ðŸ”’ Secure Chat System - Assignment #2

Course: Information Security (CS-3002, Fall 2025)
Institution: FAST-NUCES
Repository: https://github.com/maadilrehman/securechat-skeleton

ðŸ“‘ Table of Contents

ðŸŽ¯ Overview

A **console-based Secure Chat System** demonstrating practical cryptography by combining: - **AES-128** (block cipher for confidentiality) - **RSA with X.509 certificates** (authentication and digital signatures) - **Diffie-Hellman** (key agreement) - **SHA-256** (integrity)

This implementation achieves **CIANR**: **C**onfidentiality, **I**ntegrity, **A**uthenticity, and **N**on-**R**epudiation.

Threat Model

- **Adversary**: Passive eavesdropper, active MitM (replay/modify/inject), untrusted client
- **Goals**: No plaintext leakage, peer authenticity, tamper detection, replay prevention, non-repudiation via signed transcripts

âœ¨ Features

Core Security

- âœ… **PKI Infrastructure**: Self-signed CA, server & client certificates

- âœ… **Mutual Authentication**: Certificate validation with expiry and CN checks
- âœ… **Key Agreement**: Diffie-Hellman for ephemeral session keys
- âœ… **Encrypted Communication**: AES-128-ECB with PKCS#7 padding
- âœ… **Message Integrity**: SHA-256 digests with RSA signatures
- âœ… **Replay Protection**: Sequence numbers and timestamps
- âœ… **Non-Repudiation**: Append-only transcripts with signed receipts

Implementation

- âœ… **Dual-Phase Encryption**: Separate keys for control plane (auth) and data plane (chat)
- âœ… **MySQL Storage**: Salted password hashing (SHA-256)
- âœ… **Constant-Time Comparison**: Timing attack prevention
- âœ… **Graceful Error Handling**: BAD_CERT, SIG_FAIL, REPLAY errors

ðŸ—ï¸ Architecture

Protocol Phases

```
â”Œâ”€â”€â”€â
```

```
â”                                                  â”
â”  Phase 5: Encrypted Chat                        â”
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€ MSG â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¤
â”   {seqno, ts, ct, sig}                         â”
<â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€ MSG â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€
â”   [RSA sig over SHA-256(seqno||ts||ct)]        â”
â”   [Replay check, decrypt, verify]              â”
â”                                                  â”
â”  Phase 6: Non-Repudiation                       â”
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€ RECEIPT â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¤
â”   {peer, seqs, transcript_hash, sig}           â”
<â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€ RECEIPT â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€
â”   [Signed SHA-256 of full transcript]          â”
â”                                                  â”
```

Security Layers

Layer	Mechanism	Purpose
Authentication	X.509 Certificates	Verify peer identity
Key Agreement	Diffie-Hellman	Establish shared secrets
Confidentiality	AES-128-ECB	Encrypt message content
Integrity	SHA-256	Detect tampering
Authenticity	RSA Signatures	Prove sender identity
Replay Prevention	Sequence Numbers	Reject old messages
Non-Repudiation	Signed Transcripts	Provable evidence

ðŸš€ Installation

Prerequisites

- Python 3.8+
- MySQL 8.0+
- pip

Quick Setup

```
# Clone the repository
git clone https://github.com/maadilrehman/securechat-skeleton.git
cd securechat-skeleton

# Run automated setup
chmod +x setup.sh
./setup.sh

# Or manual setup:
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

Certificate Generation

```
# Activate virtual environment
source .venv/bin/activate

# Generate Root CA
python scripts/gen_ca.py

# Generate server certificate
python scripts/gen_cert.py --cn "securechat.server" --out certs/server --t

# Generate client certificate
python scripts/gen_cert.py --cn "securechat.client" --out certs/client --t

# Verify certificates
openssl x509 -in certs/ca-cert.pem -text -noout
openssl x509 -in certs/server-cert.pem -text -noout
openssl x509 -in certs/client-cert.pem -text -noout
```

Database Setup

```
# Option 1: Using provided script (interactive)
chmod +x scripts/setup_mysql.sh
./scripts/setup_mysql.sh

# Option 2: Manual MySQL commands
mysql -u root -p
CREATE DATABASE securechat CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_c
CREATE USER 'scuser'@'localhost' IDENTIFIED BY 'scpass123';
GRANT ALL PRIVILEGES ON securechat.* TO 'scuser'@'localhost';
FLUSH PRIVILEGES;
exit;

