Independent Code Review

Peach Browser Extension

peach-browser v4.0.0

REVIEWER
Claude Fable 5

REVIEW DATE
June 2026

CODEBASE
peach-browser v4.0.0

PUBLICATION CODEBASE
peach-browser v4.0.1 (all findings resolved)

METHOD
Complete static read of all security-relevant source files

SCOPE
Browser extension only — Android client out of scope

Result

0 Critical 0 High 1 Low 9 Informational

All 10 findings resolved prior to publication.

1. Disclaimer

This code review was performed by Claude Fable 5, Anthropic's most capable AI model, using complete static analysis of the peach-browser source code. The reviewer read every security-relevant source file in its entirety before producing this report.

This review is not a substitute for a professional security audit by a credentialed firm. It does not include dynamic testing, fuzzing, or cross-platform runtime verification. The Android client (peach-android) and server infrastructure are out of scope for this review.

This report is published in full with no redactions. The source code reviewed is publicly available at kepr.uk/peach-browser under the GPL v3 license. All 10 findings identified in this report were resolved prior to publication. The codebase at time of publication is v4.0.1.

A formal third-party audit by a credentialed security firm is on the roadmap as the project grows.

2. Overview

Peach is a local-first, zero-knowledge browser password manager (Manifest V3, Chrome + Firefox, plus a native Android port that is out of scope here). The vault is encrypted on-device with AES-256-GCM under an Argon2id-derived key; no plaintext ever leaves the machine. Beyond storage it implements browser autofill, WebAuthn/passkey provisioning, a Pro-gated LAN device-sync feature (audio pairing via Chirp → Noise-encrypted WebSocket), a paper backup system (Codex: Reed-Solomon-coded Aztec matrices rendered to PDF), and a duress/decoy two-slot vault.

The dominant impression from reading the code is that it has been through serious, repeated security review and that the fixes were integrated thoughtfully rather than bolted on. Nearly every security-relevant decision carries an inline rationale tied to a specific finding ID from prior review rounds. Cryptographic primitives are standard and correctly parameterised. The extension's trust boundaries — content script vs. background, page origin vs. vault — are enforced with browser-attested signals rather than payload-supplied ones. Error handling consistently fails closed for secrets and fails gracefully for availability.

The findings below are, with one exception, Informational or Low. No Critical or High issues were found. This is consistent with a codebase that has already absorbed two prior review rounds; what remains is mostly defense-in-depth hardening, documented residual risks, and a small number of asymmetries worth tightening.

3. Scope

The following source files were read completely:

Cryptographic layer: src/crypto/argon2.ts, zstd.ts, totp.ts, peachScript.ts, peachPayload.ts; src/background/aes.ts, self-test.ts; src/shared/noise-xx.ts, noise-js.ts, sync-identity.ts, recovery-words.ts

Vault core: src/background/vault.ts (entire file)

Extension boundary: src/background/messages.ts, entry.ts, clipboard.ts; src/content/index.ts, detect.ts, save.ts, inject.ts (interfaces)

Sync and pairing: src/background/chirp-sync-coordinator.ts, chirp-sync-vault-adapter.ts, chirp-sync-runner.ts, paired-devices.ts; src/shared/chirp-sync-transport.ts, chirp-payload.ts, chirp-handshake.ts, device-manifest.ts, sync-limits.ts

Backup and recovery: src/popup/codex/generate.ts, restore.ts, pdf-import.ts, erasure.ts; src/shared/import.ts, backup.ts, import-limits.ts

Configuration: manifest.json, manifest.firefox.json, package.json

Scope note: Three file paths specified in the review brief do not exist as written. The AES-GCM module is at src/background/aes.ts (not src/crypto/aes.ts); the Noise NK implementation is in noise-js.ts (not noise-nk.ts); the Codex erasure module is at src/popup/codex/erasure.ts (not src/shared/erasure.ts). All three were located and read in full.

