COFFERSHIELD

Security Architecture Whitepaper

The cryptographic design, the trust boundaries, the threat model, and — at equal length — what this design does not protect you from.

DOCUMENT VERSION	APPLICATION	BUILD REVIEWED	AUDIENCE
1.0	CofferShield 1.0.0	4d8f227422011dfd…	Technical

This document describes how CofferShield protects data, on the assumption that its reader will want to check rather than believe. Every parameter stated here is a property of the shipped build and can be read out of it: the application's own code is one unminified, unobfuscated inline script, and the vault format is published in §5.3. Where a claim rests on something the reader cannot verify from the artifact — a CI secret, a hardware behaviour, a platform API — this document says so rather than asserting it.

> **What this document is not**
>
> It is not an audit. No independent party has reviewed this design or this code, and the roadmap in §17 says when that is intended to change. It is not a proof of security; it is a description of a design, its assumptions, and its failure modes, written so that a reader can judge whether the assumptions hold in their own situation.

CONTENTS

1 Executive summary

CofferShield is a personal vault for the records a household needs and rarely has in one place: what it owns and owes, insurance, documents, passwords and recovery phrases, and an estate plan — who inherits, how each asset actually transfers, and which incapacity documents exist and where they are kept. It is sold once and installed on a computer. There is no account to create, no server to sign in to, and no copy of anything held by the vendor.

All of it lives in one file, encrypted with AES-256-GCM under a key that exists only while the application is unlocked, derived from a password only the owner knows.

Why offline

The decision not to build a server was a threat-model decision, not a cost one. A synchronising vault has to hold ciphertext somewhere its operator can reach, which creates four exposures that no amount of client-side cryptography removes:

EXPOSURE A HOSTED VAULT CARRIES	WHY IT PERSISTS DESPITE END-TO-END ENCRYPTION
A breach yields every customer's ciphertext at once	Encrypted blobs are still an offline-attack corpus. One incident produces millions of files that can be ground against forever, and the weakest passwords in the set fall first.
The operator can be compelled	Not to decrypt, but to ship altered client code to a named user. Every browser-delivered or auto-updating vault has this property; the customer cannot detect it.
Metadata accumulates	Even a service that reads no plaintext learns who has an account, when they log in, from where, on what device, and how their vault grows.
The service is a dependency	If it goes away, is blocked, or has an outage, so does access to the records — often at the moment they are needed.

Removing the server removes all four. It also removes real conveniences — synchronisation, sharing, server-side recovery, and the ability to push a fix — and §14 and §17 are honest about that trade. The exchange is deliberate: the failure modes that remain are ones the owner can see and control, rather than ones held by a third party on their behalf.

Security philosophy

1. **State the parameters.** A design that has to be secret is not a design a reader can evaluate. Algorithms, iteration counts, key sizes and file formats are published here and are readable in the shipped file.
2. **Fail closed, and say when a control is not a control.** Several mechanisms in this product are conveniences that look like protections — the session PIN, Windows Hello, the clipboard timer. Each is labelled as such in the interface and in §9 and §12, because a control the owner overestimates is worse than one they know is thin.
3. **Do not create secrets the owner did not ask for.** No telemetry, no device identifier, no crash reports, no licence check. There is nothing to leak because nothing is collected (§11).
4. **The master password is the whole boundary.** Everything else — the lock timer, the biometric shortcut, the decoy — sits inside that boundary or beside it. If the password is weak, none of it matters, which is why §6 spends more space on password strength than on ciphers.

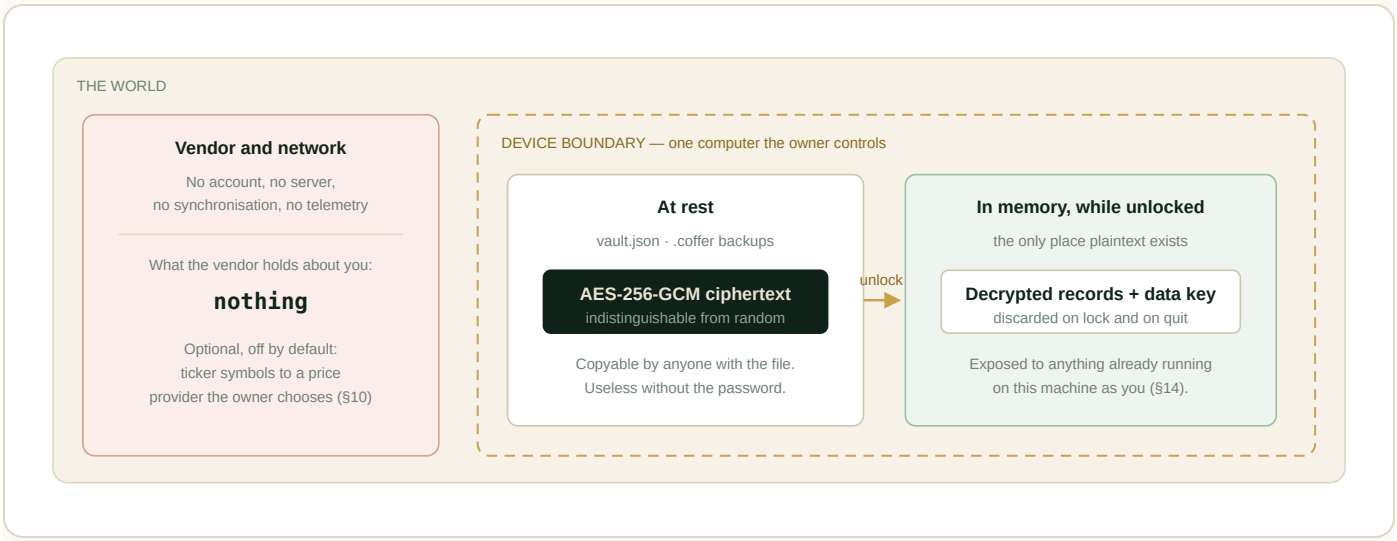

Figure 1 — Trust boundaries. There are two, and only one of them is enforced by cryptography. Everything to the right of the dashed line depends on the device itself being sound; §13 states that assumption plainly and §14 describes what happens when it fails.

2 Security principles

These are the rules the implementation is held to. Each is stated with the mechanism that enforces it and, where the principle is only partly achieved, the gap.

PRINCIPLE	HOLDS?	WHAT ENFORCES IT, AND WHERE IT STOPS
Local first	YES	Every feature works with the network cable out. The one exception is price refresh, which is off until configured and degrades to manually entered prices.
Offline by default	YES	Three independent gates keep the price feature off in a new vault; the host has no HTTP client at all, so all egress is webview-side and confined by `connect-src`.
No cloud storage	YES	No storage code paths exist other than the local file, IndexedDB and memory. A backup can be placed on a cloud drive by the owner; that is their decision, and the file is ciphertext.
No user accounts	YES	There is no registration, no licence check, no activation call, and no identifier generated at install.
No central server	YES	No CofferShield-controlled origin appears anywhere in the build, including in the CSP allowlist. There is nowhere for it to report to — though see §10.3 on what a policy of this shape does and does not close.
The owner holds the data	YES	One file the owner can copy, move, back up or delete. The export format is documented and opens on either platform.
Encrypted by default	YES	There is no unencrypted mode and no partial mode. A vault cannot be created without a password of at least 12 characters.
Least exposure	MOSTLY	The request payload for price lookups is enforced by an allowlist assertion, not by convention. The desktop host defines fifteen commands of its own, and the webview also holds the framework's default permission set — roughly a hundred built-in commands for window state, paths, events and images. §12.3 lists the fifteen and what constrains each. Nine touch the filesystem, and one of those constraints is still only partial.

PRINCIPLE	HOLDS?	WHAT ENFORCES IT, AND WHERE IT STOPS
Privacy by design	YES	Nothing is collected, so nothing is retained, so there is no retention policy to trust.
Zero knowledge	NOT APPLICABLE	The term describes a service that holds your ciphertext and cannot read it. CofferShield holds nothing, so the property is vacuous rather than achieved. Using the phrase would imply a service exists. See the note below.

On the phrase "zero knowledge"

In the password-manager industry this describes a server that stores encrypted data it is unable to decrypt. It is a meaningful claim there, because there *is* a server and the claim constrains what it can do. CofferShield has no server, so the honest statement is not "our servers know nothing" but **"there are no servers"** — which is a stronger position and a different one. This document avoids the term because borrowing it would suggest an architecture that is not present.

A principle this product does not claim

Defence against a compromised host. Some designs push key material into a hardware element so that malware on the machine still cannot extract it. CofferShield does not do this on either platform. The data key lives in the JavaScript heap of the application process while the vault is open, and anything running as the same user can read that process. §14 states the consequences.

3 System architecture

One process, one file, one key hierarchy. The diagram below is the whole product: there is no component off the page.

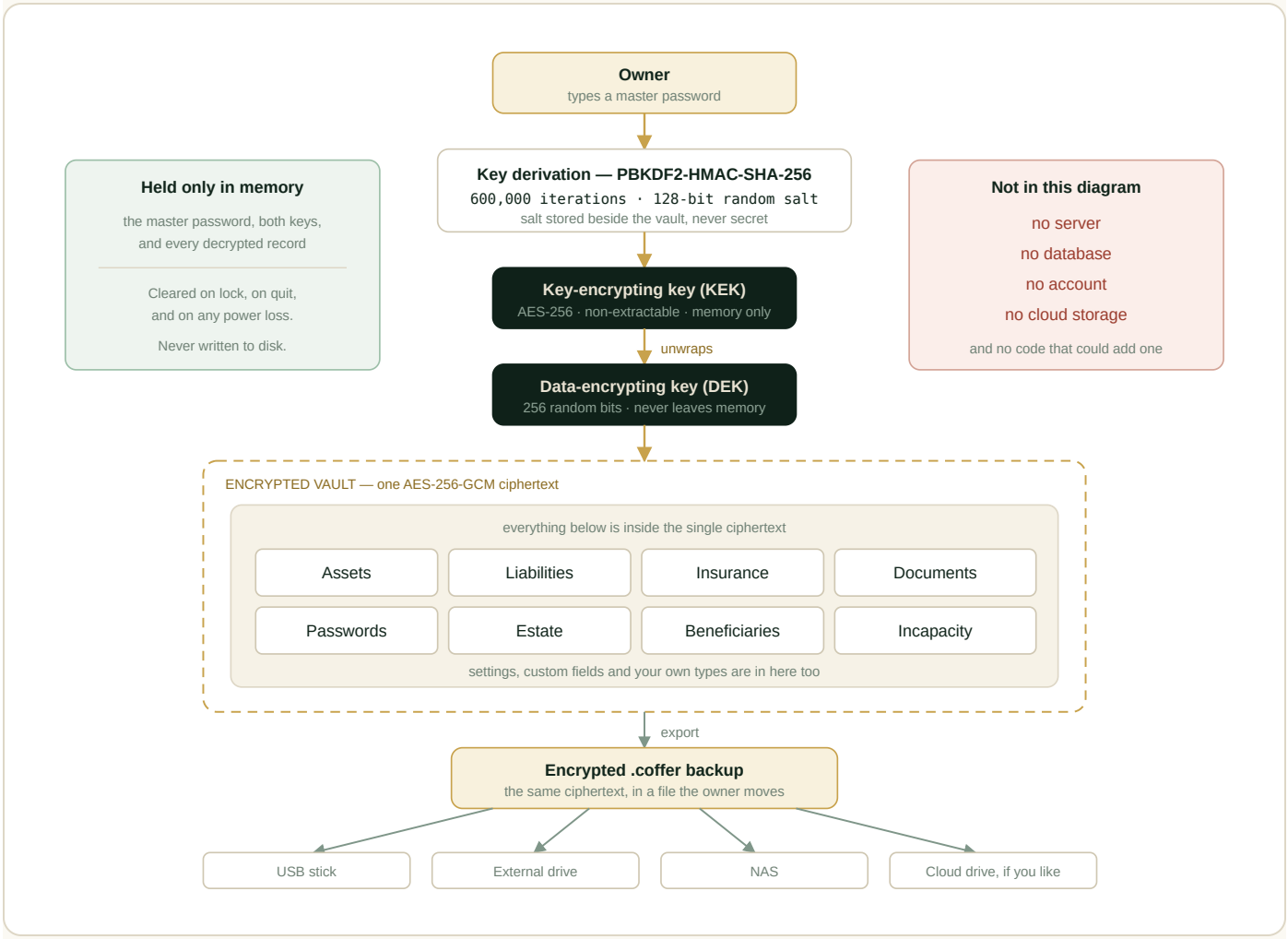

Figure 2 — System architecture. The password derives a key that unwraps a second key; only the second key touches your data. §5 explains why the split exists and what it buys.

3.1 Components

COMPONENT	WHAT IT IS	SECURITY ROLE
Application	One HTML file. The application's own code is a single unminified inline script; the file also carries a bundled copy of Mozilla's pdf.js, minified third-party code used only to draw document previews and never fetched at runtime. The shipped file is assembled from source fragments by a small splice script.	Everything cryptographic happens here, through the platform's WebCrypto implementation. Nothing is loaded at runtime, so there is no supply chain at run time — only at build time.
Desktop shell	A Tauri v2 host in Rust that owns the window, the file writes, the native dialogs and the Windows Hello prompt.	The filesystem boundary. The webview cannot open a file on its own; it asks the host, and the host validates what it is asked (§12.3).
Platform webview	WebView2 (Chromium) on Windows, WKWebView (WebKit) on macOS.	Supplies WebCrypto, and is the largest piece of code in the product that CofferShield did not write. It is also the only component that receives security updates independently — see §14.6.
Vault file	`vault.json` in the per-user application data directory. One JSON envelope holding two ciphertext slots.	The only durable artifact. Written atomically; owner-only permissions where the platform has a mode bit.
Backup files	`.coffer` files wherever the owner puts them.	Byte-for-byte the same ciphertext as the live slot, with a small cleartext header (§8.2).