# Initialize tables
python -m app.storage.db --init
```

âš™ï¸ Configuration

Environment Variables (.env)

```
# Database Configuration
DB_HOST=localhost
DB_PORT=3306
DB_USER=scuser
DB_PASSWORD=scpass123
DB_NAME=securechat

# Server Configuration
SERVER_HOST=127.0.0.1
SERVER_PORT=5555
```

```
# Certificate Paths
CA_CERT=certs/ca-cert.pem
SERVER_CERT=certs/server-cert.pem
SERVER_KEY=certs/server-key.pem
CLIENT_CERT=certs/client-cert.pem
CLIENT_KEY=certs/client-key.pem
```

ðŸ’» Usage

Starting the Server

```
# Terminal 1
source .venv/bin/activate
python -m app.server

# Expected output:
# â•”â”€â”€â”€
```

```
2. Login to existing account
> 1

Email: alice@example.com
Username: alice
Password: ********
âœ“ Sent registration request
âœ“ Registration successful

â•”â•â• Phase 4: Session Key Establishment â•”â•â•â•
âœ“ Sent DH parameters
âœ“ Received DH response from server
âœ“ Session key established

â•”â•â• Phase 5: Encrypted Chat â•”â•â•â•
Chat session started. Type your messages below.
Type '/quit' to end session.

> Hello, this is a secure message!
You: Hello, this is a secure message!
Server: Message received securely!

> /quit

â•”â•â• Phase 6: Non-Repudiation â•”â•â•â•
âœ“ Session receipt sent to server
âœ“ Transcript saved: transcripts/client_a1b2c3d4_20251116_102745.transcri
âœ“ Transcript hash: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca4959
```

ðŸ§ª Testing

Certificate Tests

```
# Run certificate validation tests
python tests/test_certificates.py

# Expected output:
# â•”â•â• Test 1: Valid Certificates â•”â•â•â•
# âœ“ Server certificate is VALID
# âœ“ Client certificate is VALID
#
# â•”â•â• Test 2: Expired Certificate â•”â•â•â•
# âœ“ Expired certificate correctly rejected
#
# â•”â•â• Test 3: Self-Signed Certificate â•”â•â•â•
# âœ“ Self-signed certificate correctly rejected
#
# â•”â•â• Test 4: CN Mismatch â•”â•â•â•
# âœ“ CN mismatch correctly detected
```

Wireshark Capture

```
# Start Wireshark or tcpdump
sudo tcpdump -i lo -w securechat.pcap port 5555

# In another terminal, run server and client
# Then analyze the capture
wireshark securechat.pcap

# Display filter to use:
tcp.port == 5555
```

Expected: All payload data should be encrypted (base64-encoded ciphertext), no plaintext credentials visible.

Tampering Test

Modify `tests/test_tampering.py`:

```
# Flip a bit in ciphertext
original_ct = chat_msg['ct']
tampered_ct = flip_bit(original_ct)
chat_msg['ct'] = tampered_ct

# Send tampered message
# Expected: SIG_FAIL error
```

Replay Attack Test

```
# Save a valid message
saved_msg = chat_msg.copy()

# Send saved_msg again
# Expected: REPLAY error (seqno not strictly increasing)
```

Non-Repudiation Verification

```
# After chat session, verify transcript
python tests/verify_transcript.py transcripts/client_*.transcript

# Expected:
# âœ“ All message signatures valid
# âœ“ Receipt signature valid
# âœ“ Transcript hash matches receipt
```

ðŸ”’ Security Features

Confidentiality

- **AES-128-ECB**: All messages encrypted with session key

- **No Plaintext Transit**: Credentials encrypted during auth phase
- **Key Separation**: Different keys for control plane and data plane

Integrity

- **SHA-256 Digests**: Computed over seqno||timestamp||ciphertext
- **Tamper Detection**: Any modification breaks signature
- **Salted Password Hashing**: SHA-256(salt || password)

Authenticity

- **X.509 Certificates**: CA-signed certificates for both parties
- **RSA Signatures**: PKCS#1 v1.5 with SHA-256
- **Certificate Validation**: Chain, expiry, and CN checks

Non-Repudiation

- **Append-Only Transcripts**: Immutable log of all messages
- **Signed Receipts**: RSA signature over transcript hash
- **Offline Verifiable**: Third party can validate entire session

Anti-Replay

- **Sequence Numbers**: Strictly increasing per session
- **Timestamps**: Unix milliseconds for freshness
- **State Tracking**: Reject seqno â‰¤ last_seen

ðŸ“ Project Structure

