Secure Chat Program Documentation

Jawad Chowdhury

CSC 380

1 Assumptions

1.1 Key Distribution Assumptions

I assume that two users have a secure method for initial key distribution. So by some outside method the users that want to talk to each other already have each other's public keys, and can verify each other's identities.

More specifically, my program displays a hash (fingerprint) of each user's long-term public key during connection. Users need to verify these fingerprints through some secure channel, so for example they could call each other to confirm they're actually talking to the right person and not some malicious third party.

Given this assumption, after identity verification, we use 3DH for communication where:

- Each party generates a long-term key pair that persists across sessions for identity
- Each party generates ephemeral key pairs for each session (PFS)
- Key authenticity is verified through public key hashes displayed during handshake
- Users manually verify these hashes match to prevent man-in-the-middle attacks

1.2 Adversary Assumptions

We assume adversaries have the following capabilities:

- **Network Access:** Can intercept, modify, inject, and replay network traffic
- **Computing Power:** Have normal computing resources but cannot break modern encryption like AES-256 or HMAC-SHA512
- **System Access:** Cannot access files or memory on your computer while the program is running
- **Physical Access:** Cannot physically access your computer, steal your hard drive, install hardware keyloggers, or perform attacks that require being physically present at your location

1.3 Other Assumptions

It seems basic but I should note that I assume the OS, hardware, network, crypto libraries (OpenSSL) work as expected and haven't been tampered with.

2 Claims

2.1 When Both Users Are Honest

2.1.1 Message Confidentiality

All messages are encrypted using AES-256 in Counter (CTR) mode. Each message gets a random 16-byte initialization vector (IV) so even if the same message gets sent twice, the encrypted output is completely different. This prevents patterns in the encrypted traffic.

The message format is: [16-byte IV][Encrypted content]

Adversaries can't decrypt messages without the 256-bit encryption key derived from the 3DH key exchange.

2.1.2 Message Integrity and Authentication

Every message includes a Hash-based Message Authentication Code (HMAC) using SHA-512 with a 64-byte key. This serves two purposes: detecting if anyone modified the message during transmission, and proving the message actually came from someone with the correct key.

My implementation calculates a 64-byte HMAC tag for each encrypted message and verifies it before decrypting. If the HMAC doesn't match, the message is rejected.

2.1.3 Mutual Authentication

The 3DH protocol provides mutual authentication by having both users exchange their long-term public keys (identity) and ephemeral public keys (session). The session key is derived using all four keys combined, so both parties must possess the correct private keys to successfully communicate. Like I said earlier in assumptions, users verify key fingerprints to ensure they're talking to the right person.

2.1.4 Replay Protection

My implementation includes replay protection through message counters. Each outgoing message gets a sequence number that increases by 1, and this counter is included in the plaintext before encryption. When receiving messages, the program checks that the counter is higher than the last one received. If someone tries to replay an old message, it will have an old counter and because counter from last one is lower than last one it'll get rejected.

2.1.5 Perfect Forward Secrecy

Each chat session generates fresh ephemeral keys that are used along with long-term keys to derive the session key. Even if someone steals a user's long-term private key later, they can't decrypt past conversations because:

- The actual encryption key is derived from both long-term AND ephemeral keys
- Ephemeral keys are deleted after the session ends
- Without the ephemeral keys, past session keys cannot be reconstructed

This means compromise of long-term keys does not compromise past session confidentiality.

2.2 When One Party Is Malicious

If one party runs a modified version of the chat program, several issues arise:

2.2.1 What a Malicious User CAN Do

A malicious party can:

- Log all received messages in plaintext (they receive messages legitimately)
- Retain session keys to decrypt intercepted messages from the same session
- Violate their own security by accepting replayed messages or skipping verification (not sure of the practical use of this though)
- Potentially send bad messages to try to cause crashes
- Misuse any information they legitimately receive

2.2.2 What a Malicious User CAN'T Do

However, a malicious party **cannot**:

- Decrypt messages intended for the honest party from previous sessions (PFS)
- Read conversations between other people (they can only decrypt messages sent to them)
- Forge messages that appear to come from the honest party without detection. This is because to send valid message, you need right HMAC. To make that HMAC, you need MAC key (64 bytes). To get MAC key you need session key. To get session key you have to perform 3DH using both parties' private keys, and the bad party only has their private keys not the honest party's private keys, so without that they can't recreate session key, and consequently HMAC. So any forged message with bad HMAC gets rejected.

2.2.3 Limitations

If someone is running malicious software, that software can misuse any information it legitimately receives. The security guarantee is that network attackers can't break the cryptography, but social engineering or endpoint compromise is outside the scope of network security protocols.

3 Conclusion

My secure chat implementation provides strong security guarantees including confidentiality, integrity, authentication, and forward secrecy when both parties act honestly. The 3DH protocol with AES-256-CTR encryption and HMAC-SHA512 authentication offers sufficient protection against adversaries.

The main limitation is that security depends on both endpoints being trustworthy and users properly verifying key fingerprints when connecting. A malicious communicating party can compromise the security of their own session but can't affect the security of other communications.