Mobile Wallet Security Audit

Vortex Mobile Wallet · Multi-Chain Non-Custodial

PROJECT	Vortex (vortexdappworld)
APPLICATION	Vortex Mobile Wallet
TYPE	Non-custodial multi-chain crypto wallet
PLATFORMS	iOS · Android · Desktop · Web · Browser Extension
ARCHITECTURE	Yarn monorepo — TypeScript / React Native
REPOSITORY	vortexdappworld-creator/VortexMobileWallet
COMMIT	dba6202 — “Initial commit”
CHAINS	BTC, ETH, Solana, Tron, BNB Smart Chain, +more
SOURCE	Open-source (MIT)
ASSESSMENT	AI-assisted static review — critical surfaces

Audit Date: June 8, 2026

Report Version: 1.2

Table of Contents

1. Executive Summary

This report presents the results of an independent, AI-assisted security review of the **Vortex Mobile Wallet** — a non-custodial, multi-chain crypto wallet. The wallet is a Yarn monorepo targeting iOS, Android, desktop (Electron), web, and a browser extension, and manages users' seed phrases, private keys, and on-chain transactions across Bitcoin, Ethereum, Solana, Tron, BNB Smart Chain and other networks.

The assessment was conducted as a **standard-pass static review of the wallet's security-critical surfaces**: key & mnemonic management and cryptography; encrypted storage at rest, authentication and biometrics; the network / RPC / API layer, secrets and telemetry; dApp connectivity, WalletConnect and message signing; and supply-chain integrity together with mobile, desktop and extension platform hardening. The review was performed by reading the relevant TypeScript, native (Java / Swift / Objective-C) and configuration source, with all findings cited to file and line.

According to our assessment, the application's security posture is:

Well Secured	
Secured	✓
Poor Secured	
Insecure	

The cryptographic core is mature, well-engineered code. The review confirmed cryptographically-secure randomness throughout, authenticated encryption (AES-256-GCM) for secrets at rest, use of audited cryptographic libraries, strong transaction-approval and blind-signing controls for dApps, hardened Electron and browser-extension configurations, and **no committed secrets, no weak RNG, no disabled TLS, and no broken signing paths**. There were **zero critical-severity and zero high-severity issues**.

The most material findings are **medium-severity at-rest protection items on mobile** — biometric unlock that is enforced in JavaScript rather than bound to the OS keychain access-control (F-01), and a reduced PBKDF2 iteration count on Android (F-02). Both require local device or encrypted-database compromise to exploit and are not remotely reachable. The remaining findings are data-leakage and defensive-hardening items in the medium, low and informational ranges.

Findings Summary

0	0	4	5	7
CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

A total of **16 observations** were recorded across the monorepo. None are critical, high-severity, or remotely exploitable. Priority remediation is **F-01** and **F-02** (mobile at-rest hardening), followed by the remaining medium data-leakage items (**F-03**, **F-04**).

2. Project Information

2.1 Application Description

Vortex Mobile Wallet is a non-custodial, multi-chain crypto wallet. Private keys and seed phrases are generated, encrypted, and stored entirely on the user's device; the wallet never transmits key material off-device. The codebase is organised as a Yarn-workspaces monorepo with a strictly enforced import hierarchy.

Security-sensitive logic is concentrated in the `core` (cryptography, key derivation, signing), `shared` (app crypto, storage, request layer), and `kit-bg` (background services: password, account, dApp provider) packages, while UI lives in `kit` and `components`.

2.2 Repository & Build

Field	Value
Application Name	Vortex Mobile Wallet
Repository	`vortexdappworld-creator/VortexMobileWallet`
Reviewed Commit	`dba6202 — "Initial commit"`
Language / Stack	TypeScript, React / React Native (Expo), Electron
Package Manager	Yarn 4.x workspaces
Platforms	iOS, Android, Desktop (macOS/Windows/Linux), Web, Chrome Extension
Android Application Id	`world.vortexdapp.wallet`
License	MIT (SPDX)
Source Availability	**Open-source — independently reviewable**

