# Petersaint security design

Keys, ciphers, limits and trade-offs · Johannson Limited

Applies to Petersaint 1.2 for iOS 16.4+ Revised 24 November 2026 (v4) Author: Jon Hurdle


IN ONE PARAGRAPH

Every file gets its own random 256-bit key and is encrypted with AES-256-GCM in 1 MiB chunks. Those file keys are wrapped by a random vault key, and the vault key is encrypted to a P-256 private key that lives in the iPhone's Secure Enclave and can't be exported. Your pattern is never used to make a key. The app checks it against an Argon2id hash, and only then asks the Secure Enclave to unwrap the vault key. That check is done by the app's code: the Secure Enclave doesn't know your pattern. While the vault is open, the unwrapped vault key is in the app's memory. Deleting the Secure Enclave key makes the vault key impossible to unwrap, which is how a wipe works. There's no server and no account, the pattern can't be reset, and the vault is kept out of device backups. Since 1.2 you can export an encrypted backup file: the vault key is sealed to an X25519 key derived from a 12-word recovery phrase that the phone doesn't keep.


This page is written for people who want to check our work. Every parameter below is taken from the app's source and build specification, not from marketing copy. Weaknesses are in section 13.


1. Threat model
2. Key hierarchy
3. Algorithms and parameters
4. File format
5. Pattern verification
6. Attempt limits and delays
7. Wipe (crypto-shred)
8. Duress pattern
9. Share extension
10. Storage, metadata and device backups
11. Encrypted backup
12. Memory, screen and clipboard
13. Known limitations
14. Code review
15. Design choices compared


## 01 Threat model

iOS Data Protection already encrypts the phone's files, and most of them are unreadable while the phone is locked. Petersaint is a second layer with its own keys, aimed at the time when the phone is *unlocked*. Three kinds of attacker need separating:

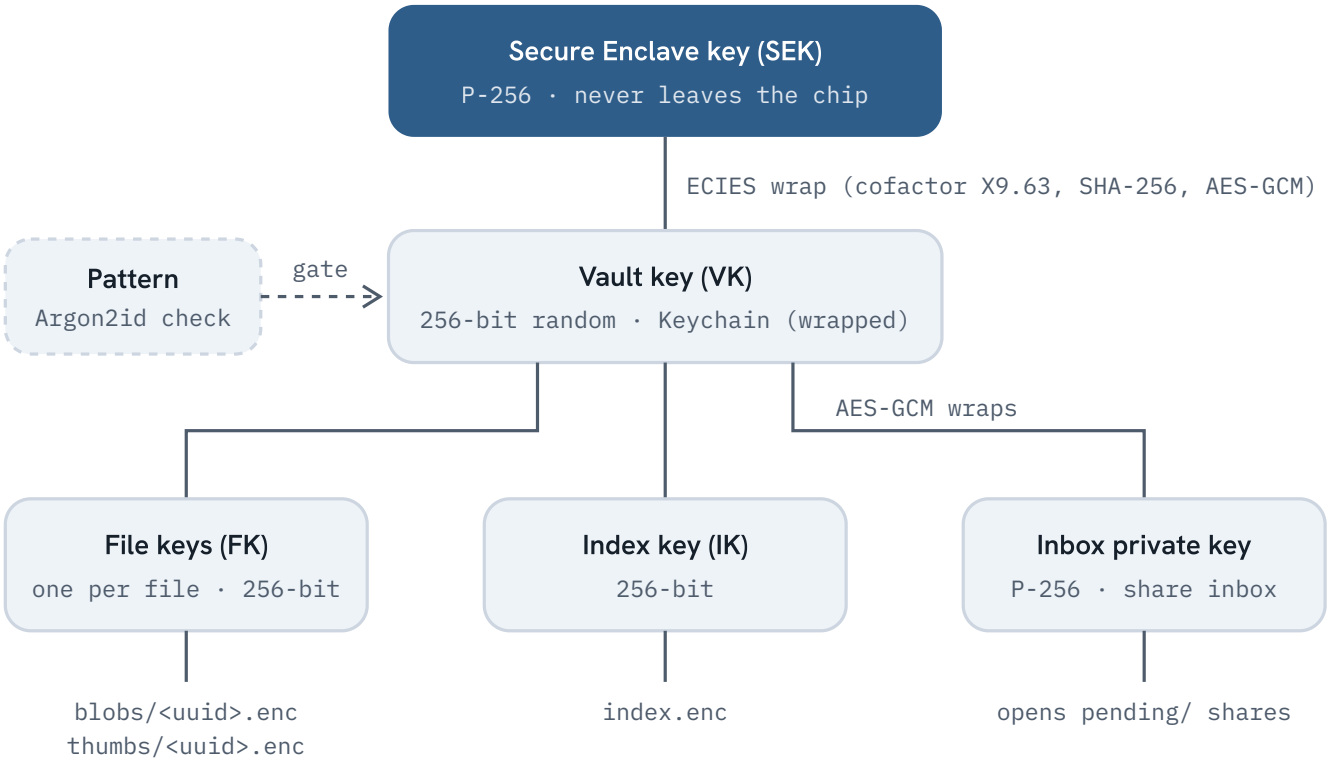
| Attacker | What stops them |
|---|---|
| Uses the phone's normal screens | The pattern, the attempt limit and the waits. All of these are enforced by the app's code, running inside the iOS sandbox. |
| Has a copy of the files (device extraction, a backup) | Encryption. The vault key can only be unwrapped by this phone's Secure Enclave, so the files alone aren't enough. See the pattern-hash limit in section 13. |
| Can run code inside the app, or a jailbroken phone | Nothing in Petersaint. Such code can call the same Secure Enclave operation the app calls, without knowing the pattern. Out of scope. |

