TENUP

# Blockchain Verification Protocol

The infrastructure for proof of truth. Any digital claim, verifiable forever.

WHITEPAPER | v2.0 | 2026

DOCUMENT STATUS & AUDIENCE

# A technical description of the verification layer

This document specifies the TENUP protocol: how arbitrary digital content is serialized, fingerprinted, committed to a public chain, revealed, and independently verified. It is written for engineers integrating the SDK, security reviewers auditing the contracts, enterprises evaluating the protocol as trust infrastructure, and institutions assessing the network.

- Engineers & integrators
- Security reviewers
- Enterprises, universities, governments
- Partners & investors

EXECUTIVE SUMMARY

# Proof instead of trust

TENUP is a general-purpose verification protocol. A publisher canonicalizes any digital artifact, hashes it with a secret nonce, and commits that fingerprint on-chain at the moment of creation. Later, the artifact is revealed and the contract proves the fingerprint matches. Anyone can re-run the check independently, forever, without trusting the publisher, the platform, or a database.

### Content-agnostic

The protocol sees bytes, never subject matter.

### Private by design

Only fingerprints go on-chain. Never raw documents.

### Client-side

Verification runs in the browser, CLI, or your backend.

### Metered

The token pays for anchoring, API access, and staking.

VISION

# The internet moves information perfectly and proves nothing.

Every layer of the web has infrastructure: transport, payments, identity, delivery. Verification never got one. Trust is still delegated to institutions, screenshots, and reputations. TENUP adds the missing layer, and makes it mathematical.

WHY VERIFICATION IS NOW ESSENTIAL

# Generation became free. Verification did not.

01

## Content is infinite

Text, images, audio, documents and reports can be produced at zero marginal cost, in any style, by anyone.

02

## Provenance is absent

A file carries no reliable record of who made it, when, or whether it was altered afterwards.

03

## Decisions depend on it

Hiring, lending, diagnosis, litigation, research and markets all run on digital claims nobody can check.

PROBLEM STATEMENT

# What the internet cannot verify

| | | | |
|---|---|---|---|
| AI outputs | Screenshots | Edited PDFs | Manipulated reports |
| Fake certificates | Backdated predictions | Modified documents | Deepfakes |
| Deleted posts | Edited research | Revised forecasts | Any digital claim |

Selective reporting, hindsight editing and silent deletion are possible everywhere, because nothing was ever sealed at the moment it was made.

SOLUTION

# Five steps, one guarantee

STEP 1

## Create content

A document, model output, record or claim.

→

STEP 2

## Generate hash

Canonical bytes plus a secret nonce, keccak256.

→

STEP 3

## Anchor on chain

The fingerprint is committed and timestamped.

→

STEP 4

## Reveal

Payload and nonce are disclosed in-window.

→

STEP 5

## Verify forever

Anyone recomputes the hash and matches the chain.

The commitment proves existence and time. The reveal proves content. Together they remove the possibility of editing history.

SCOPE

# What TENUP can verify

### AI content
Model outputs, prompts, versions

### AI predictions
Forecasts sealed before the outcome

### Trading signals
Calls committed at broadcast

### Certificates
Degrees, licences, credentials

### Research papers
Preprints, datasets, revisions

### Legal documents
Contracts, filings, evidence

### Medical reports
Results and readings, privately

### Education records
Transcripts and enrolment

### Financial reports
Statements, disclosures, audits

### Government records
Notices, permits, registries

### Intellectual property
Designs, drafts, priority dates

### Digital identity
Attestations without disclosure

### Journalism
Publication time and edits

### Scientific data
Experiment logs and results

### Enterprise reports
Internal and board reporting

### Supply chain
Custody events and origin

PROTOCOL ARCHITECTURE

# From content to public proof

Any digital content

Canonical serializer

Hash generator

Commit engine

TENUP chain contracts

Reveal engine

Verification SDK · Explorer · Public verification

## Ingress

SDK, REST, GraphQL and webhook connectors accept structured payloads or file digests from any system.