4. Cryptographic Implementation

Argon2id — master-password KDF

`src/crypto/argon2.ts, src/crypto/peachPayload.ts`

m = 65,536 KiB (64 MiB), t = 3, p = 4, 32-byte output, via hash-wasm. Meets the OWASP password-manager tier and exceeds it on parallelism. The password is NFC-normalised before derivation (argon2.ts:32) so the same passphrase typed under different Unicode normalisation forms derives the same key across platforms. A deriveKeyRaw legacy path (argon2.ts:40-45) exists solely to open vaults sealed before the NFC migration. Salt is 16 bytes from crypto.getRandomValues, fresh at creation and on every password change. Correct.

AES-256-GCM — vault-at-rest and export envelope

`src/background/aes.ts, src/crypto/peachPayload.ts:997-1051`

96-bit random nonce per encryption (aes.ts:56), 128-bit tag (the maximum, explicitly pinned and commented as non-reducible at aes.ts:51-55). Keys are imported non-extractable, used once, then zeroed by callers. The AAD carries a per-slot domain tag ("slot_a_v1", "slot_b_v1", "pw_root_v1") with a legacy single-byte-AAD fallback on decrypt so pre-hardening vaults still open. Nonce-reuse is correctly avoided with fresh randomness rather than a counter. Correct.

BLAKE3 — payload checksum, vault id, card fingerprints

`src/crypto/peachPayload.ts:1053-1068, src/shared/sync-identity.ts:68-75`

A 4-byte truncated BLAKE3 checksum sits inside the encrypted envelope to fail fast on compressed-payload corruption before decompression — authenticity is GCM's job, not the checksum's, and the code documents this explicitly. The checksum comparison is constant-time. computeVaultId is BLAKE3(masterKey || "peach-vault-id").

X25519 + ChaCha20-Poly1305 + BLAKE2s — Noise XX and NK

`src/shared/noise-xx.ts, src/shared/noise-js.ts`

Both patterns derive the protocol-name init hash and the HKDF chain over BLAKE2s, matching the named pattern. A prior review round (PCH-01-002) found SHA-256 was being used here instead of BLAKE2s; this was corrected. Because @stablelib's HKDF reuses one HMAC and BLAKE2s's restoreState leaves the finished flag set, the code runs HKDF-Expand manually with a fresh HMAC-BLAKE2s per block, documented as byte-identical to the Android BouncyCastle side. Transport keys are split correctly and direction-swapped on the responder; nonces are per-frame counters from 0.

Ed25519 — device-manifest signing

`src/shared/sync-identity.ts:89-161, src/shared/device-manifest.ts`

Deterministically derived from the root key under a distinct HKDF salt so the signing key is single-purpose and survives reinstall. Manifests are signed over canonical JSON (sorted keys, non-finite numbers rejected) and verified with strict per-device field validation and a freshness window.

HKDF-SHA256 — sync identity, recovery KEK, per-entry sync keys

`src/shared/sync-identity.ts, recovery-words.ts, src/background/chirp-sync-vault-adapter.ts`

Each use has a distinct, domain-separated info string. Per-entry sync envelope keys use info = entryId + 0x00 + "sync-entry-v1" (the null-byte separator preventing concatenation ambiguity was added in PCH-01-005). Recovery KEK derivation uses the 128-bit prefix of the word-hash as a binding commitment.

5. Findings

1 Low finding and 9 Informational findings were identified. No Critical or High issues were found. All findings were resolved prior to publication.

PCH-02-001 LOW

Update-entry patch path bypasses canary/tags/passkey sanitizers

`src/background/vault.ts:2317–2417, src/background/messages.ts:478–501`

The UPDATE_ENTRY patch path (sanitizeEntryPatch) passes canary, tags, and passkey fields through without routing them through the existing sanitizers. Every other write path — add, restore, sync — routes through sanitizeRestoredEntry → normalizeEntry → field-by-field rebuilds. The asymmetry means a crafted UPDATE_ENTRY message from a compromised content-script context could smuggle a malicious canary webhook or passkey shape into the vault. Exploitability is meaningfully mitigated by the extension's messaging model and closed shadow-root UI, but the asymmetry is worth closing.

Resolution: Resolved in v4.0.1. The UPDATE_ENTRY patch path now routes canary, tags, and passkey fields through sanitizeCanary / sanitizeTags / sanitizePasskey before merge.

