

TECHNICAL WHITEPAPER

ByzanPass: the security architecture

A technical guide to the current browser extension, device firmware, and encrypted vault format.

ByzanPass separates browsing a password directory from authorizing the release of a selected entry. This paper describes the implemented personal vault, with Chrome WebUSB as the primary browser path and explicitly identified differences for the optional Windows native host. It is an implementation description, not an independent audit or a security certification.

Source baseline: 157fcd715d253134fdce1f2019d77260f89d7e4c

Document edition: 1.0 | 3 October 2026

READING MAP

- 01 Scope and the central security claim
- 02 Components and trust boundaries
- 03 Master credentials and key derivation
- 04 Vault root, key slots, and record encryption
- 05 Authenticated storage and atomic changes
- 06 Authentication is not entry approval
- 07 Pairing and firmware consistency
- 08 Sessions, viewing windows, and reveal policy
- 09 What remains on the computer
- 10 Extension isolation, XSS, and request authorization
- 11 Field detection, matching, and fill binding
- 12 Creation, editing, and imports
- 13 TOTP and passkeys
- 14 Recovery and physical-device limits
- 15 What the design does and does not establish
- 16 Implementation map and reference standards

01. Scope and the central security claim

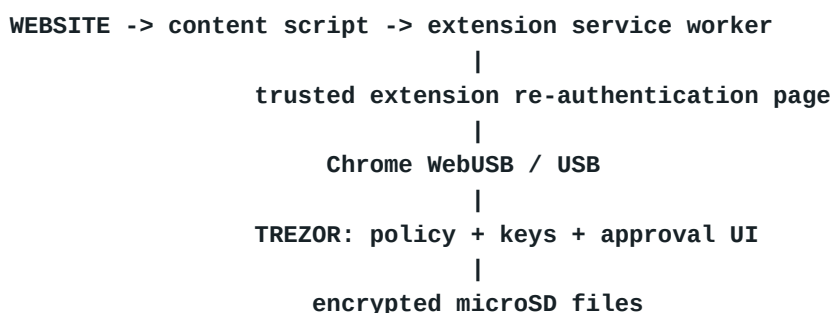
A conventional software password manager must make secrets or decryption authority available to its software client. ByzanPass keeps a required device secret and the vault-wide encryption root inside the Trezor firmware. An authenticated computer can browse directory metadata, but a private record is opened only after the firmware authorizes that selected entry.

The release unit is an entry, not necessarily one password string. An entry can contain a password, notes, card details, or other fields. Approval releases that selected payload and any permitted generated OTP codes. A malicious host can retain everything released to it. There is no bulk plaintext-read command, but an attacker can accumulate entries that the user individually approves.

The central claim assumes trusted firmware, an uncompromised device and cryptographic implementation, protected recovery material, and a user who checks the device's displayed request. It does not survive malicious firmware controlling the same keys, successful device-secret extraction, or theft of the recovery words together with the encrypted backup.

This edition describes source baseline 157fcd715d25, inspected on 3 October 2026. The latest locally recorded Safe 5 installation at inspection was development image 137f33e2. Build options and future revisions can change these properties; emulator checks and installation verification are not substitutes for an independent security review.

02. Components and trust boundaries



Where information is handled

Component	Role and exposure
Website and content script	Identify fields and request a selected fill. The destination website can read any value filled into its DOM.
Extension service worker	Own browser session state, route capabilities, validate senders, derive a PC-entered master contribution, and temporarily hold released records.
Trusted extension pages	Pairing, setup, imports, editing, and master-password entry. They are separate from a website's DOM.
Trezor firmware	Verify credentials, enforce selected-entry approvals and reveal policy, derive keys, authenticate storage, and generate OTPs or passkey signatures.
microSD card and backup	Encrypted vault objects and public format parameters. Files, sizes, object counts, and update patterns can still leak structural information.
Optional native host	An alternative identity-allowlisted stdio bridge. Its optional Windows persistence differs from the primary WebUSB package.

Names, domains, account labels, usernames, email addresses, IDs, and protection flags are directory metadata. They are encrypted on the card, but become visible to the authenticated host without a second entry approval. The app calls these fields public to distinguish them from the separately approved record. They are not safe places to hide secrets.