2.3 Audited Packages

Package	Responsibility	Security Relevance
`packages/core`	Key derivation, encryptors, signing, per-chain crypto	Critical
`packages/shared`	App crypto modules, secure storage, request layer, config	Critical

packages/kit-bg	Background services: password, account, dApp provider, DB	Critical
packages/kit	UI flows: signing confirmation, WebView, DApp connection	High
apps/mobile	iOS / Android native shells & manifests	High
apps/desktop	Electron main process, preload, updater	High
apps/ext	Browser-extension manifest, content scripts	High
patches/	56 dependency patches (supply-chain surface)	High

2.4 Key Security Dependencies

Library	Role	Status
bip39 3.1.0	Mnemonic generation (default CSPRNG)	Verified
@noble/curves, @noble/hashes, @noble/ciphers	Curves, hashing, authenticated ciphers	Verified
elliptic 6.6.1	secp256k1 signatures (canonical low-S)	Verified
react-native-get-random-values	Native CSPRNG shim	Verified
expo-secure-store / Keychain / Keystore	OS secure storage backend	See F-01
@walletconnect / Reown	dApp session transport & origin verification	Verified

3. Audit Scope & Methodology

3.1 Scope

The audit covers the **security-critical surfaces** of the Vortex Mobile Wallet at the reviewed commit, across the following five domains:

- **Key & mnemonic management and cryptography** — entropy sources, HD derivation, signing, encryptors.
- **Encrypted storage at rest, authentication, biometrics & app-lock** — KDF, cipher mode, secure-storage backends, biometric gating, clipboard, screenshot protection.
- **Network / RPC / API, secrets & telemetry** — hardcoded secrets, TLS, endpoints, analytics and crash-reporting data flows.
- **dApp connectivity, WalletConnect & message signing** — provider RPC handling, blind-signing controls, origin binding, scam detection.
- **Supply-chain integrity & platform hardening** — dependency patches, mobile manifests, Electron configuration, extension CSP and permissions.

3.2 Out of Scope

The following were explicitly out of scope for this engagement:

- Exhaustive line-by-line review of every package in the monorepo.
- The internals of third-party dependencies (beyond review of the project's own `patches/` directory).
- Dynamic / runtime testing (DAST), instrumented device testing, and live exploitation.
- Formal verification, and any backend / server-side infrastructure.
- The contents of `node_modules/`.

3.3 Methodology

The following techniques were applied during the audit:

1. **AI-assisted static source review.** Parallel domain reviewers located and read the security-critical source for each of the five domains above, producing findings with exact `file:line` evidence.
2. **Secret & credential scanning.** The repository (excluding `node_modules`) was scanned for hardcoded API keys, tokens, private keys, mnemonics, AWS/Bearer patterns, and high-entropy strings, with all hits manually triaged.
3. **Cryptographic review.** Entropy sources, KDF parameters, cipher modes, IV/salt/nonce handling, and signing serialization were inspected against current best practice (OWASP, NIST).

4. **Supply-chain review.** All 56 dependency patches were enumerated and every patch touching crypto, randomness, networking, storage, signing or security libraries was deep-inspected for weakening changes, exfiltration, or backdoors.
5. **Platform configuration review.** Android manifests, iOS `Info.plist` / entitlements, Electron `BrowserWindow` / Fuses, and extension manifest CSP / permissions were checked against platform hardening guidance.
6. **Consolidation & rating.** Findings were de-duplicated and assigned a qualitative severity reflecting *impact × exploit precondition*.

> **Note on provenance.** The repository contains a single squashed “Initial commit”, so findings are reported on the code *as-is* at the reviewed revision rather than as a diff against any prior revision history.

4. Severity Classification

Findings are classified using the following severity scale, adapted from the OWASP Mobile / Application risk-rating frameworks for a non-custodial wallet:

Severity	Definition
CRITICAL	Directly and remotely reachable path to theft of user funds or compromise of key material, requiring no special preconditions. Must be fixed before any release.
HIGH	Compromise of key material or funds reachable under realistic conditions with limited preconditions. Strongly recommended to fix before production release.
MEDIUM	Issues requiring local device or encrypted-storage compromise, information disclosure, privacy leakage, or weakening of a defensive control that does not by itself lead to remote fund loss. Should be mitigated.
LOW	Defensive hardening or minor inconsistencies a careful operator should be aware of.
INFORMATIONAL	Observations about configuration, privacy posture, or notes confirming that a relevant best practice is followed.