3.2 The trust boundary, stated precisely

A boundary is only useful if both sides are named. CofferShield's cryptographic boundary is **the master password**, and it separates these two sets:

	INSIDE THE BOUNDARY (PROTECTED BY THE PASSWORD)	OUTSIDE IT (NOT PROTECTED)
Data	Every record, document, password, recovery phrase, name, amount, note, setting, custom field and estate decision.	The existence of the file, its size to within 64 KiB, and its modification time.

	INSIDE THE BOUNDARY (PROTECTED BY THE PASSWORD)	OUTSIDE IT (NOT PROTECTED)
Threats	Someone who obtains the file, the disk, the backup, the cloud folder it was copied to, or the whole computer while it is locked.	Someone who obtains the computer while it is unlocked, or who is already executing code on it as you.
Time	While the application is closed or locked.	While the application is open.

4 Threat model

The table below is the centre of this document. It is written so that the NO rows are as easy to find as the YES rows, because a reader deciding whether this product fits their situation needs the second list more than the first.

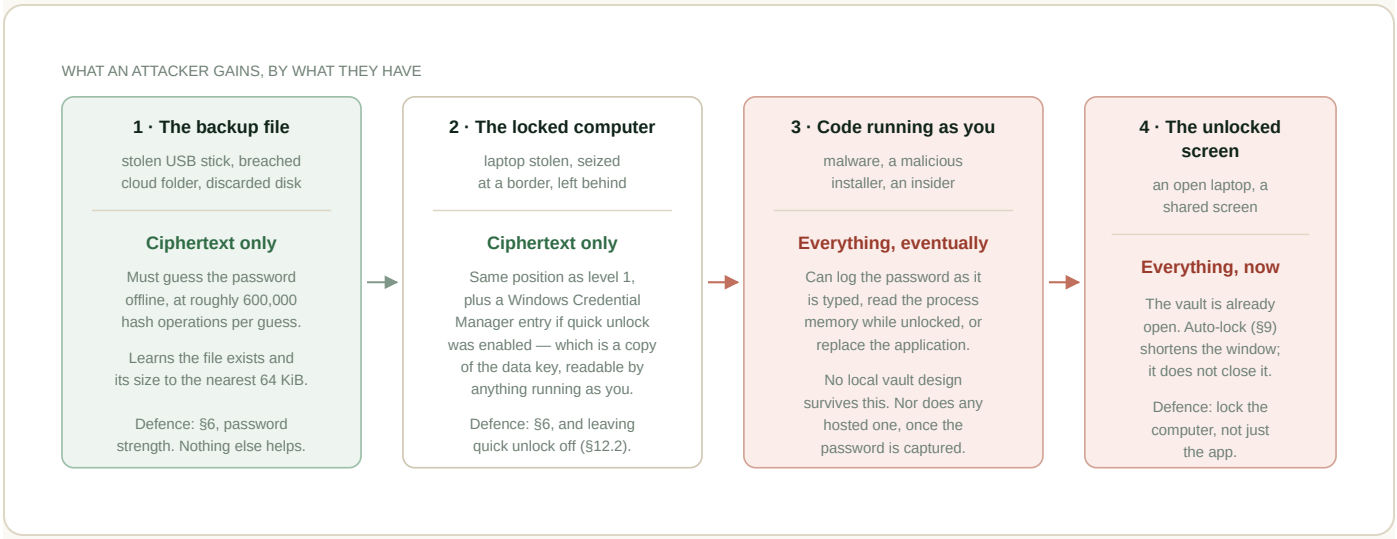

Figure 3 — The escalation ladder. Cryptography covers levels 1 and 2 completely and levels 3 and 4 not at all. The design decision that matters is where the line sits, and it sits between the second box and the third.

4.1 The table

THREAT	PROTECTED	WHAT ACTUALLY HAPPENS
Lost or stolen laptop device gone, vault closed	YES	The thief has ciphertext. Recovering it means guessing the master password against a 600,000-iteration KDF, offline. §6 gives the arithmetic. If quick unlock was enabled, see the qualification in §12.2 — the stored key changes this answer.
Stolen backup drive	YES	A `.coffer` file is the same ciphertext as the vault. Identical position to the row above. This is why backups can safely be left in places the owner does not fully control.
Cloud storage breach a backup on Dropbox, iCloud, OneDrive	YES	The provider holds an opaque blob. There is no key on their side, no recovery mechanism, and nothing for a support agent to be socially engineered into releasing.

THREAT	PROTECTED	WHAT ACTUALLY HAPPENS
Vendor database breach	YES	There is no database. This row cannot be true of CofferShield in the way it can be of a hosted product, because the asset that would be breached does not exist.
Server compromise including a compelled operator	YES	No server, and no update channel through which altered code could be delivered to a targeted user. §14.6 explains the cost of that second property.
Password reuse your password leaked from another site	YES	The master password is never transmitted, so it cannot appear in someone else's breach unless you also used it elsewhere. Credential-stuffing has no endpoint to attack. Reusing the master password on a website removes this protection entirely.
Phishing	YES	There is no login page to imitate and no account to hand over. A convincing fake CofferShield website has nothing to collect. The residual risk is being tricked into installing a counterfeit *application* — which is a supply-chain problem, and §14.7 is candid that the unsigned Windows installer makes it a real one.
Network interception	YES	No vault data crosses the network at any point. With price refresh on, the only outbound content is ticker symbols and the owner's own API key, over HTTPS, to a host on a fixed allowlist (§10.2).
Ransomware encrypts or destroys your files	PARTLY	It cannot *read* the vault without the password. It can certainly encrypt or delete the file, which is a loss of availability, not confidentiality. An offline backup is the only defence and it is a complete one. Note that ransomware is code running as you, so §14.1 applies to anything it does while the vault is open.
Physical attacker, vault locked	YES	Equivalent to the stolen-laptop row. Cold-boot and DMA attacks against a machine that was powered on with the vault open are a real class and are not addressed here; full-disk encryption with a pre-boot password is the mitigation, and it is the operating system's to provide.
Physical attacker, vault open	NO	The data is on screen. Auto-lock and lock-on-hide reduce the window (§9.2) but an attacker in front of an unlocked session has the vault.
Keylogger	NO	Captures the master password as it is typed. Everything follows. No local application can prevent this, and one that claims to is describing an operating-system feature it does not control.

THREAT	PROTECTED	WHAT ACTUALLY HAPPENS
Memory-scraping malware	NO	While the vault is open, the decrypted records and the data key are in the process heap. Keys are held as non-extractable WebCrypto objects and raw key bytes are zeroed after use, which raises the effort slightly; it is a speed bump, not a boundary.
Clipboard malware	PARTLY	A copied password is cleared after a configurable delay, 45 seconds by default. That closes the window for an opportunistic clipboard reader; it does nothing against one that polls continuously. Windows clipboard history retains a copy the application cannot reach — the interface says so at the point of copying.
Remote-access trojan	NO	Equivalent to an attacker sitting at the keyboard, with the addition that they can wait for you to unlock. See §14.1.
Screen recording / shoulder surfing	NO	Secrets are masked until revealed and re-mask after twenty seconds, which is a defence against a glance rather than a camera. The window is not marked as capture-protected, so screenshots and screen-sharing show its contents; §17 lists this as candidate work.
Malicious or crafted backup file	YES	A backup that has been altered will not decrypt: AES-GCM authenticates the ciphertext, so tampering produces a failure, not a wrong answer. Content arriving inside a valid backup is treated as untrusted input and is escaped and bounded on the way in — see Appendix B, which records that this was not fully true before this review.
Evil-maid firmware or bootkit	NO	An attacker who can modify the boot chain can replace the application. Secure Boot and full-disk encryption are the relevant controls and belong to the platform.
Nation-state adversary	NO	An adversary who can compel a device, compromise a supply chain, or apply targeted implants defeats this design, as it defeats every consumer product. What CofferShield does remove is the *bulk* collection surface: there is no repository of customers to serve one order against, and no operator to serve it to. Targeted access remains possible; scalable access does not exist.
Legal compulsion of the owner	OUT OF SCOPE	The decoy vault (§7.4) exists for the case where someone is compelled to open the application. Whether it helps is a legal question, not a technical one, and it varies by jurisdiction. Nothing in this document should be read as advice on that point.

The one-sentence version

CofferShield protects **data at rest** against anyone who obtains the file, and it does not protect **a computer that is already compromised or already unlocked**. If your threat model is the first, this design is a good fit. If it is the second, no vault product solves it and you need to fix the machine.

5 Cryptography

Nothing here is novel. Every primitive is a standard construction from the platform's own cryptographic library, used in a documented mode, with parameters chosen conservatively. That is the intended impression: a personal vault is not the place to be interesting.

5.1 Primitives

PURPOSE	CONSTRUCTION	PARAMETERS
Bulk encryption	AES-256-GCM	256-bit key, 96-bit IV drawn fresh per encryption, 128-bit authentication tag
Key derivation	PBKDF2-HMAC-SHA-256	600,000 iterations, 128-bit random salt, 256-bit output
Key wrapping	AES-256-GCM	the derived key encrypts the 32-byte data key as an ordinary message
Integrity	GCM authentication tag	verified on every decryption; a failure is a failure, never a partial read
Randomness	The platform CSPRNG via `crypto.getRandomValues`	used for every key, salt, IV, padding byte, recovery key and generated password
Recovery key splitting	Shamir secret sharing over GF(2^8)	threshold *k* of *n*, one polynomial per byte, information-theoretically secure below the threshold
Human-transcribable codes	Crockford base32 with a Fletcher-16 tail	no I, L, O or U, so a handwritten 1 or 0 cannot be misread; the checksum catches a typo before anything is attempted

Why PBKDF2 and not Argon2id or scrypt

Argon2id is the better function. It is memory-hard, which is precisely the property that degrades a GPU attack, and if CofferShield could use it, it would. It is not available in WebCrypto, so shipping it would mean either a WebAssembly implementation carried inside the file — new code in the most security-critical position in the product, unaudited, with its own supply chain — or a native dependency that would have to be trusted equally on two platforms.

The judgement made here was that a well-reviewed platform PBKDF2 at 600,000 iterations is a better risk than an unreviewed memory-hard function that this project would be maintaining itself. 600,000 is the current OWASP recommendation for PBKDF2-HMAC-SHA-256 and matches what mainstream password managers use for the same primitive. The trade is stated so a reader can disagree with it: PBKDF2 parallelises well on a GPU, and that is the reason §6 asks for a longer password than a memory-hard design would need.

5.2 The key hierarchy

The password does not encrypt your data. It encrypts a key, and that key encrypts your data. This two-level structure is why changing the master password on a vault holding hundreds of megabytes of scanned documents is instantaneous, and it has a security consequence worth stating: **the data key never changes**, so anyone who captured the ciphertext before a password change can still open it if they later learn the *old* password.

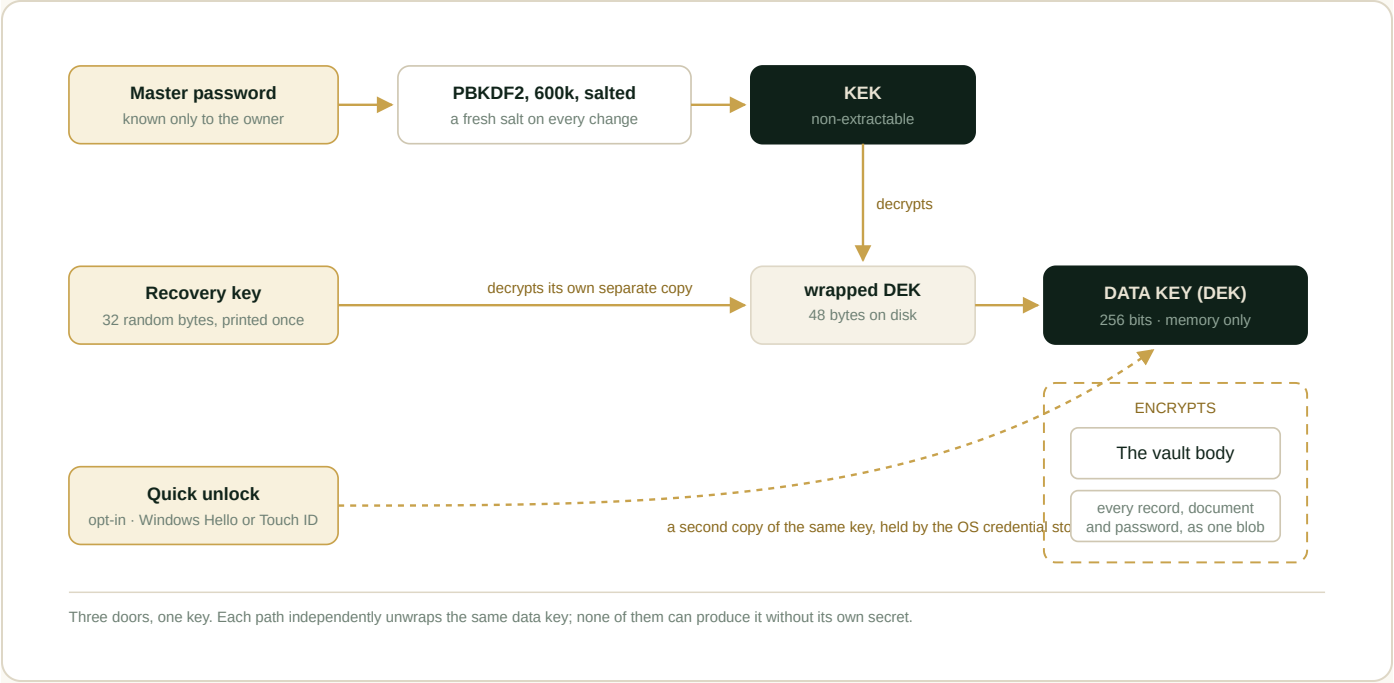

Figure 4 — Key hierarchy. Three ways in, one key out. Adding a recovery key or enabling quick unlock adds a door; §14.4 treats that as the trade-off it is.