| In scope | Out of scope |
|---|---|
| Someone holding your unlocked iPhone: a partner, a colleague, a thief who saw your passcode | Malware on a jailbroken phone reading memory while the vault is open |
| Copying the app's files off the device, or out of a backup, and reading them elsewhere | A camera filming your screen while you draw the pattern |
| Guessing the pattern on the phone | Code running in the app's process, through an iOS or app vulnerability; code signed by our developer team (see section 9) |

One thing the design deliberately allows: with an attempt limit set, anyone holding the phone can *destroy* the vault by drawing wrong patterns. They can't read it, but they can make sure you can't either. Self-destruct is off by default since version 1.1.3; leaving it off avoids this.

The app makes no network connections of its own and has no server. The App Store privacy label is "Data Not Collected".

## 02 Key hierarchy




Solid lines are key wrapping. The dashed line is a gate, not a derivation: the pattern decides whether the app asks the Secure Enclave to unwrap the vault key. It never contributes key material.

**Secure Enclave key.** A P-256 private key created with `kSecAttrTokenIDSecureEnclave`, access control `.privateKeyUsage`, accessibility `WhenUnlockedThisDeviceOnly`. iOS will use it only while the phone is unlocked, and the private key can't be exported, synced or restored to another device. No user-presence or application-password constraint is attached, so the Secure Enclave doesn't check the pattern. The app does that before it asks.

**Vault key.** 256 random bits, generated once at set-up. It's stored only in wrapped form, encrypted to the SEK's public key with `eciesEncryptionCofactorX963SHA256AESGCM`. Unwrapping needs this phone's Secure Enclave. Once unwrapped, the 256-bit vault key is ordinary app memory until the vault locks; the Secure Enclave protects the wrapping key, not the keys it releases.

**File keys.** Each item gets its own random 256-bit key, wrapped by the vault key with AES-GCM. The wrap's additional authenticated data includes the item's UUID, so a wrapped key can't be moved onto a different item.

**Index key.** A random 256-bit key, wrapped by the vault key and kept in the Keychain. It encrypts the index (file names, types, sizes, dates, folders and the wrapped file keys).

All Keychain items use `kSecAttrAccessibleWhenUnlockedThisDeviceOnly`, are not synchronisable, and since v1.0 build 2 are pinned to the app's own Keychain access group.