5. Domain-Level Verification

Each security-critical control was individually verified. **Passed** means the control is implemented to current best practice with no defect identified. **Notice** indicates a finding worth disclosure (see §6) without an immediately exploitable, remote defect.

5.1 Key Management & Cryptography

Control	Result
Mnemonic entropy via CSPRNG (no weak / custom RNG)	**Passed**
HD key derivation (BIP32 / SLIP-0010), unmodified	**Passed**
Authenticated encryption (AES-256-GCM) for secrets at rest	**Passed**
KDF strength — non-Android (PBKDF2 600k)	**Passed**
KDF strength — Android (PBKDF2 5k)	**Notice (F-02)**
Deterministic signing serialization (no `JSON.stringify` misuse)	**Passed**
No weak primitives (MD5 / SHA-1 / ECB) in security paths	**Passed**
No committed mnemonics / seeds / private keys (prod)	**Passed**
In-memory secret zeroization (Bot Wallet / CLI export)	**Passed**

5.2 Storage, Authentication & Biometrics

Control	Result
Seed / private keys encrypted in local DB (never plaintext AsyncStorage)	**Passed**
Biometric unlock bound to OS keychain access-control	**Notice (F-01)**
Cryptographic password verification (decrypt verify-string)	**Passed**
App auto-lock + in-memory password cache TTL clear	**Passed**
Screenshot / screen-recording protection	**Notice (F-03)**
Clipboard auto-clear for copied secrets	**Notice (F-05)**
No secret values written to logs	**Passed**

Legacy encoder surface (CBC / XOR retained)	Notice (F-11)

5.3 Network, RPC & Telemetry

Control	Result
No hardcoded API keys / secrets (build-injected from CI)	Passed
TLS enforced; no cleartext API/RPC (prod)	Passed
No disabled certificate validation	Passed
SSRF / metadata-endpoint protection	Passed
EVM address EIP-55 checksum validation	Passed
Crash-report scrubbing (keys / mnemonics / tokens)	Passed
Telemetry excludes addresses / balances / keys	Passed
Recipient name resolution verified on-chain	Notice (F-06)
Push-notification payload not logged (prod, native)	Notice (F-04)
iOS push Service-Extension ATS	Notice (F-07)

5.4 dApp Connectivity & Signing

Control	Result
Origin derived from trusted browser context (no cross-origin confusion)	Passed
Blind signing (`eth_sign`) always high-risk + gated	Passed
EIP-712 typed-data schema + chainId validation	Passed
dApp RPC restricted to safe-method allowlist	Passed
WalletConnect origin / Reown Verify enforcement	Passed
Phishing / scam detection with block view	Passed
Native WebView navigation guard (`originWhitelist` wildcard)	Passed*
Default signature-risk alert styling	Notice (F-08)

* Wildcard `originWhitelist` is mitigated by a per-navigation validation guard; see F-14.

5.5 Supply Chain & Platform Hardening

Control	Result
Dependency patches reviewed (56) — no weakening / backdoor	**Passed**
Pinned `resolutions` for security libs; pinned git SHAs	**Passed**
No release keystores / signing certs / service accounts committed	**Passed**
Android release hardening (allowBackup / cleartext / debuggable)	**Passed**
First-party Android FileProviders not exported	**Notice (F-09)**
iOS ATS / entitlements / FaceID usage description	**Passed**
Electron hardening + Fuses (contextIsolation / sandbox / ASAR integrity)	**Passed**
Extension CSP & least-privilege permissions	**Passed**
Desktop file-protocol traversal normalization	**Notice (F-16)**

6. Audit Findings

6.1 Critical Issues

No critical-severity issues were identified.

6.2 High Issues

No high-severity issues were identified.