5.3 The vault record

A vault file is a small JSON envelope containing two slots of identical shape. Everything in a slot except the ciphertext is public by design — a salt and an IV are not secrets, and treating them as such would be a sign of a design that had not been thought through.

Figure 5 — Vault file layout. The only secret in the file is the key you carry in your head. Everything else is published here.

5.4 Unlocking

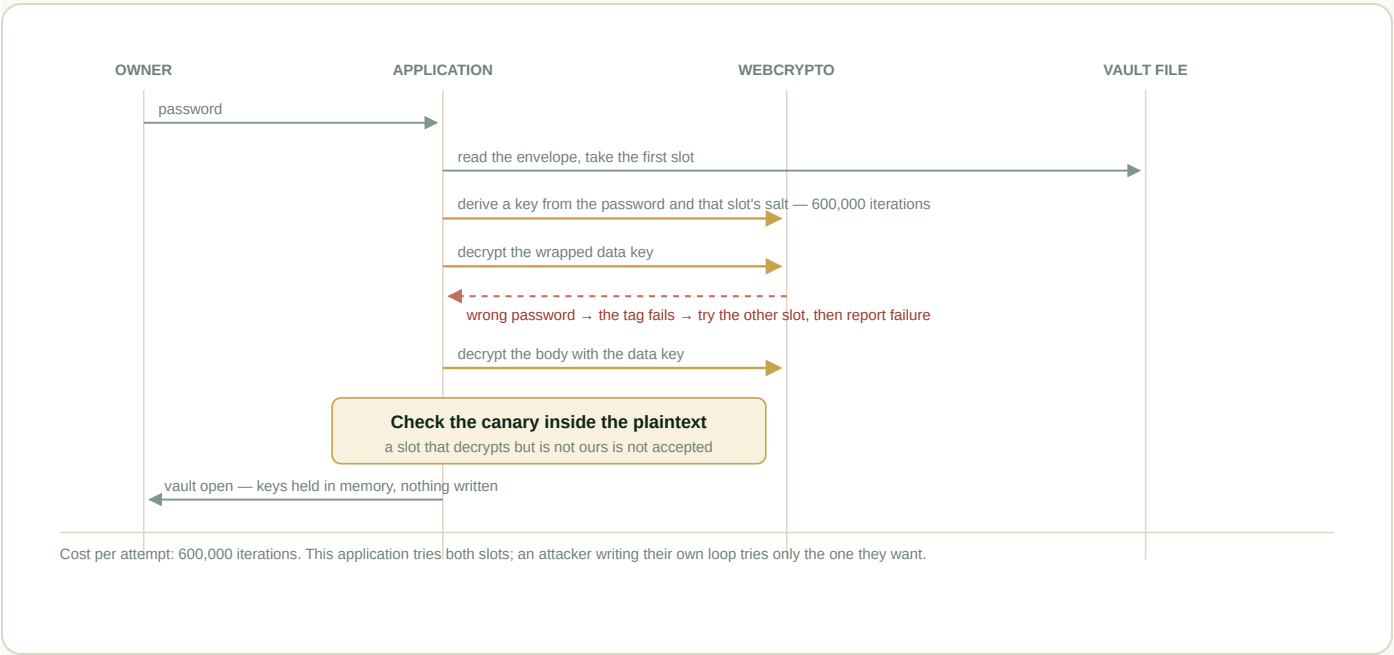

Figure 6 — Unlock sequence. Both slots are attempted with the same password, which is what allows a decoy to exist without any marker in the file saying so.

> There is no attempt limiting, deliberately
>
> CofferShield does not count failed attempts, delay retries, or wipe after *n* failures. It would be theatre. An attacker who has the file does not have to use this application to attack it — they write their own loop against the format published in §5.3. Attempt limiting only constrains someone who is politely using the front door, and the person you are worried about is not.
>
> The real control is the KDF work factor, which applies to every attempt by anybody, in any tool. A wipe-after-*n* feature would add a way to destroy your own vault by mistyping, which is a genuine risk in exchange for an imaginary one.

5.5 Saving, and why changing the password is instant

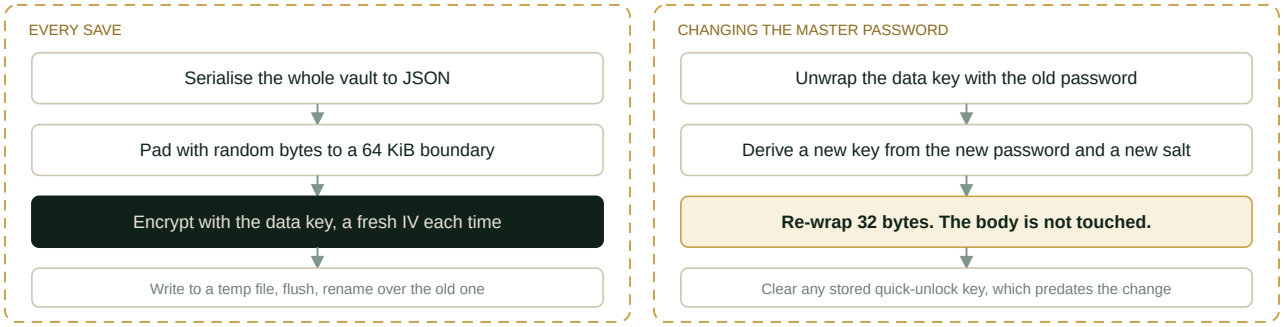

Figure 7 — Save versus re-key. A password change rewrites 48 bytes, so it takes the same time on a 2 MB vault as on a 200 MB one. The consequence is in §5.2: old ciphertext stays openable with the old password.

Two details in that diagram matter more than they look:

- **A fresh IV on every encryption.** GCM fails catastrophically if an IV is reused with the same key — it leaks the XOR of the plaintexts and can expose the authentication subkey. Every IV here is 96 fresh bits from the platform CSPRNG, never a counter, never derived.
- **Temp file, flush, rename.** A save that is interrupted by a crash or a power cut leaves the previous good vault in place, rather than a truncated file where a vault used to be. The flush before the rename is the part that makes this true on real hardware rather than only in theory.

5.6 What the cryptography does not do

- **It does not hide that a vault exists.** A file named `vault.json` in the application data directory is visible to anyone browsing the disk.
- **It does not hide the approximate size.** Padding rounds to 64 KiB, so a vault with one record and a vault with two hundred are usually indistinguishable — but a vault holding 300 MB of scanned documents is visibly a large vault.
- **It does not authenticate the application to you.** Nothing in the format proves that the program you are typing your password into is the real one. That is the job of code signing, and §14.7 explains where that currently stands.
- **It does not provide forward secrecy.** There is no ratchet and no re-keying of the body; one data key encrypts the vault for its whole life.

6 The master password

Every other control in this document is secondary to this one. An attacker with your vault file has exactly one problem to solve, and this is it.

6.1 Why it cannot be recovered

The password is never stored, never transmitted and never derivable from anything on disk. What the vault holds is a salt and a ciphertext; there is no verifier, no hash of the password, and no hint. A correct password produces a key that decrypts the wrapped data key and passes the authentication tag. A wrong one produces a key that fails it. Nothing in that process can be run backwards.

This is not a policy that could be relaxed on request. There is no vendor-held key, no escrow, no support process, and no code path that would accept one. A "reset my password" feature would require a second copy of the data key held by someone other than you — which is precisely the property this architecture exists to avoid.

> The consequence, stated once, plainly
>
> If you forget your master password and have not made a recovery key, the vault cannot be opened. Not by you, not by the vendor, not by anyone with a warrant. The data is gone. **Write the password down and store it physically before entering anything real.** A sealed envelope with your will is a good place. Your memory alone is not.

6.2 Entropy, and what the numbers actually mean

Password strength is a statement about how many guesses an attacker must make, and it is only meaningful against a stated attack rate. PBKDF2-HMAC-SHA-256 parallelises well on GPUs. The figures below use two reference adversaries:

- **A determined individual** — eight high-end consumer GPUs. At 600,000 iterations this is on the order of **10^5 guesses per second**.
- **A well-funded organisation** — roughly a thousand such GPUs, giving about **10^7 guesses per second**.

These are order-of-magnitude figures derived from published benchmarks, offered so the arithmetic can be checked rather than trusted. Hardware improves; treat the right-hand column as the number that matters.

WHAT YOU CHOOSE	ENTROPY	INDIVIDUAL, 10^5/S	ORGANISATION, 10^7/S
A word and two digits	~20 bits	seconds	instant
A memorable phrase from a book or song	~25 bits	minutes	instant
Three random words from a 2,048-word list	~33 bits	~12 hours	~7 minutes
Four random words from a 2,048-word list	~44 bits	~2 years	~8 days
Four words from CofferShield's own list 269 words, about 8 bits each	~32 bits	~1 hour	under a second
What CofferShield suggests — seven words plus a number	~66 bits	~20 million years	~200,000 years
A generated 20-character random string	~110 bits	beyond any projection worth writing down	

Two things in that table matter. The first is that four words is a common recommendation, is fine for an account behind a rate limit, and is **not** fine for a file an attacker can copy and grind at leisure. The second is that "a random word" is not a fixed quantity: it is worth about eleven bits drawn from a 2,048-word list and about eight from CofferShield's own short, deliberately plain one. That is the whole reason the built-in suggestion is seven words rather than four.

> What the built-in generator produces
>
> Pressing **Suggest one** when creating a vault produces a seven-word passphrase from the application's own wordlist, with a two-digit number attached to one word — approximately 66 bits. The list is short and deliberately plain, which is why the count is seven rather than the four or five a larger list would need. The generator inside the vault, used for website passwords, defaults to a 20-character random string, and to five words if you switch it to phrase mode. A site password is guarded by that site's rate limiting; the master password is guarded by nothing but its own length.

6.3 Choosing one

DO	WHY
Use six or more random words, or accept the suggestion	Length beats complexity against an offline attack. A long phrase of ordinary words is both stronger and easier to type correctly under stress than a short jumble of symbols.
Let the machine choose the words	People asked to pick "random" words pick related ones, and a phrase drawn from a life is a phrase drawn from a small set.
Write it down and store it physically	The realistic failure of this product is not a broken cipher. It is a person who cannot remember a password they chose two years ago.
Never reuse it anywhere	Reuse re-imports every risk this architecture removed. A breach elsewhere becomes a breach here.
Consider a recovery key as well	§7.5. It is a second door, so it must be stored like a deed rather than in a drawer — but it converts "the vault is gone" into "fetch the envelope".

The minimum accepted length is twelve characters. That is a floor against obvious mistakes, not a recommendation; twelve characters of ordinary text is well under 40 bits.

7 Vault architecture

7.1 One boundary, not several

A common design puts different data classes in different containers — documents here, passwords there, settings in the clear. CofferShield does not: there is exactly one ciphertext, and everything is inside it. That choice costs some flexibility and buys a property worth more, which is that there is no list to audit of what happens to be unencrypted this release.

