<armoredfile/>: Secure Peer-to-Peer Encrypted File Sharing

Mission Statement, User Manual, and Security Overview

The AI Machine UG
D-66333 Völklingen, Germany

May 25, 2025

Contents

1 Mission Statement

The mission of <armoredfile/> is to provide a highly secure, private, and ephemeral peer-to-peer (P2P) file sharing environment. Our core commitment is to user privacy and data control, achieved through robust end-to-end encryption of files directly within the browser, direct user-to-user communication channels facilitated by WebRTC, a strict policy against central storage of shared files by the application, and the principle of data minimization with ephemeral share codes. <armoredfile/> is designed for individuals seeking a tool to transfer files where the content remains confidential, is not stored on any central server, and where the connection is temporary and pseudonymous. We aim to empower users with control over their file data and their digital footprint during the sharing process. The name <armoredfile/> evokes the concepts of protection and security, reinforcing the application's goal to act as a digital safe-conduct for files in transit.

2 Concise User Manual

2.1 Getting Ready

1. Access the <armoredfile/> application. You can find it online at:

   https://armoredfile.aimachine.io

   Use a modern web browser that supports WebRTC and the Web Crypto API (e.g., Mozilla Firefox, Google Chrome).

2. The application interface provides options to either share a file or receive a file using a "Share Code".

2.2 Sharing a File (Sender Role)

1. **Select File:** Click on the file input area under "1. SELECT FILE TO SHARE:" or drag and drop a file onto it. The HTML element is fileInput.

2. **Initiate Share:** Once a file is selected, the SHARE SELECTED FILE button (HTML ID: initiateShareBtn) becomes active. Click this button.

3. **Local Encryption:** Your browser will then locally encrypt the file using AES-GCM with a securely generated random key (fileKey) and initialization vector (IV - fileIV).

4. **Share Code Generation:** A unique "Share Code" (which is a PeerJS ID, myPeerJsId, prefixed with armoredfile-) will be generated and displayed in the "YOUR SHARE CODE (SEND THIS TO RECIPIENT):" section (shareCodeOutputEl). This code is active for approximately 5 minutes (EPHEMERAL_TIMEOUT_MS) if no recipient connects.

5. **Copy Share Code:** Click on the displayed Share Code to copy it to your clipboard.

6. **Securely Transmit Code:** Send this copied Share Code to your intended recipient through a secure out-of-band channel (e.g., a trusted end-to-end encrypted messenger, in person). This step is crucial for security.

7. <armoredfile/> will then wait for the recipient to connect using this Share Code. The status panel (statusMessagesEl) will indicate progress.

2.3 Receiving a File (Recipient Role)

1. **Obtain Share Code:** Securely receive the "Share Code" from the sender.
2. **Enter Share Code:** Paste the received Share Code into the input field under "2. ENTER SHARE CODE (IF RECEIVING):" (`shareCodeInputEl`).
3. **Initiate Reception:** The `RECEIVE WITH CODE` button (HTML ID: `receiveFileBtn`) will become active. Click this button.
4. **Connection:** `<armoredfile/>` will attempt to establish a direct P2P connection with the sender using the provided Share Code.
5. **File Transfer and Decryption:** Once connected, the encrypted file data will be transferred. Your browser will then decrypt the file using the key and `IV` securely transmitted from the sender.
6. **Automatic Download:** Upon successful decryption, the file will be automatically downloaded by your browser.

2.4 The File Transfer Process

During the transfer, several types of data are exchanged over the secure `PeerJS` connection:

- **File Metadata:** Information like the file name, size, MIME type, the `IV` used for encryption, and total chunk count (sent as a `file_meta` message).
- **Encryption Key:** The symmetric `AES-GCM` key (as a `JWK`) used to encrypt the file (sent as a `file_key` message).
- **Encrypted File Chunks:** The file itself, broken down into smaller, encrypted chunks (typically 16KB each - `CHUNK_SIZE`) for efficient transfer. Binary data is sent after control messages.

A progress bar (`progressBarEl`) will indicate the status of the upload (for the sender) or download (for the recipient).

