

BitTool Signer

iOS application version 1.0.0 (build 1) — Source Code Security Audit

RELEASE POSTURE: NO BLOCKING FINDING IDENTIFIED

No Critical, High or Medium severity finding was established. One Low finding concerns where a consent control is enforced rather than whether it exists, and three Informational items are hardening observations. No key-extraction path, no authorization bypass and no review-to-signature divergence was found.

0	0	0	1	3
CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

Assessment identity	Value
Product	BitTool Signer for iOS, version 1.0.0 (build 1)
Bundle identifier	com.bittool.signer
Code baseline	f621372ddeb6740328af0ccfbdd5e28c3f7c9358 (tag v1.0.0)
Shipped engine SHA-256	6933d0d8a6f60faa8222506ac0bd7eef80eb884c9f73e31e9f354b4740f21077
Tracked-file manifest	d506cd4bc7fdd27fd65aa94bdfd2f227b072dc36e12c8f5d788e557f279728bd
Assessment date	19 October 2026
Prepared by	Opus 5
Assessment type	AI-assisted source review, adversarial probing, mutation fuzzing, local test execution

INDEPENDENCE: THIS IS NOT AN INDEPENDENT AUDIT. The author of this report also authored or modified a material portion of the code under review, including the remediation of findings F-01 to F-04 and R2-01 to R2-03 in the same working session. A review cannot confer assurance on its own author's work. This document is a structured self-review and must not be presented as third-party attestation. An independent audit remains outstanding.

01 / DECISION AND MATERIAL RISK

Executive assessment

The security-critical architecture is sound, and its claimed boundaries are real

The property an air-gapped signer exists to provide is that the bytes a user authorizes are the bytes that get signed. In this codebase that property is enforced structurally rather than by convention, at six independent layers inside one function, and each layer was verified in source and by execution.

Equally notable: every refusal boundary the product documents was tested and found to hold. Documentation that overstates capability is the common case in this category; here the opposite was found. `SIGHASH_NONE`, `SIGHASH_SINGLE` and `ANYONECANPAY` are refused at parse time; a SegWit v0 input without its full previous transaction is refused with an error that names the fee attack it prevents; Solana address lookup tables, EIP-7702, blob transactions, raw-hash eth_sign and unsupported chain IDs are all refused.

What matters for wallet safety

- No path was found by which a malicious QR code, file or pasted payload causes a signature over bytes other than those reviewed, or causes signing by a key other than the one displayed.
- No path was found by which recovery material leaves the device or is written anywhere unencrypted. The only deliberate exception is the plaintext phrase QR on the backup screen, which the product documents as an exception.
- Every signature requires that wallet's password at the moment of signing. There is no cached seed, no global unlock and no session that survives backgrounding.
- The one Low finding is an architectural-consistency issue: the consent gate for high-risk-but-fully-decoded requests is enforced in the Swift interface and not in the JavaScript engine. It is not reachable by an attacker in 1.0.0; it matters because the engine is the only layer this product can prove to a user.

Release recommendation

Nothing in this review blocks the release of 1.0.0. BTS-OP5-01 should be closed in 1.0.1 as a one-condition change; it is deliberately not changed in 1.0.0 because the engine artifact hash is already published and bound to the submitted build. The outstanding risks for this product are not in the code reviewed here — they are the absence of an independent audit and the absence of any completed third-party wallet interoperability test.

Evidence quality

This assessment combines source review of 3,546 lines of JavaScript and 11,858 lines of Swift, adversarial probes written against specific attack hypotheses, a 4,000-case deterministic mutation corpus against the untrusted QR boundary, and the project's own 210 engine and 168 native tests. A clean-room rebuild of the engine from the tagged commit was confirmed byte-identical to the artifact inside the archive submitted to App Store Connect.

A passing test suite is evidence about covered cases. It does not establish that all transaction semantics, all devices, all external wallets or all third-party dependencies are secure. No live-chain transaction was executed and no physical-device fault or side-channel testing was performed.

02 / SCOPE AND METHOD

What was assessed

Scope element	Coverage
JavaScript engine	15 modules, 3,546 lines: four chains, the resolver, the public index, key derivation, NFT interpretation and the QR/CBOR transport. Reviewed line by line.
Native application	50 Swift files, 11,858 lines: vault, device key store, engine bridge, authorization, lifecycle, review rendering, setup and scanner.
Tests	19 engine test files, 25 native unit files, 3 UI files — executed, and read as specifications of intended behaviour.
Build and packaging	Bundle manifest, clean-room rebuild, the submitted archive's contents, Info.plist, privacy manifest, linked frameworks.
Dependencies	Pinned runtime dependency tree, lockfile integrity, advisory status, and which packages reach the shipped bundle.