6.3 Medium Issues

MEDIUM F-01 · Biometric unlock is a JS-side gate, not bound to OS keychain access-control

Domain: Authentication / Secure Storage · **Location:** `packages/kit-bg/src/services/ServicePassword/biologyAuthUtils.ts:49`; `packages/shared/src/storage/secureStorage/index.native.ts:14–18, 46–52`; `packages/kit-bg/src/states/jotai/atoms/settings.ts:92` · **Status:** Open

Description. When biometric unlock is enabled, the user's password is persisted to the OS secure store via the plain `setSecureItem(...)` call, which does *not* set `requireAuthentication`. A biometric-bound variant (`setSecureItemWithBiometrics`, `requireAuthentication: true`) exists but is not used by `savePassword`. The FaceID / TouchID check runs independently in JavaScript and only decides whether the code then reads the keychain item. Because the keychain entry carries no biometric access-control flag, any code path able to call `getSecureItem('password')` — a malicious OTA bundle, injected JS, a rooted / jailbroken device, or keychain / backup extraction — retrieves it with no biometric prompt.

```
// biologyAuthUtils.ts:49  – setSecureItem, NOT setSecureItemWithBiometrics
await appStorage.secureStorage.setSecureItem(SECURE_STORAGE_PASSWORD_KEY, text,
options);

// index.native.ts – :14 has no requireAuthentication; :46 is the unused biometric
variant
export const setSecureItem = (key, data) => setItemAsync(key, data, keychainOptions);
setSecureItemWithBiometrics(key, data) {
  return setItemAsync(key, data, { ...keychainOptions, requireAuthentication:
true });
}
```

Impact. The stored value is wrapped with `sensitiveEncodeKey`, but that key is a UUID held in `settingsPersistAtom` and persisted to MMKV / AsyncStorage in **plaintext** — so the wrapping key sits next to the ciphertext, defeating the at-rest protection. An attacker with the conditions above can recover the wallet password without ever satisfying a biometric prompt. Exploitation requires local code execution or device compromise; it is not remotely reachable.

Recommendation. Persist the biometric-gated password via `setSecureItemWithBiometrics` (`requireAuthentication: true`) so OS-level biometrics / passcode are required to *read* the keychain item, rather than relying on a JS-side boolean. Do not treat `sensitiveEncodeKey` as a confidentiality control while it lives in plaintext storage — move it into the OS keystore or derive it from a biometric-protected key.

MEDIUM F-02 · Android uses only 5,000 PBKDF2 iterations for at-rest secret encryption

Domain: Cryptography / Data-at-Rest · **Location:** `packages/shared/src/appCrypto/consts.ts:4–7`; `packages/core/src/secret/encryptors/aes256.ts:91–95` · **Status:** Open

Description. `getSecretEncryptV2LocalTargetIterations()` returns `PBKDF2_ANDROID_LOCAL_NUM_OF_ITERATIONS` (= 5,000) on native Android, versus `PBKDF2_CURRENT_NUM_OF_ITERATIONS` (= 600,000) on every other platform — a 120× reduction. This count is used both when encrypting new V2 payloads (revealable seed, imported private keys, and the password verify-string) *and* as the lazy-upgrade threshold, so Android payloads are never upgraded past 5,000.

```
// consts.ts
export const PBKDF2_LEGACY_NUM_OF_ITERATIONS  = 5000;
export const PBKDF2_CURRENT_NUM_OF_ITERATIONS = 600_000;
export const PBKDF2_ANDROID_LOCAL_NUM_OF_ITERATIONS =
PBKDF2_LEGACY_NUM_OF_ITERATIONS;

// aes256.ts:91
return platformEnv.isNativeAndroid
  ? PBKDF2_ANDROID_LOCAL_NUM_OF_ITERATIONS   // 5000
  : PBKDF2_CURRENT_NUM_OF_ITERATIONS;        // 600000
```

Impact. OWASP's current PBKDF2-HMAC-SHA256 guidance is 600,000+ iterations. Combined with the common 6-digit numeric passcode (~10^6 keyspace), an attacker who obtains the encrypted local database can brute-force the passcode offline cheaply, recovering the seed. This is a deliberate Android performance trade-off in the codebase and applies to the largest install base. The weakened path is also not exercised by the crypto unit tests (which run non-native). *Mitigations present:* Android `allowBackup="false"` blocks cloud-backup extraction, and data lives in app-private storage — exploitation requires DB extraction via malware with storage access, physical access, or a rooted device.