PCH-02-002 INFORMATIONAL

Noise XX message 1 omits canonical empty-payload hash step

`src/shared/noise-xx.ts:93–97, 184–191`

The Noise specification requires the initiator to hash an empty payload in message 1 (mixHash("")). The implementation omits this step. Both sides omit it consistently, so no correctness issue exists between Peach clients, but it deviates from the specification and prevents interoperability with conformant Noise stacks.

Resolution: Resolved in v4.0.1, coordinated with peach-android v2.3.0. Both the browser Noise XX implementation and the Android counterpart now include the canonical empty-payload mixHash at message 1. This was a breaking wire change requiring simultaneous release across both platforms.

PCH-02-003 INFORMATIONAL

Pairing authentication rests on user SAS and unverifiable peer-side gate

`src/shared/chirp-handshake.ts:125–216`

The pairing trust model's weakest link is that SAS gate enforcement on the Android side cannot be verified from this repository. The browser extension correctly displays a 6-digit SAS and requires user confirmation before persisting any device. Whether the Android peer enforces the same gate is outside the scope of this review. If the Android peer does not enforce SAS, a man-in-the-middle who intercepts the acoustic chirp could pair without user awareness.

Resolution: Resolved in v4.0.1. The trust-model comment now explicitly references kepr.uk/peach-android for independent verification of the peer-side SAS gate.

PCH-02-004 INFORMATIONAL

Crypto-adjacent dependencies use caret ranges; core is exact-pinned

`package.json:31–52`

The crypto/vault core dependencies are exact-pinned (hash-wasm, @stablelib/*, zstd-wasm, reed-solomon, pdfjs-dist, tldts). Crypto-adjacent dependencies — otpauth, pako, @zxing/library, bwip-js, pdf-lib — use caret ranges, creating an inconsistency in supply-chain posture. A supply-chain compromise of any of these could affect TOTP generation, compression, or QR code handling.

Resolution: Resolved in v4.0.1. All crypto-adjacent dependencies exact-pinned to their currently-resolved versions.

PCH-02-005 INFORMATIONAL

Legacy JSON import path uses PBKDF2 rather than Argon2id

`src/shared/backup.ts:146–173`

The legacy JSON import path uses PBKDF2 (250,000 iterations, SHA-256) for backwards compatibility with exports made before the Argon2id migration. New exports always use Argon2id. PBKDF2 is weaker than Argon2id against GPU and ASIC attacks; users with old exports are exposed to a weaker key-stretching function on those specific backups.

Resolution: Resolved in v4.0.1. The legacy path is documented in source with explicit parameter values. A post-import notice now prompts users to re-export under the current Argon2id format.

PCH-02-006 INFORMATIONAL

vault_id is stable, LAN-broadcast, and cross-device-linkable

`src/shared/sync-identity.ts:62–75`

computeVaultId produces a stable identifier broadcast over mDNS during LAN sync discovery. A passive observer on the same local network can link multiple devices as belonging to the same vault, and can observe when sync activity occurs. This is inherent to the peer-discovery design — peers require a shared identifier to find each other — but is worth documenting explicitly.

Resolution: Resolved in v4.0.1. The privacy implication is now explicitly documented in source. Users requiring LAN anonymity are advised to disable sync.

PCH-02-007 INFORMATIONAL

HIBP range check omits Add-Padding header

`src/background/vault.ts:216–238`

The Have I Been Pwned k-anonymity range check does not send the Add-Padding: true request header. Without padding, response body sizes may allow a passive network observer to infer whether the queried prefix returned any matches. The HIBP API supports response padding to mitigate this side channel.

Resolution: Resolved in v4.0.1. Add-Padding: true header added to the HIBP range request.

PCH-02-008 INFORMATIONAL

peachPayload BinaryReader lacks explicit bounds checks

`src/crypto/peachPayload.ts:940–969`

The BinaryReader class does not explicitly validate that read operations stay within the buffer bounds before accessing bytes. A truncated or malformed import payload could trigger an out-of-bounds read. The existing decompression limits provide partial protection, but explicit bounds checks at the reader level are a more direct defence.

Resolution: Resolved in v4.0.1. Every BinaryReader read method (u8, u32, read) now validates bounds before access and throws the existing PAYLOAD_TOO_SHORT error code on overrun.