Method

Four complementary techniques, in this order: (1) line-by-line reading of the security-critical paths, starting from key derivation and the signing authorization chain rather than from the entry points; (2) hypothesis-driven adversarial probes, where a specific attack was named, encoded as a request, and executed against the real engine; (3) deterministic mutation fuzzing of the untrusted transport boundary; (4) execution of the existing suites and a clean-room rebuild.

Probes were written to fail loudly if an attack succeeded, and each is reproducible. Where a probe was refused, the refusal code and message were recorded. The probe set is listed in Appendix A.

Trust boundaries

Asset / actor	Security expectation
Recovery material	The mnemonic, BIP39 passphrase, seed, password and derived keys remain protected outside short authorized operations.
Malicious coordinator	Controls QR, file and pasted request bytes and all unsigned metadata. Must not be able to choose a signing key, misstate an economic effect, or obtain a signature over unreviewed bytes.
Signing user	May authorize a request after reading its principal summary. Correct signed bytes and an accurate review are both necessary.
Local storage adversary	May read, damage or replace container files under a separate device-compromise assumption. The vault and public index must reject tampering.
Trusted foundation	iOS, app integrity, the Secure Enclave and platform cryptographic APIs are trusted. Compromised OS or hardware is outside demonstrated assurance.

Exclusions

Physical-device fault and side-channel testing; OS and Secure Enclave internals; App Store and TestFlight binary provenance beyond the archive inspected locally; live balances, permissions and deployed bytecode; external coordinator wallet implementations; professional linguistic review of the 21 interface translations; and formal verification. Vended cryptographic dependencies were inventoried and advisory-checked but not independently audited.

03 / THE SIGNING AUTHORIZATION CHAIN

Why this section is first

Every other control in a signer is secondary to this one. If an attacker can obtain a signature over bytes the user did not review, nothing else matters. The chain was therefore reviewed before anything else, and is reported before anything else.

How the property is enforced

In Release builds the only reachable signing entry point is `signPlanTask`. The legacy `sign` operation, which accepts an optional `reviewedHash` and would therefore permit an unbound signature, is confined to a DEBUG-only allowlist in `SigningEngine.swift` and is absent from shipped builds. This was verified by reading both the allowlist and the `#if DEBUG` guard around it.

`signPlanTask` then applies six checks, each of which independently prevents a different substitution attack:

#	Check	What it prevents
1	<code>planHash === requestHash(body)</code>	A tampered plan: the plan is self-authenticating.
2	<code>plan.requestHash === requestHash(request)</code>	Swapping the request after review.
3	<code>digestOf(parsed) === plan.signingBytesDigest</code>	A request that re-parses to different signing bytes.
4	<code>seedWalletKeyId(root) === task.walletKeyId</code>	Signing with a different wallet than the plan names.
5	Full re-resolve from the decrypted seed, then <code>stable(resolved.tasks[0]) === stable(task)</code>	Any divergence between the plan and what the real seed actually resolves to.
6	<code>Per-target samePublicKey(target.publicKey, derived)</code>	Signing with a key other than the approved one.

Check 5 deserves emphasis. Rather than trusting the plan it was handed, the engine re-derives the entire public index from the decrypted seed, re-runs resolution, and requires a canonical-form equality against the task. An attacker who controls the plan, the request and the host must therefore also control the seed.

Canonicalization

The whole chain rests on `stable()`, so it was examined for injectivity and failure mode. It sorts object keys, rejects non-plain prototypes, rejects numbers that are not safe integers (forcing decimal strings for token amounts), and bounds nesting at depth 32. Critically, it has no branch for `undefined`, `bigint`, `Uint8Array` or `Date`: those reach the plain-object assertion and throw. It fails closed. A canonicalizer that silently dropped `undefined` would make two materially different requests hash identically; this one refuses to hash them at all.

VERIFIED BY EXECUTION: a `setApprovalForAll` request was reviewed, planned and signed, and the recovered address was checked against the derived address. Independently, a request whose plan had been altered was refused with `INTEGRITY`. The six checks are not aspirational.