Recommendation. Raise the Android iteration count to a security-meaningful floor (ideally matching 600,000) using the already-wired `react-native-fast-pbkdf2` native module, benchmarking for acceptable UX; and/or wrap the local key with an Android Keystore (StrongBox / TEE) key so PBKDF2 is not the sole barrier.

MEDIUM F-03 · No screenshot / screen-recording protection for secret screens

Domain: Mobile Platform Hardening · **Location:** `apps/mobile/android/.../MainActivity.java` (no FLAG_SECURE); `apps/mobile/ios/.../AppDelegate.swift` (no privacy snapshot) · **Status:** Open

Description. A repository-wide search for `FLAG_SECURE` / `preventScreenCapture` / privacy-blur found no implementation. The Android window never sets `WindowManager.LayoutParams.FLAG_SECURE`, so seed-phrase and private-key reveal screens can be screenshotted, screen-recorded, and captured in the recents thumbnail. On iOS, `AppDelegate` adds no obscuring overlay on backgrounding, so the app-switcher snapshot can capture a visible secret.

Recommendation. Set `FLAG_SECURE` on the Android window (at minimum on secret-revealing screens), and add an iOS resign-active privacy overlay / blur to keep secrets out of the app-switcher snapshot.

MEDIUM F-04 · Push-notification payloads written to OS logs in production (native)

Domain: Logging / Privacy · **Location:** `packages/kit-bg/src/services/ServiceNotification/PushProvider/PushProviderJPush.ts:170–173` (also 82–93, 180–183); `packages/shared/src/logger/config/loggerConfigManager.ts:75–79`; `packages/shared/src/logger/base/logFn.ts:91–100` · **Status:** Open

Description. The JPush provider logs whole remote / local notification objects via a `@LogToConsole()` method. `shouldLog` returns `true` unconditionally in production (`if (!isDev) return true`), and `logFn` writes to `console` when `platformEnv.isNative`.

```
defaultLogger.notification.jpush.consoleLog('JPush received remote push:',
notification);
// loggerConfigManager.ts:76 — console logging stays ON in prod
if (!this._env.isDev) { return true; }
```

Impact. On native production builds the full notification payload (title / content / extras) is written to logcat / os_log, readable by other processes with log access or over USB. Wallet notifications can carry transaction amounts, asset names, and counterparty hints.

Recommendation. Gate notification console logging to dev only, or redact body / extras fields before logging in production.

6.4 Low Issues

LOW F-05 · Copied secrets are not auto-cleared from the clipboard

Domain: Secure Storage (Mobile) · **Location:** `packages/components/src/hooks/useClipboard.tsx:55–92` · **Status:** Open

Description. `copyText` writes to the clipboard with no expiry / auto-clear; clearing only happens when the user pastes back into the app. A copied seed phrase, mnemonic word, or private key therefore persists in the system clipboard indefinitely, readable by other apps and clipboard-history features.

Recommendation. For sensitive copies, set a short auto-clear timeout and use Android 13's `EXTRA_IS_SENSITIVE` flag (and the equivalent iOS handling).

LOW F-06 · Recipient name resolution is server-trusted, not verified on-chain

Domain: Network / Transaction Integrity · **Location:** `packages/kit-bg/src/services/ServiceNameResolver.ts:24–34` · **Status:** Open

Description. ENS / .eth (and other) name resolution is delegated entirely to the backend (`GET /wallet/v1/account/resolve-name`); the client does not verify the returned address on-chain. A compromised or malicious endpoint could map a user-entered name to an attacker address for outgoing transfers. The name being resolved (the intended recipient) is also disclosed to the backend.

Recommendation. Prominently display the resolved address for explicit user confirmation before sending (the send UI appears to do this); add client-side on-chain ENS verification for EVM names.

LOW F-07 · iOS push Service Extension disables App Transport Security

Domain: iOS Platform Hardening · **Location:** `apps/mobile/ios/ServiceExtension/Info.plist:11–15` · **Status:** Open

Description. The notification Service Extension sets `NSAllowsArbitraryLoads = true`, disabling ATS for that process (commonly to fetch rich-notification media), which permits cleartext / untrusted-TLS fetches there. The main app target is correctly hardened (`NSAllowsArbitraryLoads = false`, localhost exception only).

