Drexel University

Project Part 2: Protocol Design

Secure Real-time Chat Protocol (SRCP)

Pathik Patel (Student ID: 14545203)

CS544: Computer Networks

Professor Brian Mitchell

5 June 2023

Section 1 – Service Description

1.1 Overview

The Secure Real-time Chat Protocol (SRCP) is a stateful, application-layer protocol designed to enable secure, real-time text messaging between two or more clients, as well as group chat functionality. SRCP is primarily an application-layer protocol that supports end-to-end encryption, user authentication, and message integrity checks to ensure secure and private communication. The protocol is designed to work over the TCP (Transmission Control Protocol) with TLS encryption using a client-server model.

1.2 Features

This protocol includes below features.

- Real-time communication: SRCP enables users to send and receive messages in real-time, providing a seamless chat experience.
- End-to-end encryption: SRCP uses public-key cryptography to ensure that the chat messages are encrypted and can only be decrypted by the intended recipient(s).
- User authentication: Users are required to authenticate themselves before joining a chat session to prevent unauthorized access.
- Message integrity: SRCP incorporates message integrity checks to ensure that the messages are not tampered with during transmission.
- Group chat: The protocol supports group chat, allowing multiple users to communicate in a shared environment.
- Extensibility: SRCP is designed to accommodate future improvements and extensions, such as support for file sharing or multimedia messages.

1.3 Service Architecture

The SRCP architecture consists of clients and a server. Clients are user devices running the chat application, while the server facilitates communication between clients and manages user authentication and chat sessions.

1.3.1 Client

A client is a user device running the SRCP chat application. It is responsible for:

- Establishing a connection to the server.
- Authenticating the user.
- Encrypting and decrypting messages.
- Sending and receiving messages.
- Displaying the chat messages to the user.
- Managing the user's public and private keys for encryption.

1.3.2 Server

The server plays a central role in managing the chat service. It is responsible for:

- Listening for client connections.
- Authenticating users.
- Managing chat sessions.
- Routing encrypted messages between clients.
- Storing public keys of users for encryption purposes.

1.4 Chat Session Lifecycle

1.4.1 Connection Establishment

A client establishes a connection to the server using a secure transport layer protocol, such as TCP, with TLS encryption.

1.4.2 User Authentication

Once the secure connection is established, the client sends an authentication request to the server, providing the user's credentials (e.g., username and password). The server verifies the credentials and, if valid, sends a confirmation to the client, allowing the user to join the chat session.

1.4.3 Public Key Exchange

After authentication, the client and server exchange public keys to enable end-to-end encryption of chat messages. This exchange happens over the secure TLS connection. Client sends its own public key to server and server sends public keys of other clients participated in chat to the new client.

1.4.4 Chat Session

During a chat session, clients can send and receive encrypted messages. The server is responsible for routing the messages between clients while ensuring that only authenticated users can participate in the chat session.

1.4.5 Session Termination

A chat session is terminated when a client disconnects from the server or when the server terminates the session (e.g., due to inactivity or server maintenance).

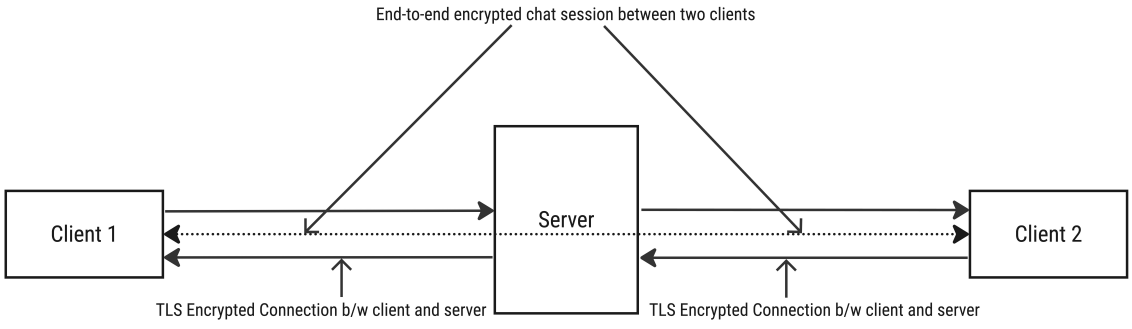

Figure 1 - End-to-end encrypted chat session

Section 2 – Message Definition (PDUs)

2.1 Overview

This section defines the Protocol Data Units (PDUs) used by the Secure Real-time Chat Protocol (SRCP). The PDUs are the building blocks of the communication between the client and the server. Protocol Data Units (PDUs) in the Secure Real-time Chat Protocol (SRCP) consist of a header and a payload. The header contains metadata about the message, such as its version, type, length and sequence, while the payload carries the actual content of the message.