## Orchestrator

Finalizes the payload, draws the nonce, computes the fingerprint, schedules the commit and queues the reveal.

## Contracts

Store commitments and Merkle roots, verify reveals against them, enforce disclosure windows, and emit events.

## Egress

Indexer, Explorer, verification badges and open-source SDKs let any third party re-run the check.

INFRASTRUCTURE NOTES

# Built for volume and continuity

## Chain

EVM L2 with low fees and fast finality. Block timestamps are the canonical clock; the UI shows pending state until k confirmations.

## Clocks

NTP-synchronised hosts, monotonic sequencing per publisher, on-chain time as the source of record.

## Storage

Hot cache for pending commitments, columnar store for revealed payloads and proofs, optional content addressing with on-chain CIDs.

## Resilience

Multi-region ingress, decoupled streams, idempotent processors, dead-letter queues and reveal-deadline alerting.

HOW VERIFICATION WORKS

# Seven steps end to end

| 01 | 02 | 03 | 04 | 05 | 06 | 07 |
|---|---|---|---|---|---|---|
| Create content | Canonical serialization | Generate hash | Commit | Blockchain timestamp | Reveal | Verification |

**Deterministic input.** Canonical serialization guarantees the same artifact always produces the same bytes, on any platform, in any language.

**Sealed, not published.** The commitment discloses nothing. The content stays private until the publisher reveals it.

**Independent check.** Verification needs only the payload, the nonce and a public RPC endpoint. No TENUP service is trusted.

STEP 02 · CANONICAL SERIALIZATION

# One artifact, one byte sequence

TENUP uses the JSON Canonicalization Scheme (RFC 8785) so that independently written clients agree on the exact bytes to hash.

- UTF-8 encoding
- Lexicographically sorted keys
- No superfluous whitespace
- ECMAScript number format, no trailing zeros
- No NaN or Infinity

```
{
  "type":      "credential.v1",
  "issuer":    "did:tenup:univ.example",
  "subject":   "sha256:9c4f…a71b",
  "issued_at": "2026-08-17T09:14:02Z",
  "claims":    { "program": "MSc" },
  "publisher": "0xPUB…KEYID"
}

// C = JCS(payload_without_nonce) as UTF-8 bytes
```

STEP 03 · HASH CONSTRUCTION

# The fingerprint

```
hash = keccak256( C(payload_without_nonce) ‖ nonce )
```

### 32-byte nonce

Drawn from a CSPRNG per artifact. Its entropy makes dictionary attacks and content guessing infeasible before reveal.

### Nonce excluded

The nonce is left out of canonicalization to avoid self-reference, then concatenated, so it can be disclosed alongside the payload.

### Publisher binding

An optional secp256k1 signature over C binds the artifact to a recognised signing key, checked off-chain by the SDK.

STEP 04 · COMMIT

# Sealing at $T_0$

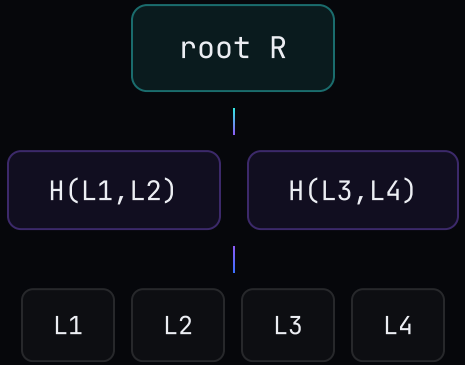
1. Build the payload without the nonce
2. Draw a 32-byte nonce
3. Compute the fingerprint
4. Call `commit(hash, meta)`
5. Publish or deliver the artifact as usual

Metadata stored on-chain is deliberately compact: integer indexes instead of strings, Unix seconds instead of dates.

```
typeId      uint32
schemaId    uint16
producerId  uint16
publisher   address
seq         uint64
minRevealAt uint40
maxRevealAt uint40
```

BATCHED COMMITMENT

# One root, millions of artifacts




$L_i = \text{keccak256}( C_i \,\|\, \text{nonce}_i )$ · pairwise sorted hashing

### Batch window

Artifacts inside a window are leaves; only the root is written on-chain.