Recommendation. Replace the blanket ATS disable with scoped `NSExceptionDomains` for the specific media CDN host(s), or require HTTPS for attachment downloads.

LOW F-08 · Strongest signature-risk styling gated behind a default-off dev setting

Domain: dApp / UX-Security · **Location:** `packages/kit/src/views/SignatureConfirm/components/SignatureConfirmAlert/MessageConfirmAlert.tsx:47–82`; `packages/kit-bg/src/states/jotai/atoms/devSettings.ts:132` · **Status:** Open

Description. `showTakeRiskAlert` short-circuits to `false` when `strictSignatureAlert` (default `false`) is off, downgrading permit / order / typed-data alert colouring from “danger” to default / warning. This is UI-only and does **not** weaken the actual gate — the confirm button still requires the user to tick “proceed at my own risk”, and the `eth_sign` critical alert always renders. Only the most severe red styling is suppressed by default.

Recommendation. Promote strict (red) styling to on-by-default for end users, or decouple severity styling from a developer-only setting.

LOW F-09 · expo-clipboard FileProvider exported without a permission (Android)

Domain: Android Platform Hardening · **Location:** merged release manifest — provider `expo.modules.clipboard.ClipboardFileProvider`, `exported="true"` · **Status:** Open

Description. This content provider merges in as `exported="true"` with no `android:permission` and no `grantUriPermissions`. It is a default of the expo-clipboard library, and all first-party FileProviders are correctly `exported="false"`. Exposure is limited (clipboard image temp files), but an exported provider is the most sensitive Android surface.

Recommendation. If clipboard image sharing is unused, override to `exported="false"` via `tools:replace` (the project already does this for its own providers).

6.5 Informational

INFORMATIONAL F-10 · Non-error console.log statements on secret-handling paths

Location: `packages/core/src/chains/nostr/CoreChainSoftware.ts:52`; `ServiceCloudBackupV2.ts:194, 198, 202`; `ServicePassword/index.ts:725`. Several password / credential paths emit `console.log`. Verified: none print the actual mnemonic, seed, private key, or password — they log operation labels, credential *type*, or booleans. No secret leakage, but they add noise on sensitive flows and run contrary to the repo's own “never log on sensitive paths” rule. **Recommendation:** gate behind `isDev` or remove.

INFORMATIONAL F-11 · Legacy AES-256-CBC and XOR encoders remain reachable

Location: `packages/core/src/secret/encryptors/aes256.ts:471–502, 583–593`; `xor.ts`. Default encryption is V2 AES-256-GCM (authenticated, random salt + nonce) — correct. But decrypt still auto-detects and accepts legacy unauthenticated AES-256-CBC, and a homemade XOR encoder remains selectable via `SENSITIVE_ENCODE_TYPE` (currently hardcoded to `'aes'`, so XOR is inactive). Retained for backward-compat / lazy upgrade; not a vulnerability today, but the unauthenticated-CBC acceptance is a downgrade surface worth tracking. **Recommendation:** keep the lazy-upgrade path that rewrites CBC payloads to V2 on unlock; consider removing the XOR encoder entirely.

INFORMATIONAL F-12 · Persistent device instance-id and first-party telemetry

Location: `packages/shared/src/request/Interceptor.ts:112–134`; `packages/shared/src/analytics/index.ts:140–159, 178–191`. Requests to the backend domains carry a persistent device instance-id header (a stable UUID) plus platform / device headers; analytics posts to `/utility/v1/track` with `distinct_id = instanceId` and (web / ext only) `currentUrl = location.href`. No wallet addresses, balances, or keys are sent. Standard first-party telemetry, but a fingerprinting / linkability consideration; `currentUrl` on web could capture sensitive query params on dApp-browser pages. **Recommendation:** confirm telemetry is consent-gated; omit `currentUrl` or strip query strings.

