SECURITY DESIGN DOCUMENT

V4 Encryption
Hardware-Bound by Design

A complete technical account of how VaultSort encrypts your files — the threat model, the cryptographic design, and the limitations we are honest about.

```
Hardware authenticator → PRF (HMAC-SHA-256) → HKDF-SHA-256 →
AES-256-KWP wrap → AES-256-GCM file encryption
```


VERSION	PUBLISHED	APPLIES TO	AUDIENCE
1.0	June 2026	VaultSort V4 format	Technical readers

VaultSort V4 Encryption — Security Design Document

Version: 1.0
Date: June 2026
Applies to: VaultSort with V4 encryption format

Introduction

This document describes the cryptographic design of VaultSort's V4 encryption format — how files are encrypted, how keys are derived, what security properties are guaranteed, and what the known limitations are. It is intended for technically literate users who want to verify VaultSort's claims before trusting it with sensitive files.

VaultSort is a file encryption application for macOS. Its V4 format is the result of a structured review of the previous V3 format, which identified a fundamental architectural weakness: hardware key touches were enforced at the UI layer but contributed no cryptographic entropy. V4 corrects this by making hardware participation mandatory at the cryptographic level.

---

1. Threat Model

1.1 What V4 protects against

Attacker has the encrypted file but not your hardware key.
A V4 file cannot be decrypted without either physical possession of a registered hardware authenticator (YubiKey or a device with Touch ID), or knowledge of the recovery code. The file encryption key is wrapped by a key that only the hardware can produce. There is no software path to derive it.

Attacker has both the encrypted file and VaultSort's credential store.
The credential store (`~/Library/Application Support/VaultSort/.../config.json`) holds registration metadata: credential IDs, public keys, and PRF salts. None of these values allow an attacker to unwrap the file key. The wrap key is derived from a hardware-internal HMAC secret that never leaves the device. Stealing the credential store without the hardware yields nothing.

File is tampered with in transit or at rest.
All V4 files use AES-256-GCM with the full JSON metadata as authenticated additional data (AAD). Any modification to the file body, the key slots, or the metadata will be detected and decryption will fail. There is no silent corruption or "decrypt with wrong key returns garbage" failure mode — the operation aborts with an authentication error.

Key slot tampering (wrapping one credential's blob over another's).
Each key slot's wrap step uses AES-256-KWP (RFC 5649), which has an authenticated integrity check value. Copying a wrapped-key blob from one slot to another fails at unwrap. The file-level GCM auth tag provides a second layer of protection.

1.2 What V4 does not protect against

Attacker has physical possession of your unlocked device.
If someone has your device, your PIN, and your fingerprint (or can compel Touch ID), they have your hardware key. VaultSort is not a defence against this scenario.

Attacker compromises your iCloud account (Touch ID users only).
Touch ID passkeys on macOS sync to iCloud Keychain by default. An attacker who gains full access to your Apple ID and iCloud account may be able to recover your Touch ID passkey from iCloud and use it from another Apple device. YubiKey users are not affected by this risk — YubiKey credentials are device-bound by hardware and cannot sync. See §4 for a full comparison.

Malicious software with file-system access running on your device.
VaultSort encrypts files at rest. It does not protect against a compromised OS, kernel-level malware, or any process that can intercept VaultSort's own IPC calls during active decryption.

Recovery code stored insecurely.
If your recovery code is stored in an unprotected location (e.g. a plaintext note, an unencrypted cloud document), an attacker who finds it can decrypt your files without any hardware. Treat the recovery code like a wallet seed phrase.

---

2. Why V4 — Addressing V3 Weaknesses

The V3 format had three structural weaknesses identified during a formal internal review in early 2026. None required immediate emergency action — V3 files were not trivially crackable — but all three represented gaps between the security VaultSort claimed to provide and what it actually provided.

2.1 YubiKey touch was a UX gate, not a cryptographic input (V3-2)

This was the primary motivation for V4.