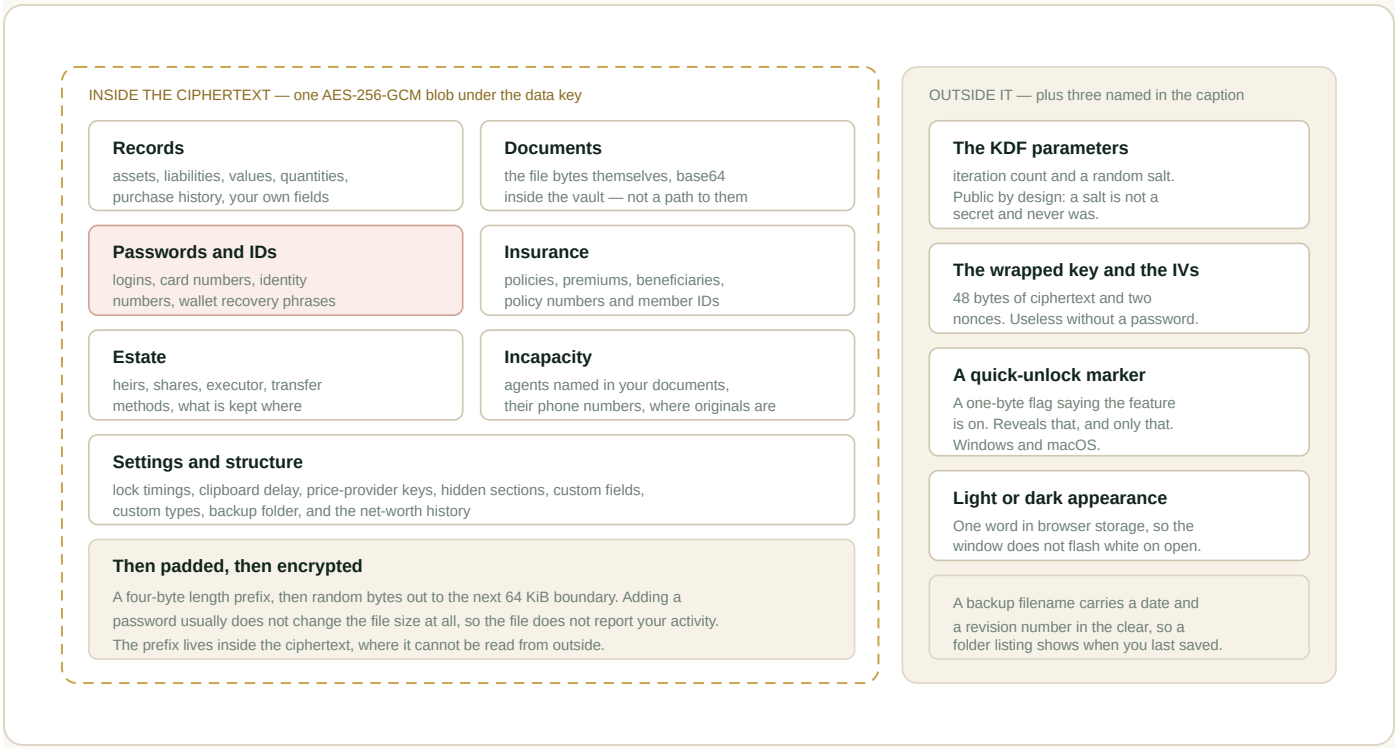

Figure 8 — The encryption boundary. Three further items sit outside the ciphertext and are named elsewhere rather than in the panel: the presence of a recovery wrap on a slot, which identifies the real one (§7.4); each slot's payload size to the nearest 64 KiB, which is not the same as the file's size; and the device label and time of day in a backup envelope (§8.2). Everything you enter is inside the ciphertext.

7.2 Where the vault lives

WHERE	WHICH EDITION	WHAT IT IS, AND WHAT IT IS NOT
A file in the per-user application data directory, written by the desktop host	Desktop	**The single authority.** Survives an uninstall and survives "clear browsing data", neither of which is an acceptable way to lose a vault.
IndexedDB inside the webview	Portable	The portable edition has no host, so IndexedDB is *its own* authority. It is **not a fallback for the desktop edition.**
IndexedDB, read once, for an upgrade	Desktop, first run only	A vault left by an older browser-era build is lifted to the file — but only when the file is genuinely *empty*. It is validated before being promoted, the write is read back and compared byte for byte, and only then is the old copy removed.
Memory	Both	Holds the working vault between saves. It is not a tier the application falls back to.

> **There is deliberately no fallback.** If the vault file cannot be read or cannot be written, the application *says so and stops*. It does not quietly open an older copy from somewhere else, and it does not report a save as successful when the file on disk still holds the previous revision. A read fallback is worse than a plain failure: a failure is visible, whereas older data presented as current is not. A storage error names which category of problem occurred — permissions, disk full, read-only, missing folder — states that nothing has been changed or deleted, and offers to try again. It never offers to create a new vault.

Writes are atomic: a temporary file, flushed to the platter, then renamed over the previous one. Where the platform has a permission mode the file is created owner-only; on Windows it inherits the ACL of the application data directory, which this application cannot improve on without taking a dependency on security-descriptor code.

7.3 Documents

An attached document is stored *inside* the vault, not referenced from it. The bytes are encrypted with everything else. Previewing a document decrypts it into memory and

renders it from an in-memory blob; **no temporary plaintext file is written to disk** at any point in that path. Nothing in adding, previewing, storing or downloading a document touches the network, and a test fails the build if any network primitive appears in that part of the source.

Being inside the one ciphertext is what sets the size limits, and the arithmetic is worth stating because it is not a policy anyone chose. A file becomes base64 inside the body, the body is encrypted, and the result is base64 again into the vault file — so raw bytes reach the disk multiplied by about 1.78, and every one of those has to be a single JavaScript string, which the engine caps at 536,870,888 characters. **64 MB per file and 256 MB of documents per vault** sit inside that with room to spare, and the application refuses a file that would take the vault past what it can re-open rather than letting the ceiling be discovered on a later unlock.

The cost of a large document is paid on every save, not only when it is added, because the whole vault is re-encrypted each time. A 64 MB file adds several seconds to every save of anything at all. That, rather than the format, is the reason to keep a scanning habit modest — and the reason a genuinely large archive belongs in its own encrypted container beside the vault rather than inside it.

Printing is the exception, and it is not one this application can close: `window.print()` hands a rendered page to the operating system's print subsystem, which spools it to disk outside the application's control. §14.5 covers this.

7.4 The decoy vault

Every vault file contains two slots. One is yours. The other is created at the same moment, with a random 24-byte password nobody ever sees, unless the owner deliberately sets a decoy password — in which case that password opens a separate, harmless vault.

Both slots have the same field structure and are written in random order, and unlocking simply tries each in turn and takes the one that decrypts and passes its canary. A second slot is always present whether or not it is in use, so the presence of one answers nothing.

They are not indistinguishable to someone examining the file. Two things separate them. **Size:** padding rounds each slot to a 64 KiB boundary, not to a common size, so a real vault holding a scanned deed sits beside a decoy of a single bucket. **The recovery wrap:** if you have made a recovery key, only the slot it belongs to carries that field. The

application warns about the first of these where you set a decoy password; this document states both.

> What the decoy is and is not
>
> It is a way to open *something* when someone is watching. It is not proof that nothing else exists, and as the paragraph above says, an examiner holding the file can usually tell the two slots apart by size. Treat it as protection against being made to open the application, not against forensic examination of the file. Whether producing a decoy helps or harms in a given legal situation is a question for a lawyer in that jurisdiction, and this document takes no position.

7.5 Recovery keys, and splitting one

A recovery key is 32 random bytes generated on the device, shown exactly once, and never stored. What is stored is a third wrapped copy of the data key, encrypted under those bytes — structurally identical to the copy the password unwraps.

The same 32 bytes can instead be split into *n* shares of which any *k* reconstruct it, using Shamir secret sharing over GF(2^8). The property that makes this safe to hand to relatives is that fewer than *k* shares reveal **nothing at all** about the key — not "less information", literally nothing. That is information-theoretic, not computational: it does not depend on any assumption about the attacker's resources.

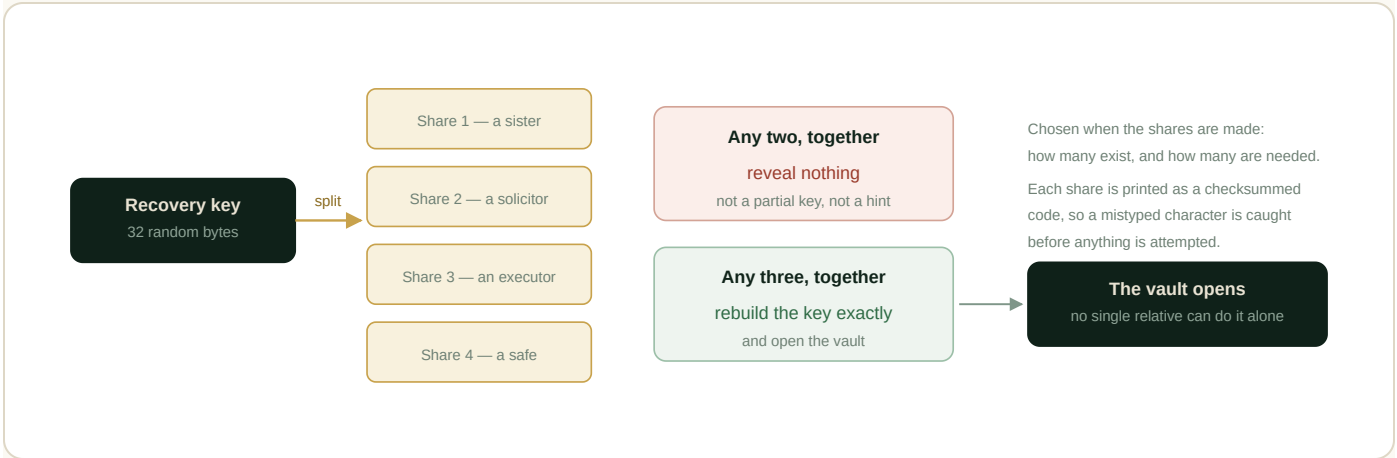

Figure 9 — Threshold recovery. Illustrated as three-of-four. The threshold and the number of shares are both chosen by the owner.

A recovery key is a second door

Anyone holding the printed sheet can open the vault without your password. It should be stored where you store a passport or a deed, not in a desk drawer, and it should be destroyed when replaced. Making one is a deliberate trade of confidentiality for availability, and it is the right trade for most households — but it is a trade.

8 Backup architecture

The backup is not a second format. It is the live slot, copied out.

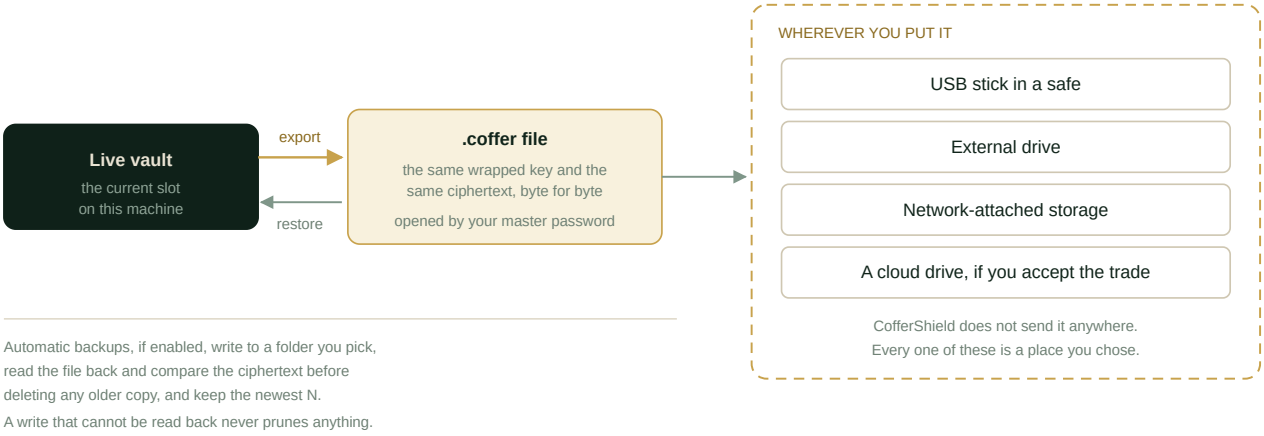

Figure 10 — Backup and restore. Because the backup is the slot itself, there is no second password to lose and no second format to get wrong.

8.1 Properties

PROPERTY	DETAIL
No cloud dependency	Export writes a file through a native save dialog. Nothing is uploaded, and there is no code that could upload it.
Cross-platform	A backup written on Windows restores on macOS and the reverse. The format carries its own version and KDF parameters, so a file written by an older build opens in a newer one.
Opened by the master password	No separate backup password to forget. The file carries the same wrapped key as the vault, so the password that opened the vault when the backup was taken opens the backup.
Tamper-evident	Modifying a backup makes it fail to decrypt. GCM does not produce a plausible-but-wrong plaintext.
Verified before pruning	Automatic backups read the file back and compare ciphertext before removing an older one, and never remove the last remaining backup.

A backup is frozen at the password of its day

Because the wrapped key travels with the file, a backup taken before a password change still opens with the **old** password. That is convenient when you have forgotten a recent change and dangerous when you changed the password because it had been exposed. If you change your master password for a security reason, take a fresh backup and destroy the older ones.

A backup file recovers your password without touching your records

Because a backup is the slot verbatim, it carries the same wrapped data key the vault has. When you have forgotten your password, that is all CofferShield takes from it: the key is unwrapped with the password that was current when the file was written, checked against the vault actually on the device, and then re-wrapped under a new password you choose. Forty-eight bytes of key envelope change. **Nothing in the vault is replaced** — every record entered after that backup was written is still there, and your recovery key and shares keep working, because the data key never changed.

Setting a vault up from a file is a different operation, and nothing can reach it from inside a vault

Installing a backup on a machine that has no vault builds a new slot around a **new data key**. A recovery key wraps the key it was made for, so any sheet printed before that file was written will not open the new vault; you are asked to set a password and offered a replacement key. That path has exactly one entrance — the first-run screen, which appears only when there is no vault on the device — so it cannot overwrite anybody's records. **Nothing reachable from an existing vault, including every branch of the forgot-password flow, can replace your records with a file.** When a file's key cannot open the vault in front of it, the application says so and stops: it does not offer to install the file over the top, because someone trying to recover a password is in no position to weigh that trade.

8.2 What a backup reveals before it is opened

Three fields sit in the clear in the envelope so that a restore screen can describe a file before asking for a password: the time it was created, a revision counter, and a device label. The filename carries a date and the same counter. Someone with the file learns roughly when you last used CofferShield. They learn nothing about what is in it.