2.5 Ephemeral Nature of the Share

- **Share Code Timeout:** If a sender initiates a share and no recipient connects using the Share Code within approximately 5 minutes (`EPHEMERAL_TIMEOUT_MS`), the Share Code expires, and the sender's encrypted file data and key are cleared from browser memory (`clearSharedFileData` method).
- **Data Clearance Post-Transfer:** After a successful file transfer, the sender's application instance also clears the encrypted file data and key. The recipient receives the decrypted file, which they can then save locally. `<armoredfile/>` does not retain any file data after the session/transfer.

2.6 Canceling an Operation / Resetting

- The `CANCEL / RESET` button (HTML ID: `cancelBtn`) allows users to terminate the current process (sharing or receiving) and reset the application to its initial state (`resetState` method).
- The application will also automatically reset after a successful transfer, a timeout, or certain critical errors, readying it for a new operation.

3 Recommended Best Practices for Security and Privacy

Achieving the objectives of secure and private file sharing with <armoredfile/> relies significantly on users adhering to certain best practices:

- **Securely Exchange Share Codes Out-of-Band:**
  - *Why:* The initial security of connecting to the correct peer depends entirely on how the Share Code is obtained by the recipient. If an attacker intercepts or provides a false Share Code, the recipient might unknowingly connect to the attacker.
  - *How:* Share the Share Code only through trusted, pre-existing secure communication channels (e.g., in-person, via another trusted E2EE messaging service). Do not share it over insecure channels.
- **Utilize a Trusted VPN Service:**
  - *Why:* A VPN can mask your real IP address from the PeerJS signaling server and any potential TURN relay servers used for NAT traversal, enhancing network-level privacy. While PeerJS connections are P2P, signaling servers are involved in establishing these connections.
  - *How:* Choose a reputable VPN provider with a strong no-logging policy and connect to it before starting <armoredfile/>.
- **Ensure No Unnecessary Logging (Application Level):**
  - *Why:* <armoredfile/> is designed with a strict no-logging principle for file content and encryption keys by the application itself.
  - *How:* Access <armoredfile/> from its official source, available at the following URL:

    https://armoredfile.aimachine.io

    Be aware that browser extensions or local developer tools, if misused by the user, could potentially capture data locally (e.g., screen capture, DOM inspection). However, these local tools cannot break <armoredfile/>'s E2EE for the file content in transit if the peer is genuine.
- **Verify Application Authenticity and Source:**
  - *Why:* To ensure you are using the legitimate, unmodified version of <armoredfile/>.
  - *How:* Always access the application via its official URL, which is:

    https://armoredfile.aimachine.io

    Be extremely cautious of unofficial websites, mirrors, or any modified versions, as these could compromise your security and privacy.
- **Maintain Endpoint Security:**
  - *Why:* The security of your device (computer, smartphone) is critical. Malware on your device can bypass any application-level security measures.
  - *How:* Keep your operating system and browser updated to the latest versions, use reputable anti-malware software, and be cautious about installed software and browser extensions.

4 Overview of Security and Privacy Elements

<armoredfile/> incorporates several key architectural and cryptographic elements to ensure secure and private file sharing:

4.1 Peer-to-Peer (P2P) Architecture

<armoredfile/> utilizes WebRTC via the PeerJS library for direct browser-to-browser connections. This P2P architecture means that encrypted file data is streamed directly between the sender and the recipient, minimizing reliance on central servers for data relay. A signaling server is used by PeerJS only for connection establishment (exchanging connection metadata like IP addresses and capabilities, often via STUN/TURN servers like stun:stun.l.google.com:19302 for NAT traversal) and does not have access to the encryption keys or the file content itself.

4.2 End-to-End Encryption (E2EE) of Files

All files shared via <armoredfile/> are end-to-end encrypted. This means the file is encrypted on the sender's device before transmission and decrypted only on the recipient's device. No intermediary, including the signaling server or any potential TURN relay servers, can access the plaintext file content.

4.2.1 File Encryption: AES-GCM

<armoredfile/> uses the Advanced Encryption Standard in Galois/Counter Mode (AES-GCM) with a 256-bit key (AES-256-GCM) for file encryption. AES-GCM is a highly secure and widely adopted authenticated encryption algorithm that provides confidentiality, integrity, and authenticity for the encrypted data. This is performed directly in the browser using the Web Crypto API.