In V3, the wrap key for the file encryption key was derived as:

```
wrapKey = scrypt(
  credential.id + ":" + credential.userHandle + ":" + credential.publicKey,
  fileSalt
)
```

All three inputs — `id`, `userHandle`, `publicKey` — are stored in plaintext in VaultSort's credential store on disk. The YubiKey touch during decryption authenticated the *user* at the application level, but the

cryptographic key used to unwrap the file was derived entirely from data that existed on disk before the key was ever touched.

The practical consequence: an attacker who obtained both the encrypted file and a copy of the credential store could decrypt the file offline, without the YubiKey, by replaying the scrypt derivation on the stored fields. The YubiKey was not a cryptographic requirement; it was a gate that could be bypassed by anyone who had already compromised the credential store.

2.2 Non-authenticated key wrap (V3-7)

V3 used AES-256-CBC with a deterministic IV to wrap the file encryption key. CBC mode provides confidentiality but not integrity — a corrupted or tampered wrapped-key blob would produce a garbage decryption key rather than a detectable error. The GCM auth tag on the file body would eventually catch the corruption, but only after the wrong key had already been used.

V4 replaces this with AES-256-KWP (RFC 5649), an authenticated key-wrapping algorithm. A tampered wrapped-key slot fails immediately at unwrap with an integrity check error, before any decryption is attempted.

2.3 Hardcoded KDF parameters (V3-8)

V3 used fixed scrypt parameters (`N=16384, r=8, p=1`) with no record of those parameters in the file. Parameters could not be upgraded without breaking all existing files. V4 moves to per-credential, self-describing parameter records — every wrapped key slot records exactly how it was derived, making future parameter upgrades clean and backward-compatible.

---

3. V4 Cryptographic Design

3.1 The core insight: hardware-bound key derivation via WebAuthn PRF

V4's fundamental change is that the wrap key for each file can only be produced with the hardware authenticator physically present and authorising the operation.

This is achieved through the **WebAuthn PRF extension** (HMAC-secret), a standardised mechanism in the FIDO2/WebAuthn specification. When a credential is registered, a 32-byte random PRF salt is stored alongside the credential ID. At encryption or decryption time, the authenticator is asked to compute:

```
prfOutput = HMAC-SHA-256(authenticatorInternalSecret, prfSalt)
```

The `authenticatorInternalSecret` is a symmetric key that lives exclusively inside the hardware authenticator. On a YubiKey, it lives in secure storage on the chip. On macOS with Touch ID, it lives in iCloud Keychain, protected by the Secure Enclave and biometric authentication. In both cases, the secret **never leaves the device** — the authenticator returns only the HMAC output.

This single output is what V3 was missing. An attacker who steals the credential store learns the PRF salt, but that alone is not enough — they would also need the authenticator's internal secret, which requires the physical device.

3.2 Wrap-key derivation

The 32-byte PRF output is not used directly as a wrap key. It is first expanded through HKDF-SHA-256:

```
wrapKey = HKDF-SHA-256(
  ikm  = prfOutput,          // 32 bytes from hardware
  salt = fileSalt,           // 32 bytes, random per file
  info = "vaultsort-v4-wrap:" + credentialId,
  L    = 32,
)
```

The per-file salt ensures that even if two files use the same credential, their wrap keys differ. The `credentialId` in the `info` field binds each wrap key to its specific slot, preventing a wrapped-key blob from one slot being substituted into another.