Chrome's current package connects directly through WebUSB and requires Chrome 137 or newer. It does not depend on a local HTTP service. Separate Chrome and Firefox native-messaging packages use a Windows helper; Firefox is not described here as a WebUSB implementation. The public website itself never connects to the vault.

03. Master credentials and key derivation

New vaults use `bcrypt_pbkdf`, the PBKDF construction, rather than the `bcrypt` password-hash format. Both Safe 5 and Model T currently target 72 rounds. The five-byte authenticated profile contains BP, a version byte, and the round count in two-byte big-endian form. Opening rejects counts outside 1-256. New production credentials must match the device's target and satisfy a minimum of 48; debug emulators alone use four rounds for tests.

P = master password

V = 16-byte vault ID

S = 16-byte credential salt

C = `bcrypt_pbkdf(P, "TVBP1" || V || S, 72, 32)`

C is a 32-byte contribution, not the vault encryption root.

The device-compatible password representation is 1-50 printable ASCII characters when opening an existing credential; a new user-chosen password must contain at least eight. These are compatibility limits, not a statement that eight characters is a strong choice. The UI also offers generated passwords. The KDF has a much smaller memory footprint than a modern memory-hard Argon2 configuration, a deliberate consequence of supporting on-device entry.

An older credential profile remains supported through host-side Argon2id with 64 MiB memory, three iterations, parallelism four, a 16-byte salt, and a 32-byte output. This compatibility path must not be confused with the current new-vault default.

If P is typed on the computer, the extension sees it and computes C. C is reusable password-equivalent material for that vault credential. A hostile extension or sufficiently compromised computer can capture either. Entering P on the Trezor keeps that derivation off the host. Buffer overwrites and dropped references reduce retention, but JavaScript strings and copies made by the browser or OS cannot be guaranteed erased.

04. Vault root, key slots, and record encryption

The vault root R is a fresh random 32-byte key. Normal authentication opens an envelope containing R using both a 32-byte device-held credential secret D and the 32-byte contribution C. The root is not sent to the browser. Purpose labels, vault identity, and immutable slot identities separate key uses.

```
H(K, M) = HMAC-SHA256(key=K, message=M)
Derive(X, salt, purpose) =
  H(H(salt, X), "trezor-vault-v4/" || purpose || 0x01)

KM = Derive(D || C, V, "master/" || S || slotID)
headerM = "TVM4" || S || slotID
masterSlot = headerM || Seal(KM, R, AAD=headerM || V)
```

Derive is a single-output-block HKDF-SHA256 extract/expand construction: the inner HMAC extracts with the salt, and the outer HMAC expands with a domain label and counter byte. Seal uses ChaCha20-Poly1305 with a random 12-byte nonce and a 16-byte authentication tag. Its serialized result is nonce || ciphertext || tag. The complete master slot is 96 bytes. Authentication must succeed before plaintext is returned.

Every immutable version of an entry gets its own random 32-byte data-encryption key DEK. A root-derived wrapping key protects the DEK; the DEK protects the entry body. The associated data binds both operations to the vault, item, and item-version identities, so moving ciphertext to a different identity fails authentication.

```
B = "trezor-vault-v4/item/" || V || itemID || versionID
KW = H(R, "trezor-vault-v4/wrap/" || B || 0x01)
record = "TVI4"
  || Seal(KW, DEK, AAD="K" || B)
  || Seal(DEK, body, AAD="I" || B)
```

The generic cryptographic envelope accepts up to 16 KiB, but the current record codec enforces a 4 KiB body, including its OTP envelope. The envelope adds 92 bytes. A compromised root can unwrap all records: independent record keys enable scoped operations, but hardware policy, rather than cryptography alone, enforces one-entry approval after the root is open.

Randomness comes through the firmware's model-specific cryptographic RNG interfaces. Model T uses its MCU hardware RNG; Safe 5 can mix additional entropy sources. Random nonces are not a guarantee against a broken RNG, and neither random keys nor domain separation compensate for a compromised firmware implementation.

