

# <infinity/>

## Chat Documentation

The AI Machine UG

February 11, 2026

Legal company name: **The AI Machine UG (haftungsbeschraenkt)**.

Short name used in this document: **The AI Machine UG**.

Application URL: <https://infinity.aimachine.io>

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Mission Statement . . . . .	2
1.2	Scope and Reality . . . . .	2
<b>2</b>	<b>Core Features</b>	<b>2</b>
<b>3</b>	<b>Security Architecture</b>	<b>2</b>
3.1	Transport and Connectivity . . . . .	2
3.2	Session Authentication and Key Schedule . . . . .	2
3.3	Message Encryption and AAD Binding . . . . .	3
3.4	Replay and Freshness Controls . . . . .	3
<b>4</b>	<b>Session Log Model</b>	<b>3</b>
4.1	Current Export Format . . . . .	3
4.2	Verification Modes . . . . .	3
4.3	What Verification Means . . . . .	3
<b>5</b>	<b>User Workflow</b>	<b>4</b>
5.1	Starting a Session . . . . .	4
5.2	Sending Messages . . . . .	4
5.3	Exporting and Verifying Logs . . . . .	4
<b>6</b>	<b>Operational Notes and Limitations</b>	<b>4</b>
<b>7</b>	<b>References</b>	<b>4</b>

# 1 Introduction

<infinity/> is a browser-based peer-to-peer chat application focused on authenticated session establishment, encrypted transport, and user-verifiable exported session logs.

## 1.1 Mission Statement

<infinity/> aims to provide practical private chat with explicit cryptographic verification controls for exported records. The current design emphasizes session confidentiality in transit and optional post-session verification of exported logs.

## 1.2 Scope and Reality

- Real-time chat is peer-to-peer via PeerJS/WebRTC.
- Secure messaging now requires a user-provided shared secret and a key-confirmation handshake.
- Exported logs are encrypted and signed in the current format.
- Legacy plaintext logs can still be verified in compatibility mode.

# 2 Core Features

- End-to-end encrypted P2P chat transport.
- Shared-secret-bound ECDH key derivation.
- Mutual key confirmation before a link is treated as authenticated.
- Replay and stale-message defenses using per-peer sequence and timestamp checks.
- Encrypted+signed session log export.
- Verification support for signed current logs and unsigned legacy logs.

# 3 Security Architecture

## 3.1 Transport and Connectivity

<infinity/> uses PeerJS over WebRTC data channels. The application primarily ships with a vendored local peerjs.min.js asset and retains CDN fallback loading logic for availability.

## 3.2 Session Authentication and Key Schedule

For each peer link, the following flow is implemented:

1. ECDH public key exchange (P-256).
2. Shared-secret-bound key derivation using HKDF.
3. Derivation of two session keys:
  - encryption key (AES-GCM message encryption/decryption)
  - auth key (HMAC-based key confirmation)
4. Mutual key confirmation messages (challenge/proof/ack) before marking the link authenticated.

### 3.3 Message Encryption and AAD Binding

Messages are encrypted with AES-GCM and include per-message metadata:

- 12-byte IV
- sequence number
- sender timestamp

Additional Authenticated Data (AAD) binds sender ID, target ID, sequence, timestamp, and protocol version so that metadata tampering causes decryption/authentication failure.

### 3.4 Replay and Freshness Controls

Inbound messages are rejected if:

- envelope structure is invalid,
- sequence is not strictly increasing,
- timestamp is outside configured skew tolerance,
- payload fields exceed validation limits.

## 4 Session Log Model

### 4.1 Current Export Format

Current export behavior is encrypted and signed:

- Event-level hash chain using `prevEventHash`.
- Event hash (SHA-256 canonicalized event content).
- Event signature (Ed25519 over event hash).
- Merkle root over event hashes.
- Full payload encryption using PBKDF2-derived AES-GCM key.

### 4.2 Verification Modes

- **Signed+Encrypted (current)**: decrypt payload, validate chain, verify signatures, validate Merkle root.
- **Unsigned Legacy (compatibility)**: validate event hash integrity and Merkle root without signature guarantees.

### 4.3 What Verification Means

- Signed current logs provide cryptographic authenticity relative to the embedded signer key in the export payload.
- Legacy unsigned verification provides integrity checks only and does not provide signer authenticity.

## 5 User Workflow

### 5.1 Starting a Session

1. Choose or generate handle.
2. Enter shared secret.
3. Enter realm and obtain active peer ID.
4. Connect to target peer by full ID.

### 5.2 Sending Messages

Messages become available once at least one authenticated secure link is established (lock indicator).

### 5.3 Exporting and Verifying Logs

1. Export uses encrypted+signed format (requires shared secret for encryption).
2. Verification prompts for shared secret when needed.
3. Current signed exports and legacy plaintext exports are both supported.

## 6 Operational Notes and Limitations

- Security still depends on safe out-of-band exchange of peer IDs and shared secret.
- Signaling infrastructure and STUN/TURN remain external dependencies.
- Log authenticity is tied to included signer key material in each export; out-of-band key trust policies are still recommended for stronger identity assurances.

## 7 References

- NIST FIPS 197 (AES): <https://csrc.nist.gov/publications/detail/fips/197/final>
- NIST SP 800-38D (GCM): <https://csrc.nist.gov/publications/detail/sp/800-38d/final>
- RFC 5869 (HKDF): <https://www.rfc-editor.org/rfc/rfc5869>
- RFC 8018 (PBKDF2): <https://www.rfc-editor.org/rfc/rfc8018>
- Web Crypto API: [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Crypto\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API)
- PeerJS: <https://peerjs.com/>
- WebRTC: <https://webrtc.org/>