9 Session security

Everything in this section is about the window between unlocking and locking. None of it is cryptography; all of it is about narrowing that window.

9.1 Key lifetime

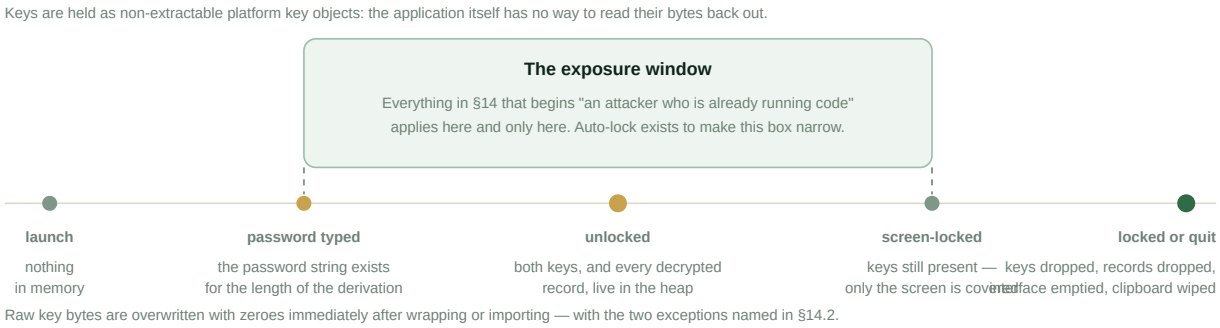

Figure 11 — Key lifetime. Non-extractable keys and zeroed buffers raise the effort of a memory attack; they do not prevent one, and §14.2 names two paths where a copy is deliberately kept alive.

9.2 The three lock states

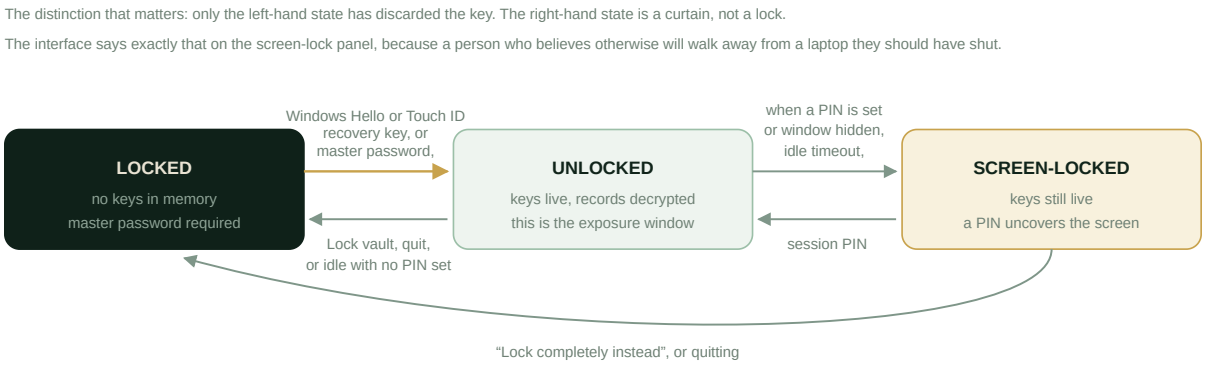

Figure 12 — Lock states. Screen-locked is a convenience state and is described as one wherever it appears.

9.3 The controls, and how strong each one is

CONTROL	DEFAULT	WHAT IT DOES, AND WHAT IT IS WORTH
Auto-lock on idle	`5 minutes`	Discards the keys after inactivity. Click, keypress and pointer activity reset the timer; scrolling and mouse movement alone do not, so reading a long page can trip it. Selecting "Never" now genuinely means never — see Appendix B.
Lock when hidden	`5 minutes`	Starts a timer when the window is hidden or minimised, and cancels it if you come back. This is the control that covers walking away with the app behind a browser.
Session PIN	`off`	**A convenience, not a security control.** It covers the screen and is compared as a plain string in memory. It is never written to disk and cannot decrypt anything. With a PIN set, an idle timeout covers the screen rather than dropping the keys — which is a deliberate weakening of auto-lock in exchange for not retyping a long password all day. The interface says so.
Lock vault	—	Immediate. Drops both keys and all decrypted data, empties the interface, wipes the clipboard, and clears any pending recovery key material.
Clipboard clearing	`45 seconds`	Overwrites the clipboard after the delay, and immediately on lock. **Best-effort:** an operating system can refuse a clipboard write from an unfocused application, and closing the app before the timer fires leaves the value in place. Windows clipboard history keeps a copy this application cannot reach.
Masked values	`always`	Passwords, card numbers, routing numbers, identity numbers and security answers are masked until revealed, and re-mask automatically after 20 seconds. A recovery phrase is the exception: it stays visible until you hide it, because writing down 24 words against a timer produces errors.
Copy versus reveal	—	In the passwords section, copying a secret schedules a wipe and copying an ordinary field does not — the distinction comes from the data model. Elsewhere it is chosen per button: a copied insurance policy number is not wiped, and its reveal has no 20-second timer.

Three honest notes on this section

One: every control here operates inside the process. None of them helps against software that is already reading that process. **Two:** auto-lock protects against an unattended screen, which is a real and common risk, and against nothing else. **Three:** locking CofferShield is not the same as locking your computer, and the second is the one that matters when you leave the room.

10 Offline architecture

The network is where most vault compromises come from, so the useful question is not "is the connection encrypted" but "how many bytes leave at all". For a default CofferShield installation the answer is **zero**: no request of any kind is made until the owner turns price sources on. This section describes that, and the optional exception in detail.

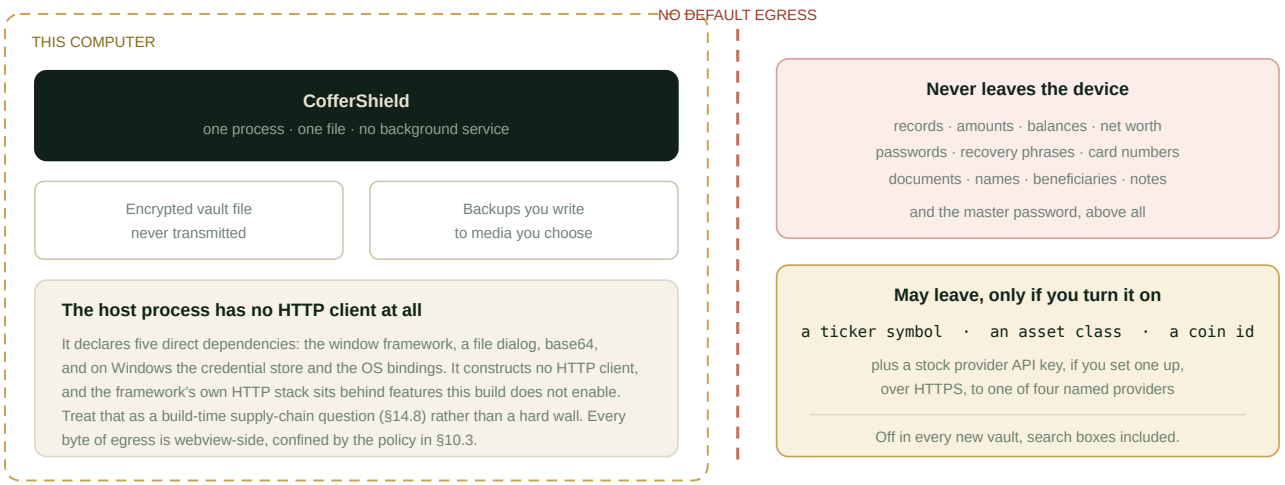

Figure 13 — Egress. The left column is what the product is; the right column is what can cross the line once the owner turns the feature on. Text typed into a symbol-search box goes to the same providers, under the same switches (§10.2).

10.1 Why an offline design reduces attack surface

ATTACK CLASS	REQUIRES	PRESENT HERE?
Credential stuffing, brute force against a login	An authentication endpoint	ABSENT
Session hijacking, token theft, CSRF	Sessions and cookies	ABSENT
Server-side injection, deserialisation, SSRF	Server-side code	ABSENT
Bulk data exfiltration	A repository of many customers	ABSENT

ATTACK CLASS	REQUIRES	PRESENT HERE?
Malicious update pushed to a targeted user	An update channel	ABSENT — and §14.6 explains the price of that
TLS interception of vault contents	Vault contents in transit	ABSENT
Third-party script or CDN compromise	Runtime-loaded code	ABSENT — nothing is fetched at runtime
Malicious dependency at build time	A dependency	PRESENT — see §14.8

10.2 The one exception, in detail

Price refresh updates the value of holdings that have a ticker or coin symbol, and it is off in every new vault. Stock prices additionally require a provider and an API key the owner obtained themselves; crypto prices go to CoinGecko, which works without a key, so that path is gated by the two switches rather than by three. The symbol-search box on a record form is gated by the same switches — it is a network request like any other, and it sends whatever you typed, which is not necessarily a ticker.

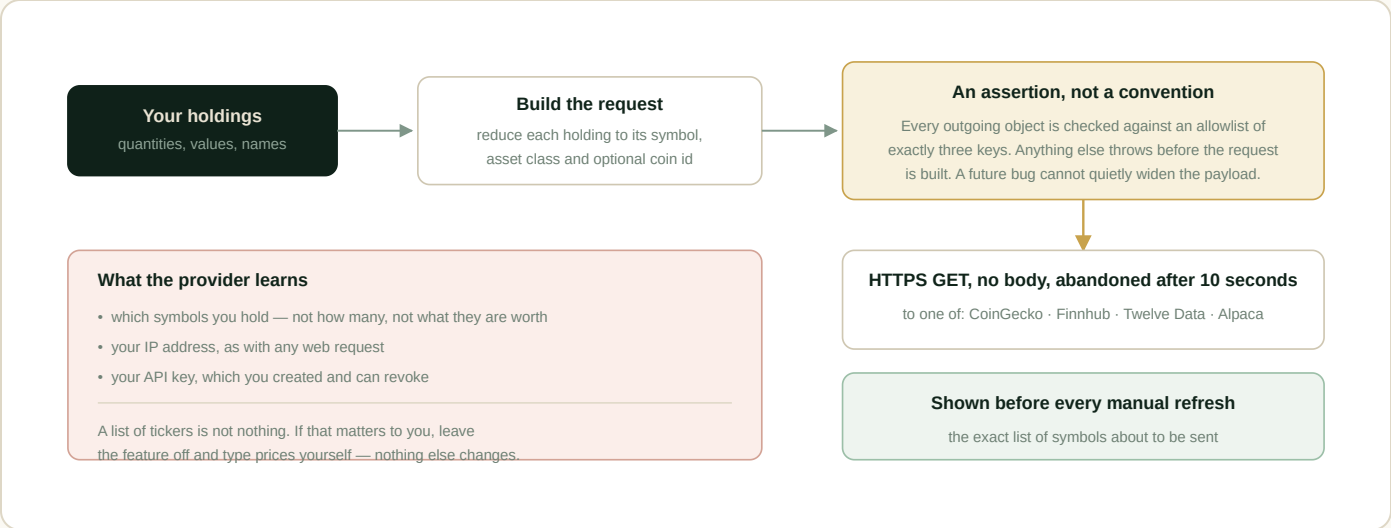

Figure 14 — Price refresh. The payload restriction is enforced in code and fails loudly, which is the difference between a design intention and a control.

10.3 The content security policy

The application ships with a policy, and the desktop shell delivers the same policy as a header, with one addition — `frame-ancestors`, which a browser ignores in a meta

element. Both are enforced when both apply, and a test fails if the two drift apart.

DIRECTIVE	VALUE	EFFECT
`default-src`	`'self'`	Nothing loads from anywhere else.
`script-src`	`'self' 'unsafe-inline'`	Structural: the application is one inline script with no build step to hash against. **This means the policy is not an XSS control** — see the note below.
`connect-src`	`'self'`, the IPC scheme, and four named provider hosts	The one directive that constrains network exfiltration. No wildcard, and no remote host beyond those four. `ipc:` is a scheme-only source and is the local bridge to the desktop shell.
`object-src`	`'none'`	No plugins, no embeds.
`base-uri`	`'none'`	An injected `<base>` cannot re-point relative URLs.
`form-action`	`'none'`	Closes form submission, which is an exfiltration channel `connect-src` does not cover.
`frame-ancestors`	`'none'`	Set by the shell, where it can be enforced. A browser ignores this directive in a meta element.
`style-src`	`'self' 'unsafe-inline'`	Also structural: the interface carries several hundred inline style attributes, which WebView2 blocks without this. `style-src-attr 'unsafe-inline'` is set alongside it.
`img-src`	`'self' data: blob:`	Covers the inline icon and in-memory document previews. No remote images, so no pixel can be used as a beacon.
`font-src`	`'self' data:`	No remote fonts.
`frame-src`	`blob:`	For in-memory previews. No remote frame can be created.

Being precise about what this policy is for

With `'unsafe-inline'` present, this CSP does not stop injected script from executing. Pretending otherwise would be the kind of claim this document is written to avoid. What it does is remove the value of executing: injected code cannot load a second stage, cannot post a form to an attacker, cannot open a connection to any host outside the allowlist, and cannot re-point the document's base. What it does **not** close is top-level navigation: no CSP directive covers that today, so script that could run could still send the window to an attacker's URL with data in the query string. The defence against injection itself is therefore output escaping in the application, which is tested — Appendix B records two places where it had failed and how they were closed.