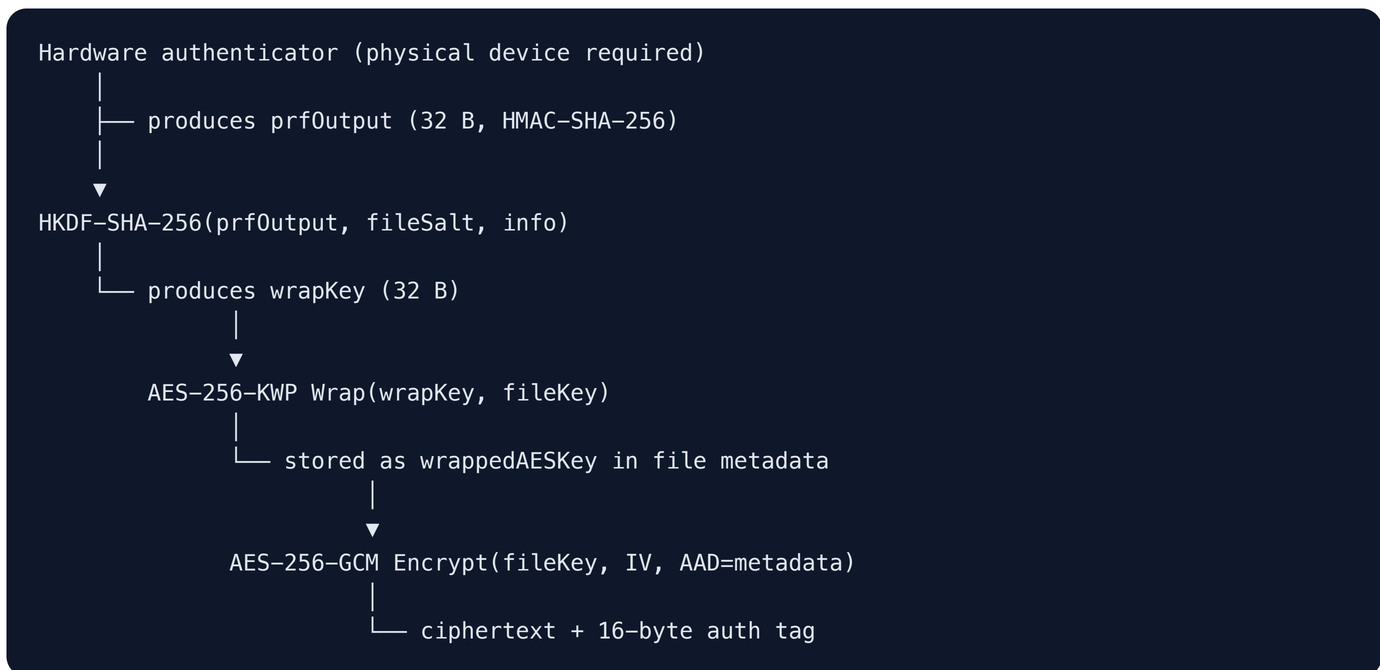
3.3 Key wrapping

The 32-byte file encryption key is wrapped using **AES-256-KWP** (RFC 5649 — also known as AES Key Wrap with Padding). This produces a 40-byte authenticated ciphertext. Any attempt to tamper with this blob is detected at unwrap time with a specific integrity error.

3.4 File encryption

The file body is encrypted with **AES-256-GCM** using a randomly generated 256-bit file key and a randomly generated 96-bit IV, both unique per encryption. The full metadata JSON (including all wrapped key slots) is passed as authenticated additional data (AAD), so any modification to the metadata — adding a slot, removing a slot, changing a parameter — invalidates the GCM authentication tag and decryption fails.

3.5 Complete per-file key derivation chain



```
Hardware authenticator (physical device required)
    │
    ├── produces prfOutput (32 B, HMAC-SHA-256)
    │
    ▼
HKDF-SHA-256(prfOutput, fileSalt, info)
    │
    └── produces wrapKey (32 B)
              │
              ▼
        AES-256-KWP Wrap(wrapKey, fileKey)
              │
              └── stored as wrappedAESKey in file metadata
                        │
                        ▼
              AES-256-GCM Encrypt(fileKey, IV, AAD=metadata)
                        │
                        └── ciphertext + 16-byte auth tag
```

3.6 Multi-key design

A V4 file can contain multiple wrapped key slots — one per authorised credential, plus an optional recovery slot. Each slot wraps the same 32-byte file key independently. Decryption succeeds with any single slot whose credential is available. This means a file can be accessible by Touch ID, a YubiKey, and a recovery code simultaneously, without any of them sharing key material.

Important: files are encrypted for one credential at a time, not all registered credentials simultaneously. This is a deliberate design decision — requiring every registered hardware key to be physically present at encrypt time would make encryption impractical for users with multiple keys. After encrypting with a primary credential, an "Add backup key" action allows the file encryption key to be additionally wrapped for a second credential while both credentials are present.