05. Authenticated storage and atomic changes

The current vault-v4 directory is an encrypted, authenticated snapshot structure, with immutable item versions, encrypted directory pages and manifests, object hashes, and revision information. Its tree uses 4 KiB pages and a 10,000-entry bound. Parent references authenticate child ciphertext hashes; the manifest commits to the directory root. The card is treated as untrusted input, with bounds and authenticated identity checks before its directory authorizes a record.

An update writes new encrypted objects before publishing the on-device commit. Alternating commit records occupy public storage slots authenticated by HMAC under protected device state; they contain the snapshot identity, sequence, revision, high-water entry number, and ciphertext hash. Interrupted work can leave unreferenced files, but should not silently promote a partially written snapshot. Record release is checked against the authenticated current entry and version. Corrupt current state does not justify falling back to an older password slot or a convenient older record.

The device's retained, authenticated commit state gives rollback resistance against an older or replaced card during normal use. It does not make a removable backup globally fresh. On explicit recovery, a complete, coherent older backup can authenticate successfully and restore older data. There is no independent trusted counter or external ledger proving that it is the newest copy.

Password rotation publishes a new credential envelope with a new credential secret. When the initial setup secret still doubles as an envelope secret, it also rotates the private-state authentication key before retiring that envelope secret. The vault root and recovery envelope remain stable, so records need not all be re-encrypted. This revokes old credentials through current device state; it does not erase old backups or revoke a root or old credential secret that an attacker already extracted.

The present card page format is TVPG0002. This document is not a compatibility promise for every earlier development format; older TVPG0001 backups require their matching older firmware rather than silent conversion.

06. Authentication is not entry approval

- The host first checks public status and paired-device consistency, then authenticates a master contribution or requests explicit attachment to an already locally opened vault.
- The firmware establishes a host session and allows directory operations. Authenticating through the extension does not enable local password browsing on the device.
- For a selected read, firmware validates the current entry, immutable version, stored metadata, and requested operation before presenting its identity on the touchscreen.
- The physical approval is bound to a digest of the operation, vault/item/version context, and exact ciphertext. That one-use approval is consumed before the data key is unwrapped.
- Configured master and PIN checks are applied, the ciphertext tag is verified, and only then is the selected entry returned. Denial, cancellation, a stale selection, or authentication failure prevents release.

```
approvalDigest = SHA256(  
    "trezor-vault-v4/approval/" || operation || B  
    || SHA256(recordCiphertext)  
)
```

The main USB interface uses Trezor's framed message transport over 64-byte interrupt endpoints. The extension claims the vault interface, not the independent FIDO interface. Public status recovery may drain a bounded set of eight recognized out-of-order public replies; private reads and mutations are not blindly replayed as a connection-recovery shortcut.

USB access permission is only permission to talk to the device. A session token is authorization state, not a vault decryption key or a general encrypted tunnel protecting plaintext from the host. The returned entry crosses into the computer's trust domain. The host may be able to solicit many prompts; the user must evaluate the identity actually displayed by the device.

07. Pairing and firmware consistency

Pairing pins a device identity key and an approved reported build. Subsequent checks use a fresh 32-byte nonce and an Ed25519-signed statement. Verification binds the purpose, key generation, reported build, and current vault/setup/credential/authentication-challenge fields. Invalid signatures, weak keys, malformed fields, and inconsistent public status are rejected.

A firmware change must be explicitly accepted as the exact old-build/new-build transition for the paired key. The current extension records the new approval only after the device successfully authenticates the password in that session. It does not retain a separate local master-password verifier to approve firmware changes.

WebUSB pairing records use an IndexedDB envelope protected by a non-exportable AES-GCM key. Non-exportable means the API will not export that key; it does not stop malicious code with the extension's own privileges from invoking the key or changing application behavior.

These checks establish consistency with what this PC previously paired and accepted. They are not independent firmware attestation: code controlling the identity key can sign its own claims. A hash confirms bytes only when checked against a trustworthy reference; it is not proof of safe behavior. Firmware distribution, review, and the installation decision remain part of the trusted system.