11 Privacy

Each statement below was established by searching the shipped build for the mechanism that would be required, not by consulting a policy document. A reader can repeat every one of these searches.

CLAIM	STATUS	HOW IT WAS ESTABLISHED
No analytics	TRUE	The build was searched for every mainstream analytics and product-telemetry SDK by name. No match. There is no vendor-controlled origin in the CSP allowlist, so a beacon would be blocked by the shell even if one existed.
No telemetry or crash reporting	TRUE	No error-reporting service is present. The host has no logging framework and writes no diagnostic file. A crash produces nothing that leaves the machine.
No advertising, no trackers	TRUE	No third-party script, iframe, pixel or remote font. The application loads no external resource of any kind at runtime.
No account	TRUE	No registration, no licence check, no activation, and no identifier generated at install.
No synchronisation	TRUE	No storage code path other than the local file, the webview store and memory.
No background uploads	TRUE	Seven network call sites exist in total, all in the price module, all reachable only when the owner has enabled and configured it. There is no background worker and no scheduled task.
No update check	TRUE	No updater plugin, no endpoint, no public key for signed updates, and no version ping. §14.6 treats this as a limitation as well as a property.

The practical consequence is that CofferShield has no data-retention policy, because there is no data to retain. There is no subject-access request to make and nothing to delete, since nothing was ever collected. That is a stronger position than a promise about how collected data is handled, and it is the whole reason for accepting the costs listed in §14.

12 Platform security

12.1 The two platforms are not equivalent

	WINDOWS	MACOS
Webview engine	WebView2 (Chromium), updated by Microsoft independently of this app	WKWebView (WebKit), updated with the operating system
Vault location	Per-user application data	Application Support in the user's library
File permissions	Inherited from the parent directory ACL: your user, plus SYSTEM and administrators	Owner-only, set explicitly by the application
Biometric quick unlock	Windows Hello, opt-in, with the qualification in §12.2	Touch ID, opt-in. In its stronger mode this is **better** than the Windows equivalent, because the system enforces the check rather than the app requesting it — see §12.4
Installer signing	UNSIGNED — SmartScreen will warn (§14.7)	Signing and notarisation are configured in the release pipeline (§14.7)
Sandbox	None	None — the App Sandbox entitlement is not claimed
Minimum version	Windows 10 1803, set by WebView2's own support floor	macOS 11, declared in the bundle configuration and set by the CSS the interface relies on

12.2 Windows Hello quick unlock

When enabled, the operating system's credential store holds a second copy of the data key. Both enrolling one and retrieving one require the owner to pass a Windows Hello check first — face, fingerprint, or the device PIN Windows already accepts. The result is interpreted strictly: only an explicit success proceeds, and every other outcome, including an unrecognised one, is treated as failure.

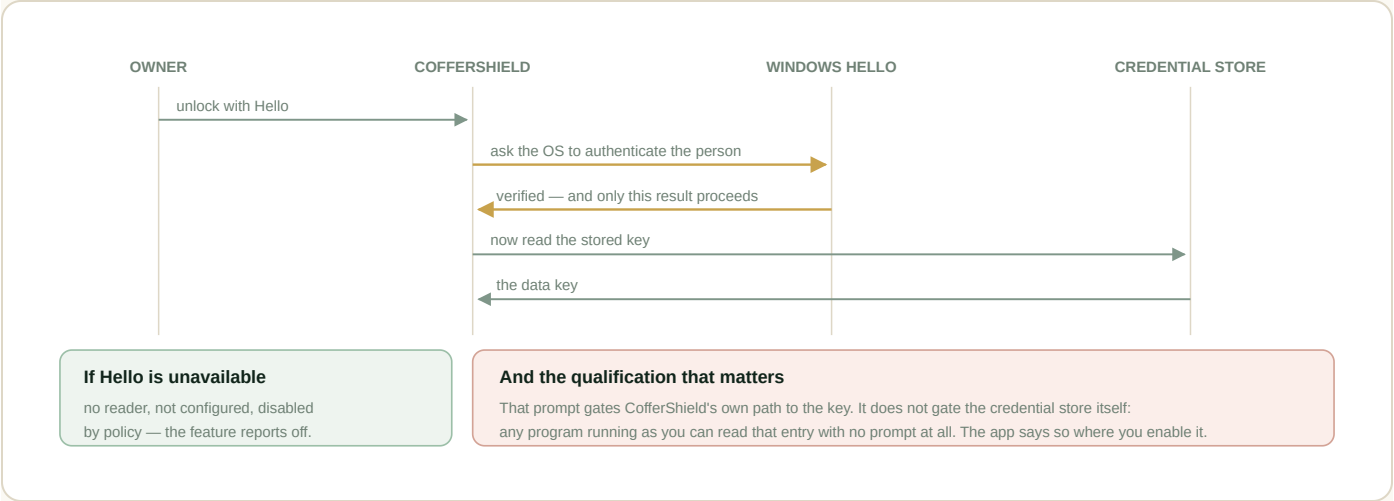

Figure 15 — Quick unlock. Treat this as a convenience that shortens a password you already chose, not as an additional layer of protection.

> If your threat model includes malware on your own machine, leave quick unlock off
>
> Enabling it places a copy of the data key somewhere that any process running as you can read, with no biometric challenge. That trades a real reduction in security for a real gain in convenience. It is off by default, the interface states the trade at the point of enabling, and changing the master password clears the stored copy — because someone changing their password because they are worried about a laptop should not find that laptop still opens with a fingerprint.

12.3 The IPC boundary

No filesystem plugin and no dialog capability is granted to the webview, so it cannot open a file or raise a dialog on its own. What it has instead is fifteen commands defined by this application, plus the framework's default permission set — roughly a hundred built-in commands covering window state, paths, events, menus and images, among them one that reads an image from a path and one that toggles the inspector. The fifteen below are where this application's own validation lives.

COMMAND GROUP	WHAT THE WEBVIEW CAN ASK FOR	CONSTRAINT IN THE HOST
Save a file	"write these bytes, ask the owner where"	A native save dialog. The webview never names the destination.
Vault read / write / delete	A key, not a path	The key is stripped to letters, digits, hyphen and underscore and used as a filename inside the application data directory. Traversal and absolute paths are impossible by construction. A read or write that fails is **reported as a failure**; the application has no second place to look. §7.2.
Backup write	A filename and the bytes — **no folder**	The filename must start with the application's prefix, end in `.coffer`, be a single path component, and carry no traversal or separator characters. Anything else is refused before a path is built. The folder is not a parameter: the host resolves the one the owner approved, or errors. §14.10.
Backup read	Nothing at all	It takes no arguments and reads only the file this process last wrote, so the caller can compare ciphertext before pruning.
Backup prune	A count — **no folder**	Deletes files matching the backup naming pattern inside the approved folder, never the last one. §14.10.
Folder questions	A picker, and one yes-or-no question	`pick_folder` raises a native picker and returns what the owner chose; it is the *only* way a folder becomes usable, and the host canonicalises it before keeping it. The question of whether the chosen folder is the install directory takes no path and compares canonical paths. The directory-existence command that accepted any path is gone. §14.10.
Quick unlock	Enrol, fetch, clear, availability	Enrol and fetch both require a Windows Hello success first.
Window theme	Light or dark title bar	Cosmetic.

12.4 Touch ID on macOS

Quick unlock on macOS has two storage modes, and which one you get changes the security story enough that the interface is told which one it got rather than left to assume the better one.

MODE	HOW THE KEY IS PROTECTED	REQUIRES
Enclave	The data key sits in the data-protection keychain behind an access control requiring biometry from the **currently enrolled** set. macOS refuses to return the item without a successful Touch ID. The check is enforced below the application, so neither CofferShield nor anything else running as the owner can bypass it. This is materially stronger than the Windows design in §12.2.	A signed build carrying a keychain access group
Keychain	The ordinary login keychain, with a Touch ID prompt the application raises itself. Equivalent in shape to Windows: the prompt guards CofferShield's path to the key, not the stored item. macOS still ties the item to this application and prompts when another application reads it, which Windows Credential Manager does not.	A Touch ID sensor with a registered fingerprint

The enclave path is attempted first. If the platform refuses it, the key is stored the weaker way **and the security note in Settings says so**. That rule is the load-bearing one: storing weakly while the interface claims the enclave would be worse than having no feature at all, and it is the reason this went unimplemented for as long as it did.

Two consequences worth knowing. Binding to the currently enrolled set means **adding or removing a fingerprint invalidates the saved key** — by design, and the error message says so. And changing the master password clears the stored copy on both platforms, because the data key does not change when the password does (§5.2), so a credential captured beforehand would otherwise keep working afterwards.

> Written and reviewed, not yet proven on hardware
>
> The macOS biometric path has not been executed on a Mac by this project — macOS-gated code could not be compiled in the environment it was written in, which is the same constraint that kept the feature out of earlier releases. Treat its first release as untested rather than as established, and see the test sequence in `MAC.md`. The Windows path in §12.2 is unaffected.

12.5 macOS entitlements

ENTITLEMENT	WHY
`com.apple.security.cs.allow-jit`	WKWebView runs JavaScriptCore, which needs JIT pages under the hardened runtime. Without it the window opens blank. This is the narrowest of the JIT-related exceptions; the looser `allow-unsigned-executable-memory` and `disable-library-validation` are **not** claimed.
`com.apple.security.network.client`	Needed only if the owner turns on price refresh. Nothing else in the application touches the network.

Those two are the complete list. The App Sandbox is **not** claimed, which means the process has ordinary user-level filesystem access; that is a consequence of letting the owner keep automatic backups in a folder of their choosing without security-scoped bookmarks. It is a real reduction in containment and it is listed in §14.

13 Security assumptions

Every security design rests on assumptions. Leaving them unstated is how a product ends up protecting something other than what its owner believed. These are CofferShield's, in the order that they matter.

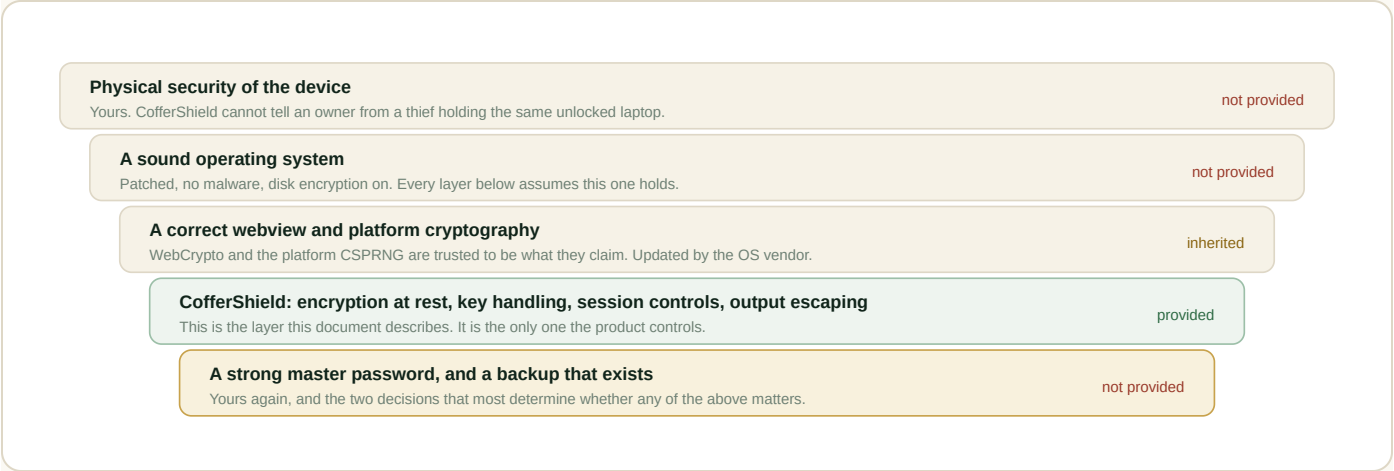

Figure 16 — Layers. Only the fourth band is CofferShield's work. The outer bands are assumptions, and a failure in any of them defeats the middle.

ASSUMPTION	WHY IT MATTERS	IF IT DOES NOT HOLD
The computer is not already compromised	The plaintext and the key exist in this process while the vault is open. Process memory is readable by other software running as the same user.	Total loss. Everything in §14.1 applies. This is the assumption that carries the most weight.
The operating system is current	The webview supplies WebCrypto and enforces the content security policy. A vulnerable webview undermines both.	Depends on the flaw. A webview code-execution bug reduces to the row above.
The platform's randomness is sound	Every key, salt, IV and generated password comes from the OS CSPRNG. Predictable randomness would make keys guessable regardless of password strength.	Catastrophic and undetectable from inside the application. This assumption is shared by essentially all software on the machine.
The master password is strong and not reused	It is the only secret protecting the file. §6 quantifies what "strong" means against an offline attack.	The at-rest protection fails on a timescale set by the password, not by the cipher.

ASSUMPTION	WHY IT MATTERS	IF IT DOES NOT HOLD
The device is physically controlled	An unlocked screen is an open vault. Auto-lock narrows the window; it does not remove it.	Anyone with hands on an unlocked session has the data.
The installer that was run was genuine	A counterfeit build could do anything. Nothing in the file format authenticates the application to the owner.	Total loss, silently. §14.7 is honest that the unsigned Windows installer makes this harder to verify than it should be.
Backups exist and are stored apart	Encryption protects confidentiality. It does nothing for availability against theft, hardware failure or ransomware.	The data is lost. This is by some distance the most common way people actually lose a vault.

