XRPL TIMESTAMP SERVICE

Cryptographic File Notarization on the XRP Ledger

WHITE PAPER · v1.0

xrpltimestamp.com
nfo@xrpltimestamp.com

Privacy-first · Client-side commitment · XRPL-anchored · Self-verifiable

Abstract

XRPL Timestamp is an open, privacy-preserving file notarization service that anchors cryptographic commitments on the XRP Ledger. A file never leaves the user's device: the client computes a double-SHA-256 commitment over the file hash and a random nonce, then submits only that commitment to the ledger via a standard AccountSet transaction. Settlement occurs in approximately four seconds at a cost of 1 XRP, producing a publicly verifiable, tamper-evident proof of existence that any party can independently check without trusting the service operator.

This document describes the cryptographic design, the end-to-end protocol, the current implementation, and a roadmap of future extensions including user accounts, multi-party escrow, Merkle batch notarization, and automated ledger-account reclamation.

1. Introduction

Proof of existence — the ability to demonstrate that a document existed in a specific form at a specific point in time — is a well-established legal and compliance requirement. Traditional solutions rely on trusted third parties (notaries, timestamping authorities, registrars) whose records can be altered, lost, or disputed.

Blockchain timestamping replaces trusted intermediaries with cryptographic certainty. By anchoring a hash of a document on an immutable public ledger, any holder of the original can prove, without further trust assumptions, that the document existed before the block was confirmed.

XRPL Timestamp extends this concept with two additional guarantees:

- **Privacy:** The file hash is salted with a random nonce before being committed. The commitment itself reveals nothing about the file, and only the commitment reaches the network.
- **Self-sovereignty:** The printed or saved certificate contains every value (file hash, nonce, commitment, transaction hash, ledger sequence) needed to independently reconstruct and verify the proof — no dependency on xrpltimestamp.com after the fact.

2. Threat Model & Privacy Guarantees

2.1 What the service learns

The API endpoint receives only the cryptographic commitment — a 32-byte SHA-256 digest of (SHA256(file) || nonce). It cannot reverse the commitment to recover the file or its hash; it cannot correlate two commitments to the same file unless the nonce is reused (which the client prevents).

The service does record the IP address of the submitter for fraud-prevention purposes, consistent with standard web-service operation. No file content, filename, or file hash is ever transmitted.

2.2 What an observer learns

Anyone reading the XRP Ledger sees the raw AccountSet MemoData, which contains:

```
{ "v": 1,
  "alg": "sha256d-nonce",
  "commitment": "e3b0c44298fc1c149afb..." }
```

Without the nonce, the commitment is computationally indistinguishable from a random string. The nonce is delivered only to the user; it does not appear on-chain.

2.3 Integrity assumptions

The security of the scheme rests on the collision resistance of SHA-256 (NIST-standardized, no known practical attack) and the immutability of the XRP Ledger's closed ledgers. A ledger sequence, once closed and validated by a supermajority of UNL validators, cannot be altered retroactively.

3. Cryptographic Protocol

3.1 Commitment scheme

Let F be the raw bytes of the file to be timestamped, and let N be a 128-bit cryptographically random nonce generated in the browser.

```
H₁  = SHA256( F )          -- file fingerprint
C   = SHA256( H₁ ‖ N )     -- commitment (sent to API)
```

```
Memo = JSON { v:1, alg:'sha256d-nonce', commitment: hex(C) }
MemoData = hex( UTF8( Memo ) )
```

The construction SHA256(SHA256(F) || N) is chosen because:

- SHA256(SHA256(·)) provides length-extension attack resistance without requiring HMAC.
- The nonce prevents pre-image attacks on the commitment: an adversary cannot build a rainbow table of common documents and match them to on-chain commitments.
- The file hash H_1 is stored client-side (never transmitted) and printed on the certificate, allowing the holder to verify the file without contacting the service.

