Fractum

an open-source CLI for Threshold-Based Cold Storage of Critical Secrets

AES-256-GCM + **Shamir Secret Sharing**

`Repo: github.com/katvio/fractum`

THE PROBLEM

Everything you own is protected by one secret

The one-secret pile:


Password-manager master key


Infra private keys, Break-glass creds


Cloud KMS root


Cryptocur seed phrase


Family photos

So how do you back up

THAT ?

→One copy is **fragile.**

→Many full copies are **dangerous.**

LANDSCAPE

What exists - and the gap Fractum targets

All tools require securing a master key:

- File-encryption utilities
- Encrypted containers (VeraCrypt, .dmg)
- Password managers
- Enterprise secret managers (Vault)
- Wallet Shamir backups (SLIP-39)
- Raw Shamir tools (ssss lib)

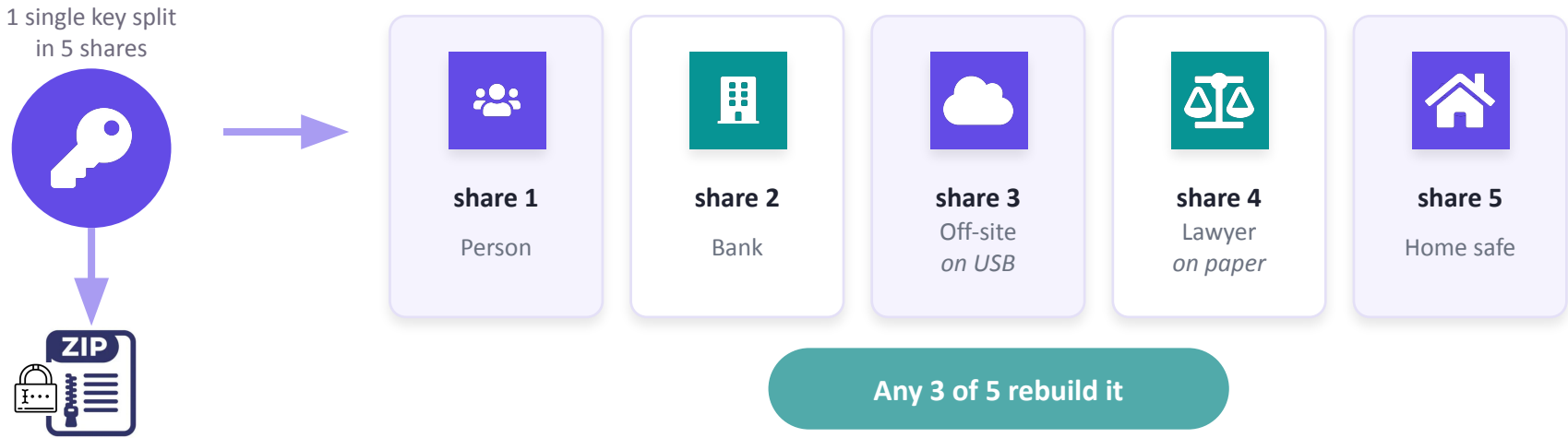
how is get stored today…

- **Single KMS / cloud vault**
  One provider, one account, one network.
- **One USB / paper in a safe**
  A single point of failure: fire, loss, theft, bit-rot.
- **Naïve split "just cut it into pieces"**
  Lose ONE piece → lose 100%
- **Copies everywhere**
  multiplying the attack surface

THE CORE IDEA

Best way to protect you master key?

Splitting it using Shamir's algo



Any **K of N** shares recover it — **fewer than K learn nothing.**

Long-term cold storage that's resistant to:

- loss, theft, disaster, & coercion

→ all at once.

Fractum is file-agnostic and self-contained for decade-scale recovery.
The novelty isn't AES or Shamir
—> it's the packaging and operations.