## 03 Algorithms and parameters

| Purpose | Algorithm and parameters |
|---|---|
| File contents | AES-256-GCM, 1 MiB chunks, fresh 96-bit random nonce per chunk, 128-bit tag |
| Index | Same chunked AES-256-GCM, under the index key |
| Thumbnails | Encrypted JPEG, same chunked format, under a key derived from the file key with HKDF-SHA256 and its own label, so a thumbnail can't stand in for its file |
| File key wrap | AES-256-GCM under the vault key, AAD bound to item UUID |
| Vault key wrap | ECIES, P-256 Secure Enclave key, cofactor ECDH, X9.63 KDF with SHA-256, AES-GCM |
| Pattern check | Argon2id (v1.3) via libsodium `crypto_pwhash`: 64 MiB memory, 3 passes, 1 lane, 32-byte output, fresh 16-byte salt every time a pattern is set, constant-time compare. Stored record: salt ‖ hash (48 bytes), plus one grid byte for 3×6 |
| Recovery phrase (1.2) | 128 bits from `SecRandomCopyBytes`, encoded as 12 words with the standard BIP-39 English wordlist and checksum. The bundled wordlist's SHA-256 is checked at load |
| Recovery key (1.2) | HKDF-SHA256 over the 128 bits, with Petersaint's own salt and info, to an X25519 private key. Not BIP-39's PBKDF2 seed, so it shares nothing with a wallet made from the same words |
| Backup key block (1.2) | Fresh ephemeral X25519 key per export, ECDH with the recovery public key, HKDF-SHA256 (salt = ephemeral ‖ recipient public keys), AES-256-GCM over the vault key with the file header as AAD |
| Share inbox | Ephemeral P-256 ECDH per file, HKDF-SHA256, 64-byte output split into a metadata key and a content key |
| Randomness | CryptoKit `SymmetricKey(size: .bits256)` (system CSPRNG) for keys; libsodium `randombytes_buf` for salts |
| Libraries | Apple CryptoKit and Security framework; swift-sodium pinned to exactly 0.9.1 (libsodium 1.0.18) for Argon2id. No other third-party code. |

The building blocks are standard and come from Apple's frameworks and libsodium. The way they're put together (key hierarchy, file format, share inbox, pattern and duress logic) is Petersaint's own and hasn't had a paid independent audit.

## 04 File format

Every encrypted file (blob, thumbnail, index) shares one chunked format:

```
header   = magic ‖ version ‖ chunkSize (u32, BE) ‖ chunkCount (u64, BE)
chunk[i] = nonce (12) ‖ ciphertext ‖ tag (16)
AAD[i]   = header ‖ itemUUID ‖ i
```

- The chunk count is fixed in the header and authenticated with every chunk, so chunks can't be dropped, reordered, duplicated or moved between files.
- The reader checks the last chunk and refuses trailing data, so truncation and extension are both detected.
- Chunk size is 1 MiB when writing. The reader accepts at most 16 MiB and rejects zero values, and since build 2 its header arithmetic is overflow-checked before any tag is verified.
- Nonce reuse: under a single key, the heaviest user is the index key, which is re-encrypted on every change. Assuming around a million index chunks over a vault's life (an estimate, not an enforced limit), the chance of any two random 96-bit nonces colliding is about $2^{-57}$. A 4 GB video is about 4,000 chunks under its own key.
- Authentication isn't freshness. GCM proves a file hasn't been altered, not that it's the latest version: an older, valid copy of the index or a blob written back in place will decrypt. See section 13.

## 05 Pattern verification

The pattern is a **gate, not a key**. Drawing it does three things, in order:

1. Increment and save the wrong-attempt counter in the Keychain, before any hashing starts.
2. Hash the drawn pattern with Argon2id and the stored salt, and compare in constant time (`sodium_memcmp`).
3. Only on a match: reset the counter, ask the Secure Enclave to unwrap the vault key, then unwrap the index key and load the index.

**Input.** The ordered point indices as ASCII joined by hyphens, e.g. `0-1-2-5-8`, prefixed `3x6:` on the tall grid, so the two grids can never produce the same input.

**Grids.** 3×3 (minimum 4 points) or, from v1.1, 3×6 (minimum 6 points). One continuous drag, no point reused; a straight line through an unvisited point collects it. A 3×3 grid with at least 4 points allows 389,112 patterns. Real people choose far fewer of them, so treat that as a ceiling. 3×6 allows many more; we haven't published a count. The grid is recorded as one byte at the end of the stored hash, so 3×3 records are byte-for-byte what v1.0 wrote.

**Reduced feedback while drawing.** By default the trail isn't drawn, points don't light up, and there are no haptics or sounds that would reveal how many points were drawn. None of this stops someone watching your finger. A half-drawn pattern is discarded if the app goes inactive mid-draw (v1.1).

