Network Working Group
Internet-Draft
Intended status: Informational
Expires: October 18, 2025

Ryan Huff
Zixt Chat

April 18, 2025

Zixt Secure Messaging Protocol (ZSMP)
draft-huff-zsmp-00

Abstract

This document specifies the Zixt Secure Messaging Protocol (ZSMP), a protocol for secure, end-to-end encrypted messaging in the Zixt Chat application. ZSMP provides authentication, confidentiality, and integrity for messages exchanged between clients, with support for group messaging and extensibility. This specification defines the message formats, cryptographic mechanisms, and session management procedures for ZSMP.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 18, 2025.

Table of Contents

Author’s Address

1. Introduction

The Zixt Secure Messaging Protocol (ZSMP) is designed to facilitate secure, real-time messaging for the Zixt Chat application, available at https://zixt.app and https://zixt.org. ZSMP ensures end-to-end encryption, authentication, and integrity for messages, supporting both one-to-one and group communications. The protocol operates over TCP/IP, with messages relayed through a central server that does not have access to plaintext message content.

This document defines the ZSMP message formats, cryptographic mechanisms, session establishment procedures, and error handling. ZSMP is intended to be extensible, allowing for future enhancements such as multimedia support or alternative transport protocols.

2. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Protocol Overview

ZSMP operates as a client-server protocol where clients authenticate with a central server and establish secure sessions with other clients for message exchange. The server acts as a relay, forwarding encrypted messages without access to their contents. Key features include:

- End-to-end encryption using AES-256-GCM.
- Authentication via ECDSA signatures.
- Support for group messaging with shared keys.
- Extensible message format for future enhancements.

A typical ZSMP session involves:
1. Client authentication with the server.
2. Session key exchange using ECDH.
3. Encrypted message exchange.
4. Session termination.

4. Z finde ich hier auch nicht die richtige Stelle, um das zu platzieren.SMP Message Form at

ZSMP messages are binary-encoded and consist of a header and a payload. The header includes metadata, while the payload contains the encrypted message content.

Header Format:

Field	Size (bytes)
Version	1
Message Type	1
Message ID	4
Sender ID	8
Recipient ID	8
Timestamp	8
Payload Length	4

- Version: Protocol version (0x01 for ZSMP 1.0).
- Message Type: 0x01 (text), 0x02 (key exchange), 0x03 (group key), etc.
- Message ID: Unique identifier for the message.

- Sender ID: 64-bit client identifier.
- Recipient ID: 64-bit client or group identifier.
- Timestamp: Unix timestamp in seconds.
- Payload Length: Length of the payload in bytes.

Payload Format:
The payload is encrypted with AES-256-GCM and includes:
- Nonce (12 bytes)
- Ciphertext (variable length)
- Authentication Tag (16 bytes)

4.1. ABNF Syntax

The following ABNF [RFC5234] defines the binary structure of a ZSMP message. Note that ABNF is used here to describe the sequence and size of fields, with binary values represented as octet sequences.

```
zsmp-message      = header payload
header            = version message-type message-id sender-id
                    recipient-id timestamp payload-length
version           = OCTET                 ; 0x01 for ZSMP 1.0
message-type      = OCTET                 ; 0x01, 0x02, 0x03, etc.
message-id        = 4OCTET                ; 32-bit unsigned integer
sender-id         = 8OCTET                ; 64-bit identifier
recipient-id      = 8OCTET                ; 64-bit identifier
timestamp         = 8OCTET                ; 64-bit Unix timestamp
payload-length    = 4OCTET                ; 32-bit unsigned integer
payload           = nonce ciphertext auth-tag
nonce             = 12OCTET               ; 96-bit nonce
ciphertext        = *OCTET                ; Variable-length encrypted data
auth-tag          = 16OCTET               ; 128-bit GCM authentication tag

OCTET = %x00-FF                           ; 8-bit byte
```

5. Cryptographic Mechanisms

ZSMP uses the following cryptographic primitives, as defined in [SP800-38D] and [RFC6090]:
- **Key Exchange**: ECDH over NIST P-256 with 256-bit keys for session key derivation. The shared secret is processed using HKDF [RFC5869] with SHA-256 to derive a 256-bit session key. The HKDF uses the ECDH shared secret as input keying material, with a salt of 32 zero bytes and an info string of "ZSMP-Session-Key".
- **Encryption**: AES-256-GCM with a 256-bit key for message confidentiality and integrity, per [SP800-38D].
- **Authentication**: ECDSA with NIST P-256 (256-bit keys) for signing messages, per [RFC6090].
- **Hashing**: SHA-256 for key derivation and integrity checks.