All multi-byte integers are sent in big-endian (network) byte order. The length field in the PDU header indicates the length of the payload in bytes.

Each PDU payload structure corresponds to a specific message type. The AuthRequest and AuthResponse payloads are used for user authentication. The KeyExchange payload is used for exchanging public keys for encryption. The Message payload is used for sending chat messages. The Disconnect payload is used for initiating a disconnection.

When constructing a PDU, the appropriate payload structure is serialized to bytes and appended to the PDU header. When parsing a PDU, the payload bytes are deserialized to the appropriate structure based on the message type in the PDU header.

Note that the payloads should be encrypted (except for AUTH_REQUEST and AUTH_RESPONSE) using the previously exchanged public keys, ensuring end-to-end encryption of chat messages.

2.2 PDU Header

The PDU header has the following structure:

```
typedef struct {
    uint8_t Version;   // Protocol version
    uint8_t Type;      // Message type
    uint16_t Length;   // Length of the payload
    uint32_t Sequence; // Sequence number
} Header;
```

- version (1 byte): The version of the SRCP protocol. The initial version is 1.
- type (1 byte): The type of the message, as defined in the Message Types section below.
- length (2 bytes): The length of the payload in bytes.
- sequence (4 bytes): A monotonically increasing sequence number, used to detect message loss or duplication.

2.3 Message Types

The following message types are defined for the SRCP protocol:

- 0x01: AUTH_REQUEST – Sent by the client to request authentication.
- 0x02: AUTH_RESPONSE – Sent by the server to indicate the authentication result.
- 0x03: KEY_EXCHANGE – Used by the client and server to exchange public keys.
- 0x04: MESSAGE – Represents an encrypted chat message.
- 0x05: MESSAGE_ACK – Sent by the recipient to acknowledge receipt of a chat message.
- 0x06: DISCONNECT – Sent by the client to indicate the termination of a chat session.

2.4 PDU Payloads

The following subsections define the structure of the payloads for each message type.

2.4.1 AUTH_REQUEST Payload

```
typedef struct {
    char Username[32]; // Null-terminated username (max 31 characters)
    char Password[32]; // Null-terminated password (max 31 characters)
} AuthRequestPayload;
```

This is used for authentication requests. It carries a username and a password, both up to 31 characters long (plus a null terminator). The server typically checks these credentials to authenticate the user.

2.4.2 AUTH_RESPONSE Payload

```
typedef struct {
    uint8_t Status; // Authentication status (0: success, 1: failure)
} AuthResponsePayload;
```

The server uses this payload to communicate the result of an authentication attempt. The Status field indicates whether the authentication was successful (0) or failed (1).

2.4.3 PUBLIC_KEY Payload

```
typedef struct {
    char Username[32]; // Null-terminated username (max 31 characters)
    unsigned char Key[512]; // Public key (256 bytes - RSA-2048)
} PublicKeyPayload;
```

This payload carries a public key associated with a username. This could be used for public key infrastructure-based authentication or for setting up encrypted communication. The public key is 256 bytes

long, suitable for RSA-2048.

2.4.4 MESSAGE Payload

```
typedef struct {
    uint32_t Timestamp;    // Message timestamp (Unix epoch time)
    uint8_t Sender[32];    // Length of the sender's username
    uint8_t Recipient[32]; // Length of the recipient's username
    uint16_t TextLen;      // Length of the encrypted text
    uint8_t Data[2048];        // Encrypted text message
} MessagePayload;
```

This is used for sending messages. It contains a timestamp in Unix epoch time format, the lengths of the sender's username, the recipient's username, and the encrypted text. The Data field is variable-length and contains the sender's username followed by the encrypted text.

2.4.5 MESSAGE_ACK Payload

```
typedef struct {
    uint32_t Sequence; // Sequence number of the acknowledged message
} MessageAckPayload;
```

This payload acknowledges the receipt of a message. The Sequence field contains the sequence number of the message being acknowledged.

2.4.6 DISCONNECT Payload

```
typedef struct {
    uint8_t Reason; // Reason for disconnection (0: user request, 1:
server request)
} DisconnectPayload;
```

This is used to indicate a request for disconnection. The Reason field specifies whether the request came from the user (0) or the server (1).

Section 3 – Deterministic Finite Automata (DFA)

The Deterministic Finite Automata (DFA) for the Secure Real-time Chat Protocol (SRCP) represents the various states and transitions of the protocol during a chat session. This DFA is crucial to understand the flow of the protocol and how it responds to different message types.