INFORMATIONAL F-13 · Sample mnemonics committed in the Developer Gallery (production-excluded)

Location: `packages/kit/src/views/Developer/pages/Gallery/Components/stories/DotMap.tsx:41, 51, 61`. Hardcoded example mnemonics used for component previews. These are demo phrases, not user secrets, and the entire Developer / Gallery tree is stubbed out in production via a Metro resolver swap to `router.empty.ts` when `UNION_BUILD=true`. No security impact; noted for completeness. **Recommendation:** none required; optionally replace with obviously-fake placeholder strings.

INFORMATIONAL F-14 · Native WebView uses originWhitelist=['*'] (mitigated by per-navigation validation)

Location: `packages/kit/src/components/WebView/NativeWebView.tsx:334`; `InpageProviderWebView.native.tsx:231`. The native WebView sets `originWhitelist={['*']}`. In isolation this permits any scheme, but every top-frame navigation is validated by `onShouldStartLoadWithRequest → validateWebviewSrc → uriUtils.parseDappRedirect`, which denies `javascript:` URLs, blocks punycode hosts, rejects localhost / private-IP HTTP, and routes deep links separately. Flagged for awareness since the wildcard relies on the navigation guard remaining correct. **Recommendation:** keep as-is; ensure `onShouldStartLoadWithRequest` is wired for any new bridge-enabled WebView surface.

INFORMATIONAL F-15 · Firebase client config files committed

Location: `apps/mobile/android/app/google-services.json`; `apps/mobile/ios/.../GoogleService-Info.plist`. These contain Firebase project identifiers and API keys. Per Google's model these are client-side identifiers (not secrets); committing them is common but discloses project configuration. No true secrets (release keystores, service-account JSON) are committed. **Recommendation:** acceptable; restrict the Firebase API keys by app bundle ID / SHA in the Google Cloud console.

INFORMATIONAL F-16 · Desktop file-protocol interceptor lacks explicit path-traversal normalization

Location: `apps/desktop/app/app.ts:644–682`. The `interceptFileProtocol` js-sdk branch builds a path via `path.join(staticPath, 'js-sdk', fileName)` without rejecting `../`. The URL is the app's own private protocol serving bundled assets to the trusted renderer (contextIsolation + sandbox on, nodeIntegration off), so it is not attacker-controlled in practice. Defense-in-depth only. **Recommendation:** add an explicit check that the resolved path stays within the build directory.

7. Automated & Tooling Analysis

The following checks were performed across the repository (excluding `node_modules`) in addition to manual review.

7.1 Secret & Credential Scanning

Check	Result	Notes
Hardcoded API keys / Bearer tokens / AWS keys	**Clean**	None in tracked source.
Committed private keys / seeds / mnemonics (prod)	**Clean**	Only Gallery demo phrases (F-13), prod-excluded.
Root `.env`	**Clean**	Tracked-as-empty; real `.env` gitignored.
Build-time secrets (Sentry DSN, Covalent, JPush)	**Clean**	Injected from CI `secrets.*` at build.
High-entropy strings in minified bundles	**Triaged**	Matches are crypto test vectors — false positives.
Firebase client config	**Notice**	Client identifiers, not secrets — F-15.

7.2 Cryptography Checks

Check	Result	Notes
Entropy source (mnemonic / IV / salt / nonce)	**CSPRNG**	No `Math.random` in key paths.
Cipher mode (default)	**AES-256-GCM**	Authenticated; random salt + nonce per payload.
KDF (non-Android)	**PBKDF2 600k**	32-byte salt; password SHA-256 pre-hash.
KDF (Android)	**PBKDF2 5k**	120× weaker — F-02.
Signature canonicalization	**Low-S**	via `elliptic` / noble.
Legacy / downgrade encoder surface	**Present**	Unauthenticated CBC still accepted on decrypt — F-11.

7.3 Supply-Chain & Patch Review

Check	Result	Notes
Dependency patches (56)	**Clean**	All legitimate bugfix / hardening; no exfiltration / eval / subprocess.
Security-relevant patches (crypto / RNG / TLS / signing)	**Clean**	None weaken security; one disables a vulnerable native addon.
`resolutions` / pinned versions	**Clean**	elliptic 6.6.1, noble, axios 1.15.2, viem 2.37.6, etc.
Git-sourced dependencies	**Clean**	Pinned to specific commit SHAs.
Install hooks (pre / post-install)	**Clean**	Standard patch-package / env flow.