3.7 Recovery slot

The optional recovery slot stores the file key wrapped by a key derived from a user-supplied passphrase:

```
wrapKey = Argon2id(
  password = recoveryCode,    // 20-char Crockford base32, 100 bits of entropy
  salt     = recoverySlotSalt,  // 16 bytes, per-credential, stored in app
  m        = calibrated,        // ≥ 65536 KiB (OWASP 2026 minimum)
  t        = calibrated,        // ≥ 3 iterations
  p        = 1,
  hashLen  = 32,
)
```

Parameters are calibrated at first setup to target ~1 second of computation on the user's machine, then recorded in the slot metadata so the same parameters are used on recovery regardless of hardware

speed. The recovery code is shown to the user exactly once at setup and is not stored anywhere in VaultSort.

---

4. Touch ID — Security Model and Comparison

Touch ID support uses the **WebAuthn platform authenticator** path — the same PRF-based mechanism as YubiKey, with two important differences in the underlying storage.

4.1 YubiKey (cross-platform credential)

The authenticator's internal HMAC secret lives in protected storage on the physical YubiKey chip. It cannot be exported, cloned, or retrieved. It is not transmitted over any network. Possession of the YubiKey is the only way to produce the PRF output.

Property	Value
Storage	On the YubiKey hardware
Syncable	No
Remote attack surface	None
Lost device scenario	Files encrypted with this key only are inaccessible unless a backup credential or recovery code exists

4.2 Touch ID (platform credential via iCloud Keychain)

The passkey is stored in iCloud Keychain, protected by Apple's end-to-end encryption and gated by biometric authentication on the device. On macOS 14+, the PRF extension is supported and VaultSort uses the same cryptographic path as YubiKey.

Property	Value
Storage	iCloud Keychain (end-to-end encrypted)
Syncable	Yes — available on all Apple devices signed into the same Apple ID
Remote attack surface	iCloud account compromise allows recovery of the passkey from another Apple device
Lost device scenario	Passkey is available on other Apple devices via iCloud sync

The iCloud sync property is both a convenience and a risk. Users who want decryption to work on a new Mac after replacing their primary machine will benefit from the sync. Users whose threat model includes a compromised Apple ID should register at least one YubiKey as a backup and treat Touch ID as a convenience credential, not a primary security perimeter.

VaultSort surfaces this distinction in the credential list and warns users during Touch ID registration. It does not block Touch ID registration for users who accept this trade-off — the decision belongs to the user.

4.3 Minimum OS requirement for Touch ID support

PRF support in the macOS WebAuthn provider requires **macOS 14 Sonoma or later**. On macOS 13 and earlier, Touch ID registration for V4 encryption is blocked with a clear error message. Existing YubiKey credentials and V3 files are unaffected.

---

5. File Format Reference

5.1 Header layout

```
Offset  Length  Field
──────  ──────  ─────────────────────────────────────────────
0       21      Magic bytes: "VAULTSORT-WEBAUTHN-v4" (UTF-8)
21      4       Metadata length (uint32, big-endian)
25      N       Metadata JSON (also used verbatim as GCM AAD)
25+N    32      File salt (random per-file, used as HKDF salt)
57+N    12      GCM IV (random per-file)
69+N    …       Ciphertext (AES-256-GCM)
end-16  16      GCM authentication tag
```

Total fixed overhead: 85 bytes plus metadata JSON.

5.2 Wrapped key slot schemas

Hardware credential slot (WebAuthn PRF):

```
{
  "kind": "webauthn-prf",
  "credentialId": "<base64url WebAuthn credential ID>",
  "wrapAlgorithm": "aes-256-kwp",
  "hkdf": {
    "hash": "sha-256",
    "info": "vaultsort-v4-wrap:<credentialId>"
  },
  "wrappedAESKey": "<base64, 40 bytes>"
}
```

Recovery passphrase slot:

```
{
  "kind": "recovery-passphrase",
  "credentialId": "recovery-<uuid>",
  "wrapAlgorithm": "aes-256-kwp",
  "kdf": "argon2id",
  "m": 65536,
  "t": 3,
  "p": 1,
  "salt": "<base64, 16 bytes>",
  "wrappedAESKey": "<base64, 40 bytes>"
}
```

No other slot kinds are valid in V4.