### Per-item reveal

A reveal supplies payload, nonce, leaf and Merkle proof; the contract checks the proof against the stored root.

### Cost profile

Gas per artifact falls logarithmically with batch size, making high-volume anchoring economical.

STEP 06 · REVEAL AND ON-CHAIN CHECKS

# The contract does the checking

- Recompute hash* from the revealed payload and nonce
- Single commit: assert hash* equals the stored hash
- Batch: verify the Merkle proof against the stored root
- Assert block.timestamp within [minRevealAt, maxRevealAt]
- Mark revealed and emit the event with payload bytes or CID

### minRevealAt

Prevents instant disclosure that would leak a private payload prematurely. Typically minutes.

### maxRevealAt

Forces timely disclosure. Configurable per artifact class, from hours to years.

### Expired

A commitment never revealed in-window is flagged publicly, so withholding unfavourable artifacts is visible rather than silent.

STORAGE & PRIVACY MODES

# You choose what is ever disclosed

MODE A

## Fingerprint only

Nothing but the hash ever leaves your systems. The holder of the document proves it themselves. Suitable for medical, legal and identity records.

MODE B

## Compact on-chain

Small payloads are emitted as event bytes for maximum longevity, with no external dependency for retrieval.

MODE C

## Content-addressed

Larger artifacts are stored off-chain with the CID anchored on-chain, plus periodic cold-storage snapshots for redundancy.

In every mode the chain holds fingerprints and metadata. Raw documents, patient data and personal records are never written to a public ledger.

SMART CONTRACT ARCHITECTURE

# Minimal surface, maximal durability

```
// SPDX-License-Identifier: MIT
contract TenupVerification {
  struct Record {
    bytes32 hashOrRoot;   // leaf or merkle root
    CommitMeta meta;
    bool isBatch;
    bool revealed;
  }
  function commit(bytes32 leaf, Meta m) external;
  function commitBatch(bytes32 root, Meta m) external;
  function revealSingle(Meta m, bytes c, bytes32 n) external;
  function revealLeaf(Meta m, bytes c, bytes32 n,
                      bytes32[] proof) external;

  event Committed(bytes32 key, bytes32 root, bool batch);
  event Revealed(bytes32 key, bytes32 leaf, bytes ref);
}
```

### Keyed by publisher and sequence

key = keccak256(publisher, seq); commits are write-once.

### No payload storage

Content travels through events or CIDs, never contract storage.

### Indexer-friendly

Per-leaf reveal state is tracked off-chain or via an optional bitmap extension.

### Upgrade discipline

Multisig plus timelock; the verification core stays deliberately small.

CLIENT-SIDE VERIFICATION

# Verification runs on the verifier's machine

TYPESCRIPT

```ts
import { verify } from "@tenup/verify";

const result = await verify({
  payload,            // revealed artifact
  nonce,              // 32-byte hex
  rpc: RPC_URL        // any public node
});

result.status    // "verified"
result.anchored  // 2026-08-17T09:14:02Z
result.txHash    // 0x7c1a…
```

PYTHON

```python
from tenup import jcs, keccak

def fingerprint(payload, nonce):
    c = jcs(payload).encode("utf-8")
    n = bytes.fromhex(nonce[2:])
    d = keccak(c + n).hex()
    return "0x" + d

# compare to the anchor
assert fingerprint(p, n) == anchor
```

Both libraries are open source and dependency-light. The canonicalizer, the hash and the proof check are all a verifier needs; TENUP's own services are never in the trust path.