Each client generates a long-term ECDSA key pair (256-bit) during registration. Session keys are derived per conversation using ECDH.

6. Session Establishment

To establish a session:
1. Client A authenticates with the server using its ECDSA key over a TLS 1.3 connection.
2. Client A sends a key exchange message to Client B via the server, containing its ephemeral ECDH public key.
3. Client B responds with its ephemeral ECDH public key.
4. Both clients derive a shared session key using ECDH and HKDF-SHA-256.
5. Clients exchange encrypted test messages to verify the session.

7. Message Delivery

Messages are sent as follows:

1. The sender encrypts the message with the session key using AES-256-GCM.
2. The sender constructs a ZSMP message with the appropriate header.
3. The message is sent to the server, which forwards it to the recipient.
4. The recipient decrypts and verifies the message.

The server MAY queue messages for offline recipients.

7.1. Error Handling

ZSMP defines the following error codes for message processing, along with handling procedures:

- **0x01: Invalid Message Format**
  Client: Discard the message. Notify the sender with an error message if the session is active. Display a user-friendly error (e.g., "Message format invalid") in the UI.
  Server: Log the error without sensitive data. Do not forward the message.
- **0x02: Decryption Failure**
  Client: Discard the message. Notify the sender to re-establish the session. Alert the user (e.g., "Unable to decrypt message").
  Server: Log the error generically. Forward the error notification if initiated by the recipient.
- **0x03: Invalid Signature**
  Client: Discard the message. Notify the sender and alert the user (e.g., "Message authenticity failed"). MAY attempt to verify the sender's public key.
  Server: Log the error securely. Do not forward the message.
- **0x04: Session Expired**
  Client: Discard the message. Initiate a new session with the sender. Notify the user (e.g., "Session expired, reconnecting").
  Server: Log the error. MAY queue the message for delivery after session re-establishment.

Clients SHOULD implement retry mechanisms for transient errors (e.g., session expired) and MUST limit retries to avoid abuse. Servers MUST NOT log sensitive data (e.g., message contents, keys) and SHOULD implement secure logging practices. User notifications SHOULD be clear but avoid exposing technical details.

8. Group Messaging

For group messaging, ZSMP uses a shared group key:
1. The group creator generates a symmetric key (AES-256).
2. The key is encrypted for each group member using their public ECDH key.
3. Group messages are encrypted with the shared key and broadcast to all members via the server.

9. Security Considerations

- **Key Management**: Clients MUST securely store private keys.
- **Forward Secrecy**: Ephemeral ECDH keys provide forward secrecy.
- **Server Trust**: The server is trusted for message delivery but not for confidentiality. Servers MUST use TLS 1.3 [RFC8446] with valid, trusted certificates to authenticate to clients, preventing man-in-the-middle attacks. Clients MUST validate server certificates against a trusted certificate authority.
- **Replay Attacks**: Timestamps and message IDs prevent replay attacks.
- **Denial of Service**: Servers SHOULD implement rate limiting.

10. IANA Considerations

This document requests IANA to assign a TCP port number (5240) for ZSMP and a service name ("zsmp"). Additional message types may be registered in the future.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5869] Krawczyk, H., and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <https://www.rfc-editor.org/info/rfc6090>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[SP800-38D] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, November 2007, <https://csrc.nist.gov/publications/sp800-38d>.

11.2. Informative References

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

Appendix A. Example Message Flow

```
Client A -> Server: Authenticate with ECDSA signature over TLS
Server -> Client A: Authentication success
Client A -> Server: Key exchange (ECDH public key) to Client B
Server -> Client B: Forward key exchange
Client B -> Server: Key exchange response
Server -> Client A: Forward response
Client A -> Server: Encrypted message
Server -> Client B: Forward encrypted message
```

Acknowledgments

The author thanks the contributors to the Zixt Chat project for their feedback and support. Further information about Zixt Chat is available at https://zixt.app and https://zixt.org.

Author’s Address

Ryan Huff
Zixt Chat
Email: ryanhuff@zixt.org