The paired-code-derived two-color gradient is a visual recognition aid. The pairing code is not a secret, and a hostile page can copy a gradient or imitate a dialog. Master-password inputs therefore belong in a top-level extension page or on the Trezor, never in an ordinary website overlay.

08. Sessions, viewing windows, and reveal policy

Current defaults and independent timers

Control	Current behavior
Password session	12-hour fixed expiry by default; ordinary activity does not extend it.
Sensitive view	Two minutes from release by default, bounded by session expiry. Closing, refresh, navigation, or lock clears the retained view as applicable.
Host liveness	A heartbeat every 10 seconds renews a 45-second firmware host lease; loss of a browser window is not guaranteed to be detected instantly.
Reveal PIN reuse	A fixed 120-second grant for the same record ID/version and exact host session; never skips its physical approval.
Global reveal allowance	No limit by default (stored as 0), or 1-99 successful Chrome reveals within a rolling hour before a fresh PIN is required.

The sensitive-view timer governs data the extension retains. It cannot erase a clipboard copy, a value already filled into a website, a screenshot, or a copy stolen by malware. Copying, ordinary activity, and refreshing OTP codes in an existing approved panel do not restart that timer. Monotonic and wall-clock checks account for sleep and prevent backward clock changes from extending an existing view.

Entries can require a PIN, master re-authentication, or both. WebUSB offers computer entry first, with a choice to type on the Trezor instead. A new protected computer read obtains the entry approval, checks the master contribution, then requests the PIN if required. A wrong master prevents the later PIN/release. The optional native-host protected-entry flow currently supports device entry only.

PIN reuse is deliberately narrower than an unlocked vault: it is tied to the same record number, ID, version, and host session and ends when another entry is read, on relevant policy changes, session end, or expiry. It is disabled for master-protected entries and local device reveals. An entry requiring both master and PIN therefore repeats both checks for every new read. Local on-device entry reveals always require a fresh PIN.

The global allowance uses device monotonic time and successful reveal events held in RAM. A successful fresh reveal PIN resets the allowance; the next successful reveal consumes its first slot, and events age out after an hour. A valid same-entry PIN reuse does not consume another allowance slot. Restart resets this RAM history but returns the device to its boot PIN protection.

A lost host lease clears host authority. A separately opened local vault can remain available locally; a PC-only session does not turn into local access. An active operation has a bounded allowance of up to 12 minutes tied to that exact session, so a legitimate device prompt is not confused with an idle host. A malicious host holding a valid token can send heartbeats: Connected is a liveness indicator, not proof that a benign Chrome window is visible.

09. What remains on the computer

The current Chrome WebUSB package does not persist an unlock contribution across browser restarts. A new browser session must authenticate again or obtain device approval to use an already locally opened vault. Browser directory/account metadata is memory-only; old persistent directory keys are proactively deleted rather than loaded. Locked state responses contain no directory entries.

Settings and pairing records persist. Site icons are fetched by the extension, not the Trezor; current icon storage is discarded on lock, fresh worker startup, and when internet fetching is disabled. Requests omit credentials and referrers, reject redirects, and bypass normal HTTP caching, but the contacted server can still learn the requested domain and client IP.

The optional Windows native host has two separate persistence features. Its disposable directory cache contains only public metadata, protected by AES-GCM plus current-user DPAPI and a device-provided cache key. Exact snapshot/root/revision/count matching after authentication is required; the cache never authorizes a private read.

Its optional Remember on this PC feature stores a DPAPI-protected password-derived contribution with expiry and credential binding. It is not the literal password, but is password-equivalent material. Malware running as the signed-in Windows user may be able to unwrap it. Software expiry and DPAPI are not a new hardware approval boundary. This persistence feature is absent from the primary WebUSB package.

10. Extension isolation, XSS, and request authorization

There is no cookie-authenticated web vault endpoint to which a website can submit a conventional CSRF request. The relevant boundaries are extension origins, resource accessibility, runtime-message sender checks, operation-specific capabilities, browser-verified frame/document identity, and device approval. CORS is not the vault's authorization mechanism.