The gap: one workflow that is…

- file-oriented
- local-first
- offline-ready
- threshold-based
- portable
- inspectable FOSS
- Multi-OS
- battle-tested libs
- Built to outlive us

Who it's for

Organisations

- Break-glass credentials
- Root CA / offline keys
- DB & password-manager exports
- Legal / finance recovery docs

Individuals

- Crypto seed phrases
- Master backup keys
- Emergency documents
- Family photos & irreplaceable files

ADOPTION

Employed by ppl handling the highest-value keys

Trezor/Coinbase Dfns/Ledger

ICANN's DNS records & PKI Root CA key

Defense & Military industry

Banks/Finance

HOW IT WORKS

What makes it different - the operational model

Offline
zero network code

Self-bundled
src + wheels + bootstrap
in each ZIP

Portable / durable
recoverable in 10+ yrs
No PyPI, no vendor

Multi-OS
Linux · macOS · Win · Docker

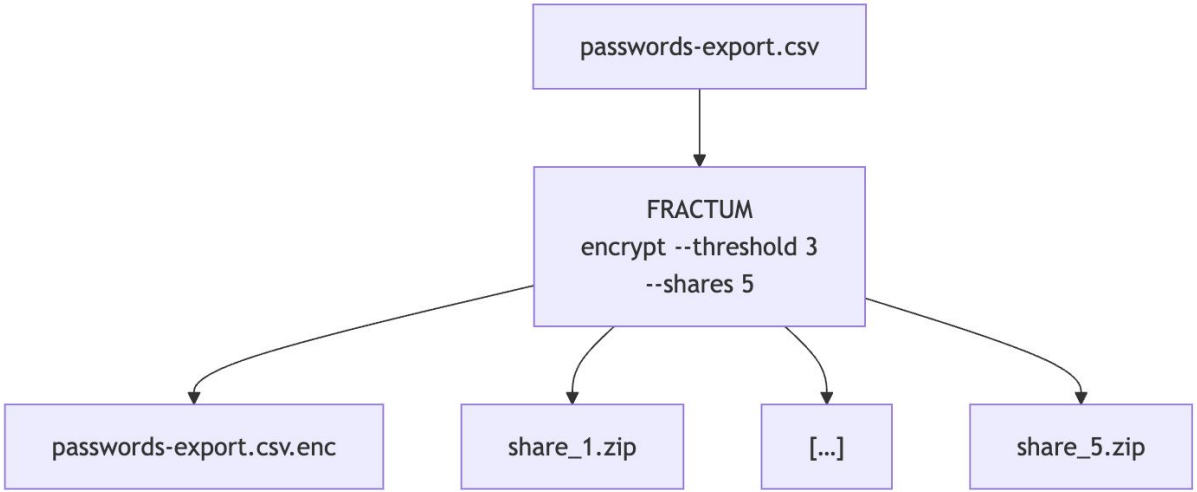
Auditable
~1,900 LOC readable Python

FOSS
Apache-2.0

Battle-tested crypto lib
PyCryptodome, not homemade

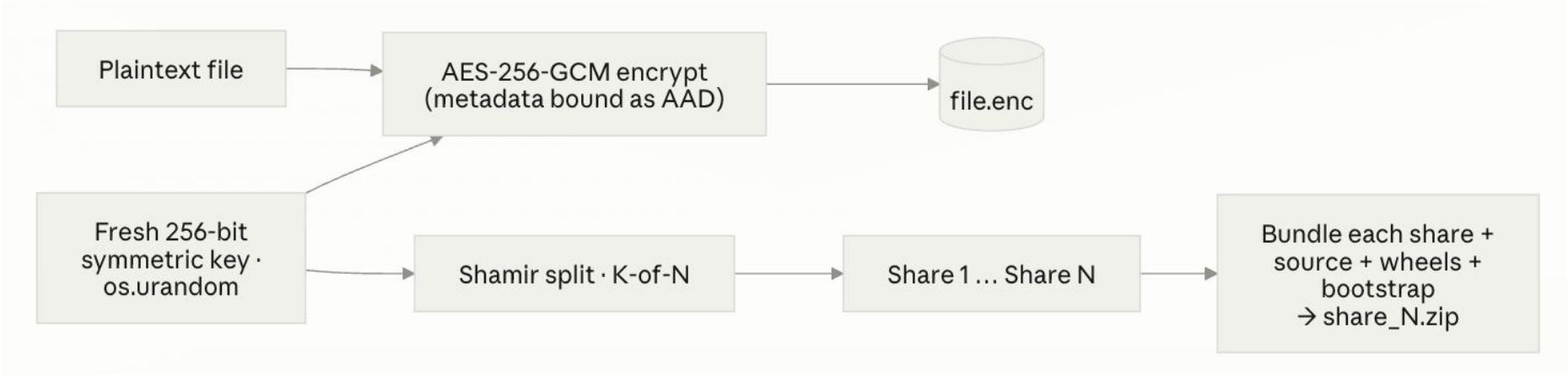
Encrypt the file and split the key

Encrypt the file with a fresh AES-256-GCM key, split **that key (not the file)** with Shamir, and bundle everything needed to recover it ; forever, offline.

