

# EncryptFire Security Review

Version 1.0 – Self-Review with AI Validation

## 1. Purpose of This Document

This security review outlines the architecture, encryption methodology, and data-handling practices of the EncryptFire application. It is intended to give technical users, legal fiduciaries, and privacy-conscious individuals a transparent overview of how the tool operates — and what risks it mitigates or avoids entirely.

This is a self-review based on manual inspection and AI-assisted validation.

## 2. Scope of Review

This review focuses on the following:

- Client-side architecture (no backend services or APIs)
- Cryptographic implementations and library usage
- File output structure and password protection
- Browser storage, local file handling, and airgap compatibility
- Risk surfaces and recommended operational practices
- Code visibility and verifiability

Target users include:

- Individuals using self-custodied crypto wallets
- Estate planners and fiduciaries seeking a secure continuity solution
- Privacy-focused users aiming to reduce exposure to cloud-based vulnerabilities

## 3. Summary of Findings

### 3.1 In-Browser Isolation

EncryptFire is fully self-contained and runs 100% in-browser with no internet access or backend dependencies. When opened via `encryptfire.html`, it makes:

- No network requests
- No telemetry or analytics calls
- No external library loads (all dependencies are bundled locally)

The entire system can be verified and operated in airplane mode on an airgapped machine.

It is specifically designed for use on laptops or desktops, not mobile devices — both for security reasons and due to the file handling and storage methods required.

### 3.2 Cryptographic Operations

EncryptFire uses strong, modern cryptography:

- AES-256-CBC is used for encrypting file content
- AES-128 is used for PDF password protection (standardized)
- PBKDF2 key derivation with per-file salt and IV ensures high entropy and resistance to brute-force attacks
- All operations are performed using the crypto-js library (served locally)

AES-256-CBC uses a symmetric key algorithm with cipher block chaining, offering strong confidentiality when combined with random IVs and PBKDF2-derived keys. Files are encrypted using unique salts and initialization vectors to ensure non-repetition of ciphertext, even for identical input.

All encrypted outputs are Base64-encoded to ensure compatibility across devices and platforms.

### 3.3 File Generation & Output

- PDF files are generated using pdfmake and secured with passwords.
- Fonts are embedded via vfs\_fonts.js for print-ready compatibility.
- No content is saved to IndexedDB, localStorage, sessionStorage, or cookies unless explicitly enabled by the user.

Users may optionally choose to print password-protected files or store them offline on USB devices.

### 3.4 Password Security

- Passwords are 32-character randomized strings with uppercase, lowercase, numbers, and symbols.
- Entropy exceeds most password manager standards and is sufficient to resist brute-force attempts even by advanced threat actors.
- There is no password recovery mechanism. Redundant backups (multi-location copies) are essential.

Users can store encrypted passwords in a time-release system (e.g., Google Inactive Account Manager), with the password to that system stored offline.

### 3.5 PDF Handling

EncryptFire uses:

- pdftmake for generating password-protected files
- vfs\_fonts.js for embedding fonts directly into the file
- pdf.min.js for decrypting and displaying encrypted PDFs without needing Adobe Acrobat, for reading in-browser

PDF password protection is enforced via the encrypt flag in pdftmake, which uses AES-128 encryption at the file level. Attempting to open these PDFs with an incorrect password results in denial of access at the viewer level (e.g., Acrobat, Preview). This method is widely supported and used for secure document exchange across industries.


PDF content may be optionally stored encrypted using 256 bit AES-CBC. This is the recommended workflow for most users, however unencrypted contents allows for direct reading with password access. The choice exists for the users preference and use case.

### 3.6 Source Code Visibility


- The entire app is viewable via browser 'View Source'
- All files are delivered via a compressed (.zip) format to be run locally
- All code is unobfuscated and commented
- Users may inspect encryption logic in files like encrypt.js, crypto-utils.js, and pdf-export.js

Anyone can verify there is no network activity, telemetry, or remote data storage via developer tools panel in the browser.

## 4. Risks and Limitations

 **User Error:** As with any cold storage system, accidental loss of passwords or poor backup practices can lead to permanent loss of access.

- Examples: printing passwords without protection, storing keys and encrypted files together, emailing backups

 **No Recovery:** There is no central recovery system. This is intentional and a necessity for hardened cold storage. Redundant multi-location storage is necessary for risk mitigation.

⚠ Browser Trust Assumption: EncryptFire assumes the browser used is clean, malware-free and running no extensions that may read user input or live memory.

⚠ Dependency Integrity: Libraries such as crypto-js, pdfmake, and pdf.min.js are open source, widely trusted and are in popular use worldwide. All are served locally to facilitate the security requirements of the app running air gapped (not connected to the Internet).

## 5. Recommended Verification Steps

Technical users can validate the app using DevTools:

- Network Tab: Confirm zero outgoing requests on any action
- Sources Tab: Inspect JavaScript logic for encryption/decryption
- Console: Confirm password generation entropy
- File Handling: Use a hex editor to confirm that file contents differ even for the same input (proving random IV and salt)

You may also test the PDF password protection by attempting to open files with incorrect credentials.

## 6. Summary Statement

EncryptFire is a transparent, self-contained cryptographic tool that enables hardened wallet cold storage with a unique cipher map, encryption and distributed physical backups. By leveraging layered AES encryption, PDF protection, and no browser storage, it achieves a high level of security without any reliance on cloud services or centralized infrastructure. Further, it provides a mechanism for recovery that does not require the app (if used with PDF password protection only). For users who demand the highest encryption all content may be secured with 256 bit AES-CBC.

EncryptFire's core mission is privacy-by-design and transparent security. Users can verify its behavior locally, with nothing transmitted or stored online — no backend, no telemetry, and no hidden processes. It delivers security, privacy, and transparency in a standalone app that runs entirely in the browser.

Crypto Security Tools  
8 The Green, Ste B  
Dover, DE 19901  
[CryptoSecurityTools.com](https://CryptoSecurityTools.com)