i220943_AhmedHannan-A02 (Main Report — content)

Title page

- Course: Information Security — Assignment 2
- Project: Secure Chat (Application-layer PKI, DH, AES, RSA signatures, Non-Repudiation)
- Student: `RollNumber — FullName`
- Date: `<fill date of submission>`
- Files submitted: `securechat_starter.zip` (or your repo zip), `RollNumber-FullName-Report-A02.docx`, `RollNumber-FullName-TestReport-A02.docx`, MySQL dump, `.env.example` etc. (see Submission Checklist section)

Executive summary

This project implements a secure client–server chat demonstrating Confidentiality, Integrity, Authenticity and Non-Repudiation (CIANR) per the assignment. It uses an application-layer PKI (root CA, X.509 certs), classical Diffie–Hellman for ephemeral key agreement, AES-128 (CBC + PKCS#7) for data confidentiality, SHA-256 for hashing, and per-message RSA signatures for integrity/authenticity. The system includes registration/login over an ephemeral AES session, transcript logging, and signed SessionReceipts for offline verification.

Design & Architecture

- **Transport:** Plain TCP (no TLS). The entire security stack is implemented at the application layer as required.

- **PKI:** `scripts/gen_ca.py` (root CA) and `scripts/gen_cert.py` (issue server/client certs). CA cert stored at `certs/ca.cert.pem`. Private keys are kept locally (`certs/*.key.pem`) and **must not** be committed.
- **Auth + DB:** Registration and Login are done inside an ephemeral AES session; credentials never cross the wire in plaintext. Users stored in MariaDB (`securechat` database, `users` table) as `salt VARBINARY(16)` + `pwd_hash CHAR(64)` where `pwd_hash = hex(SHA256(salt || password))`.
- **Session crypto:** Classical DH parameters generated at connection, client and server publish DH values, shared secret `Ks` derived, session AES key `K = Trunc16(SHA256(Ks))` (first 16 bytes).
- **Message format:** JSON with `{ "type":"msg", "seqno": n, "ts": unix_ms, "ct": base64, "sig": base64 }`. Each message `ct` is AES-CBC(iv||ciphertext) with PKCS#7; `sig` is RSA(PKCS1v15,SHA256) over `SHA256(seqno || ts || ct)`.
- **Transcript & Receipt:** Both sides append messages to transcript lines `seqno|ts|ct_b64|sig_b64|peer_cert_fingerprint`. On session close, compute `SHA256(transcript_bytes)` and sign it with local RSA private key to create a SessionReceipt JSON.

Implementation summary & files

- `scripts/gen_ca.py` — generate root CA (PEM key + cert).
- `scripts/gen_cert.py` — issue end-entity certs (server/client).
- `server.py` — server implementation (PKI validation, DH, auth phase, chat loop, transcript & receipt creation).
- `client.py` — client implementation (cert-send, verify, DH, register/login, interactive chat, transcript & receipt creation).
- `db/db.py` — MySQL (MariaDB) helper and helpers `create_user`, `get_user_by_email`.

- `auth.py` — `generate_salt`, `hash_password`, `verify_password`.
- `verify_receipt.py` — offline receipt verifier.
- Test scripts: `test_invalid_cert_client.py`, `test_replay_attack.py`, `test_tamper_attack.py`.
- `transcripts/` — contains transcript log files and JSON receipts created during sessions.

(Include each of the above files in your GitHub/ZIP as required; do **not** commit `certs/*.key.pem`.)

How to run (commands — copy into README)

Prepare venv and deps

```
python3 -m venv venv
source venv/bin/activate
pip install cryptography pycryptodome pymysql
```

Generate CA and certs

```
python3 scripts/gen_ca.py --outdir certs --cn "SecureChat Root CA"

python3 scripts/gen_cert.py --ca-key certs/ca.key.pem --ca-cert
certs/ca.cert.pem \
  --cn server.example --outdir certs --type server

python3 scripts/gen_cert.py --ca-key certs/ca.key.pem --ca-cert
certs/ca.cert.pem \
  --cn client.example --outdir certs --type client
```

Start server (Terminal 1)

```
export DB_HOST=localhost DB_USER=chatuser DB_PASS='StrongPassword123!'
DB_NAME=securechat
```

```
source venv/bin/activate
python3 server.py --server-cert certs/server.example.cert.pem \
                  --server-key certs/server.example.key.pem \
                  --ca-cert certs/ca.cert.pem
```

Server console output (observed during testing — include in report):

```
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
```

Start client (Terminal 2)

```
export DB_HOST=localhost DB_USER=chatuser DB_PASS='StrongPassword123!'
DB_NAME=securechat
source venv/bin/activate
python3 client.py --mode login --email test@example.com --password
mysecret
```

Client console output (observed):

```
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
```

Security decisions & reasoning

- Used strong RSA key sizes (RSA 3072 for end-entity in scripts), CA key 4096.
- AES-128 used per spec (derived from truncated SHA-256). CBC chosen with random IV per message; PKCS7 padding applied.
- Signatures performed with RSA PKCS#1 v1.5 (PKCS1v15) per assignment.
- Password storage follows assignment: per-user random salt (≥16 bytes) and stored as hex SHA-256(salt || password).
- All authentication information is transported only inside the ephemeral AES session (after DH). The certificate exchange and verification occur before any credentials are accepted.

Files to submit (per assignment PDF)

IS_Assignment_2

- GitHub repo zip (your fork) — no private keys, include `.env.example`.
- MySQL schema dump (`securechat_schema_dump.sql`) and sample records.
- `RollNumber-FullName-Report-A02.docx` (this main report).
- `RollNumber-FullName-TestReport-A02.docx` (test report below).
- Any PCAPs and screenshots referenced in the Test Report.

- `scripts/gen_ca.py`, `scripts/gen_cert.py`, `server.py`, `client.py`, `db/db.py`, `auth.py`, `verify_receipt.py`, test scripts.

Appendix — Message formats & exact JSON used

- Hello exchange:

Client → Server:

```
{ "type":"hello", "client_cert":"...PEM... (base64)",
"nonce":"<base64>" }
```

  ○

Server → Client:

```
{ "type":"server_hello", "server_cert":"...PEM... (base64)",
"nonce":"<base64>" }
```

  ○

DH params:

```
{"type":"dh_params","p":"<decimal p>","g":"<decimal
g>","server_pub":"<base64>"}
```

-

Auth inside AES:

```
{ "type":"register" | "login", "email":"...", "username":"..."
(register only), "password":"<plaintext password inside AES>" }
```

-
- Encrypted message:

  ```
  { "type":"msg", "seqno": n, "ts": unix_ms, "ct":
  "<base64(iv||ciphertext)>", "sig":
  ```