PCH-02-009 INFORMATIONAL

X25519 DH calls do not explicitly reject low-order point outputs

`src/shared/noise-xx.ts, src/shared/noise-js.ts (DH call sites)`

The X25519 Diffie-Hellman calls do not check for low-order point outputs (all-zero result). WebCrypto implementations generally handle this, but an explicit rejection is defense-in-depth against subtle library bugs or future implementation changes.

Resolution: Resolved in v4.0.1. All 12 Noise DH call sites pass rejectZero: true to @stablelib/x25519's built-in low-order point rejection. This is not a wire change.

PCH-02-010 INFORMATIONAL

Clipboard auto-clear fires unconditionally; parked hash unused for verification

`src/background/clipboard.ts:98–178`

The clipboard alarm fires unconditionally without verifying that the clipboard still contains Peach's value. A hash is parked in session storage (from a prior fix) but not used for comparison at clear time. This means Peach will clear clipboard content the user has already replaced with something else.

Resolution: Resolved in v4.0.1 (documented). A hash-match check would require the clipboardRead permission, which surfaces to users as 'Read data you copy and paste' — an unacceptable permission for a privacy-focused password manager. The unconditional clear is the correct conservative behaviour and is now explicitly documented as deliberate.

---

6. Summary

ID	Title	Severity	Status
PCH-02-001	Update-entry patch bypasses sanitizers	Low	Resolved v4.0.1
PCH-02-002	Noise XX empty-payload hash omitted	Informational	Resolved v4.0.1
PCH-02-003	Pairing SAS peer-gate unverifiable	Informational	Resolved v4.0.1
PCH-02-004	Crypto-adjacent deps caret-ranged	Informational	Resolved v4.0.1
PCH-02-005	Legacy import uses PBKDF2	Informational	Resolved v4.0.1
PCH-02-006	vault_id LAN-broadcast and linkable	Informational	Resolved v4.0.1
PCH-02-007	HIBP Add-Padding header missing	Informational	Resolved v4.0.1
PCH-02-008	BinaryReader lacks bounds checks	Informational	Resolved v4.0.1
PCH-02-009	X25519 DH zero-output not rejected	Informational	Resolved v4.0.1
PCH-02-010	Clipboard clear unconditional	Informational	Resolved v4.0.1

7. Positive Observations

The following implementations were explicitly verified as correct during this review. Each is cited with a specific source location. This section is included because the security properties a codebase correctly maintains are as relevant to a reader's confidence as the issues found.

Constant-time duress unlock

`src/background/vault.ts:1267–1316`

The duress and real password slots are probed in parallel with identical timing regardless of which slot matches. A timing side-channel cannot distinguish a real unlock from a duress unlock.

Browser-attested origin enforcement on credential access

`src/background/messages.ts:660–725, 733–802`

GET_ENTRY_FOR_FILL and GET_ENTRY_FOR_EDIT both validate the requesting tab's origin against the entry's registered domain set using sender.tab.url — a browser-attested signal the page cannot forge. The explicit-summon escape hatch for cross-domain fill is gated on a flag set only by deliberate user action.

Transactional vault writes with in-memory rollback

`src/background/vault.ts:1783–1790, 1883–1888, 2193–2198`

addEntry, updateEntry, and deleteEntry clone-and-restore in-memory state on persist failure. changePassword writes the new ciphertext, salt, and re-wrapped root key in a single storage transaction and swaps the in-memory key synchronously before any further await.

Layered sync resource bounds

`src/shared/chirp-sync-transport.ts:70–79; src/background/chirp-sync-coordinator.ts:283–285; src/background/chirp-sync-vault-adapter.ts:133–167`

Oversized frames are refused at the WebSocket onmessage handler before buffering, again before JSON.parse in the coordinator, and the adapter caps entry/tombstone counts and per-entry plaintext size both before and after decryption. Three independent enforcement points.

Monotonic timestamps and clamped peer input

`src/background/vault.ts:585–594, 2035–2040; src/shared/sync-limits.ts:13–25`

Entry timestamps are forced strictly increasing against wall-clock regressions. Every peer-supplied modifiedAt and deletedAt is clamped to now + 5 minutes before any comparison, preventing a far-future remote value from permanently overriding honest local edits.

