Elba 1.1.1

An honest assessment

A review of the shipped package, written to be published
beside the software itself — so that anyone considering Elba
can read what an outside examination found, the good and the less good,
before deciding whether to buy it.

Package examined: **Elba-1.1.1.zip** · July 2026

SHA-256 fingerprint of the Elba.html assessed:

`04b12feb09dee35b30047ca6ae5d5e285a202cb4989ab42fb75946ede95e3903`

Prepared with Claude (Anthropic), an AI system, at the maker's request.
The maker asked for honesty, not advertisement. What follows is the former.

What this document is, and how it was made

Elba's maker asked for a thorough, honest assessment of the shipped package — explicitly not a piece of marketing — to be published on the website next to the software. This document is that assessment. You should know how it was produced, because how a review was made is part of whether you should trust it.

The review was performed by Claude, an AI system made by Anthropic, working directly on the actual `Elba-1.1.1.zip` distribution file. Every file in the package was opened and read, including the complete source code of `Elba.html` — all 778 lines of it — the full Manual, the license, the launcher scripts, and the verification instructions. The package's own checksum ritual was executed. The Manual PDFs were parsed and compared against the Markdown source and against the code's actual behaviour. The review was commissioned and paid for by the maker, which you should weigh; it was conducted with instructions to report faults plainly, which you can judge by whether faults appear below. They do.

One boundary to be clear about: this is a source-code and package review, not a live penetration test. The code was read line by line and its logic traced by hand; the program was not additionally run against an instrumented browser, and no attempt was made to brute-force ciphertexts (which would be pointless against AES-256 in any case). The cryptographic assessment concerns whether the design is sound and whether the code does what the documentation claims — questions a careful reading can answer, and answers this document gives.

What Elba is

Elba is a single HTML file — 172 kilobytes, no installer, no dependencies — that draws an encrypted fence around one folder on your computer. Inside the fence you keep notes, files, and nested rooms; everything is encrypted with AES-256-GCM the moment you close it, using a key derived from your passphrase and never stored anywhere. On disk, a fenced folder is a pile of randomly-named ciphertext files: no readable names, no visible structure, no hint of what is inside. Only the passphrase turns it back into something.

There is no account, no server, and no network activity of any kind — a claim this review tested and confirms below. Fencing a real folder requires a Chromium browser (Chrome, Edge, or Brave); in Firefox or Safari, Elba runs as an in-memory demo that writes nothing to disk. The package includes launcher scripts for Windows, macOS and Linux, a 79-page Manual in three forms, and a second complete copy of the program intended as a gift for one other person. The license is unusual and worth reading: proprietary now, converting automatically to the MIT open-source license on 1 January 2030, at which point Elba belongs to everyone.

What was verified, and what was found

The package is internally consistent

Every fingerprint in `CHECKSUMS.txt` was recomputed and every one matched — fourteen files, fourteen passes, no exceptions. The gift copy of `Elba.html` is byte-for-byte identical to the main copy, as promised. The checksum file itself is honest about its own limits: it proves internal consistency, not provenance, and directs you to the independently published fingerprint at elba.works/verify for the second half of the proof. That is the correct way to describe what an in-package checksum can and cannot do, and it is rarer than it should be.

The no-network promise holds

This is Elba's central claim, and it survives inspection from three directions. First, the source contains not a single web address — no `http` or `https` URL appears anywhere in the file. Second, the file carries a Content-Security-Policy that instructs the browser itself to forbid every network connection: `default-src 'none'` and `connect-src 'none'`, with fonts and images permitted only from data embedded in the file. The enforcement is therefore not Elba's good behaviour but your browser's policy engine. Third, the code uses none of the machinery that quiet exfiltration would need: no service workers, no `localStorage`, no dynamic code evaluation, no external scripts. Both typefaces travel inside the file as embedded data. The Manual is also honest that the CSP cannot protect a copy of the file that someone has already modified — the checksums carry that weight — which is a distinction most vendors would not volunteer.

