# IronVault: The Tamper-Proof DataWarehouse Reference Guide

## 1. Executive Summary: The "Integrity Anchor" Concept

IronVault is a storage architecture designed to eliminate "blind trust." It separates the **Payload** (the actual data) from the **Proof** (the evidence of its validity).

* **The Core Philosophy:** Data is stored off-chain for performance, but its cryptographic identity is anchored on-chain for immutability.
* **The Result:** A system where **Non-Repudiation** is guaranteed. Mathematical proof replaces administrative trust. If a file changes by a single bit, the system knows, alerts, and heals itself.

## 2. The Layered Object Architecture

The system treats data not as files, but as **Secure Objects** that pass through four distinct, decoupled layers.

### Layer A: Transformation (The Engine)

* **Role:** Prepares the raw data for storage.
* **Rule:** *Compress First, Encrypt Second.*
* **Compression:** Uses **Zstandard (Zstd)** to reduce storage footprint and attack surface.
* **Encryption:** Uses **Envelope Encryption** (AES-256 GCM or Post-Quantum Kyber). A unique key is generated for every object, wrapped by a Master Key (HSM).

### Layer B: Distribution (The Shield)

* **Role:** Ensures availability and performance via **Software RAID / Erasure Coding**.
* **Logic:** The encrypted artifact is sliced into shards (e.g., 10 Data + 4 Parity).
* **Benefit:** Shards are scattered across different physical hardware. You can lose entire servers without data loss.

### Layer C: The Anchor (The Truth)

* **Role:** Provides the immutable evidence.
* **Blockchain:** Stores the **Hash** (Digital Fingerprint) of the encrypted artifact.
* **Index:** Stores the **Metadata** (Location Map, Author, Timestamp).
* **Rule:** *Never* encrypt/compress the Hash itself (it requires maximum entropy). *Always* encrypt the Metadata.

### Layer D: Persistence (The Vault)

* **Role:** The physical resting place.
* **Hot Tier:** NVMe/Flash for instant analytics.
* **WORM Tier:** **Write Once, Read Many** storage (Tape/Object Lock). Guarantees immunity to ransomware and deletion.

## 3. Operational Lifecycle: The 5 Flows

### 1. The Write Flow (Ingestion)

1. **Transform:** Input $\rightarrow$ Compress $\rightarrow$ Encrypt $\rightarrow$ Encrypted\_Artifact.
2. **Fingerprint:** Calculate Hash of Encrypted\_Artifact.
3. **Distribute:** Split Artifact into Shards (Data + Parity).
4. **Persist:** Write Shards to Disk + WORM.
5. **Anchor:** Commit Hash to Blockchain + Metadata to Index.
6. **Ack:** Return "Success" only after Anchor confirmation.

### 2. The Read Flow (Retrieval)

1. **Locate:** Query Index for Shard locations.
2. **Reassemble:** Fetch Shards $\rightarrow$ Reconstruct Encrypted\_Artifact.
3. **Verify:** **Compute Hash of Artifact** and compare with **Blockchain Anchor**.
4. **Decrypt:** Unwrap Key $\rightarrow$ Decrypt $\rightarrow$ Decompress $\rightarrow$ Return Data.

### 3. The Tamper Flow (Detection)

* **Scenario:** A rogue admin modifies a file on disk.
* **Detection:** Background "Scrubber" process hashes the disk file.
* **Conflict:** Disk Hash $\neq$ Blockchain Anchor Hash.
* **Action:** Alert Security. Mark file as "Compromised."

### 4. The Self-Healing Flow (Recovery)

* **Trigger:** Tamper detection or Bit Rot.
* **Action:** Retrieve remaining healthy Parity Shards.
* **Reconstruct:** Mathematically rebuild the missing/corrupted bytes.
* **Overwrite:** Replace the bad file on disk with the healed version.

### 5. The Audit Flow (Evidence)

* **Challenge:** "Prove this file hasn't changed since 2025."
* **Proof:** Hash the current file in real-time. Match it against the 2025 Blockchain Transaction ID.
* **Verdict:** Mathematical certainty.

## 4. The Configuration Matrix: Tunable Paranoia

IronVault is modular. You define the "Intensity" of the pipeline based on the asset's value.

| **Feature** | **1. I Don't Care** | **2. Speed Demon** | **3. Balanced** | **4. Fort Knox** | **5. Ultra Paranoid** |
| --- | --- | --- | --- | --- | --- |
| **Use Case** | Temp / Trash | Real-time Caching | Corporate Docs | Military / Legal | **Espionage / Black Ops** |
| **Compression** | None | LZ4 (Fastest) | Zstd (Level 3) | Zstd (Level 19) | **PAQ (Max Ratio)** |
| **Encryption** | None | Disk-Level Only | AES-256 | AES-GCM (Envelope) | **Kyber (Post-Quantum)** |
| **Integrity** | CRC32 | Async Batch Hash | SHA-256 | SHA-3 (Keccak) | **HMAC-SHA3 (Salted)** |
| **Distribution** | RAID 0 (Stripe) | RAID 10 (Mirror) | RAID 6 (Double Parity) | Erasure (10+4) | **Geo-Dispersed** |
| **Metadata** | Plaintext DB | In-Memory (Redis) | SQL TDE | App-Side Encrypted | **Shamir's Secret** |
| **Obfuscation** | None | None | None | None | **Chaff & Padding** |

## 5. Deep Dive: The "Ultra Paranoid" Mode

For use cases where the *existence* of data or *traffic patterns* are sensitive.

1. **Salted Hashes (HMAC):**
   * *Concept:* Instead of Hash(File), store Hash(File + Secret\_Salt).
   * *Why:* Prevents "Existence Proof" attacks. If a hacker has a leaked copy of a file, they cannot check the Blockchain to see if *you* also have it, because their hash won't match your salted hash.
2. **Traffic Chaff (Noise):**
   * *Concept:* The system randomly writes fake, encrypted garbage data to storage and blockchain.
   * *Why:* Defeats Traffic Analysis. An attacker watching your network cannot distinguish between a "Nuclear Launch Code" upload and random background noise.
3. **Shamir's Secret Sharing (Index):**
   * *Concept:* The Encryption Keys and Metadata are not stored in one database. They are mathematically split into 5 pieces.
   * *Why:* Requires 3 out of 5 distinct "Generals" (or server nodes) to agree before a filename can even be read.
4. **Padding:**
   * *Concept:* All files are padded to fixed sizes (e.g., exact 1GB blocks).
   * *Why:* Prevents "Side-Channel" attacks where the *size* of a file gives away its content.