```
securechat-skeleton/
â”œâ”€â”€ app/
â”‚   â”œâ”€â”€ client.py                # Client application
â”‚   â”œâ”€â”€ server.py                # Server application
â”‚   â”œâ”€â”€ crypto/
â”‚   â”‚   â”œâ”€â”€ aes.py               # AES-128-ECB encryption
â”‚   â”‚   â”œâ”€â”€ dh.py                # Diffie-Hellman key exchange
â”‚   â”‚   â”œâ”€â”€ pki.py               # Certificate validation
â”‚   â”‚   â””â”€â”€ sign.py              # RSA signing/verification
â”‚   â”œâ”€â”€ common/
â”‚   â”‚   â”œâ”€â”€ protocol.py          # Pydantic message models
â”‚   â”‚   â””â”€â”€ utils.py             # Utility functions
â”‚   â””â”€â”€ storage/
â”‚       â”œâ”€â”€ db.py               # MySQL database handler
â”‚       â””â”€â”€ transcript.py       # Transcript management
â”œâ”€â”€ scripts/
â”‚   â”œâ”€â”€ gen_ca.py               # Generate Root CA
â”‚   â”œâ”€â”€ gen_cert.py             # Generate certificates
â”‚   â””â”€â”€ setup_mysql.sh          # Database setup script
â”œâ”€â”€ tests/
â”‚   â”œâ”€â”€ test_certificates.py    # Certificate tests
```

```
â”‚   â””â”€â”€ manual/
â”‚       â””â”€â”€ NOTES.md            # Manual testing notes
â”œâ”€â”€ certs/                    # Certificate storage (gitignored)
â”œâ”€â”€ transcripts/              # Session transcripts (gitignored)
â”œâ”€â”€ .env                      # Environment configuration (gitign
â”œâ”€â”€ .env.example              # Example configuration
â”œâ”€â”€ .gitignore                # Git ignore rules
â”œâ”€â”€ requirements.txt          # Python dependencies
â”œâ”€â”€ setup.sh                  # Automated setup script
â””â”€â”€ README.md                 # This file
```

ðŸ› Troubleshooting

Common Issues

Certificate Errors

```
Error: BAD_CERT: Certificate expired
Solution: Regenerate certificates with longer validity
```

Database Connection Failed

```
Error: Can't connect to MySQL server
Solution:
1. Check MySQL is running: sudo systemctl status mysql
2. Verify credentials in .env match database user
3. Ensure database exists: mysql -u root -p -e "SHOW DATABASES;"
```

Import Errors

```
Error: ModuleNotFoundError: No module named 'app'
Solution: Run from project root: python -m app.server
```

Port Already in Use

```
Error: Address already in use
Solution:
1. Find process: lsof -i :5555
2. Kill it: kill -9 <PID>
3. Or change SERVER_PORT in .env
```

Debugging

Enable verbose logging:

```
# Add to .env
DEBUG=True
LOG_LEVEL=DEBUG
```

Check logs:

```
# Server logs
tail -f logs/server.log

# Client logs
tail -f logs/client.log
```

ðŸ“š References

- SEED Security Labs - PKI
- RFC 3526 - DH Parameters
- RFC 5280 - X.509 Certificates
- Cryptography Python Library

ðŸ‘¨â€ðŸ’» Development

Commit Guidelines

This project follows semantic commit messages:

```
feat: Add certificate validation
fix: Correct replay detection logic
docs: Update README with usage examples
test: Add tampering test case
```

Minimum 10 meaningful commits required for submission.

Code Style

- Follow PEP 8
- Type hints encouraged
- Docstrings for all functions
- No secrets in version control

ðŸ“ License

This project is for educational purposes as part of FAST-NUCES Information Security course.

ðŸ‘¥ Authors

- **Course Instructor**: [FAST-NUCES Faculty]
- **Student Implementation**: [Your Name] - [Roll Number]

ðŸ™ Acknowledgments

- FAST-NUCES Information Security Course
- Python Cryptography Library maintainers
- SEED Security Labs

Note: This implementation is for educational purposes. For production use, employ battle-tested libraries like TLS/SSL instead of custom application-layer crypto.