PROTOCOL COMPONENTS

# The nine primitives

### Commit
Seals a fingerprint and its time without disclosing content.

### Reveal
Discloses the artifact and proves it matches the seal.

### Hash
keccak256 over canonical bytes; collision-resistant identity.

### Merkle tree
Batches many artifacts under one on-chain root.

### Nonce
32 bytes of entropy that make pre-reveal guessing infeasible.

### Canonical form
Deterministic serialization so every client agrees on the bytes.

### Explorer
Public record of every commitment, reveal and status change.

### SDK
Open-source verifier and anchoring clients for TS and Python.

### APIs
REST, GraphQL and webhooks for anchoring and verification at scale.

VERIFICATION EXPLORER

# Every proof, publicly inspectable

| ARTIFACT | PUBLISHER | ANCHORED | PROOF | STATUS |
|---|---|---|---|---|
| credential.v1 | univ.example | 09:14:02Z | 0x7c1a…9b4e | Verified |
| research.preprint | lab.example | 09:11:47Z | 0x2f88…c105 | Verified |
| ai.output | assistant-api | 09:10:22Z | 0xa413…7d20 | Awaiting reveal |
| signal.v3 | tenup-signals | 09:08:03Z | 0x91cd…4fa7 | Committed |
| supplychain.custody | port.example | 08:59:41Z | 0x60ba…13ce | Expired |

- Committed — sealed, awaiting the disclosure window
- Awaiting reveal — window open
- Verified — content matches the seal
- Expired — never disclosed in time

SDK & APIS

# Six ways in

### TypeScript SDK

Anchor and verify from Node or the browser. Works with any RPC provider.

### Python SDK

For data pipelines, research workflows and ML systems that emit artifacts.

### REST

POST /anchor, GET /proof, GET /verify. Batch endpoints for bulk anchoring.

### GraphQL

Query commitments, reveals and status history across publishers and types.

### Webhooks

Push notifications on commit confirmation, reveal, expiry and finality.

### Enterprise

Dedicated throughput, private publishers, key custody options and SLAs.

APPLICATIONS

# One protocol. Many industries.

Each application uses the same commit, reveal and verify primitives. What changes is the artifact schema, the disclosure window, and who checks the proof.

FLAGSHIP APPLICATION 01

# AI trading signals

Signals are the hardest test of the protocol: the incentive to edit history is immediate and financial. Every call is committed at broadcast and revealed after the trade window, so performance is computed only from sealed, disclosed calls.

```
{
  "type":       "signal.v3",
  "ts":         "2026-08-17T09:08:03Z",
  "pair":       "BTCUSDT",
  "direction":  "LONG",
  "entry":      { "from": 64250, "to": 64290 },
  "take_profit": [64520, 64780, 65150],
  "stop_loss":   63940,
  "model":       "ob-v3.2",
  "confidence":  0.74,
  "publisher":   "0xPUB…KEYID"
}
```

### No hindsight editing

The call existed, in this exact form, at this exact block.

### No hidden losers

Unrevealed commitments are flagged Expired and penalised in analytics.

### Auditable performance

Hit rate, profit factor, expectancy and drawdown derived from verified history only.

APPLICATIONS 02 — 05

# Knowledge and credentials

## AI research

Anchor model versions, prompts, evaluation runs and results before publication. Claims about what a system produced become checkable.

## Universities

Degrees, transcripts and enrolment records verified by any employer in seconds, with no registry call and no document ever exposed.

## Scientific research

Pre-register hypotheses and datasets; reveal after the study. Priority and integrity are provable, not asserted.

## Intellectual property

Designs, drafts and source can be sealed at creation, establishing a timestamped priority date without disclosing the work.

APPLICATIONS 06 — 10

# Institutions and records

## Legal

Contract versions, filings and evidence chains sealed at signature time, so later alteration is detectable.

## Healthcare

Reports and readings anchored as fingerprints only. Patient data stays in the clinical system.

## Insurance

Claims, inspections and photographs timestamped at capture, reducing post-hoc dispute.

## Government

Notices, permits and registry entries citizens can verify without contacting the issuing office.

## Financial reporting

Statements and disclosures anchored at issue, giving auditors a fixed reference point.