The manifest restricts scripts to bundled extension resources. The WebUSB build additionally permits WebAssembly evaluation for the bundled compatibility KDF; it does not load arbitrary remote JavaScript. CSP disables plugin objects and external base URLs and constrains form actions. Inline styles remain permitted. Privileged vault and re-authentication pages further restrict connections to the extension origin.

Dynamic values are assigned as text or escaped before template insertion. Imported data are displayed through `textContent`. These are concrete XSS defenses, not a claim that every rendering sink is bug-free. A future unsafe sink, compromised dependency, or malicious extension update could defeat the browser side.

The service worker checks explicit action allowlists for each privileged page and content-script context, together with sender identity, URL, tab, frame, and document as appropriate to each operation. The master re-authentication page is a top-level extension tab whose grant is bound to the original request and requesting document. Navigation, closing, stale records, lock, and timeout cancel that grant.

The only web-accessible extension HTML page is the approved-account display frame, `approved.html`. It has no website `postMessage` `secret-query` API. Its worker data request must match the issued grant, expected page and hash, tab, frame, selected record/version, and owning document. Website code cannot simply ask that frame for an arbitrary password.

The extension necessarily has broad HTTP/HTTPS host permissions and all-frame content scripts for form detection and autofill, plus storage, active-tab, clipboard-writing, alarms, scripting, navigation, and Chrome

favicon capabilities. These are powerful permissions. The hardware release boundary limits bulk extraction under trusted firmware; it does not make a malicious extension update harmless.

11. Field detection, matching, and fill binding

Detection is local heuristic code. It combines HTML autocomplete tokens with input type, names, IDs, labels, nearby text, and form structure to classify login names/email, current and new passwords, OTP fields, identity/contact/address fields, and payment-card fields. This is an independent implementation, not imported Proton Pass detection code or a claim of identical accuracy on every site.

New entries default to exact host and subdomains. A match permits the saved host and its dot-delimited descendants, not arbitrary strings containing that name. Exact host is available for a narrower match; local addresses need special handling. Matching proposes candidates; it never replaces device authorization.

Before approval, the extension snapshots the selected document, form, and field. After approval and while filling, it checks document identity and URL, browser-reported frame ancestry, element identity/signature, and form membership. If a page navigates or replaces the selected form or fields, the fill is refused rather than redirected into the new target.

Embedded-frame routing uses browser APIs and isolated content-script observations, not a site's claimed iframe URL. Card numbers and CVC are restricted to the selected card origin; only permitted cardholder/expiry metadata can cross to an allowed top origin. Credential delivery requires HTTPS except explicitly supported local loopback hosts.

Simulated typing emits ordinary DOM input events, not trusted physical keyboard events. Once a value is in a site field, that site can read it. Binding prevents a class of target-switch attacks while the user approves; it cannot stop an already hostile intended destination. Multi-step sign-in is not an automatic cross-document approval grant: the Google example enters the email first, then selects and approves the login on the password step.

Save suggestions are bounded in-memory drafts and require explicit save/device approval. Ordinary login and address drafts last five minutes; cards last three. Generated-password drafts have a 30-minute allowance and limited same-site redirect handling. Card capture accepts number, expiry, and holder, with 12-19 digit Luhn validation; it never captures CVC or PIN. Recent real interaction is required for card/address submission capture, and the extension does not submit those forms itself.

12. Creation, editing, and imports

Creating or editing an entry on the computer exposes that plaintext to the computer before encryption. Device-side storage encryption protects the stored copy, not the input path. Existing entry protections are displayed with the item and edited in its edit view; changing reveal protections requires device approval and a fresh device PIN. If the entry already requires master re-authentication, changing its protections also requires the current master check.

Importers support KeePass CSV/XML, LastPass CSV, Bitwarden CSV/JSON, 1Password CSV/1PUX, browser CSV, and the project's JSON format. Direct KDBX import is not currently in the format allowlist. Export files and parsed values necessarily exist on the importing computer.