04 / FINDING REGISTER

0 CRITICAL	0 HIGH	0 MEDIUM	1 LOW	3 INFORMATIONAL
---------------	-----------	-------------	----------	--------------------

ID	Severity	Finding	Status
BTS-OP5-01	Low	High-risk consent is enforced only in the layer that cannot be verified	Open, fix recommended for 1.0.1
BTS-OP5-02	Informational	A localization key, and therefore a Markdown renderer, is constructed from engine output	Open, latent
BTS-OP5-03	Informational	The password KDF is not memory-hard	Accepted risk, strongly mitigated
BTS-OP5-04	Informational	Intermediate Ed25519 key material is not zeroized	Open, inherent limitation

Severity definitions used

Level	Definition for this product
Critical	Established broad key compromise or systemic unauthorized asset loss. None established.
High	Material potential asset consequences, or a core informed-consent failure reachable by an attacker. None established.
Medium	Security-relevant bounded integrity or availability impact without the above exposure. None established.
Low	A control that is present but enforced in a weaker position than the product's own trust model implies, with no attacker-reachable path in this version.
Informational	Hardening, defence-in-depth or disclosure observations without an established exploit path.

No CVSS vector, CVE, external certification or claim of real-funds theft is made. Severity reflects attacker reachability in version 1.0.0 as shipped, not the theoretical worst case of the underlying pattern.

05 / LOW — BTS-OP5-01

High-risk consent is enforced only in the layer that cannot be verified

Code references: engine/src/resolver.js (signPlanTask); BitToolSigner/Features/Sign/RequestReviewView.swift:188; BitToolSigner/Models/Models.swift:374

Observation

The interface and the engine disagree about which requests require explicit consent.

TransactionReview.requiresAcknowledgement is defined as `decodeStatus != .fullyDecoded || riskLevel == .high`, and the Sign button is disabled until the box is ticked. The engine's check is narrower:

```
if (['partiallyDecoded', 'opaque'].includes(parsed.review.decodeStatus))
```

```
    check(riskAcknowledged === true, 'RISK', 'Confirm this request's decoding risks before signing');
```

A request that is `riskLevel: 'high'` but `decodeStatus: 'fullyDecoded'` therefore passes the engine with no acknowledgement at all. That class is not obscure: it is exactly the set of fully-understood dangerous operations — unlimited token approvals, collection-wide `setApprovalForAll`, Safe permission changes, token burns.

Demonstration

`setApprovalForAll(0xABab..., true)` on an unregistered contract — reviewed as `riskLevel: high`, `decodeStatus: fullyDecoded`, carrying a HIGH RISK warning — was planned and then signed with the `riskAcknowledged` parameter omitted entirely:

```
riskLevel: high | decodeStatus: fullyDecoded
```

```
plan status: ready
```

```
>> SIGNED with riskAcknowledged omitted: 375a3fff911fe09f93996b8d...
```

Impact and reachability

In BitTool Signer 1.0.0 this is not reachable by an attacker. A malicious QR code cannot bypass a disabled SwiftUI button, and the interface gate is correct. No user of the shipped app can sign a high-risk request without ticking the box.

It is reported because of what this product specifically claims. The published verification story states that the engine is the component a user can rebuild and hash-compare, and that the Swift layer "can only be reviewed by reading, not by comparing hashes". The consent control for the most dangerous request class lives entirely in the unverifiable layer. A tampered host, a future alternative host, or an ordinary refactor that drops one SwiftUI modifier would remove the control with no change to the artifact a user is told to verify.

Recommendation

Extend the engine's condition to the risk level, so the verifiable layer enforces at least as much as the unverifiable one:

```
const needsConsent = ['partiallyDecoded', 'opaque'].includes(parsed.review.decodeStatus)
```

```
    || parsed.review.riskLevel === 'high';
```

```
if (needsConsent) check(riskAcknowledged === true, 'RISK', '...');
```

This is deliberately not applied in 1.0.0: the engine artifact hash is already published in the `v1.0.0` tag and bound to the archive submitted to App Store Connect, and changing `signer.js` would invalidate the verification material users are directed to. It should ship in 1.0.1 with a regression test asserting that a high-risk fully-decoded request is refused without acknowledgement.