14 Security limitations

This section exists because a security document that only lists strengths is marketing. Each item is a genuine limitation of this design, stated without hedging.

14.1 A compromised computer defeats everything

If software is running on the machine as your user, it can log the master password as you type it, read the decrypted vault out of process memory while it is open, take screenshots, or replace the application with one that reports what you enter. No local vault can prevent this, and neither can a hosted one once the password is captured. Nothing in this document should be read as protection against it. Specifically, CofferShield does not defend against:

- **Keyloggers** — hardware or software. The master password is typed on the same keyboard as everything else.
- **An already-infected machine.** Installing a vault on a compromised computer moves your secrets somewhere convenient for the attacker.
- **Screen recording and remote-access tools.** What is displayed can be captured. The window is not marked capture-protected, so it appears in screen shares and screenshots.
- **A compromised operating system, hypervisor or firmware.** Everything above assumes the layer below is honest.

14.2 Memory is not protected

While the vault is open, decrypted records live in the JavaScript heap. Keys are held as non-extractable platform key objects, so the application itself cannot read a key's bytes, and raw byte buffers are zeroed immediately after wrapping or importing on the password and backup paths.

Two paths are exceptions, and are named here rather than glossed. Enrolling or retrieving a quick-unlock key moves the data key through a base64 string in order to cross the bridge to the host, and a JavaScript string cannot be erased. And opening the vault with a recovery key deliberately keeps the raw key in memory until a new password is set

or the vault locks — so that the owner is not asked for the password they came to that screen because they had forgotten.

None of this stops a debugger, a memory-scraper or a crash dump. Managed runtimes also copy and relocate memory during garbage collection, so a guarantee of complete erasure is not available and is not claimed.

14.3 An unlocked session is an open vault

Auto-lock, lock-on-hide and the session PIN reduce the window in which that is true. None of them closes it. The session PIN in particular is a screen cover, not a lock — the key stays in memory while it is showing, and the interface says so where it is used.

14.4 Every extra door is a door

A recovery key opens the vault without the password. A quick-unlock enrolment puts a copy of the data key in the operating system's credential store. Both are opt-in, both are genuinely useful, and both enlarge the attack surface by exactly as much as they enlarge convenience. Neither is enabled by default, and both are described in the interface at the moment of enabling rather than in a footnote.

14.5 Printing leaves the boundary

The wallet card, the heir sheets, the emergency sheet and the recovery-key sheet are all designed to be printed, and printing is a plaintext operation by definition. Beyond the paper itself, the operating system's print subsystem spools the rendered page to disk at a location this application cannot reach or clean up. Print to a printer you control, and treat the paper as you would treat the contents.

14.6 There is no way to send you a fix

The absence of an update channel is a deliberate privacy property: no install can be counted, no user tracked, and no update endpoint hijacked to deliver code. It is also a real limitation. **If a vulnerability is found in this application, there is no mechanism to reach you with a patch.** You would have to learn of it independently and download a new build.

One component does update itself, and it is the largest: the platform webview. On Windows, WebView2 receives security updates from Microsoft independently of this application. The application layer does not.

14.7 Distribution integrity

PLATFORM	STATUS	WHAT THIS MEANS FOR YOU
Windows	INSTALLER UNSIGNED	SmartScreen will show a warning and the publisher will read as unknown. More importantly, there is no signature to verify, so a substituted installer cannot be distinguished from the real one by inspection. Verify the published hash of the file you downloaded before running it. Code signing is the top item in §17 for a reason.
macOS	CONFIGURED, NOT ASSERTED	The release pipeline is set up to sign with a Developer ID and submit for notarisation, and will do so when the signing credentials are present. This document does not assert that any particular published build was signed; check that the copy you have opens without a Gatekeeper warning, which is the observable test.

14.8 Build reproducibility and provenance

Dependencies are declared as version ranges and there is no committed lockfile, so two builds of the same source may not be byte-identical. There is no build attestation and no software bill of materials. A reader cannot currently verify that a published binary was produced from the published source. §17 lists this.

14.9 Developer tools are off in release builds

Through revision 4 the shipped build could open a web inspector — a deliberate choice so a rendering fault on a real machine could be examined rather than guessed at. The cost was that anyone with a moment at an *unlocked* window could open the inspector and read the decrypted vault out of the heap. **Revision 5 turns it off:** the framework feature is not enabled in the release build, and the command that would toggle it is denied in the capability as well, so neither route remains. It is still available as an opt-in for local debugging.

This changes nothing about a locked vault. An attacker at an unlocked screen still has the data on screen, which is why the answer to §14.3 is still "lock your computer".

14.10 The host's filesystem reach, and what closed it

Through revision 4 the backup commands validated the filename but not the folder: the folder was whatever the caller passed, checked only to exist. Script executing inside the webview could write a file whose name matched the backup pattern into any directory the process could reach, or delete matching files from one.

Closed in revision 5. A folder becomes usable exactly one way — the owner picks it in a native dialog, and the host canonicalises it and keeps it. `backup_write` and `backup_prune` no longer take a folder at all, `backup_read` already took no argument, and the directory-existence command is gone. Canonicalisation is the part that matters: a symlink cannot be approved under one name and written through under another. The framework's image-from-path command — a second arbitrary-path read this application never used — is now denied explicitly in the capability rather than left on because the default set includes it.

What remains is smaller, and worth stating: approval lasts for the process, so a fresh launch re-approves the stored folder before automatic backup will write. The host does not persist the approval itself.

14.11 What a determined, targeted adversary can do

An adversary who can compel a device, intercept a shipment, or place an implant on your machine defeats this product. That is true of every consumer security product, and a document that implied otherwise would be doing you a disservice. What this architecture removes is the *scalable* attack: there is no service to breach, no repository to seize, and no operator to compel. Attacks must be conducted one person at a time, against one machine at a time.

If any of the following is true, this product is not the right answer

You need multi-user access with per-user permissions. You need an audit trail of who read what. You need to revoke access to data already shared. You need central administration or policy enforcement across a fleet. You need guaranteed recovery when an employee leaves. These are all real requirements, and they all require a server. CofferShield is a single-owner tool and does not pretend otherwise.

15 Security best practices

Ranked by how much difference each makes, rather than by how easy it is.

Essential

DO THIS	BECAUSE
Use a long master password — six or more random words, or accept the suggestion	It is the entire at-rest boundary. §6 has the arithmetic; the difference between four words and seven is the difference between days and millennia.
Write the password down and store it physically	Forgetting it is the most likely way you will lose this data, by a wide margin. A sealed envelope with your will.
Export a backup, and keep it somewhere else	Encryption does not protect against a failed drive, a fire, or ransomware. A backup does, and it is safe to leave in a place you do not fully control.
Never reuse the master password	Reuse re-imports every risk this design removes. A breach elsewhere becomes a breach here.
Keep the operating system patched	The webview is the largest piece of code in the product and it is the platform's, not this application's.

Strongly recommended

DO THIS	BECAUSE
Turn on full-disk encryption — BitLocker or FileVault	It protects everything else on the machine, and it covers the print spool, temporary files and swap that this application cannot reach.
Lock your computer, not just the app	Auto-lock protects an unattended CofferShield window. Only the OS lock protects an unattended computer.
Make a recovery key and store it like a deed	It converts "the vault is gone" into "fetch the envelope". Treat it as equal in value to the password itself, because it is.
Verify the installer before running it	Especially on Windows, where there is currently no signature to check. Compare the published hash.

DO THIS	BECAUSE
Test a restore once	An untested backup is a hope. Restoring on a second machine, once, converts it into a fact.

Consider, depending on your situation

DO THIS	BECAUSE
Leave Windows Hello quick unlock off	If malware on your own machine is part of your threat model, the stored key is worth more to an attacker than the convenience is worth to you (§12.2).
Shorten auto-lock to one minute	On a shared desk or in an office. The cost is retyping; the benefit is real.
Split a recovery key among several people	If continuity for your family matters more than the risk of a single holder acting alone (§7.5).
Keep the price feature off	If a third party learning which tickers you hold is a concern. Nothing else changes; you type prices yourself.
Keep documents out of the vault	If you would rather the file stayed small and easy to back up frequently. Record where the originals are instead.

16 Frequently asked questions

Can CofferShield recover my password?

No, and not as a matter of policy — as a matter of arithmetic. The password is never stored or transmitted, and the vault contains no verifier that could be attacked or unlocked from our side. If you made a recovery key, use that. If you did not, the data cannot be recovered by anyone.

Can hackers read my vault?

Not from the file alone. An attacker holding your vault or backup has AES-256-GCM ciphertext and must guess your master password against a 600,000-iteration KDF, offline. With a strong password, that is not feasible. An attacker who is *already running code on your computer* is in a completely different position — see §14.1, which is written plainly for exactly this question.

Can CofferShield employees see my data?

There is nothing for them to see. No account exists, no data is transmitted, and no server holds a copy. This is not a promise about internal access controls; it is the absence of anything to control access to.

What happens if your company disappears?

Your vault keeps working. It is a local file opened by a local application with no licence check, no activation and no server dependency. Nothing expires and nothing calls home. The format is documented in §5.3 and uses only standard primitives, so it can be read by other software if it ever needs to be.

Can I open my vault without internet?

Yes — that is the normal case. Every feature except optional price refresh works with the network disconnected, and price refresh degrades to the prices you last entered.

Can ransomware steal my vault?

It cannot read it without your password. It can certainly encrypt or delete the file, which is why an offline backup matters. Note that ransomware is code running as you, so while the

vault is open §14.1 applies to anything else it chooses to do.

Is my data encrypted on my own disk, or only in backups?

Both, identically. There is no unencrypted state at rest. The live vault and the backup are the same ciphertext.

What if I lose the computer with the vault on it?

The thief has an encrypted file. Restore your backup on a new machine with the same master password. If quick unlock was enabled on the lost machine, read §12.2 — that changes the answer, and it is a reason to consider leaving the feature off.

Why is there no two-factor authentication?

Two-factor authentication protects an *authentication event* against a server. There is no server here and no login to intercept — the password is not checked against anything, it derives a key. A second factor would have to be stored on the device to be usable offline, which would make it a second thing on the same machine rather than a second factor. Quick unlock and the recovery key are the mechanisms in that space, and §14.4 is honest that each is a door rather than a lock.

Has this been independently audited?

No. Not by anyone. §17 states the intention to change that; until it does, this document is a self-description and should be weighed as one. Appendix B lists what a review conducted while writing it actually found, which may be more informative than the assurance itself.

Why should I trust a single-file HTML application?

Partly because you do not have to trust it blindly: it is one readable file, unminified, with no build step between the source and what runs. That is unusual and it is a deliberate transparency choice. It also has a real cost, documented in §10.3 — a single inline script means the content security policy cannot restrict script execution, only what executing script can reach.

17 Future security work

Listed in the order they would most improve the product's security posture. Nothing here is a commitment to a date.

WORK	WHY IT MATTERS
Code signing on Windows	The largest single gap. Today there is no signature for a customer to verify and no way to distinguish a substituted installer by inspection. Every other item on this list matters less than this one.
Independent cryptographic review	The design in §5 has not been examined by anyone outside this project. A review of the key hierarchy, the slot format and the recovery-key construction is the assurance this document cannot provide about itself.
Independent penetration testing	Focused on the IPC boundary and on injection paths into the interface, which is where this project's own review found the defects listed in Appendix B.
Reproducible builds and published hashes	A committed lockfile, pinned toolchain and build attestation, so a published binary can be tied to published source.
Confirm Touch ID on Mac hardware	The macOS biometric path now exists (§12.4) and has never been run on a Mac. Until it has, it is written code rather than a proven feature. The sequence to run is in `MAC.md`.
Argon2id key derivation	Memory-hardness is the property that most degrades the GPU attack in §6. Worth doing when it can be done without carrying an unaudited implementation inside the file.
Hardware security key support	A FIDO2 device holding a second factor for the data key. Genuinely useful, and genuinely hard to do without creating a way to lose the vault by losing a key.
Capture protection on the window	Marking the window so screenshots and screen-sharing show nothing. Straightforward; the open question is whether owners would rather keep the ability to screenshot their own records.
A signed security advisory channel	Not an auto-updater — a place a customer can check, so that §14.6 stops being a dead end.

