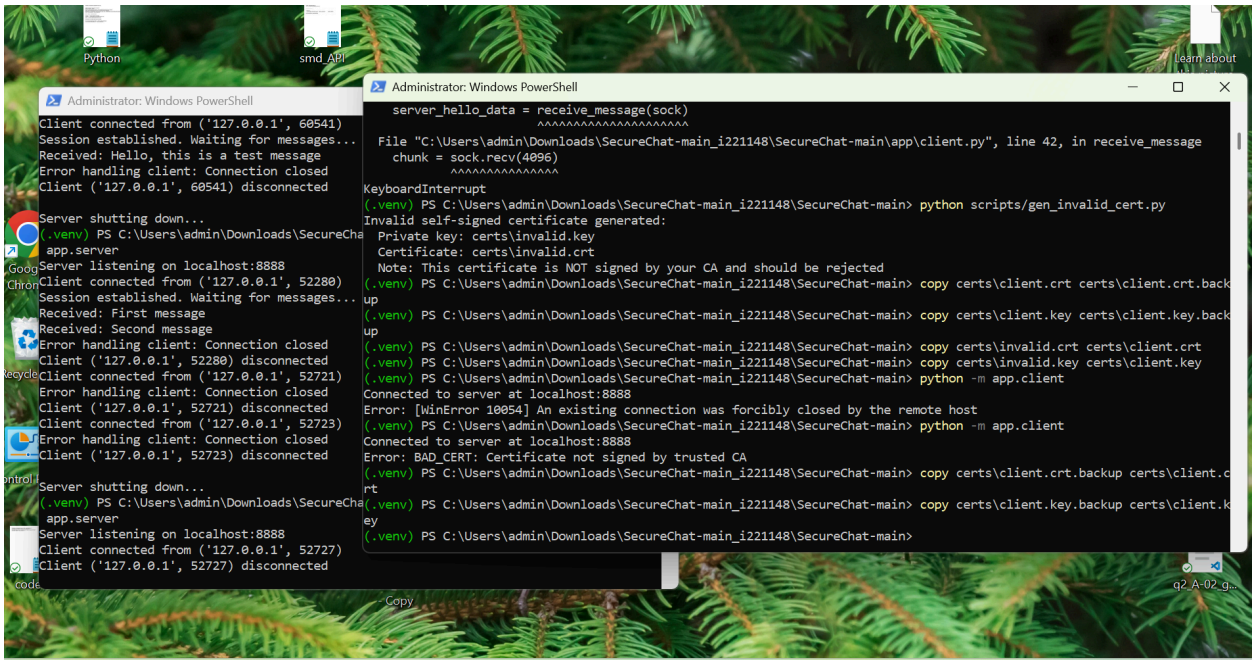
Expected Output:

```
Connected to server at localhost:8888
Error: BAD_CERT: Self-signed certificate rejected
```

Step 6: Restore Valid Certificates

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

Evidence: Screenshot showing `BAD_CERT` error.



Test 4: Non-Repudiation Verification

Tests offline verification of transcript and SessionReceipt.

Step 1: Run a Chat Session

Terminal 1:

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

Terminal 2:

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

- Login with your credentials
- Send 2-3 messages
- Type `quit` to end session

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
```

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

Expected Output:

```
======================================================================
TRANSCRIPT & SESSION RECEIPT VERIFICATION
======================================================================

1. Loading certificate: certs/client.crt
2. Validating certificate against CA: certs/ca.crt
   ✓ Certificate is valid and trusted

3. Loading transcript: transcripts/client_localhost_8888.export.txt
   Found 3 message entries

4. Verifying message signatures:
   Message 1: ✓ Signature valid
   Message 2: ✓ Signature valid
   Message 3: ✓ Signature valid

   ✓ All message signatures are valid

5. Computing transcript hash:
   Computed hash: abc123def456...

6. Loading SessionReceipt:
   Auto-detected receipt file: transcripts/client_receipt_localhost_8888.json
   Peer: client
   First seq: 1
   Last seq: 3
   Transcript hash: abc123def456...