06 / INFORMATIONAL FINDINGS

BTS-OP5-02 — A localization key is constructed from engine output

Code reference: BitToolSigner/Features/Activity/ActivityTab.swift:225

```
label: LocalizedStringKey(display.counterpartyLabel ?? "To")
```

`LocalizedStringKey` is not an inert string type. SwiftUI resolves it against the string table and then renders the result as Markdown. Constructing one from engine output means engine text reaches a Markdown renderer, where syntax such as a link or emphasis would be interpreted rather than displayed.

Every value this field can currently hold was enumerated from the engine source. All are closed-set literals — `To`, `Spender`, `Operator`, `Contract`, `Marketplace`, `Mint`, `Receiver`, `New account`, `To token account` — so no request data reaches the renderer today and the finding is latent, not live. It is reported because the hazard is invisible at the call site: a future label that interpolates a symbol, a contract name or any request-derived text would become a Markdown injection into the transaction history with no other code change.

Recommendation: render it with `Text(verbatim:)` after an explicit lookup, or keep the type as `String` and translate through the same `ReviewText.phrase` path the review page already uses for engine text.

BTS-OP5-03 — The password KDF is not memory-hard

Code reference: BitToolSigner/Core/VaultCrypto.swift

The vault derives its password key with PBKDF2-HMAC-SHA256 at 600,000 iterations and a 32-byte random salt. That iteration count sits exactly at the current OWASP floor for PBKDF2-SHA256, and the implementation is careful: the count is range-checked on derivation, so a tampered header cannot force a weaker KDF, and the password buffer is cleared with `memset_s`.

PBKDF2 is nonetheless cheap to parallelize on GPUs and ASICs relative to a memory-hard function. For a wallet whose encrypted file could be exfiltrated, `scrypt` or `Argon2id` would raise the cost of an offline password search by orders of magnitude.

This is rated Informational rather than Low because of the architecture around it, which substantially removes the exposure. The password layer is wrapped in a second AES-GCM layer under a 32-byte device key held in the Keychain as `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly` with `kSecAttrSynchronizable = false`. An attacker holding only the vault file cannot begin a password search, because they must first defeat a hardware-bound key that never leaves the device, is excluded from backups and becomes inaccessible if the device passcode is removed. The GPU-friendliness of PBKDF2 is therefore not the binding constraint.

Recommendation: adopt `Argon2id` in a future vault format version, as defence against the case where the device key is also compromised. Not urgent.

BTS-OP5-04 — Intermediate Ed25519 key material is not zeroized

Code reference: engine/src/keys.js (solanaKey)

The SLIP-0010 derivation loop zeroizes its working variable `I` on every iteration and in a `finally` block, which reads as deliberate and complete hygiene. It is not complete: `I.slice(0, 32)` and `I.slice(32)` each allocate a fresh copy, and the buffer built by `concat(...)` contains the parent private key. None of those three copies is cleared, so parent key material remains on the JavaScript heap until garbage collection.

The practical exposure is small. Each sensitive operation runs in a `JavaScriptCore` context that is discarded immediately afterwards, and the project already discloses that "Swift strings and JS garbage collection also prevent a complete zeroization guarantee". The finding is reported because the visible partial zeroization invites a reader to conclude more than is true.

Recommendation: either zeroize the intermediate buffers explicitly, or replace the `finally` block with a comment stating plainly that zeroization here is best-effort and that context disposal is the actual control.

Observation — TRON signatures carry no network binding

A TRON `raw_data` protobuf contains no chain identifier, so a signature produced for mainnet is replayable on any TRON-compatible chain that accepts the same `raw_data`. This is a property of the TRON protocol, not a defect in this code, and the application discloses it: the engine refuses to assert network identity offline and the compatibility documentation states the limitation. No change is recommended beyond continuing to disclose it.

07 / CONTROLS ASSESSED AS EFFECTIVE

A review that reports only findings misrepresents a codebase. The following controls were examined specifically and found to be correctly implemented.