The DFA for SRCP is as follows:

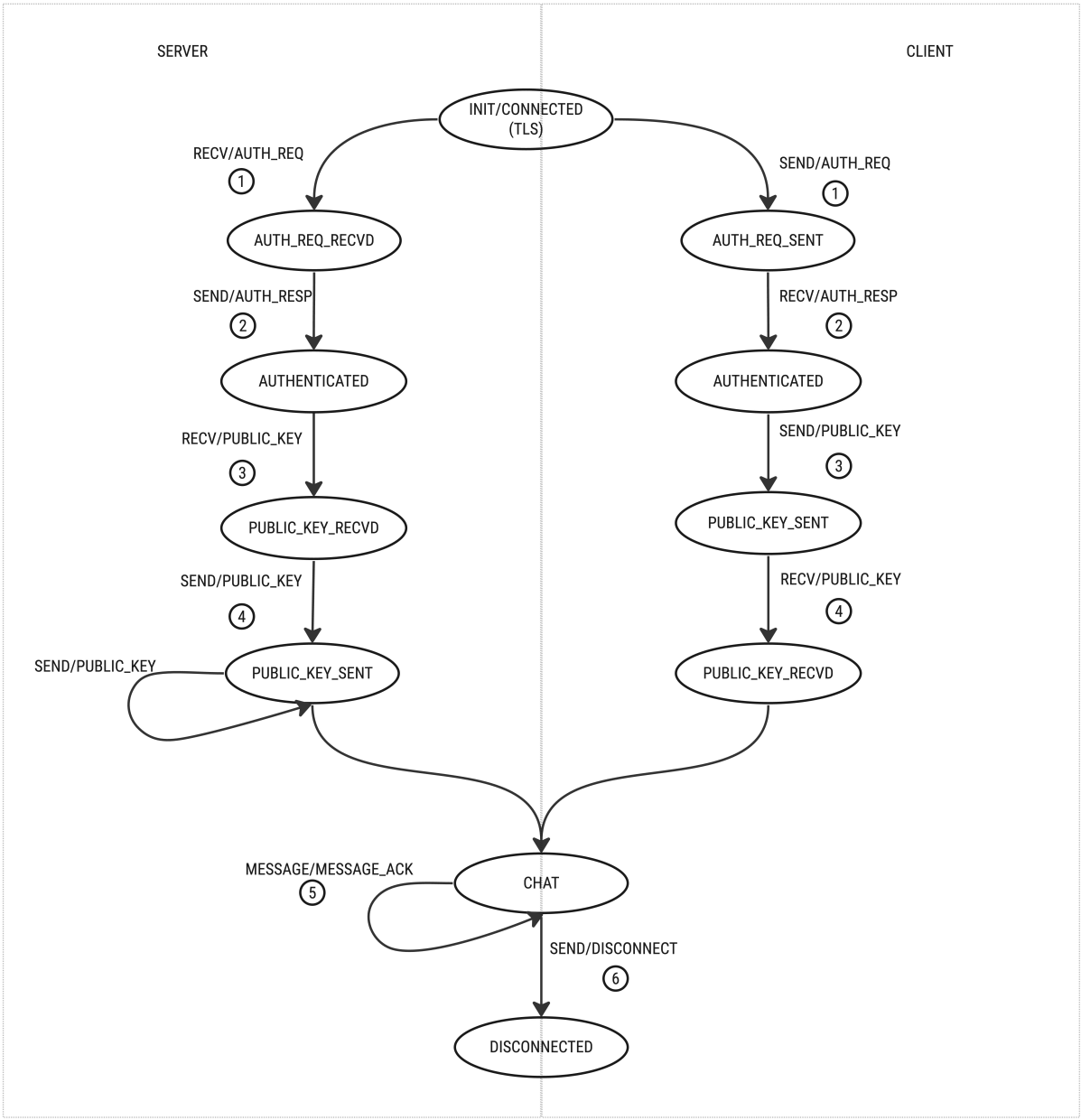

Figure 2 - Deterministic Finite Automata (DFA)

The protocol states are:

1. INIT: The initial state, where the client establishes a connection to the server.
2. AUTH_REQ_SEND: The client sends an authentication request to the server.
3. AUTH_REQ_RECVD: The server receives an authentication request from the client.
4. AUTHENTICATED: The client is authenticated, and public key exchange can take place.
5. PUBLIC_KEY_SENT: The client sends its Public key to server. For server, Server sends public key of other chat participants to client.

6. PUBLIC_KEY_RECVD: The server has received public key from client. for client, client has received other participant's public key from the server.
7. CHAT: The main state where chat messages are exchanged between clients.
8. DISCONNECTING: The client or server initiates a disconnection.
9. TERMINATED: The chat session has been terminated.

The transitions between the states occur based on the message types:

1. INIT to AUTH_REQ_SEND: This transition occurs when the client decides to begin the authentication process. The client sends an authentication request message to the server.

2. AUTH_REQ_SEND to AUTH_REQ_RECVD: This transition occurs when the server successfully receives the authentication request message sent by the client.