7. Verifying receipt hash matches transcript:
   ✓ Receipt hash matches computed transcript hash

8. Verifying receipt signature:
   ✓ Receipt signature is valid

======================================================================
VERIFICATION RESULT: ✓ ALL CHECKS PASSED
======================================================================
```

Step 3: Test Tampering Detection

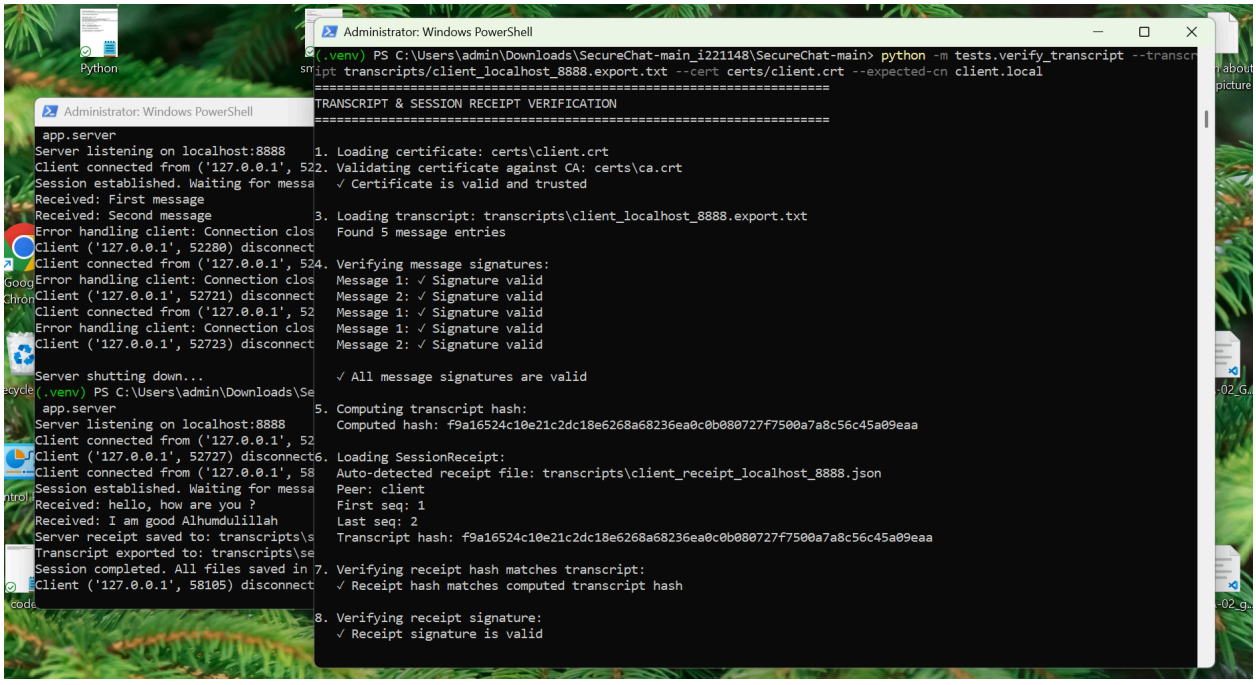
Edit the transcript file (change any character), then run verification again:

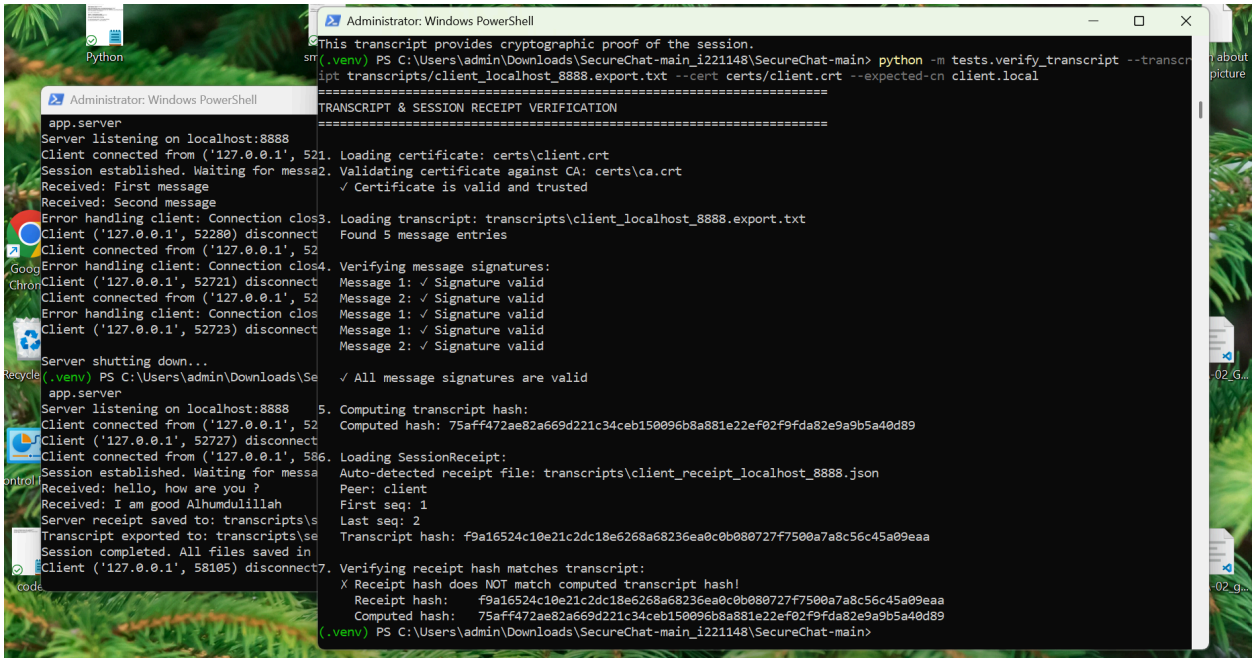
```
# Edit transcript file (add/remove a character)
# Then verify again
python -m tests.verify_transcript \
  --transcript transcripts/client_localhost_8888.export.txt \
  --cert certs/client.crt \
  --expected-cn client.local
```

Expected Output:

```
7. Verifying receipt hash matches transcript:
   ✗ Receipt hash does NOT match computed transcript hash!
     Receipt hash:    abc123def456...
     Computed hash:   xyz789ghi012...
```

Evidence: Screenshots showing successful verification and tampering detection.





Test 5: Wireshark Capture

Captures network traffic to verify encrypted payloads (no plaintext).

Step 1: Start Wireshark

1. Open Wireshark
2. Select loopback interface (`Loopback: lo0` on Mac, `Npcap Loopback Adapter` on Windows)
3. Start capture

Step 2: Run Chat Session

Terminal 1:

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

Terminal 2:

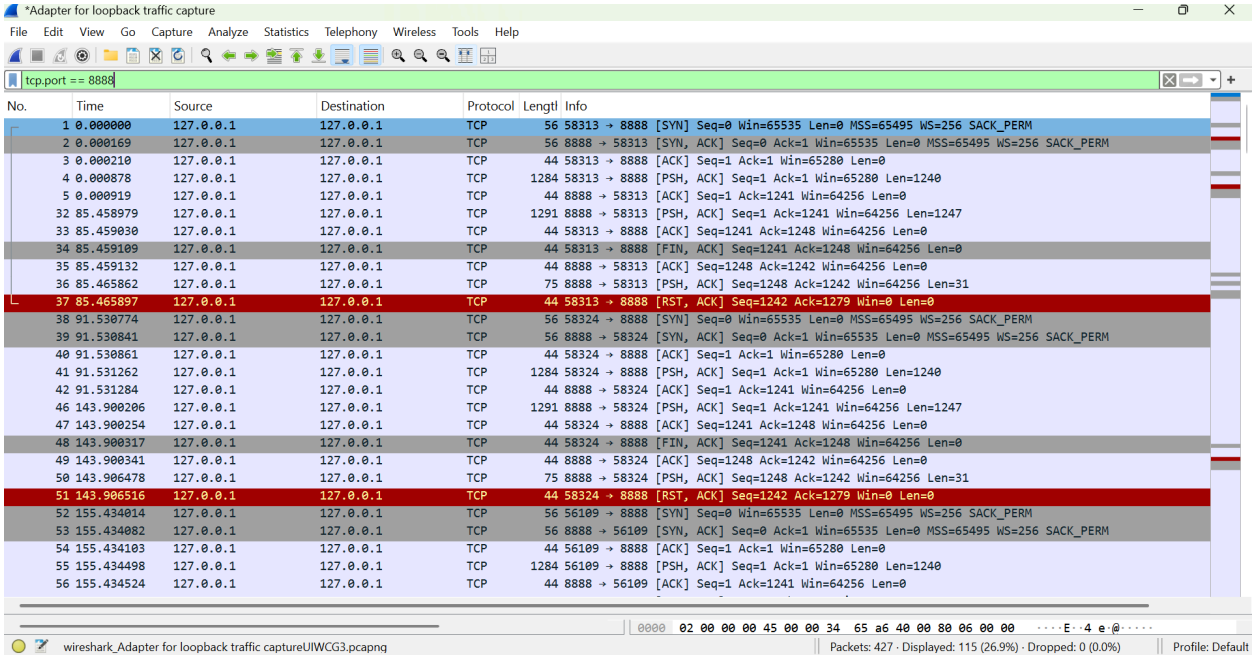
```
python -m app.client
```

- Login and send messages
- Type `quit` to end

Step 3: Stop Capture and Analyze

1. Stop Wireshark capture
2. Apply filter: `tcp.port == 8888`
3. Look for TCP packets with data
4. Expand packet → TCP → Data
5. Verify payload is encrypted (base64 encoded, no readable plaintext)

Evidence: Screenshot showing encrypted payload in Wireshark.



Problem: `BAD_CERT: Certificate not signed by trusted CA`

Solution:

1. Regenerate certificates:

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

2. `python scripts/gen_cert.py --cn client.local --out certs/client`