---

6. Algorithm Selection Rationale

Algorithm	Role	Rationale
AES-256-GCM	File body cipher	AEAD — provides both confidentiality and authentication in one pass. Metadata-as-AAD makes the authenticated region cover the entire file structure. Unchanged from V3.
HKDF-SHA-256	Wrap-key derivation	PRF output is already 256-bit uniform entropy. HKDF is the standard "expand uniform input into keying material" primitive. Clean, audited, no parameter tuning required.
AES-256-KWP (RFC 5649)	Key wrapping	NIST-standard authenticated key wrap primitive, purpose-built for wrapping symmetric keys. Replaces V3's unauthenticated AES-CBC.
Argon2id	Recovery KDF	OWASP-current memory-hard function. Resists both GPU and ASIC brute force. Parameters are recorded per-slot and calibrated at setup time.
WebAuthn PRF	Hardware key derivation	Standard FIDO2 extension, implemented by both YubiKey 5 series (HMAC-secret over CTAP2) and macOS 14+ platform authenticators. No proprietary protocols.
Crockford base32	Recovery code encoding	Human-friendly (excludes ambiguous characters 0/O, 1/I/L), case-insensitive, 20 characters at 5 bits each = 100 bits of entropy.

---

7. Backward Compatibility

V4 is a new file format, not a replacement that invalidates existing files. VaultSort reads V1, V2, V3, and V4 files. V4 builds do not write V3 (or earlier) format files for new encryptions, but all existing files remain fully accessible.

Existing YubiKey credentials registered before V4 do not automatically gain V4 support. A credential registered under V3 did not go through the PRF probe at registration time, so no PRF salt was stored for it. Such credentials are labelled in VaultSort's credential list and can still decrypt V3 files. To use a credential with V4 files, the authenticator must be re-registered through the V4 registration flow, which probes for PRF support and records the PRF salt.

V3 files stay in V3 format and remain fully readable — there is no automatic upgrade on decrypt. To migrate a file to V4, decrypt it and re-encrypt it with a V4-capable credential. The Key Sync panel in Key Settings re-wraps key slots across a folder (so all registered credentials can decrypt existing files) but does not change the file format.

---

8. What This Document Does Not Cover

- **VaultSort's application security model** — how the Electron app's IPC surface is hardened, sandboxing, entitlements. These are implementation details rather than cryptographic design questions.
- **Network security** — VaultSort encrypts files locally. No file content is transmitted over a network by the app. The WebAuthn ceremony uses a local HTTP server on `localhost` for the browser-based portion of the key ceremony.
- **Key management policy** — how many keys to register, when to rotate a recovery code, and how to store a recovery code are user decisions. VaultSort provides nudges and warnings but does not enforce a policy.
- **Post-quantum cryptography** — V4's wrap chain is symmetric throughout (HMAC, HKDF, AES-KW). Symmetric primitives are well-positioned relative to known quantum attacks (Grover halves effective AES key size, AES-256 remains ~AES-128-equivalent). The WebAuthn signature verification step uses P-256 ECDSA; a post-quantum adversary breaking ECDSA cannot bypass the hardware PRF requirement. PQC hedging for the wrap step is deferred to a future format version.

---

9. Independent Verification

The complete V4 specification — byte layouts, algorithm parameters, derivation pseudocode, and test vectors — is maintained in VaultSort's source repository at `ignore/Yubi/phase-iv_and-v/phase-iv/v4_format.md`. The implementation is in `src-electron/modules/yubikey-encryption.ts` and `src-electron/modules/webauthn-service.ts`.

The credential store can be inspected at
`~/Library/Application Support/VaultSort/.../config.json`.
It contains credential IDs, public keys, and PRF salts — no secrets. The file encryption key and the PRF output are never written to disk.