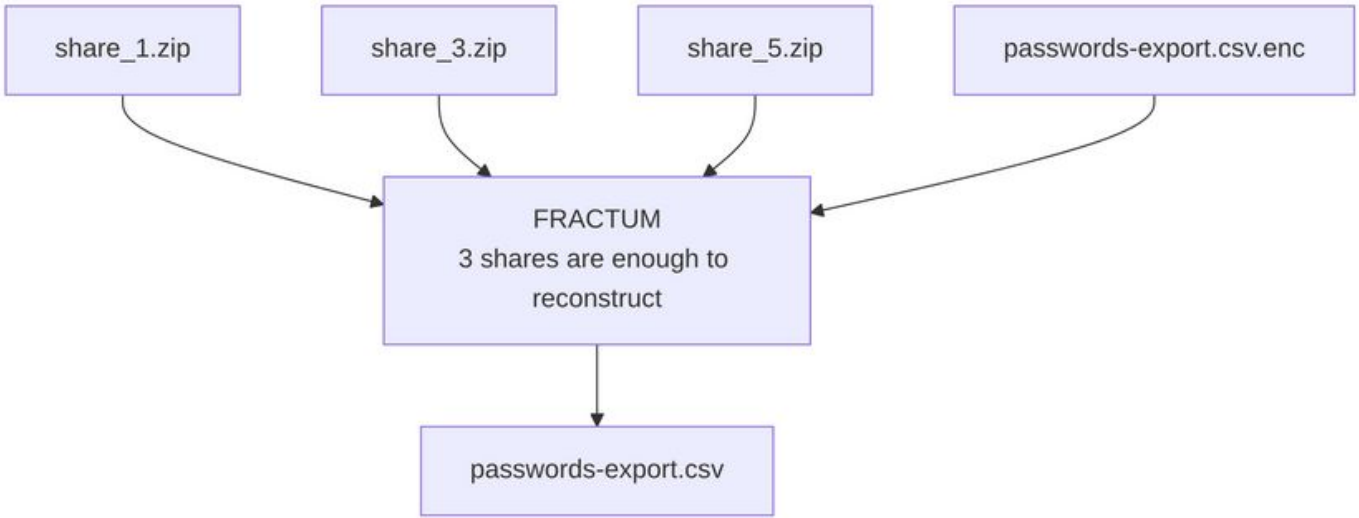


HOW IT WORKS

Encrypt the file and split the key

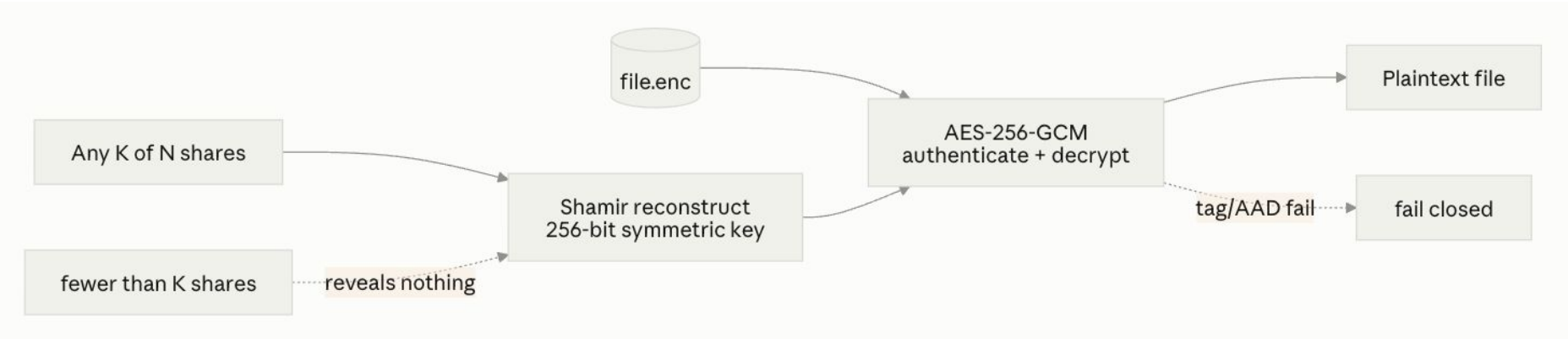


Recovery: reconstruct the key, then decrypt



The .enc alone isn't enough. One share alone isn't enough. **Recovery needs the .enc + at least K compatible shares.**

