MSG

Technical Whitepaper & Developer Guide

Version 2.0.0
Status: Stable / Production

January 12, 2026

Abstract

MSG is a privacy-first, zero-knowledge messaging platform designed to eliminate digital footprints. Unlike traditional messaging apps that rely on server-side storage and metadata retention, MSG operates on a **"Blind Relay" architecture**. The server functions solely as a transient data pipe, storing encrypted messages only until delivery or for a maximum of 48 hours in volatile storage. This document outlines the cryptographic primitives, system architecture, API specifications, and security mechanisms of the protocol.

Contents

1 Executive Summary

Core Philosophy:

- **Ephemeral by Design:** Data exists only as long as necessary.
- **Zero-Knowledge:** The server processes encrypted binary blobs without access to decryption keys for message content.
- **Identity Agnostic:** No phone numbers, emails, or government IDs are linked to accounts.

The platform empowers users with self-sovereign identity management, where keys are generated client-side and never exposed to the centralized relay in plain text (except for encrypted backup blobs).

2 Security Architecture

2.1 Cryptographic Primitives

The protocol utilizes the **Web Crypto API** (SubtleCrypto) for high-performance, native browser-based cryptography. This ensures that cryptographic operations are handled within the secure context of the browser engine.

Component	Algorithm	Specification
Key Exchange	ECDH	NIST P-256 Curve (secp256r1)
Symmetric Encryption	AES-GCM	256-bit Key, 12-byte random IV
Key Storage	JWK	JSON Web Key format
Transport Security	TLS 1.3	Standard HTTPS/WSS

Table 1: Cryptographic Standards

2.2 Identity & Key Generation

Unlike centralized apps that issue keys, MSG empowers the client to generate authority locally.

1. **Generation:** Upon registration, the client generates an ECDH Key Pair (`P-256`) locally.
2. **Format:** Keys are exported as **JWK** (JSON Web Key) objects.
3. **Persistence:**
   - **Private Key:** Stored in the user's browser `localStorage` as `priv_{username}`.
   - **Public Key:** Sent to the server to allow other users to derive shared secrets.
   - **Key Backup:** The private key is encrypted and stored on the server to facilitate multi-device login (Host-Proof Storage).

2.3 End-to-End Encryption (E2EE) Flow

Message confidentiality is guaranteed via a **Shared Secret** derived locally by both peers.

1. **Derivation:**
   - Sender uses `Sender_PrivateKey` + `Receiver_PublicKey`.

- Receiver uses `Receiver_PrivateKey` + `Sender_PublicKey`.
- Both result in the mathematically identical **AES-256 Shared Key**.

2. **Encryption:**
   - A random 12-byte **IV** (Initialization Vector) is generated for *every* message.
   - Message text is encrypted using `AES-GCM`.
   - `Payload = IV (12 bytes) + Ciphertext`.
3. **Transmission:** The Base64-encoded payload is sent via Socket.io/REST. The server sees only opaque strings.
4. **Decryption:** Receiver splits the payload to extract the IV and uses the Shared Key to decrypt the ciphertext.

3 System Design

3.1 Technology Stack

- **Frontend:** React, Framer Motion (Animations), Vite (Bundler).
- **Backend:** Node.js, Express, Socket.io (Real-time).
- **Database:** MongoDB (Mongoose ORM).
- **P2P:** WebRTC (Peer-to-Peer Video/Audio).

3.2 Database Schema (MongoDB)

Data storage is minimized to the bare essentials required for routing.

User Model (`User.js`)

Field	Type	Description
`username`	String	Unique anonymous identifier.
`friendCode`	String	Public discoverable ID (e.g., USER-1234).
`publicKey`	String	Public ECDH key for encryption.
`privateKey`	String	Backed up private key (Encrypted blob).
`contacts`	Array	List of usernames the user has interacted with.

Message Model (`Message.js`)

Uses MongoDB TTL (Time-To-Live) indexes for auto-deletion.

Field	Type	Description
`text`	String	**Encrypted** ciphertext (Server cannot read this).
`senderId`	String	Routing metadata.
`expireAt`	Date	**Auto-Delete:** Indexed to remove doc 48h after creation.
`isSaved`	Boolean	If `true`, bypasses the "Nuke" command (but not 48h TTL).

4 API Specification

4.1 Authentication Endpoints

`POST /register`
Creates a new anonymous identity.

- **Payload:** `{ username, password, publicKey, privateKey }`
- **Behavior:** Hashes password (bcrypt), generates `friendCode`, stores keys.
- **Response:** `{ _id, username, friendCode }`

`POST /login`
Retrieves identity and keys.

- **Payload:** `{ username, password }`
- **Behavior:** Verifies hash. **Critical:** Returns the `privateKey` to the client to restore E2EE capability on a new device.
- **Response:** `{ username, friendCode, privateKey, contacts }`

4.2 Messaging Endpoints

`GET /messages/:userId`
Fetches chat history.

- **Behavior:** Retrieves encrypted messages where `userId` is sender OR recipient.
- **Security:** Content returned is fully encrypted.

`DELETE /messages/nuke` (The Nuke Protocol)
Immediate destruction of conversation history.

- **Payload:** `{ myId, otherId }`
- **Behavior:**
  1. Deletes ALL messages between the two users from DB (unless `isSaved: true`).
  2. Emits real-time `chatNuked` event to connected sockets.

5 Real-Time Protocol (Socket.io)

The WebSocket layer handles instant message delivery and WebRTC signaling.

5.1 Events

Event	**Direction**	**Payload**	**Description**
`addNewUser`	Client → Server	`username`	Registers socket connection map.

`sendMessage`	Client → Server	`senderId, recipientId, text, time`	Saves to DB and relays to recipient.
`getMessage`	Server → Client	`Message Object`	Incoming encrypted message.
`chatNuked`	Server → Client	`{ target }`	Signal to clear local UI/Cache immediately.

5.2 P2P Signaling (Video)

WebRTC signaling bypasses DB storage entirely. The `callUser` and `answerCall` events exit the MSG server architecture to establish direct peer-to-peer media streams (Mesh topology).

6 Client-Side Mechanics & Storage

6.1 Local Storage Strategy

To maintain "Zero-Knowledge" properties while ensuring usability, sensitive keys are managed as follows:

- `priv_{username}`: The raw JWK Private Key. **Critical Security Asset.**
- `my_friend_code`: Public shareable ID.

Warning: If a user clears their browser cache (and has not synced the key via Login), the ability to decrypt past messages is lost forever.

6.2 The "Nuke" Implementation

The Nuke feature is a dual-action delete mechanism:

1. **Server-Side:** The `DELETE` API call wipes the MongoDB documents.
2. **Client-Side:** The `chatNuked` socket event triggers a frontend state update to wipe the React view state immediately, ensuring no visual remnants remain on either device.

7 Build & Deployment

7.1 Prerequisites

- **Node.js:** v16+
- **MongoDB:** v4.4+ (Replica Set recommended for Transactions)
- **Vite:** v4+

7.2 Environment Variables

```
MONGO_URI=mongodb+srv://...
PORT=3000
```

Listing 1: Server .env Configuration

```
const SERVER_URL = "https://msg-p0th.onrender.com"; // Adjust for Prod
```

Listing 2: Client Config (Login.jsx)

7.3 Compliance & Legal

- **GDPR:** The architecture is compliant by default as it adheres to "Data Minimization" and "Right to Erasure" (via Nuke/TTL).
- **Export Control:** Application uses standard AES-256 encryption. Check local regulations regarding encryption export.