Key management and cryptography

Control		Assessment
Wallet entropy		Generated natively via <code>SecRandomCopyBytes</code> , never in JavaScript. Failure throws rather than falling back. 24-word default is 256-bit.
Engine randomness		Bridged from <code>SecRandomCopyBytes</code> . The crypto object is frozen and non-configurable, the native entry point is deleted after capture, installation happens before the bundle is evaluated, and unavailability throws. <code>Math.random</code> is also backed by native entropy, so no bundled library can introduce a weak source.
BIP39 derivation		Native NFKD plus PBKDF2-HMAC-SHA512. Verified against @scure/bip39 and the official bip32JP Japanese vectors including a compatibility-character passphrase.
SLIP-0010 (Ed25519)	Hand-written derivation checked against the specification: correct ed25519 seed HMAC key, correct 0x00	key ser32(index) child data, big-endian index with the hardened bit, and all-hardened path enforcement.
BIP340 auxiliary randomness		Taproot signing passes fresh <code>randomBytes(32)</code> . ECDSA paths use deterministic RFC 6979 nonces, so the all-zero auxiliary value passed to <code>signIdx</code> is inert rather than a nonce-reuse hazard.
Vault		PBKDF2-HMAC-SHA256 at 600,000 rounds, then two AES-GCM layers with per-layer domain-separated AAD that authenticates the stored header bytes. Cleartext and password buffers cleared with <code>memset_s</code> .
Device key		32 random bytes, <code>WhenPasscodeSetThisDeviceOnly</code> , non-synchronizing. Removing the device passcode renders wallets unopenable — the intended and verified behaviour.
Failed-attempt throttling		Persisted before the expensive derivation, so terminating the process cannot reset the counter. Exponential delay from the fifth attempt to a 300-second cap.

Public metadata integrity

`validatePublicIndex` does not merely parse: it re-derives. Each stored xpub is re-parsed and its public key, chain code, parent fingerprint, depth and child index are cross-checked against the extended key, and the presence of a private key is rejected. The structure is pinned exactly — six Bitcoin nodes matching the profile-to-path mapping, three Solana keys at distinct non-small-order points, one hundred TRON addresses at exact indices. Derivation is restricted to registered non-hardened 0/i and 1/i branches, so no caller can request an arbitrary hardened path. The file itself is authenticated with HMAC-SHA256 under a Keychain key, and tampering forces an explicit repair rather than silently degrading.

Transaction review integrity

For EIP-712 — historically the most dangerous surface in this category — the digest and the displayed values are computed from the same normalized objects. `displayTyped` only converts `BigInt` to string; it performs no other transformation. Display cannot diverge from what is hashed, which is a structural guarantee rather than a matching pair of code paths. Type enforcement is strict: a `bool` must be a real boolean, an address exactly 20 hex bytes, integers unambiguous, duplicate field names refused, and the declared `EIP712Domain` field order governs the separator.

Bitcoin signing holds an unsigned-transaction invariant across the whole operation, refuses any sighash other than `ALL` or `DEFAULT`, requires the signing key to satisfy the committed input script, restricts paths to mainnet, refuses duplicate signatures, and verifies each signature after producing it.

Defensive parsing

The hand-written canonical CBOR decoder bounds depth at 24, nodes at 20,000, arrays at 20,000, and rejects duplicate map keys, non-canonical integer encodings and indefinite-length items. Transport limits are explicit: 131,072-byte payloads, 2,048 frames, 4,300 characters per frame, 1,048,576 retained characters. The `dispatch` wrapper converts any uncontrolled exception into a generic refusal that leaks no internal detail.

08 / EXECUTED VALIDATION

Validation	Result
Engine suite	210 / 210 passed. No failures, skips or cancellations.
Native unit tests	168 tests in 23 suites passed.
Adversarial probes (Appendix A)	11 hypotheses. 10 refused as designed; 1 confirmed BTS-OP5-01.
Mutation corpus, QR boundary	4,000 deterministic mutations. 4,000 controlled refusals, 0 accepted, !0 uncontrolled exceptions.
Clean-room engine rebuild	<code>npm ci --ignore-scripts</code> plus build in a throwaway copy of the tagged commit: byte-identical.
Fresh-clone rebuild	Clone of tag <code>v1.0.0</code> , built from scratch: byte-identical.
Submitted archive	The <code>.app</code> in the App Store Connect archive carries the same <code>signer.js</code> hash, <code>ITSApplUsesNonExemptEncryption = false</code> , 21 localizations, privacy manifest present, team <code>F5X3QH95Y6</code> .
Linked frameworks	Neither <code>CFNetwork</code> nor <code>Network.framework</code> is linked. The offline claim is structurally true, not merely a policy.

Remediation verification: prior findings

The two earlier review rounds are independently relevant to this one, because several of their findings were regressions introduced by earlier remediation. All were re-tested at this baseline.