Recovery: reconstruct the key, then decrypt



Overview CLI

```
# Encrypt: 3-of-5 threshold scheme
fractum encrypt secret.txt -t 3 -n 5 -l mylabel

# Decrypt: point at a DIRECTORY of shares .zip)
fractum decrypt secret.txt.enc -s ./shares
```

`-t / --threshold`	K — shares needed (≥ 2)
`-n / --shares`	N — shares created (K≤N≤255)
`-s / --shares-dir`	directory of shares to recover
`-l / --label`	optional (defaults to filename)
`-b / --bundle-encrypted`	copy .enc into each ZIP
`--full-metadata`	embed label / set-id / integrity
`-m / --manual-shares`	paste Base64 values of the share

Why AES-256-GCM

AEAD

Confidentiality and integrity in one primitive

128-bit authentication tag

Any change to ciphertext / nonce / tag ⇒ decryption fails closed.

Metadata as AAD

Header (label, version, set-id) is authenticated. Flip a byte ⇒ decrypt fails.

Shamir Secret Sharing: threshold

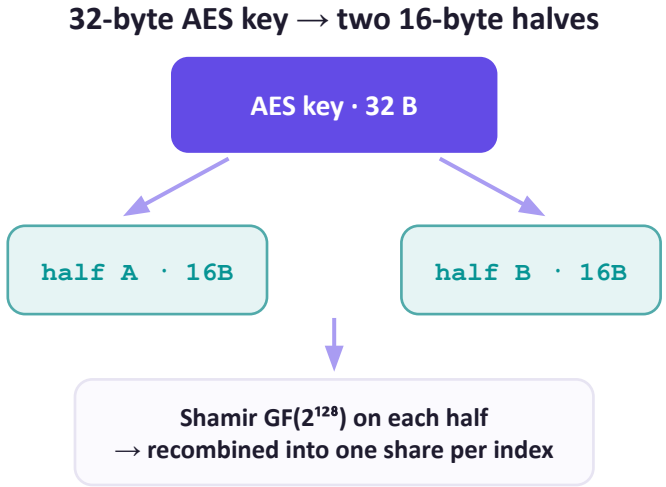
Below threshold ⇒ ZERO information

Not “computationally hard”: mathematically nothing.
Impossible even for a quantum computer.

K ≥ 2

K ≤ N ≤ 255

-Per-share SHA-256 verified before reconstruction.
-PyCryptodome Shamir over GF(2^{128}).



Entropy: where the AES key comes from

- **Difficult to add entropy above the kernel CSPRNG**
- **Kernel CSPRNG everywhere** Linux getrandom(2) · macOS getentropy(2) · Windows BCryptGenRandom → full 256-bit min-entropy.
- **When low entropy WOULD bite:** sub-second-old VMs, distroless without /dev/urandom, cloned-VM RNG state.

The sensitive moment (SPOF) + memory limits

K-of-N removes single points of failure at rest. But everything converges on one machine, twice:

At ENCRYPT

- plaintext + AES key + ALL N shares
  — together on the machine

At DECRYPT

- K shares + reconstructed key + plaintext
  — together on the machine

That host IS the single point of compromise. Memory hygiene is best-effort, not a guarantee: mutable bytearray keys, multi-pass overwrite, mlock (non-fatal).
Limits remain: OS swap, crash dumps, cold-boot, a compromised host.

Best practice: the throwaway machine

Dedicated, air-gapped, disposable host
e.g. a Raspberry Pi you can wipe or physically destroy.

Docker --network=none
The load-bearing control: a compromised code still can't exfiltrate.

No persistence
Minimal apps, controlled swap. Destroy / wipe after the session.