**Why not derive the key from the pattern?** Because the pattern space is small. A key derived from it can be attacked offline by anyone with a copy of the files, at whatever speed their hardware allows. Here a copy of the

files is useless without this phone's Secure Enclave. The cost of that choice is in section 13.

## 06 Attempt limits and delays

| Setting | Behaviour |
|---|---|
| Self-destruct: 3, 5 or 10 | Opt-in. The Nth consecutive wrong pattern destroys the key. Vaults created before 1.1.3 keep their stored limit, which was 5 unless changed |
| Single-strike | Limit of 1, behind a bold warning |
| Never (default for new vaults since 1.1.3) | The key is never destroyed by wrong patterns; waits keep growing instead. Shown in Settings as Self-destruct: Never |
| Delays | 0, 0, 2 s, 10 s, 30 s before each attempt. With self-destruct off (the default since 1.1.3), then 1, 2, 5, 10, 30 min, then 1 hour per attempt |
| Counter | In the Keychain, not UserDefaults. Reset only by a correct pattern; never by restart, time or app update |

Like the pattern check, the limit and the waits are enforced by the app's code, not by hardware. They hold against someone using the phone's screens, not against code running inside the app. Each attempt is counted and saved before hashing starts, and attempts are handled one at a time.

**Clock changes.** Each delay is recorded against both the wall clock and the phone's time since start-up, which keeps counting during sleep and can't be changed in Settings. A delay ends only when both agree. After a restart, the full delay for that step starts again. In testing, moving the clock forward or back doesn't shorten it.

**Reinstalling** doesn't reset anything usefully: a fresh install goes to set-up, and set-up deletes the old Secure Enclave key and Keychain items first.

## 07 Wipe (crypto-shred)

A wipe destroys keys, not files. Order matters:

1. Delete the Secure Enclave key. Once that succeeds, the stored vault key can't be unwrapped anywhere, so the files on disk can't be decrypted.
2. Delete every Keychain item the app holds: the wrapped keys, the pattern and duress hashes, and the attempt counter.
3. Zero the in-memory keys and drop the decrypted index.
4. Rename the vault folder, then delete it on a background queue.

Each step runs even if an earlier one fails, and nothing is logged. Steps 1 and 2 together mean no wrapped key survives unless both deletions fail. The wipe can't undo anything captured earlier: a vault key read out of memory,

a file someone exported or screenshotted, or a copy taken while the vault was open. Triggers: the attempt limit, the duress pattern, or "Destroy vault" in Settings (which asks for the real pattern). Not triggers: restart, app or iOS update, low storage, time passing.

We say "the key is destroyed, making the files unrecoverable". We don't say "securely erased": flash storage gives no such guarantee for the ciphertext, which is exactly why the design relies on destroying the key.

## 08 Duress pattern (Pro)

For personal safety: if someone is pressuring you to open the vault, a second pattern you set in advance destroys the real vault and opens an empty one, as if freshly installed.

- It has its own Argon2id hash with its own salt, and must differ from the real pattern.
- It's checked before the attempt counter, so it never counts as a failure. It is checked after any delay that's running, so it doesn't bypass the lock screen.
- No confirmation, animation or log. Since v1.1 the old vault's files are deleted in the background, so a duress unlock no longer gets slower as the vault grows. We haven't shown that its timing can't be told apart from a real unlock.
- Pro status only switches the feature on in the interface. It's held in memory, and nothing in the crypto code checks it.

It's a destructive alternative unlock, not a guarantee of safety. An empty vault may itself raise questions, and the decoy isn't meant to defeat forensic examination. We don't claim it's undetectable.

## 09 Share extension: add, never read

You can send files to Petersaint from other apps without opening the vault. By design the share extension holds only a public key: it can encrypt files *to* the vault but has nothing that decrypts.

- At set-up the app creates an inbox P-256 key pair. The private half is wrapped by the vault key. The public half sits in `Vault/inbox.pub` in the App Group container.
- For each shared file the extension makes an ephemeral P-256 key, runs ECDH against `inbox.pub`, and derives 64 bytes with HKDF-SHA256 (both public keys as salt, versioned info string). That splits into a metadata key and a content key. AAD binds the file's UUID.
- Files wait in `pending/`. At the next unlock the app decrypts them with the inbox private key, re-encrypts each under a fresh file key, and removes the pending copy. Malformed, oversized or duplicate pending files are rejected.
- The extension's entitlements include no Keychain access group, and the app's own Keychain queries are pinned to the app's group.

