Introducing Shroud: Encrypted, Ephemeral, Zero-Knowledge Messaging

Imagine a messaging app designed for one-time, secure communications where nothing is stored permanently, no sensitive data is ever in plain sight, and only the communicating parties hold the decryption keys. That's Shroud. Let's walk through how it works.

---

1. Ephemeral Sessions & Zero-Logs

- One-Time Sessions:
    - Session Creation: Every session is created on demand using a unique session ID and an accompanying termination token.
    - Limited Lifetime: Sessions automatically expire after 15 minutes of inactivity, ensuring that stale data is purged.
    - Ephemeral Data: All session data—users, messages, and even key exchange parameters—reside only in memory. Once the session ends (either manually or due to inactivity), all information is destroyed.
- Zero-Logs Philosophy:
    - No Persistent Storage: The server never stores any plain-text messages or long-term user identifiers.
    - Temporary Memory: Even the session messages, while kept in memory during active communication, vanish immediately when the session is terminated.

This design means that even if someone were to gain access to the server's memory, they would only see encrypted, ephemeral data with no long-term logs.

---

2. End-to-End Encryption & Zero-Knowledge

- Client-Side Encryption:
    - Encryption Happens Locally: All encryption and decryption take place on the client side. The server merely routes encrypted blobs—it never sees the unencrypted content.
    - Zero-Knowledge: Because the server does not have the keys or the decryption process, it is "zero-knowledge" regarding the content of the messages.
- Key Exchange via ECDH:

    - Ephemeral Diffie–Hellman (ECDH) on P-256: Each user generates an ephemeral key pair when joining a session.
    - Public Key Exchange: When a user joins, their public key (exported as a JSON Web Key) is shared with the other participant.
    - Shared Secret Derivation: Using ECDH and HKDF (with the session ID as salt), both parties derive the same shared AES-GCM key.
    - No Key Exposure: At no point does the server see any part of the key material.
- Message Encryption with AES-GCM:
    - Per-Message IV: Every message is encrypted with a fresh random initialization vector (IV) to ensure message uniqueness.
    - AES-GCM Benefits: Provides both confidentiality and authentication (integrity), so messages can't be tampered with without detection.

In short, even if intercepted, the data remains unintelligible since only the two communicating clients possess the decryption key.

---

3. Robust Communication & Data Integrity

- Real-Time Messaging with Socket.IO:
    - Low-Latency Exchange: Messages and file transfers are transmitted in real time.
    - Participant Management: The app supports a strict two-user limit per session, ensuring that keys and messages remain private between the intended parties.
- Message and File Integrity:
    - Message Deletion: Users can delete their own messages. This deletion is broadcast to all participants, ensuring that no stale or unwanted data lingers.
    - Secure File Transfers: Files are encrypted using the same shared key, then decrypted on the client side. Additionally, there's client-side validation of file type and size to prevent abuse.

---

4. Session Control & User Management

- User Identification:
    - Random, Ephemeral IDs: Users are given randomly generated IDs (e.g., "SH" followed by random hex digits) that are used only for the duration of the session.
- Session Termination:
    - Termination Token: Each session is paired with a unique termination token, which must be provided to end the session. This prevents unauthorized session closures.

    - Graceful Disconnects: If a user disconnects unexpectedly, a background task gives them a 15-second grace period before they are removed from the session—at which point an announcement is sent to the remaining participant.
- Auto-Refresh:
    - Preventing Stale Sessions: If a session is not joined within a certain timeframe, it is automatically refreshed to ensure that only active, intentional sessions are maintained.

---

5. Additional UI & Security Enhancements

- Secure Frontend Practices:
    - Input Sanitization: All messages are sanitized (using techniques like escaping HTML and linkifying URLs) to prevent cross-site scripting or injection attacks.
    - User Feedback: Visual notifications (toasts) inform users of key events (e.g., new session generated, session ended, or errors).
- File Handling Security:
    - Client-Side File Encryption/Decryption: Files are read as binary data, encrypted using AES-GCM with a unique IV, and then securely transmitted. Recipients decrypt files only on their end.
    - Validation: Only specific file types and sizes are allowed, reducing the risk of malicious file uploads.

Summing Up

- Zero-Knowledge Principles:
  Shroud is designed so that all encryption occurs on the client side. The server handles only encrypted data, meaning it "knows" nothing about the actual contents of your communications.
- Ephemeral, Secure, and Private:
  With a strict session lifetime, robust key exchange via ECDH, AES-GCM encryption for both messages and files, and complete user anonymity, Shroud embodies the ethos of secure, one-time communication.
- User Empowerment:
  By allowing either party to destroy the session (and thereby all data) at any time, Shroud empowers its users to maintain control over their privacy. Once the session is over, nothing remains—no logs, no keys, no recoverable messages.

By combining state-of-the-art cryptography with an ephemeral, no-trace philosophy, Shroud offers a secure, private communication channel where the only people who ever "know" the conversation are you and the person you're communicating with.