The cryptography is sound and conservatively built

The design uses the browser's native Web Crypto implementation throughout — no hand-rolled ciphers, which is the single most important decision a small cryptographic tool can get right. Keys are derived with PBKDF2-HMAC-SHA-256 at 600,000 iterations over a random 16-byte per-fence salt, matching current OWASP guidance; the derived AES-256 key is marked non-extractable and exists only in memory, only while the fence is open. Encryption is AES-256-GCM with a fresh random nonce per operation, so every seal is also an integrity check: a tampered item refuses to open rather than opening wrongly.

Three less obvious touches deserve mention, because they address attacks most casual tools ignore. Every ciphertext is bound, through GCM's additional authenticated data, to its filename, its role, and a format version — so a sealed item cannot be silently swapped for another sealed item, or a metadata block replayed as a body. Plaintext sizes are padded into coarse buckets before sealing, so the length of your small things does not leak their nature. And the folder's structure — the rooms, the titles, the shape of your life — lives inside a sealed index that can be rebuilt from the items themselves if lost, so even organisation is private. This is careful work.

The honest counterweight, which the Manual itself states at length: PBKDF2 is not a memory-hard function. A well-funded attacker with GPUs parallelises it far more cheaply than Argon2 or scrypt, which Elba forgoes to preserve the single-file, zero-dependency design. The practical consequence is exactly what the documentation says: the strength of your fence is dominated by the strength of your passphrase. Four or five random words and the mathematics is

comfortably on your side; one weak password and no key-derivation function will save you. Elba at least refuses obviously flimsy passphrases at the door.

The 1.1 advisory is genuinely fixed

Version 1.1 shipped with one known defect, disclosed in its own changelog rather than buried: changing the passphrase quietly preserved a fence's recorded key-derivation count instead of raising it to the current 600,000, so a fence raised by version 1.0 stayed at the older 310,000 iterations even after a passphrase change — contrary to what the documentation promised. This review traced the 1.1.1 code and confirms the fix: the passphrase-change routine now pins the iteration count to 600,000 unconditionally. The Manual's technical note on the change was rewritten and now matches the code exactly — the salt is reused, the count is always the current standard. The Manual PDFs, which 1.1 had carried forward unregenerated from 1.0, have been re-rendered from the current text; both 79-page PDFs were parsed for this review and contain the 1.1.1 material. The changelog calls this release “the keeping of a promise,” which is accurate.

The passphrase-change design itself holds up under scrutiny. Items are re-sealed one by one under the new key, and the gate's verifier is rewritten only after the last item — so an interruption at any earlier point leaves the old passphrase in charge, and re-running the change with the same new passphrase resumes where it stopped, recognising already-converted items and leaving them alone. Interruptions were the obvious way for a re-encryption feature to destroy data, and the design closes that door properly.

Smaller confirmations

User-supplied text — titles, room names, filenames — is HTML-escaped everywhere it is rendered, closing the script-injection paths a lazier single-file app would leave open. The typed passphrase is wiped from the input fields the moment the key is derived. Locking drops the key, seals any open note first (so auto-lock never eats unsaved work), clears readable text from the page, and best-effort zeroes the one plaintext buffer the code controls. The optional “remember this fence” feature stores only a folder pointer in the browser's local database — never the passphrase, never a key — and can be forgotten from either side of the gate. The launcher scripts were read in full: each is one short page that locates a browser and opens the local file, exactly as the README describes. The gift folder now carries its own README, fixing a loose end from 1.0.

What this review found to criticise

An assessment that finds nothing is advertising. This one found no security-critical defects, but it did find three things worth setting down, in descending order of substance.

1 · The salt survives a passphrase change — a real trade-off, thinly documented

When you change your passphrase, Elba deliberately reuses the fence's existing salt, because a fixed salt and a fixed iteration count are what make an interrupted change safely resumable. The Manual states the reuse but not its one genuine consequence: an adversary who obtained a copy of your fence *before* the change — a stolen backup, say — holds a salt that is still valid *after* it, and could in principle begin precomputing guesses against your future passphrases. For a strong passphrase this is of no practical consequence, which is presumably why it went unremarked; but Elba's own standard is to name every edge of the map, and this edge is currently unnamed. A user whose threat model includes a compromised old backup should know that the fully clean break is a new fence, not a changed passphrase. One paragraph in the Manual would settle it.