Import bounds

Boundary	Limit
Ordinary input file	650 KiB
Expanded archive content	4 MiB
Parsed source rows	5,000
One accepted device batch	256 entries
Preview lifetime	180 seconds, also bound to the current directory snapshot
Current vault record count	10,000
Encoded record body	4 KiB

Parsing uses strict UTF-8, checked CSV structure, bounded XML depth/node counts, bounded ZIP decompression and CRC validation, and schema checks. Archive paths are not extracted to the filesystem. Unknown authority fields, invalid schemas, and out-of-bounds records are rejected instead of silently truncated.

A device-approved import authorizes a bounded batch of new records. The device allocates IDs and fresh record keys, stages encrypted objects, and commits the accepted batch. It does not grant permission to read existing secrets or overwrite arbitrary entries. A snapshot change invalidates the preview. Interruption can leave unreferenced encrypted objects; before successful publication the previous committed snapshot remains authoritative.

13. TOTP and passkeys

TOTP seeds pass through the computer during enrollment, then are retained in the device-side encrypted record representation. The normal read path returns generated codes rather than the seed. Supported algorithms are HMAC-SHA1, HMAC-SHA256, and HMAC-SHA512 with six or eight decimal digits, periods of 15-120 seconds, and seeds of 10-64 bytes. Defaults are SHA1, six digits, and 30 seconds. The implementation uses the time counter, dynamic truncation, and decimal reduction specified by RFC 6238.

```
counter = uint64_be(floor(unixTime / period))
code = dynamicTruncate(HMAC(seed, counter)) mod 10^digits
```

The PC supplies time at authentication; the device then measures elapsed time and refuses a code request more than 90 seconds from that approved baseline. This constrains requests relative to the baseline, not independently verified real time. Displaying UTC does not make it authoritative. The local interface also permits explicitly setting time. Wall time is not the security clock for the rolling PIN allowance or host lease. A compromised host receiving an approved OTP can use it during its validity interval.

Passkeys use the separate FIDO HID interface and a vault-owned CTAP2.0 authenticator derived from Trezor's implementation. Credential signing supports P-256/ES256 and Ed25519/EdDSA. A separate random 32-byte passkey root is held in internal storage, wrapped under a key derived from R and the vault ID with the passkey-root-wrap purpose. A live PIN-unlocked vault session is needed to unwrap it. Credential keys are derived from that passkey root and per-credential material, bound to the relying-party ID. The host gets

authenticator responses and signatures, not private signing keys. The implementation also supports hmac-secret.

Passkey authorization and user-verification rules belong to the authenticator, not the password-view timer. User verification uses the on-device PIN with a 180-second freshness window; not every assertion requires another PIN. The authenticator uses packed self-attestation, a zero signature counter, and clear backup-eligibility and backup-state flags. It does not advertise clientPin. Passkeys and security-key credentials are device-only and are not recovered by a password-vault backup. Register an independent recovery method with important services.

A raw local FIDO client has more visibility than an ordinary web page. While unlocked, supported silent preflight requests can reveal whether a guessed relying-party ID has a stored credential and expose its credential ID and user handle. The silent response has no valid signature, user-presence or user-verification assertion, names, account count, or hmac-secret result. This is an explicit metadata limitation, not secret-free behavior.

14. Recovery and physical-device limits

The recovery path deliberately bypasses the original device and master password. The device generates 16 random bytes, encodes them as 12 BIP39 words, and uses the BIP39 seed function with an empty passphrase to obtain recovery key material. The words are entered through local device UI in the supported recovery workflow and are not returned in a browser message.

```
Q = BIP39-seed(normalized12words, "")
KR = Derive(Q, V, "recovery/" || recoveryID)
headerR = "TVR4" || recoveryID
recoverySlot = headerR || Seal(KR, R, AAD=headerR || V)
```

The recovery slot is 80 bytes. A complete encrypted vault-v4 backup plus the correct words is enough to reconstruct R and decrypt the vault independently of the original Trezor. The BIP39 checksum catches only some transcription mistakes; successful envelope authentication establishes that the words match the backup. Words alone do not contain the records.

Safe 5 and Model T enforce the same application-level entry approval policy, but their physical security differs. Safe 5 has secure-chip-assisted PIN/storage protection; Model T does not. The separate Safe 5 secure-element master-password attempt limiter is disabled in the latest recorded development candidate. Its enabled diagnostic probe is not enforcement.