**Trust limit.** The App Group is writable by any code signed by our developer team, and the design trusts that code. Something that could replace `inbox.pub` would receive files shared until the next unlock. Since v1.1 the app writes `inbox.pub.sha256` beside it and the extension refuses to share if they disagree. That's a tamper and corruption check, not authentication: an attacker who can replace both files can redirect new shares to their own key.

## 10 Storage, metadata and device backups

```
<App Group>/Library/Application Support/Vault/
  index.enc          encrypted index
  blobs/<uuid>.enc   encrypted files
  thumbs/<uuid>.enc  encrypted thumbnails
  inbox.pub          share inbox public key (+ .sha256)
  pending/           share-extension drop zone
```

- On-disk names are random UUIDs. In the vault, original names, types, dates and folders exist only inside the encrypted index. Pending shares carry their names encrypted too.
- Someone with the files can still see how many blobs there are, roughly how big each one is, and file-system timestamps.
- Files and folders use `FileProtectionType.complete`, and the app's default data protection entitlement is Complete. Pending share files use `completeUnlessOpen` so a share can finish if the phone locks mid-write.
- The vault folder is marked excluded from iCloud and computer backups, and the mark is re-applied to the folder at every launch. This is deliberate: a device backup is a copy we can't protect. The only supported copy is the encrypted backup file in section 11.
- **Originals aren't moved.** Importing copies a file into the vault. The original stays in Photos, iCloud Photos or Files until you delete it (the app offers to delete Photos originals, which then sit in Recently Deleted for 30 days).
- The index is rewritten atomically (temp file, fsync, rename). A corrupt index fails closed: the unlock fails rather than showing an empty vault.

## 11 Encrypted backup (optional, since 1.2)

Backup is off until you turn it on. It adds one secret, a 12-word recovery phrase, and one file format. It adds no server, no account and no network code: the app still makes no connections, and a backup goes only where you save it.

### Recovery phrase and key

- Turning backup on generates 128 random bits and shows them once as 12 BIP-39 words. You type three of them back before backup is on.
- The bits are run through HKDF-SHA256 to make an X25519 key pair. **The phone keeps only the public key**, in the Keychain (when-unlocked, this device only). The phrase and private key are held in memory while the words are on screen, then zeroed.
- So the phone can make backups without the phrase, and nothing on the phone can open one.
- "New Recovery Phrase" makes a new pair; older backup files still need the older phrase. Turning backup off deletes the public key and leaves existing files valid.

### The backup file ( `.psbackup` )

| Part | Contents |
|---|---|
| Header | Magic `PSBK`, format version, fingerprint of the recovery public key, creation date. Plaintext |
| Key block | Ephemeral X25519 public key, and the vault key sealed with AES-256-GCM under a key from HKDF over the ECDH secret. The header is the AAD, so it can't be changed |
| Index | The wrapped index key and the encrypted index, byte for byte as stored on the phone |
| Records | Each encrypted file and thumbnail, byte for byte, keeping its UUID. Nothing is decrypted to make a backup, and each file's own AAD still verifies after a restore |
| Footer | Item count, total length and an unkeyed SHA-256 of everything before it, to catch truncation and damage early. Integrity comes from GCM, not from this hash |

Export asks for your pattern, streams the file into a protected temporary folder that only ever holds ciphertext, and hands it to the iOS save sheet. The temporary copy is removed when the sheet closes, on lock, and at launch.

### Restore

- Offered only on the welcome screen, when there is no vault. It is never on the lock screen, so nobody holding the phone can restore a backup of their own over your vault.
- Order: read the header, check the phrase's checksum, compare the key fingerprint ("different phrase" stops here), unseal the vault key, then verify the footer and every chunk of every record into a staging folder.
- Only when everything verifies does the app create a new Secure Enclave key, wrap the vault key to it, and rename staging into place. An interrupted restore leaves no vault: staging is cleared at the next launch.
- A backup carries no pattern, duress pattern, attempt counter or settings. A restored vault gets a new pattern and default settings.