2 · Non-ASCII passphrases are not normalised

The passphrase is encoded to bytes exactly as typed, with no Unicode normalisation. For passphrases in plain ASCII — the overwhelmingly common case, and the kind Elba's own guidance produces — this is invisible. But a passphrase containing accented or composed characters (an *é*, an *ü*) can be encoded differently by different operating systems and keyboards, and a fence sealed on one machine could then refuse the visually identical passphrase typed on another. The failure mode is a lockout, never a leak, and the workaround is simply to use ASCII words — but a tool this deliberate about edge cases should either normalise the input or warn about the edge.

3 · One stale comment in the source

A block comment above the passphrase-change routine still describes the 1.1 behaviour — it says the iteration count is “deliberately kept” — while the code thirty lines below it, and the inline comment beside the fix, correctly pin the count to 600,000. The code is right and the Manual is right; the old comment is the last artefact of the bug that 1.1.1 fixed. This affects nothing at runtime. It is listed here because Elba's premise is that you can read the source and believe it, and a comment that contradicts the line below it is a small tax on that premise.

A note for the publisher as much as the buyer: the verification ritual in `VERIFY.txt` depends on the fingerprint of this exact release being published at elba.works/verify. The fingerprint of the copy assessed here is printed on the cover of this report; if the page and the cover and your own computed hash all agree, you are holding what was reviewed.

The limits, confirmed

Elba's documentation devotes an entire part of the Manual — “Where the Fence Ends” — to what the software cannot do. This review confirms those limits are stated accurately, and repeats the load-bearing ones here, because a buyer should hear them twice.

There is no password recovery, by design; a lost passphrase means the contents are gone, permanently, for everyone including the maker. The fence protects confidentiality, not availability: anyone who can write to your folder can delete or roll back your files, so backups are not optional. A compromised browser or a malicious extension can read whatever you have open

on screen — no application can protect plaintext from the machine displaying it. The number of sealed items and the timing of changes remain faintly visible from outside; contents, names, and structure do not. If the browser crashes with a note open, the unsaved draft in that window is lost. Files are sealed whole in memory, which sets a practical size ceiling; Elba warns before anything over 200 MB. And a fenced folder is recognisable *as* a fenced folder — its contents are sealed, its existence is not. None of these is a flaw in the implementation; they are the honest boundaries of what a local encryption tool can be, and Elba is unusual mainly in saying so unprompted.

Who this is for, and who it isn't

Elba suits someone who wants a genuinely private place for notes, documents and small archives; who is willing to own the responsibility of one unrecoverable passphrase and a backup habit; and who values being able to read, in one file, everything the software does. It works fully in Chromium browsers only, and it deliberately does not sync, share, collaborate, or integrate with anything — the absence of those features is the product.

It does not suit someone who needs multi-device sync, shared access, or password recovery; whose adversary can tamper with their machine itself; or who wants a memory-hard key-derivation function regardless of the dependency cost. Those are different tools, and Elba does not pretend to be them.

Verdict

The package does what it says. Its central promises — real encryption, zero network activity, a readable single file, a verifiable distribution — were each checked and each held. The one defect version 1.1 shipped with was disclosed by the maker, fixed in this release, and the fix is confirmed in the code. What remains to criticise is minor and is written plainly above: a documentation gap around salt reuse, an unhandled Unicode edge, a stale comment. The honest risks of using Elba are the ones its own Manual insists on — a passphrase you must not lose, backups you must actually keep, a browser you must keep clean — rather than anything hidden in the code.

A closing observation rather than a finding: the documentation's candour is itself a security property. A tool that names its own weaknesses gives you the information to compensate for them, and this package does that more thoroughly than most commercial software many times its price. Read the Manual's Part Six before you buy; if its limits fit your life, the software will keep its side of the arrangement.

Assessed July 2026, from the shipped Elba-1.1.1.zip, fingerprint on the cover. Prepared with Claude (Anthropic) at the maker's request, with instructions to be honest rather than kind. Elba is published by Meanwhile VOF, Amsterdam · elba.works