7.4 Platform Configuration Checks

Surface	Result	Notes
Android release manifest	**Hardened**	`allowBackup=false`, no cleartext, no debuggable; FileProvider F-09.
iOS `Info.plist` / entitlements	**Hardened**	Main-app ATS on; no `get-task-allow`; Service-Ext ATS F-07.
Electron `webPreferences`	**Hardened**	contextIsolation + sandbox on, nodeIntegration off, webSecurity on.
Electron Fuses	**Locked**	runAsNode off, ASAR integrity + onlyLoadAppFromAsar on, cookie encryption on.
Extension CSP / permissions	**Strict**	`script-src 'self' 'wasm-unsafe-eval'`; no `<all_urls>`; no externally_connectable.

7.5 Test Coverage

The repository ships crypto unit tests (`secret-aes256.test.ts` and others) covering the V2 GCM envelope and the 600,000-iteration KDF path. Note that these tests run in a non-native (Jest) environment, so the Android-weakened 5,000-iteration path (F-02) is **not** covered by automated tests.

8. Conclusion

The **Vortex Mobile Wallet** is a competently engineered, non-custodial multi-chain wallet built on a mature codebase. The cryptographic core exhibits consistent defensive engineering: cryptographically-secure randomness throughout, authenticated AES-256-GCM encryption for secrets at rest, a strong 600,000-iteration PBKDF2 KDF on non-Android platforms, audited cryptographic libraries, robust dApp transaction-approval and blind-signing controls, EIP-712 typed-data validation, a hardened Electron configuration with locked Fuses, and a strict browser-extension CSP.

The audit identified **zero critical and zero high-severity findings**. The most material items (F-01, F-02) are medium-severity *at-rest protection* weaknesses on mobile that require local device or encrypted-database compromise to exploit; neither is remotely reachable. The remaining medium findings concern data-leakage hygiene (F-03, F-04), and the low and informational findings are defensive-hardening observations rather than exploitable defects.

The auditor positively notes that (a) the project is open-source and independently reviewable, (b) no secrets, weak randomness, disabled TLS, or broken signing paths were found, and (c) the dApp / transaction-approval surface — historically the highest-risk area for wallets — is well controlled.

On the basis of the static review, secret scanning, cryptographic review, supply-chain review and platform-configuration review performed during this engagement, the auditor's overall assessment is that the application is **Secured** for the use case it documents, subject to remediation of the medium-severity at-rest hardening items below.

Well Secured	
Secured	✓
Poor Secured	
Insecure	

Recommendations — Summary

1. Bind the biometric-gated password to OS keychain access-control (`requireAuthentication: true`) and stop relying on a plaintext-stored wrapping key. (F-01)
2. Raise the Android PBKDF2 iteration count to a security-meaningful floor and/or wrap the local key with an Android Keystore key. (F-02)

3. Add screenshot / app-switcher privacy protection on secret screens, and stop logging push-notification payloads in production. (F-03, F-04)
4. Apply the remaining low / informational hardening items as routine maintenance.

9. Disclaimer

This audit report is provided **“as is”** for informational purposes only and does not constitute legal, financial, or investment advice. The audit was performed on the specific revision of the source code identified in §2; any subsequent modification — including updates, redeployments, or interactions with previously unaudited components — invalidates the conclusions of this report.

This was an **AI-assisted, standard-pass static review of selected critical surfaces**. No dynamic, runtime, instrumented-device, or live-exploitation testing was performed, and the repository's single squashed commit precluded a diff against prior revision history. A static review of this nature may miss issues that require runtime context or deep cross-module data-flow analysis, and it is **not a substitute for a full manual audit by a specialist security firm** before a production release that custodies real user funds. Line numbers reflect the state of the codebase at the time of review and may drift as the code changes. The auditor has used reasonable efforts to identify issues but cannot guarantee the absence of all defects, and assumes no responsibility for losses arising from use of the application.