## Handling the phrase on the phone

- Third-party keyboards are refused throughout the app, so the phrase and item names can't reach one. Apple's keyboards still work.
- Autocorrect and learning are off in the phrase fields. After a phrase is pasted, the clipboard is emptied. The app never copies the phrase and has no Copy button for it.
- The words appear on one screen only. A screenshot of it triggers a warning with an offer to delete it or make new words; recording and mirroring are covered like the rest of the app.

### What backup changes about wipe and duress

A backup file is a full copy of the vault and outlives a wipe. Self-destruct, the duress pattern and "Destroy vault now" destroy the key on the phone; they can't reach a file you saved somewhere else. Both wipe paths also delete the recovery public key, so the empty vault looks like a fresh install with backup off. If a copy that survives a wipe is a risk for you, don't turn backup on, or keep the file somewhere only you control.

> **What a backup file shows without the phrase.** Not contents, names, folders, types or dates of items. It does show: when the file was made and its total size; how many items it holds; each file's size, roughly each thumbnail's size, and which items have one; and the recovery key's fingerprint, so two files made with the same phrase can be linked. Hiding these would need a new format version.

- **The phrase is the whole key.** Anyone with a backup file and its 12 words can open the vault on any iPhone, with no pattern. Keep them apart.
- **Backups are manual.** A file holds the vault as it was at export. Items added later aren't in it until you export again.
- **Two phones restored from one backup share the vault key** and every existing file key. Items added afterwards on each get their own keys.
- **Any valid BIP-39 phrase is accepted** under "I Already Have a Phrase". The app tells you to use only the words Petersaint gave you, never a crypto wallet's.

## 12 Memory, screen and clipboard

- **Keys in memory.** Key buffers the app controls (vault key, index key, inbox key, every file key handed out) are overwritten on lock or wipe, as far as Swift allows (see section 13). A lock also stops a PDF or text decrypt that's already running, at the next chunk.
- **Viewers.** Photos, video, PDFs and text are decrypted into memory only. Video streams through a custom loader that reads chunks on demand and stops when the vault locks.
- **No copy-out.** PDFs are drawn as page images with no text layer, so there is no Copy, Look Up, Translate or Share. The text viewer isn't selectable.
- **Camera.** Photos taken in the app go from memory straight into the vault, never to the camera roll. Location is removed unless you turn it on.
- **Screen.** A cover hides content while iOS reports screen recording or mirroring, and in the app-switcher snapshot. iOS doesn't let apps block screenshots and only reports one after it's taken, so from v1.1 the app offers to delete it from Photos. Neither can guarantee nothing was captured.
- **Locking.** With auto-lock set to Immediately (the default), the vault locks when the app goes to the background. With 1 or 5 minutes, it's checked when the app comes back, and it also locks when iOS reports the phone is locking, if the app is running at that moment.
- **Logging.** The app's code makes no logging calls, and there's no analytics or crash-reporting SDK. iOS's own crash reports could still exist.

## 13 Known limitations

> **Software-enforced gate.** The pattern, attempt limit and waits are checked by Petersaint's code. Anyone who can run code inside the app on an unlocked phone can ask the Secure Enclave for the vault key without the pattern. That needs an iOS or app vulnerability or a jailbreak, but it's the boundary of what we protect.

> **No recovery without your own backup.** Forget the pattern, or lose or replace the phone, and the files on it are gone. There is no reset and no support override. The only way back is a backup file you exported and its 12-word phrase (section 11). This is the price of keeping the only key on one device.