3. AUTH_REQ_RECVD to AUTHENTICATED: This transition happens when the server verifies the authenticity of the client. The server sends a confirmation message to the client indicating successful authentication.

4. AUTHENTICATED to PUBLIC_KEY_SENT: After being authenticated, the client sends its public key to the server for secure communication. Similarly, the server sends the public keys of other chat participants to the client.

5. PUBLIC_KEY_SENT to PUBLIC_KEY_RECVD: This transition happens when the server receives the client's public key or when the client receives other participants' public keys from the server.

6. PUBLIC_KEY_RECVD to CHAT: After receiving the public keys, the client and server transition to the main chat state, where messages can be securely exchanged.

7. CHAT to DISCONNECTING: This transition occurs when either the client or the server decides to end the communication. A disconnection request message is sent from the initiating party.

8. DISCONNECTING to TERMINATED: This final transition happens when the disconnection process is completed. The session is terminated, and a termination confirmation message is sent.

These transitions can change based on the specific protocol or system design. They can be more complex, including error states or additional steps for enhanced security.

Section 4 – Extensibility

The Secure Real-time Chat Protocol (SRCP) is designed with extensibility in mind to accommodate future improvements, extensions, and features. This section outlines the key aspects of the protocol that promote extensibility.

4.1 Protocol Versioning

The version field in the PDU header allows for protocol versioning. The initial version of the protocol is 1. As new features or changes are introduced, the version number can be incremented to distinguish between different versions of the protocol. Clients and servers can negotiate the protocol version during the connection establishment to ensure compatibility between them.

4.2 Message Types

New message types can be easily added to the protocol by defining a new message type identifier and associated PDU payload structure. For example, future extensions could include file sharing, multimedia messages, or presence information. The extensible message type system allows SRCP to evolve and support new features without disrupting existing functionality.

4.4 Modular Encryption and Authentication

SRCP uses public-key cryptography for end-to-end encryption and a username/password-based authentication mechanism. These components can be replaced or extended with alternative encryption and authentication methods without affecting the core functionality of the protocol. For example, future extensions could include support for additional encryption algorithms, single sign-on (SSO) integration, or two-factor authentication (2FA).

4.5 Backward Compatibility

To ensure backward compatibility, new features and extensions should be designed in a way that does not break the functionality of older clients and servers. This can be achieved by preserving the original behavior of the protocol while introducing new optional fields or messages that can be safely ignored by clients and servers that do not support the new features.

By considering extensibility in the design of SRCP, the protocol can adapt to new requirements and technologies over time, ensuring its continued relevance and utility in the rapidly evolving field of real-time communication.

Section 5 – Security Implications

The Secure Real-time Chat Protocol (SRCP) incorporates various security features to protect the privacy and integrity of chat messages. However, it is crucial to understand the potential security implications and risks associated with the protocol. This section discusses the security measures implemented in SRCP and their potential limitations.

5.1 End-to-end Encryption

SRCP employs public-key cryptography to provide end-to-end encryption of chat messages. This ensures that messages can only be decrypted by the intended recipient(s) and remain protected even if intercepted during transmission. However, it is essential to use strong encryption algorithms and key management practices to maintain the confidentiality of messages. Additionally, the protocol must be updated periodically to address any vulnerabilities discovered in the encryption algorithms used.

5.2 User Authentication

The protocol requires users to authenticate themselves with a username and password before joining a chat session. This helps prevent unauthorized access to chat rooms. However, the security of this mechanism relies on the strength of the user's password and the server's ability to protect stored credentials. Future extensions could incorporate more robust authentication methods such as two-factor authentication (2FA) or integration with single sign-on (SSO) systems to enhance security.

5.3 Message Integrity

SRCP incorporates message integrity checks to ensure that messages are not tampered with during transmission. This can be achieved using cryptographic hash functions and digital signatures. However, the security of these mechanisms depends on the strength of the cryptographic algorithms used and the proper implementation of the signing and verification process.

5.5 Potential Limitations and Risks

While SRCP incorporates various security measures, it is essential to be aware of potential limitations and risks:

Key management: Proper management of public and private keys is crucial for maintaining the security of end-to-end encryption. This includes securely generating, storing, and exchanging keys. Server trust: Users must trust the server to manage authentication, public keys, and message routing. A compromised server could potentially impact the privacy and integrity of chat messages. Metadata leakage: While SRCP encrypts the content of messages, metadata such as sender, recipient, and message timestamps may still be visible to the server and potential eavesdroppers. This could reveal information about the communication patterns of users. Software vulnerabilities: The security of SRCP depends on the correct implementation of the protocol and its underlying technologies, such as encryption and authentication mechanisms. Vulnerabilities in the software could expose users to potential attacks. By understanding the security implications of SRCP, developers and users can make informed decisions about the protocol's appropriateness for specific use cases and take necessary precautions to mitigate potential risks.