Prior finding	Status	Verification at this baseline
F-01 Safe value on <code>DELEGATECALL</code>	Fixed	Both Safe entry points zero the value; the ignored amount is reported, not displayed as paid.
F-02 Solana burn omission	Fixed	Burns are first-class destructive effects, lead the summary, and force high risk.
F-03 Fee-format overflow	Fixed	String carry; no <code>Int</code> parsing. A fee beyond <code>Int.max</code> renders instead of trapping.
F-04 Burn plan unsignable	Fixed	Absent optional fields omit the key. Verified through the full <code>resolveSigningPlan</code> .
R2-01 Safe execution context	Fixed	<code>forwards</code> propagates descendant native spending through nested delegates; both entry points agree.
R2-02 Change dilutes the fee rule	Fixed	The relative rule is recomputed against the verified payment after change classification.
R2-03 Solana first-burn summary	Fixed	Burns group by mint and total; a single headline amount appears only when one number describes the transaction.

A notable pattern across all three rounds: four of the seven prior findings were introduced by the remediation of an earlier finding. Two of those — F-03 and the `forwards` `BigInt` defect found during R2-01's own remediation — were crashes while rendering the review page, before any authorization. This codebase's dominant historical risk has not been cryptographic error; it has been regression during remediation of economic-display logic. The regression suites now pinning each of those cases are the most valuable artifact the project has.

09 / SUPPLY CHAIN AND BUILD INTEGRITY

The shipped engine matches a clean rebuild

The 1,181,026-byte `signer.js` reproduces exactly from the tagged commit after `npm ci` with lifecycle scripts disabled. Its SHA-256 matches the declared bundle manifest and the resource inside the submitted archive, confirmed in three independent locations.

```
6933d0d8a6f60faa8222506ac0bd7eef80eb884c9f73e31e9f354b4740f21077
```

Check	Observed result
Production advisories	<code>npm audit --omit=dev</code> : 0 vulnerabilities at audit time.
Complete dependency tree	6 advisory entries: 3 moderate, 3 low, 0 high or critical. These are development and reference packages, not six wallet exploits.
Runtime reachability	The affected packages and their vulnerable paths are absent from the package roots contributing to the bundle. No runtime exploit path was established.
Dependency pinning	Runtime dependencies are exact-pinned and the lockfile records integrity hashes.
Release packaging	Version 1.0.0, build 1, <code>com.bittool.signer</code> . No unexpected test or debug files packaged.

The limit of reproducibility, stated plainly

An App Store executable cannot be reproduced byte for byte: Apple re-signs it, FairPlay-encrypts the main executable, and the Swift toolchain makes no determinism guarantee. The project states this rather than obscuring it, and directs verification at the one component that can be checked — the unencrypted `signer.js` resource, extractable from an `.ipa` and comparable against a source build. That is an honest and technically correct division, and it happens to cover the code that decides what a user is signing. It does not cover the Swift layer, which is the reason BTS-OP5-01 is reported at all.

10 / ACTION PLAN

Priority	Action	Acceptance evidence
Next release	Enforce high-risk consent in the engine (BTS-OP5-01)	A regression test asserting a high-risk, fully-decoded request is refused without acknowledgement; republished artifact hash.
Next release	Render counterpartyLabel verbatim (BTS-OP5-02)	No LocalizedStringKey constructed from engine output anywhere in the tree.
Hardening	Clarify or complete zeroization in solanaKey (BTS-OP5-04)	Either explicit clearing of intermediates, or a comment stating the control is context disposal.
Future format	Evaluate Argon2id for the vault (BTS-OP5-03)	A vault format version with migration, benchmarked on the oldest supported device.
Before real funds	Independent third-party security audit	An engagement report from a firm with no authorship relationship to the code.
Before claiming compatibility	Complete one third-party wallet round trip per claimed chain	Wallet, version, platform, chain, pairing payload, request, response, independent signature verification, real camera loop.

Residual limitations requiring separate work

- Physical-device memory, capture, background and backup behaviour; Keychain and access-control behaviour under real device states; jailbreak, malicious OS and fault or side-channel resistance.
- Controlled contract execution for Safe accounting, current on-chain permissions, live resource fees and real coordinator round trips.
- Independent review of the cryptographic dependencies, and provenance of the production build and distribution artifacts.
- Semantic review of the 21 interface translations beyond structural validation of placeholders and completeness.