## Digital identity

Attestations proved against an anchor rather than by handing over the underlying documents.

APPLICATIONS 11 — 15

# Media, industry and enterprise AI

## Journalism

Publication time, source material and every subsequent correction on the public record.

## Media provenance

Images and video anchored at capture, so a synthetic re-cut cannot inherit the original's proof.

## Enterprise AI

Every model output in a regulated workflow anchored with its model version and inputs, for audit and incident review.

## Supply chain

Custody events, inspections and origin claims sealed at each handover.

## Enterprise reporting

Board packs, ESG figures and internal metrics anchored when issued, not when questioned.

## Predictions

Any forecast — economic, clinical, operational — sealed before the outcome is known.

# Trust versus proof

TRADITIONAL

## Requires trust

- Records held by the party being checked
- Timestamps issued by the same party
- Edits and deletions leave no trace
- Verification means contacting an institution
- Availability depends on one company's servers

TENUP

## Provides mathematical proof

- Fingerprints held on a public, append-only ledger
- Timestamps set by block consensus
- Any alteration changes the hash and fails the check
- Verification is a local computation
- Proofs outlive the company that created them

TOKEN UTILITY

# The protocol is the product

TENUP is the metering and coordination asset of the network. It pays for anchoring work, gates access tiers, and aligns the operators and model publishers who depend on the protocol.

### Verification credits

Metered anchoring and proof retrieval.

### API access

Tiered rate limits and throughput.

### Enterprise usage

Committed volume with contractual SLAs.

### SDK & platform

Developer credits and sandbox quotas.

### Staking

Publishers stake to list; poor conduct is slashed.

### Marketplace

Verified model and data listings with revenue share.

### Governance

Reveal windows, batch policy, fee splits, listings.

### Usage sink

Net protocol revenue routed to timelocked buyback and burn.

TENUP is a utility token for access, staking and governance. It does not represent equity, debt, or a right to revenue.

SECURITY & THREAT MODEL

# Every attack has a named mitigation

### Hindsight editing

Commit-reveal with a secret nonce and public verification.

### Content guessing pre-reveal

32 bytes of nonce entropy per artifact.

### Withholding unfavourable artifacts

maxReveal windows plus public Expired flags counted against the publisher.

### Front-running commits

Harmless: a commitment discloses no content.

### Publisher spoofing

Off-chain signature binds the artifact to a registered key; publisher in meta must match.

### Chain reorganisation

Pending state until k confirmations; explorer tracks finality.

### Key compromise

HSM-held publisher keys, on-chain publisher registry, rotation; multisig and timelock for contracts.

### Contract bugs

External audits before mainnet, public bug bounty, minimal on-chain complexity.

ROADMAP

# Protocol first, network last

PHASE 1

## Foundations

Verification contracts
Verifier SDK and CLI
Public Explorer

PHASE 2

## Developer platform

REST and GraphQL APIs
Batch anchoring
Webhooks and dashboards

PHASE 3

## Flagship applications

Signal verification
Credentials and certificates
AI output anchoring

PHASE 4

## Institutions

Enterprise deployments
Universities and governments
Compliance tooling

PHASE 5

## Global network

Multi-chain anchoring
Token governance
AI verification network

FUTURE VISION

# The verification layer for AI and digital information.

Every AI output, every prediction, every model version.

Every certificate, transcript, research paper and report.

Every legal document and every important digital claim.

Verifiable at creation. Checkable by anyone. Permanent.

NEXT STEPS

# Where to start

## Developers

Install the SDK, anchor your first artifact on testnet, and verify it from a second machine.

## Enterprises

Pick one document class, run it through fingerprint-only mode, and measure the dispute cost it removes.

## Reviewers

Read the contracts and the canonicalization spec; both are small by design and open to audit.

Disclaimers. This document is informational and does not constitute investment advice or an offer. Protocol parameters, phases and figures are subject to change. Verifiable history is not a performance promise; applications built on TENUP carry their own risks, including total loss of capital in trading contexts.