4.2.2 Symmetric Key Handling

For each file sharing session, the sender's browser securely generates a unique symmetric AES-GCM key (fileKey using crypto.subtle.generateKey). This key is:

- Used to encrypt the file locally on the sender's machine.
- Exported in a standard JSON Web Key (JWK) format (crypto.subtle.exportKey("jwk", this.fileKey)).
- Securely transmitted to the recipient over the PeerJS data channel (as part of the file_key message). PeerJS data channels are themselves encrypted using Datagram Transport Layer Security (DTLS), ensuring the JWK (containing the file encryption key) is protected in transit between the peers.

The recipient's browser then imports this JWK (crypto.subtle.importKey("jwk", message.key, ...)) to decrypt the received file data.

4.2.3 Initialization Vector (IV)

A unique 12-byte (96-bit) Initialization Vector (IV) is cryptographically securely generated by the sender for each file encryption operation (crypto.getRandomValues(new Uint8Array(12))). The IV is crucial for the security of AES-GCM and ensures that encrypting the same file multiple times will produce different ciphertext. The IV is sent to the recipient (as an array within the file_meta message) over the secure PeerJS data channel.

4.3 No Persistent File Storage by the Application

<armoredfile/> is designed to be stateless regarding file storage.

- Files are loaded into the sender's browser memory for encryption and chunking.
- Encrypted chunks are streamed to the recipient.
- The recipient's browser assembles and decrypts these chunks in memory.
- Once the transfer is complete or the session is terminated (e.g., via timeout, cancellation, or browser close), the file data and associated encryption keys are cleared from the browser memory of the application instance (e.g., `this.clearSharedFileData()` which nullifies `this.encryptedFile`, `this.fileKey`, `this.fileIV`).
- <armoredfile/> does not store any part of the file or the encryption keys on any central server.

4.4 Ephemeral Share Codes and Data

The "Share Codes" generated by <armoredfile/> are temporary. If a connection is not established by a recipient within approximately 5 minutes (`EPHEMERAL_TIMEOUT_MS`), the code expires, and the associated sharing session on the sender's side is terminated, clearing any related data (`handleEphemeralTimeout` method). This ephemerality limits the window of opportunity for connection and enhances privacy.

4.5 No Application-Level Logging of Sensitive Data

A core design principle of <armoredfile/> is the minimization of data collection. The production version of <armoredfile/> at its official URL is intended not to perform any logging of file content, encryption keys, or other sensitive user or session metadata to the browser console (beyond transient diagnostic messages like those from `console.log` in the provided source, which are primarily for development/debugging and not persistent storage accessible to the application developers). Users should be aware that their own browser's developer tools or extensions could potentially log data if activated by the user, but this is outside the control of the <armoredfile/> web application itself.

4.6 Pseudonymous Connection Identifiers

The "Share Codes" used by <armoredfile/> are `PeerJS` IDs generated to be unique random strings. These IDs are constructed with a prefix `armoredfile-`, followed by characters derived from the current timestamp and a random component (e.g., a Share Code might appear as `armoredfile-kjpvxy0a-b3cde4f`). While these IDs are necessary for establishing the P2P connection, they are not inherently linked to a user's real-world identity by the <armoredfile/> system itself. Users are responsible for sharing these codes securely and can enhance pseudonymity by using VPNs and practicing other operational security measures.

5 References

For further information on the cryptographic standards and technologies used or relevant to <armoredfile/>, the following resources are recommended:

- **NIST FIPS 197:** Advanced Encryption Standard (AES).
  https://csrc.nist.gov/publications/detail/fips/197/final
- **NIST SP 800-38D:** Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.
  https://csrc.nist.gov/publications/detail/sp/800-38d/final
- **Web Crypto API (MDN Web Docs):** JavaScript API for browser cryptography.
  https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API
- **PeerJS Library:** Simplifies `WebRTC` peer-to-peer connections.
  https://peerjs.com/
- **WebRTC Project:** Open-source project for Real-Time Communications in browsers and mobile applications.
  https://webrtc.org/

Document End