A Appendix A — Parameters at a glance

A.1 Cryptographic parameters

PARAMETER	VALUE
Symmetric cipher	AES-256-GCM
Initialisation vector	96 bits, fresh from the CSPRNG on every encryption
Authentication tag	128 bits, verified on every decryption
Key derivation	PBKDF2-HMAC-SHA-256
Iterations	600,000
Salt	128 bits, random per slot, re-drawn on every password change
Data-encrypting key	256 bits from the CSPRNG, wrapped under the derived key
Key extractability	Non-extractable platform key objects; raw bytes zeroed after use, with the two exceptions in §14.2
Padding	Random, to a 64 KiB boundary, with a length prefix inside the ciphertext
Recovery key	256 bits, Crockford base32 with a Fletcher-16 checksum
Recovery key splitting	Shamir over GF(2^8), threshold k of n
Minimum master password	12 characters
Suggested master password	7 words plus a number — approximately 66 bits
Slots per vault	2, identical in structure, order drawn from the CSPRNG at creation

A.2 Session and storage defaults

SETTING	DEFAULT	RANGE
Auto-lock on idle	5 minutes	1 min – 1 hour, or never
Lock when hidden	5 minutes	1 – 15 min, or never
Clipboard clearing	45 seconds	15 – 90 s, or never
Masked value re-mask	20 seconds	fixed
Session PIN	off	4+ characters, memory only
Quick unlock	off	Windows Hello; Touch ID on macOS, in one of two modes (§12.4)
Price refresh	off	opt-in, per provider
Automatic backup	off	folder chosen by the owner
Maximum document size	64 MB per file	fixed; 256 MB of documents per vault (§7.3)
Network request timeout	10 seconds	fixed; the request is abandoned, not aborted

A.3 The build this document describes

ARTIFACT	SHA-256
dist/index.html	4d8f227422011dfdbf6df8c20745e21b453e7c7ed0441797351102278b8eda25
src-tauri/src/main.rs	2728abdf3bcbe03095138c5cda9669b179e2c13369263f2408a641b848f3fe5c
src-tauri/tauri.conf.json	e5803346b4409b54a4e8385ad2321cae9708682dd50b2975694d86fc447ee1b0
src-tauri/entitlements.plist	eba132b18ca55caf8e8449717f61d88d381cb407557aa9f784a36de44394e745

Application version 1.0.0 · vault schema version 2 · backup format version 2. Automated test suite at the time of writing: 229 tests, 0 failures, run at the shipping 600,000 iterations as well as at a reduced count, and verified to behave identically on both platform webview engines.

B Appendix B — Findings from this review

This document was written by reading the shipped build rather than the intentions behind it. That exercise found defects. They are published here for two reasons: a reader deciding whether to trust a self-assessment is better served by seeing what it caught than by being told it was thorough, and a fixed defect that nobody mentions tends to come back.

All items below were found and fixed before this document was published, and each has a test that fails if it returns. Two rounds are recorded: the first from reading the application, the second from an adversarial pass over a draft of this document against the code, which is where B.2's last three entries came from.

B.1 The one that mattered

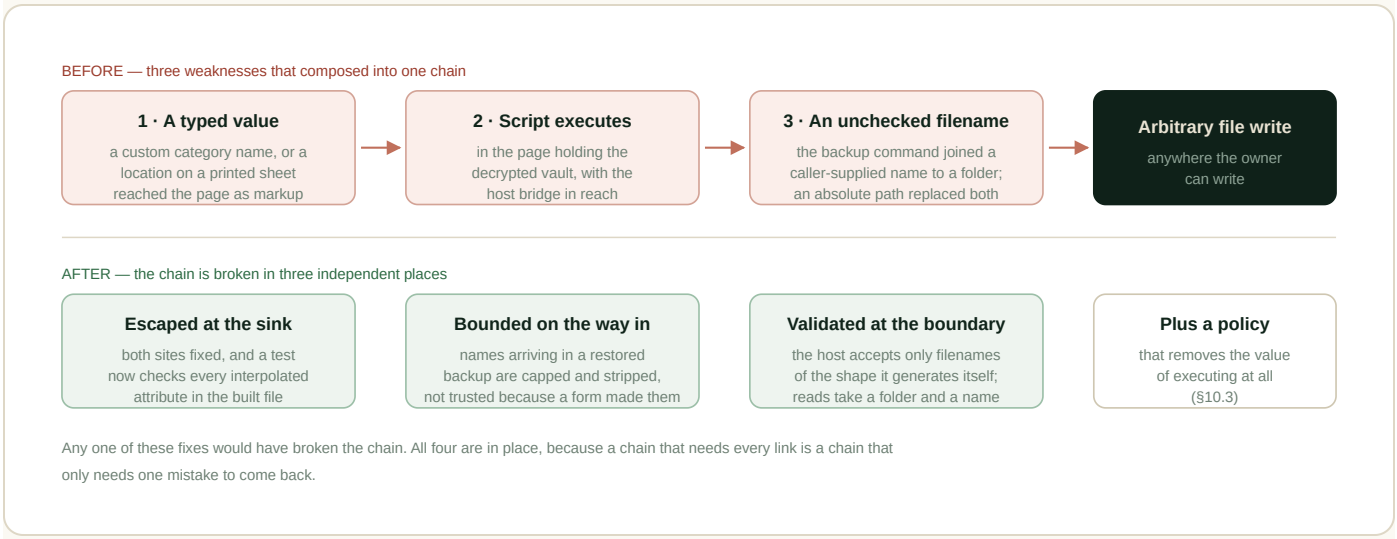

Figure 17 — The chain, and how it was closed. No exploitation is known; it was found by reading the code for this document, not by an incident. §14.10 states the part of the third link that is narrowed rather than removed.

FINDING	EFFECT	FIX
Two output-escaping failures	A custom insurance category name reached a tooltip attribute unescaped, and a "where things are kept" location reached the printed heir sheet unescaped. Both are values the owner types, and both can also arrive inside a restored backup.	Escaped at both sites. A test now scans the built file for any interpolated `title` attribute that is not escaped.

FINDING	EFFECT	FIX
Unvalidated backup filename in the host	The command joined a caller-supplied filename to the owner's chosen folder. An absolute path discards the folder entirely, and traversal escapes it a segment at a time.	The host now accepts only filenames that start with its own prefix, end in `.coffer`, are a single path component, and carry no traversal or separator characters. Five unit tests in the host cover that validator directly, including both of the shapes above; a test in the application suite asserts the commands call it.
Arbitrary file read in the host	The read-back command took a full path and read whatever was there.	It now takes no argument at all and opens only the file this process last wrote. There is no argument shape left to abuse. The *write* and *prune* commands still take a folder that is not constrained — §14.10 states what that leaves open.
An unused command in the IPC surface	A directory-listing command was registered and never called.	Removed. An unused command is still a reachable one.

B.2 The others

FINDING	EFFECT	FIX
Enrolling a quick-unlock key required no consent	Retrieving the stored key required Windows Hello; replacing it did not. One call could substitute the saved key.	Enrolment now requires the same check as retrieval, before anything is stored. It also means nobody enrols without first proving the reader works.
The application shipped without its own CSP	The desktop shell delivered a policy; the file opened directly in a browser had none.	The policy now ships in the file as well, and the two are tested to agree. Both gained `object-src`, `base-uri` and `form-action`, which close channels `connect-src` does not.
"Auto-lock: never" locked anyway	A stored zero was read as "unset" and silently replaced with five minutes. The setting screen was describing behaviour that did not occur.	Zero is now a choice. The failure was in the safe direction; a setting that lies is still a defect.

FINDING	EFFECT	FIX
A claim about hardware the build did not use	An unreachable branch of the interface stated that the key was held in the macOS Secure Enclave, in a build where quick unlock was disabled on macOS. No user could see it — but the string was there.	Removed at the time. macOS Touch ID has since been implemented (§12.4), so the test that guarded the phrase became a stronger one: the enclave claim may only be shown when the host reports that it actually achieved enclave storage, and an unrecognised mode must claim nothing.
A stale ciphertext left behind after migration	Lifting a vault from the browser store into a file did not remove the original, leaving a second copy ageing in the webview profile under whatever password was current when it was written.	The file is read back and compared before the old copy is removed, so a half-finished migration never destroys the original.
The suggested master password was too short for an offline attack	The wordlist is short and friendly — about eight bits a word — and the suggestion used five words, giving roughly 49 bits. That is inside reach of a well-funded attacker for a high-value target.	Seven words, roughly 66 bits. A test recomputes the figure from the live wordlist and fails if either the list or the count moves the result below 60 bits.
A symbol search reached the network with prices switched off	The equity search branches were gated behind a provider and a key. The crypto branch was not, because CoinGecko needs neither — so typing in the "Search coin" box on a record form sent that text, whatever it was, to a third party in a vault where price refresh had never been turned on.	Gated on the same switches as every other request, and given the same timeout. A test drives the real module with a fetch stub and fails if any call is attempted while prices are off.
Slot order came from `Math.random`	The order of the real and decoy slots is presented as something an examiner cannot exploit, so it should not have come from a generator that makes no such promise.	Drawn from the platform CSPRNG, like every other random value in the product.
The interface recommended a weaker password than this document does	The create-vault screen suggested "four or five unrelated words" while §6 calls four words inadequate for a file that can be copied.	The screen now says six or more, and explains why in one line.

B.3 Known and not changed

These were examined and left as they are. Each is a judgement rather than an oversight, and each is documented in the body of this whitepaper so that a reader can disagree.

ITEM	WHY IT STANDS
Developer tools enabled in release builds	Diagnosing a rendering fault on a real machine is worth more than the marginal protection removing them would give against someone who already has an unlocked window. §14.9.
No attempt limiting on the unlock screen	It would constrain only an attacker politely using this application, while adding a way to destroy your own vault by mistyping. §5.4.
PBKDF2 rather than Argon2id	A reviewed platform primitive beats an unreviewed in-house one in the most critical position in the product. §5.1, and it is on the roadmap.
The window is not capture-protected	It would also stop owners screenshotting their own records. §17 lists it as an open question rather than a decision.
Windows file permissions are inherited	Improving on the directory ACL would mean taking a dependency on security-descriptor code to protect a file that is ciphertext. §7.2.
The backup folder argument is not constrained	Constraining it means the host remembering the folder the owner picked across restarts. Worth doing; not done. §14.10 states the residual reach rather than leaving the reader to find it.
macOS Touch ID has not been run on a Mac	Written, reviewed and reasoned about, but macOS-gated code could not be compiled in the environment it was written in. §12.4 carries the warning, rather than the release notes carrying a silence.

C Appendix C — Verifying these claims

Most of this document can be checked without trusting it. The application is a single file of readable source, so the following are not thought experiments.

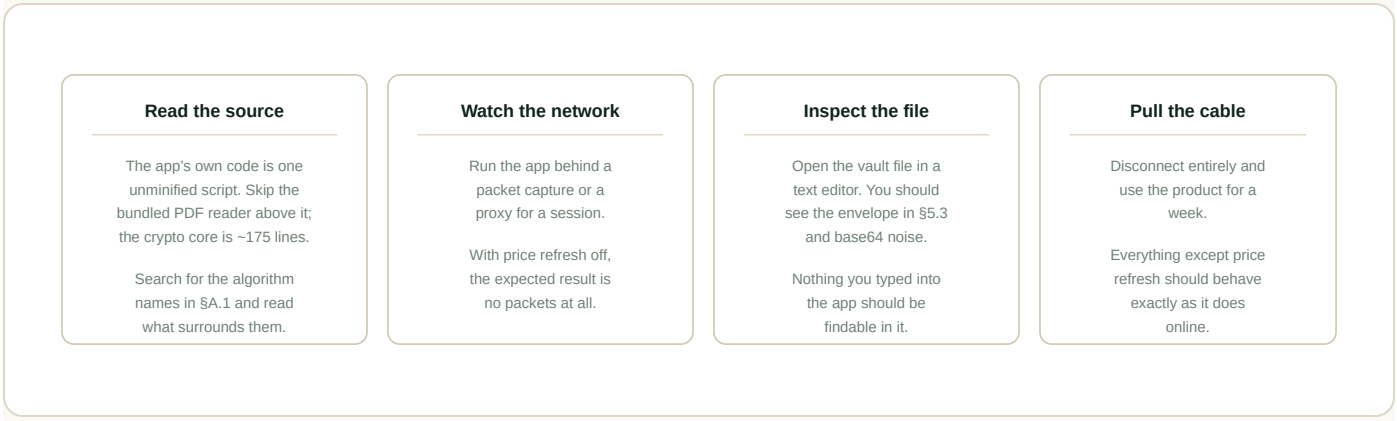

Figure 18 — Four checks. None requires special tooling, and the third one takes a minute.

C.1 Specific things to look for

CLAIM IN THIS DOCUMENT	HOW TO CHECK IT YOURSELF
600,000 PBKDF2 iterations	Open the vault file and look for `"kdf":{"it":600000`. The count is stored in the clear in each slot, because it has to be.
AES-256-GCM with a fresh IV	Save twice without changing anything. The slot's top-level `iv` changes both times, and so does `ct`. (A slot has up to three `iv` fields; the other two belong to the wrapped keys and do not change on a save.)
Padding to 64 KiB	Note the file size. Add a password. In most cases the size does not change.
Two slots always present	Count the entries in the `slots` array. There are two, whether or not you set a decoy password.
Nothing plaintext at rest	Search the vault file for a distinctive string you entered. It is not there.
No telemetry	Search the application file for the name of any analytics product you can think of, and watch the network as above. With price sources off, nothing should appear even while you type into a symbol-search box.

CLAIM IN THIS DOCUMENT	HOW TO CHECK IT YOURSELF
The CSP	It is a meta tag in the head of the application file, and the same policy appears in the desktop configuration.
Backups are the same ciphertext	Export a backup and compare its `ct` field with the live slot's. They match.

C.2 What you cannot verify from the artifact

- **That the binary you installed was built from this source.** Reproducible builds are on the roadmap; until then this is a trust assumption, and on Windows there is not yet a signature to anchor it. §14.7 and §14.8.
- **That the platform's cryptography does what it claims.** WebCrypto and the operating system's random number generator are trusted. Every application on the machine trusts them too.
- **That a particular published macOS build was notarised.** The pipeline is configured for it; the observable test is whether Gatekeeper accepts the copy you have without complaint.

> ### Reporting a security issue
>
> If you find something in this design or this implementation that is wrong, we would rather hear it than not. Please include what you did, what you expected, and what happened, and give us a reasonable window to fix it before publishing. Given §14.6, please also tell us how you would like to be told when it is fixed. Contact: **[security contact]**.