The implemented firmware master-password delay allows two failures, delays 30 seconds after the third, then doubles to a 15-minute maximum. Authenticated persistent attempt state survives lock and power loss; after reboot, the full outstanding penalty starts again. A successful password check resets the count. This is separate from the device PIN limiter and is not an off-device guessing defense once an attacker has extracted the necessary secret and ciphertext.

Extraction of the device's credential secret together with the encrypted card enables offline master-password guesses. Recovery-word theft together with a complete backup offers a direct recovery route. Device tampering, firmware exploitation, weak passwords, and recovery-material handling therefore remain relevant even when the browser is denied bulk reads.

15. What the design does and does not establish

Threat outcomes under the stated assumptions

Attacker capability	Consequence
Copied encrypted card only	Cannot use the normal master path without the required device secret and contribution; structure and file metadata remain observable.
Captured master password plus card	Still lacks the device secret for normal decryption. A connected trusted device can nevertheless be asked to prompt the user.
Compromised extension or PC	Can capture typed/imported/generated secrets, directory metadata after authentication, and every released record; can mislead the user or alter fills.
Malicious firmware or successful key extraction	May bypass application approval policy. Pairing signatures do not prove firmware honesty.
Backup plus recovery words	Can recover the password vault without the original device or master password.
Unattended, locally opened device	Local new reveals still require a fresh PIN; existing displayed information and separate authenticator policies need their own consideration.

There is no claim of immunity to malware, XSS, supply-chain compromise, physical extraction, phishing, side channels, or denial of service. The design concentrates vault-wide decryption authority on the device and narrows each approved computer release. It cannot retract a secret once the computer has learned it.

Source review, unit and integration tests, browser tests, emulator checks, and installation fingerprints provide different kinds of evidence. A passing test suite only covers its cases; an emulator is not physical fault-injection testing; a matching installed fingerprint is not an audit. This whitepaper makes no independent certification claim and does not describe an enterprise service.

16. Implementation map and reference standards

The source paths below identify the implementation behind this edition. They are a reading map for the stated baseline, not links to a publicly distributed source bundle. Cryptographic formulas in this paper follow the current code rather than older design sketches.

Source reading map

Area	Primary files
Master credentials and envelopes	firmware/overlay/core/src/apps/misc/vault_kdf.py, vault_credentials.py, vault_keyslots.py, vault_core.py
Items, commits, recovery	vault_item_crypto.py, vault_record.py, vault_directory_v4.py, vault_storage_v4.py, vault_v4.py, vault_recovery.py, vault_change_master_password.py
Reveal policy and identity	vault_reveal_policy.py, vault_attempts.py, vault_identity.py; extension/webusb_trust.js
Browser sessions and messages	extension/background.js, webusb_backend.js, webusb_transport.js, sensitive_view.js, reveal_protection.js
Field and frame handling	extension/content.js, autofill_fields.js, autofill_frames.js, content_helpers.js, save_drafts.js
Import and rendering	extension/webusb_import.js, import.js, shared.js, vault_view.js
OTP and authenticator	vault_otp.py and vault_passkey_*.py under the firmware misc application
Optional native persistence	native_host/remembered_key.py, directory_cache.py, vault_backend.py

OpenBSD bcrypt_pbkdf(3)

The PBKDF construction used for current device-compatible master credentials.

https://man.openbsd.org/bcrypt_pbkdf.3

RFC 5869: HKDF

HMAC-based extract-and-expand key derivation; the envelope construction uses one SHA-256 output block.

<https://www.rfc-editor.org/rfc/rfc5869>

RFC 8439: ChaCha20 and Poly1305

The authenticated-encryption primitive used by vault envelopes.

<https://www.rfc-editor.org/rfc/rfc8439>

RFC 6238: TOTP

Time-based one-time-password counter and truncation rules.

<https://www.rfc-editor.org/rfc/rfc6238>

Chrome content scripts

Browser isolation concepts relevant to detection and filling; isolation does not hide a filled DOM value from its page.

<https://developer.chrome.com/docs/extensions/develop/concepts/content-scripts>