- **The pattern hash is an offline oracle.** Someone who extracts the Keychain (for example, a forensic image taken while the phone is unlocked, with its passcode) could try every 3×3 pattern against the Argon2id hash in hours. Argon2id makes each guess expensive; it doesn't make the pattern space bigger. Learning the pattern doesn't unwrap the vault key off the phone, but with the phone back in hand they can simply draw it. The 3×6 grid and longer patterns raise the cost.
- **Anyone holding the phone can destroy the vault.** With an attempt limit set, a few wrong patterns wipe it, by design. If losing the files to someone else's tampering worries you more than them being read, leave self-destruct off (the default since 1.1.3).
- **No rollback protection.** There's no version counter on the index. Writing an older copy of `index.enc` back, with its blobs, brings back an older state of the vault, including items since deleted.
- **Zeroing is best effort.** Keys are overwritten through pointers that CryptoKit and CoreFoundation expose. That works today but relies on implementation details. Decoded images, PDF documents and AVFoundation buffers aren't zeroed.
- **Auto-lock delay.** With auto-lock set to 1 or 5 minutes, the vault key can stay in memory while the app is suspended, past the time the phone locks. "Immediately", the default, avoids this.
- **Photos import makes a short-lived plaintext copy.** A file picked from Photos is briefly written to the app's temp folder (protection class Complete) while it's encrypted, then deleted. Leftovers are cleared at launch. Deleting a file isn't secure erasure of the flash blocks it used.
- **Files you open are parsed by iOS.** Images, video and PDFs are decoded by Apple's frameworks. A crafted file shared into the vault is only as safe as those decoders.
- **Deleting one item isn't a crypto-shred.** The index key doesn't rotate. Someone who copied the files earlier and later gets the vault key could read an item you've since deleted. Only a whole-vault wipe destroys keys.
- **Settings can be loosened while the vault is open.** Raising the attempt limit, turning self-destruct off, or extending auto-lock doesn't ask for the pattern. Changing the pattern, duress, export and destroy do.
- **Disguise is cosmetic.** The alternative icons hide the app on the Home Screen, but the share sheet and privacy cover still say "Petersaint". It's not a security feature.
- **libsodium is a prebuilt binary,** trusted by git commit hash rather than built from source.

## 14 Code review

On 19 October 2026, before release, the code was reviewed against 24 specific threats by a reviewer that had no part in writing it: an AI coding agent working in a fresh session, with the full source and spec. It ran the test suite,

added failing tests to prove each finding, and inspected the Release binary for debug code.

It found no path for someone without the pattern to read vault contents. It reported ten findings, all about availability, the delay clock and defence in depth. Eight were fixed in v1.0 build 2 and 3; the other two (duress timing and the inbox key check) were fixed in v1.1. Its notes on residual risk are the basis for section 13.

On 21 November 2026 the 1.2 backup code (the phrase, the key block, the file format, restore, and the screens that show or accept the phrase) was reviewed in the same way against 26 threats. It found no way to open a backup file without the 12 words and no way for a restore to replace or half-build a vault. It reported no critical or high findings. Two medium findings (auto-lock stopping long exports and restores, and third-party keyboards seeing the phrase) and one low finding (a pasted phrase left on the clipboard) were fixed before release. One low finding, what a backup file shows without the phrase, is accepted for this format version and described in section 11.

That's internal evidence, not assurance. An AI review of 24 chosen threats isn't a paid third-party audit, and passing tests don't prove there are no undiscovered bugs. Some checks in it need a signed build on a real iPhone.

## 15 Design choices compared

Vault apps make different trade-offs. None of these is wrong; they suit different people. This is where Petersaint stands.

| Choice | Common alternative | Petersaint |
|---|---|---|
| Where the key comes from | Derived from the PIN or pattern with PBKDF2 or similar | Random vault key, wrapped by a Secure Enclave key; pattern is a gate only |
| Key per file | One master key for every file | A separate random key for every file |
| Pattern hashing | PBKDF2 (CPU-hard) | Argon2id, 64 MiB (memory-hard) |
| Backup | Automatic encrypted cloud backup | Optional encrypted file you export and store yourself; the vault stays out of device backups |
| Recovery | Recovery phrase or account reset | 12-word recovery phrase with a backup file; no pattern reset and no account |
| Wrong guesses | Delays or lockouts | Growing delays, plus an optional key-destroying limit |

A key derived from your PIN or pattern can be re-created on another phone, which makes backup and recovery easier to build. A Secure Enclave key can't leave the phone, and that's the point. Petersaint's backup works around

that by sealing the vault key to a second key that only your 12 words can rebuild. If you want backups that happen automatically, a vault built around a cloud may suit you better.

Background reading: Apple Platform Security: the Secure Enclave, RFC 9106: Argon2, NIST SP 800-38D: GCM. For a plain-English version, see what "encrypted" really means.

---

Petersaint is made by Johannson Limited, a company registered in England and Wales, no. 07760655. iPhone and App Store are trademarks of Apple Inc.