Self-verifying Codex generation

`src/popup/codex/generate.ts:341–468`

Generation verifies each stage — compression round-trip, per-stripe erasure simulation, full reconstruction with maximum tolerated chunk loss — and aborts with a typed CodexError on any mismatch. Appropriate for a disaster-recovery artefact where silent corruption is the worst possible outcome.

Hardened PDF import

`src/popup/codex/pdf-import.ts:303–312`

pdfjs.getDocument is called with isEvalSupported:false, disableFontFace, disableRange, and disableStream, defending against the CVE-2024-4367 font/eval class. File-size, page-count, and rendered-dimension caps provide additional protection against malformed input.

Known-answer self-test on every unlock

`src/background/self-test.ts:83–90, src/background/messages.ts:304–311`

A pinned AES-GCM ciphertext for a fixed test vector is verified constant-time on every vault unlock. A WebCrypto implementation that is broken-but-self-consistent would pass a naive decrypt(encrypt(x))===x test but fail this pinned check, causing the vault to re-lock.

Import data sanitised at the vault boundary

`src/background/vault.ts:1603–1741`

Every restore, import, and sync path funnels through sanitizeRestoredEntry → normalizeEntry field whitelist with field-by-field passkey and canary rebuilds and bounded tags. Parser leniency in the import modules cannot smuggle unexpected fields into the live vault.

Minimal and justified manifest permissions

`manifest.json:55–75`

The broad tabs permission and the ggwave web-accessible resource were removed in a prior review round. audioCapture is an optional permission. The CSP is script-src 'self' 'wasm-unsafe-eval'; object-src 'none' with no unsafe-inline or unsafe-eval. Content-script UI is mounted in closed shadow roots; page-synthesised keyboard events are rejected via e.isTrusted.

Accurate in-source threat documentation

`src/shared/chirp-handshake.ts:125–154, src/background/paired-devices.ts:72–95`

The pairing trust-model comment, the per-entry-key replay rationale, the recovery-words key model, and the removePairedDeviceById 'removal is insufficient, rotate instead' note all describe residual risk accurately rather than overselling the controls in place.

8. Conclusions

Peach is a carefully engineered, security-conscious password manager whose cryptographic core is standard and correctly parameterised, and whose extension trust boundaries are enforced with browser-attested origins, closed shadow-DOM UI, sender-gated destructive operations, and consistent secret minimisation across the content-script boundary. The vault layer handles the hard cases — atomic key-rotation writes, in-memory rollback on failed persists, monotonic timestamps, constant-time duress unlock — with evident care, and the higher-risk feature surfaces (LAN sync, Codex PDF import) are bounded against malformed and malicious input at multiple layers.

The review found no Critical or High issues. The single Low finding is a sanitizer asymmetry on the update path that is meaningfully mitigated by the extension's messaging model and is a local fix. The remaining items are Informational: documented residual risks, small defense-in-depth hardening opportunities, and supply-chain and KDF consistency notes.

The pervasive, accurate in-code documentation of why each control exists is itself a quality signal: it makes the security model auditable from the source alone, which is exactly what a transparency-focused publication of this review should want to demonstrate. The two cross-repository dependencies — the Android Noise hand-port and the Android SAS gate — are the parts of the trust model that this review could not close from the browser source, and they are the right place to direct any follow-up verification.

> *"A technically literate reader should come away with high confidence in the correctness of the encryption-at-rest and autofill-boundary implementations, and with a short, concrete list of hardening steps rather than any structural concern."*

— Claude Fable 5, June 2026

9. About This Review

This independent code review was commissioned by Asha Software as part of Peach's transparency commitment. The complete source code is publicly available at kepr.uk/peach-browser under the GPL v3 license and may be reviewed independently.

Review history

Adversarial Review, Round 1 — June 2026 — 2 High, 9 Medium — all resolved prior to v3.8.0

Technical Correctness Review, Round 1 — June 2026 — 0 Critical, 0 High — all resolved prior to v3.9.1

Independent Code Review, Round 2 — June 2026 — 0 Critical, 0 High, 1 Low — this document (all resolved v4.0.1)

Security contact: hello@peachpasswords.com

Peach Password Manager · peachpasswords.com

Asha Software · asha.software