Release decision

No finding in this report blocks 1.0.0. The reviewed code implements its security-critical properties structurally, and its documented refusal boundaries were tested and hold. The gap between this position and one that justifies holding significant funds is not closed by code quality: it is closed by an independent audit and by real interoperability evidence, neither of which exists yet. The project's own documentation states this, which is itself the most reassuring finding in this review.

This assessment is a point-in-time structured self-review by the author of much of the code under review. It is not an independent audit, transfers no custody responsibility, and does not guarantee the absence of undiscovered defects.

APPENDIX A / ADVERSARIAL PROBES

Each probe encodes a named attack hypothesis and was executed against the engine at this baseline. All are reproducible from the tagged commit.

Hypothesis	Result	Observed
EIP-712 domain chainId differs from transport chainId, enabling cross-chain replay	Refused	CHAIN_ID EIP-712 chain ID does not match the signing request
PSBT input requests SIGHASH_NONE, so the signature omits outputs	Refused	SIGHASH Only SIGHASH_ALL or Taproot SIGHASH_DEFAULT accepted
PSBT input requests SIGHASH_SINGLE	Refused	SIGHASH as above
PSBT input requests SIGHASH_ALL with ANYONECANPAY	Refused	SIGHASH as above
PSBT input requests SIGHASH_NONE with ANYONECANPAY	Refused	SIGHASH as above
witnessUtxo amount contradicts nonWitnessUtxo, misstating the fee	Refused	Rejected at PSBT construction; inconsistent prevouts are unrepresentable
SegWit v0 input supplies only witnessUtxo, so the amount is unverifiable	Refused	PREVOUT Full previous transactions required to prevent repeated-signing fee attacks
Solana v0 message carries an address lookup table	Refused	LOOKUP_TABLE requires independently trusted account data
EVM raw-hash eth_sign	Refused	UNSUPPORTED Unsupported EVM request kind
EVM unsupported chain ID 999	Refused	NETWORK Unsupported network
EVM type 4 transaction (EIP-7702)	Refused	UNSUPPORTED Only legacy, 2930 and 1559 supported
High-risk, fully-decoded request signed with no acknowledgement	Confirmed	Signature produced — BTS-OP5-01

Mutation corpus

Ten protocol seeds (UR crypto-psbt, eth-sign-request, bytes, BBQR, base64 PSBT, protocol JSON, raw hex, a bech32 address, crypto-account) subjected to 4,000 deterministic single- and multi-character mutations, including null and non-BMP insertions, and submitted through dispatch.

```
mutations: 4000 | controlled refusals: 4000 | accepted: 0 | uncontrolled throws: 0
```

Every mutation produced a controlled, coded refusal. No mutation was accepted, and none produced an uncaught exception or an error message disclosing internal state.

APPENDIX B / PRIMARY REFERENCES

- OWASP Mobile Application Security Verification Standard — control-family framework; no conformance claim is made. <https://mas.owasp.org/MASVS/>
- SLIP-0010, Universal private key derivation from master private key — checked against the Ed25519 derivation. <https://github.com/satoshilabs/slips/blob/master/slip-0010.md>
- BIP340 Schnorr signatures — auxiliary randomness guidance. <https://bips.dev/340/>
- BIP174 / BIP370 Partially Signed Bitcoin Transactions. <https://bips.dev/174/>
- EIP-712 Typed structured data hashing and signing. <https://eips.ethereum.org/EIPS/eip-712>

- Safe v1.4.1 Executor.sol — CALL passes value; DELEGATECALL retains the executing account context. <https://github.com/safe-global/safe-smart-account>
- OWASP Password Storage Cheat Sheet — PBKDF2-HMAC-SHA256 iteration guidance. <https://cheatsheetseries.owasp.org/>
- Apple Platform Security — Keychain data protection classes. <https://support.apple.com/guide/security/>
- RFC 8949 §4.2.1 Deterministic CBOR encoding. <https://www.rfc-editor.org/rfc/rfc8949.html>

Prepared by Opus 5 as an AI-assisted code security assessment of BitTool Signer for iOS, version 1.0.0 (build 1), at commit f621372. Conclusions are restricted to the stated baseline, the executed checks and the static evidence described. No audit-firm affiliation, independent human attestation or formal security certification is represented, and the independence limitation on the cover page applies to every conclusion in this document.