DEMO pt1

```
 ~/pro_wks/fractum  main *6 !1 ?2  fractum --version
fractum version 1.4.0

 ~/pro_wks/fractum  main *6 !1 ?2  echo "top-secret recovery phrase" > secret.txt ;  cat secret.txt
top-secret recovery phrase

 ~/pro_wks/fractum  main *6 !1 ?3  fractum encrypt secret.txt -t 3 -n 5 -v
Using label: secret (spaces replaced with underscores)
Created shares directory
Generated share set ID: 77fbc8e34fad728a
Generated shares: 5
Encrypted file: secret.txt.enc
Created archive: /Users/[redacted]/pro_wks/fractum/shares/share_1.zip
Created archive: /Users/[redacted]/pro_wks/fractum/shares/share_2.zip
Created archive: /Users/[redacted]/pro_wks/fractum/shares/share_3.zip
Created archive: /Users/[redacted]/pro_wks/fractum/shares/share_4.zip
Created archive: /Users/[redacted]/pro_wks/fractum/shares/share_5.zip

 ~/pro_wks/fractum  main *6 !1 ?4  ls shares/ secret.txt.enc

secret.txt.enc

shares/:
share_1.zip     share_2.zip     share_3.zip     share_4.zip     share_5.zip
```

DEMO

DEMO pt2

```
~/pro_wks/fractum  main *6 !1 ?4  unzip -l shares/share_1.zip
Archive:  shares/share_1.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
      181  07-02-2026 12:58   share_1.txt
    23195  07-02-2026 12:58   LICENSE
     3842  07-02-2026 12:58   bootstrap-macos.sh
     1437  07-02-2026 12:58   Dockerfile
     4851  07-02-2026 12:58   README.md
      887  07-02-2026 12:58   setup.py
       95  07-02-2026 12:58   .dockerignore
     5096  07-02-2026 12:58   bootstrap-linux.sh
     3250  07-02-2026 12:58   bootstrap-windows.ps1
    98188  07-02-2026 12:58   packages/click-8.1.8-py3-none-any.whl
  2268954  07-02-2026 12:58   packages/pycryptodome-3.23.0-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
  1799636  07-02-2026 12:58   packages/pycryptodome-3.23.0-cp37-abi3-win_amd64.whl
  2495627  07-02-2026 12:58   packages/pycryptodome-3.23.0-cp37-abi3-macosx_10_9_universal2.whl
       79  07-02-2026 12:58   src/config.py
      137  07-02-2026 12:58   src/__init__.py
     4700  07-02-2026 12:58   src/crypto/encryption.py
     5490  07-02-2026 12:58   src/crypto/memory.py
      226  07-02-2026 12:58   src/crypto/__init__.py
      207  07-02-2026 12:58   src/utils/__init__.py
     1106  07-02-2026 12:58   src/utils/integrity.py
       48  07-02-2026 12:58   src/cli/__init__.py
     3716  07-02-2026 12:58   src/cli/core.py
    10391  07-02-2026 12:58   src/cli/interactive.py
    33375  07-02-2026 12:58   src/cli/commands.py
     2622  07-02-2026 12:58   src/shares/metadata.py
     4722  07-02-2026 12:58   src/shares/archiver.py
      259  07-02-2026 12:58   src/shares/__init__.py
     7516  07-02-2026 12:58   src/shares/manager.py
---------                     -------
  6779833                     28 files

~/pro_wks/fractum  main *6 !1 ?4  unzip -p shares/share_1.zip share_1.txt
{
  "share_index": 1,
  "share_key": "HEk3l1LvWvrLk9jCYf7bg0UW8nS7eNKqlfZiwvG73eg=",
  "threshold": 3,
  "hash": "6ebb3606deb2647bb12d74dc371e7415c347667a4d61f8e2e45bdb56db6f8efe"
}%
```

DEMO

DEMO pt3

```
~/pro_wks/fractum main *6 !1 ?4 mkdir two && cp shares/share_1.zip shares/share_2.zip two/

~/pro_wks/fractum main *6 !1 ?5 fractum decrypt secret.txt.enc -s ./two -v

Looking for shares in: /Users/[redacted]/pro_wks/fractum/two
Found share set ID in metadata: 77fbc8e34fad728a
Found 2 ZIP archives
Processing 2 share files/archives
Found shares for 1 different labels
  - _unlabeled: 2 shares (threshold: 3)
Found shares for 1 different labels
  - _unlabeled: 2 shares (threshold: 3)
Trying shares with label: _unlabeled
Error trying label _unlabeled: Insufficient number of shares: 2 < 3
Error: No compatible shares found. None of the available shares could decrypt the file.

~/pro_wks/fractum main *6 !1 ?5 mkdir out

~/pro_wks/fractum main *6 !1 ?5 cp secret.txt.enc out/

~/pro_wks/fractum main *6 !1 ?6 cp shares/share_1.zip shares/share_3.zip shares/share_5.zip out/

~/pro_wks/fractum main *6 !1 ?6 cd out ; fractum decrypt secret.txt.enc -s . -v ; cd ..

Looking for shares in: /Users/[redacted]/pro_wks/fractum/out
Found share set ID in metadata: 77fbc8e34fad728a
Found 3 ZIP archives
Processing 3 share files/archives
Found shares for 1 different labels
  - _unlabeled: 3 shares (threshold: 3)
Found shares for 1 different labels
  - _unlabeled: 3 shares (threshold: 3)
Trying shares with label: _unlabeled
Successfully verified key with label: _unlabeled

Using shares for label: _unlabeled
Found 3 shares (need 3)
Key reconstructed from shares
File successfully decrypted: /Users/[redacted]/pro_wks/fractum/out/secret.txt

~/pro_wks/fractum main *6 !1 ?6 shasum -a 256 secret.txt out/secret.txt

0ef11f0988dc77ec2da993345be332b9836fc9260ebb1d138bb456dc6c2d256d  secret.txt
0ef11f0988dc77ec2da993345be332b9836fc9260ebb1d138bb456dc6c2d256d  out/secret.txt
```

Post-Quantum Ready: by construction

No public-key crypto anywhere. No RSA / ECC / DH ⇒ Shor has nothing to attack.

AES-256 vs Grover

≈ 128-bit effective ⇒ NIST Category 5.

Shamir below threshold

Information-theoretic ⇒ immune to Shor and Grover.

GCM tag (128-bit)

No meaningful quantum speedup on GHASH.

"Post-Quantum Ready for the symmetric layer, by construction"

DESIGN CHOICE

Why Python, not Rust

The Python trade

- Battle-tested libs: *PyCryptodome* AES-GCM & Shamir, **12+ years** of scrutiny.
- Auditable: small, readable app logic
- Supply chain: vendored wheels + SHA-256, fully offline; no per-machine build.

The honest concession

- Rust offers memory safety, and single binay story
- Our trade a minimal auditable app over a mature Shamir library, for decade-scale recovery.
- pip / PyPI is itself a supply-chain surface: so we vendor + checksum + go offline.

To be precise:

PyCryptodome ships a compiled C extension (the AES / GHASH core).
Honest claim: the application logic is auditable Python; we don't ship our own compiled binary.

Open source ≠ trust solved: verify, don't trust

Verify, don't trust. Open source gives review, auditability, reproducibility goals. It does not prove the release is authentic, deps are clean, the host is clean, or the workflow was followed.

It does NOT prove…

- dependencies are clean
- the host is clean
- the workflow was followed

"Open source doesn't solve trust: it makes trust inspectable."

Verifiable today

- GPG-signed tags (fingerprint on the site)
- Wheels checksummed vs PyPI's SHA-256
- Digest-pinned Docker base
- SHA-pinned GitHub Actions

```
gpg --fingerprint D009F6290DCFDAB6
(cd packages && \
  sha256sum --check \
  CHECKSUMS.sha256 --strict)
```

EXPERIMENTAL

Experimental verification: AI + attestation

The idea

- Each release: run a strong model (e.g. Opus 4.8) with a public security-audit prompt over the source
- Publish a signed, timestamped, immutable attestation (blockchain-anchored, e.g. GenLayer)
- A “green badge” on the release page

Defense-in-depth, not proof:

We're exploring independent verification mechanisms.

Not a replacement for:
signatures · human review · external audit.

GET INVOLVED

Contribute

LOOKING FOR:

- Cryptographers and Security auditors
- Formal Proofs (Coq etc)
- Operational-security practitioners
- Rust cryptographers

WHAT'S NEXT

Roadmap

Independent security audit - *external company + independent cryptographers*

Migrate Shamir to an audited implementation - *evaluate Privy's SSS*

Ocaml/Coq: Formal verification - *for selected components*

Harden the pipeline — *reproducible builds · signed checksums*

Thanks - questions?

Questions?

github.com/katvio/fractum

docs: fractum.katvio.com

```
signing key:  179D F24B 6E6E 3D8A EFD9 A795 D009 F629 0DCF DAB6
```