3.2 Verification algorithm

Given a certificate containing (H_1, N, C, tx_hash), a verifier runs:

1. Compute H_1' = SHA256(candidate_file).
2. Assert H_1' == H_1 (file integrity check).
3. Compute C' = SHA256(H_1' || N).
4. Assert C' == C (commitment integrity check).
5. Retrieve the AccountSet transaction tx_hash from any XRPL node.
6. Assert MemoData hex-decodes to JSON containing commitment == C.
7. Assert the transaction ledger_index and close_time fall within acceptable bounds.

> ⚠️ *All seven steps can be performed with standard open-source tooling (ripple-binary-codec, any SHA-256 implementation). No proprietary software is required.*

3.3 Transaction record

Transaction type	AccountSet
Fee	12 drops (0.000012 XRP)
MemoType	hex('xrpl-timestamp/v1')
MemoFormat	hex('application/json')
MemoData	hex(JSON commitment object)
Ledger close	≈ 4 seconds
Service charge	1 XRP (paid by user before worker submits)

4. System Architecture

4.1 Components

Browser client	Computes H_1 and C entirely in WebCrypto; never transmits raw file data
PHP REST API	Receives commitment; issues 402 Payment Required; queues jobs in MySQL
MySQL database	Persistent job store: status, commitment, tx_hash, ledger metadata
Node.js worker	Holds private key; polls MySQL; submits AccountSet; updates records
XRPL network	Mainnet via wss://xrplcluster.com; Reliable Submission pattern
Xaman / QR	User-facing payment UX; any XRPL wallet is also accepted
Apache / TLS	HTTP/2, mpm_event + PHP-FPM, Let's Encrypt (Certbot)

4.2 Request lifecycle

8. User selects a file. Browser computes `H₁` and `C`. Commitment is POSTed to /api/stamp.php.
9. API validates input, inserts a QUEUED job row, returns a payment address and job_id.
10. User pays 1 XRP to the service address (via Xaman QR or any wallet).
11. Worker detects the incoming payment, associates it with job_id by amount + destination tag, transitions job to PROCESSING.
12. Worker submits AccountSet with `MemoData` containing the commitment. Reliable Submission pattern: LastLedgerSequence set 4 ledgers ahead; re-submit on RETRY_LATER until validated or expired.
13. Worker writes tx_hash, ledger_index, ledger_close_time, and account to the database. Job transitions to COMPLETED.
14. Browser polls /api/status.php, receives the COMPLETED record, renders the certificate.

4.3 Key isolation

The private key of the signing account never touches the PHP API layer. It lives exclusively in the worker process environment, loaded from a .env file with 600 permissions owned by the service account. Compromise of the web server does not expose the signing key.

5. Current Implementation

5.1 Delivered artefacts (v1)

The following 15 source files form the complete v1 system:

client/client.c	C CLI client for batch or scripted notarization
client/Makefile	Build configuration
api/stamp.php	POST /api/stamp — job creation, 402 flow
api/status.php	GET /api/status — job polling
api/verify.php	GET /api/verify — standalone commitment verification
api/db.php	PDO database abstraction
api/helpers.php	Input validation, hex utilities
api/.htaccess	mod_rewrite rules, CORS headers
worker/worker.js	Node.js fire-and-forget batch worker
worker/merkle.js	Merkle tree helper (batch extension)
worker/package.json	Node dependencies
worker/xrpl-worker.service	systemd unit for pm2-managed process
sql/schema.sql	MySQL schema with indexes
docs/SETUP.md	Full deployment guide
docs/xrpl-tx-example.jsonc	Annotated example transaction

5.2 Frontend

index.html	Landing page / explainer
stamp.html	Main stamping tool
certificate.html	Print-ready proof certificate
how-it-works.html	Step-by-step protocol explainer
press.html	Press kit and media assets

5.3 Infrastructure

- Ubuntu 22.04 LTS on Linode VPS
- Apache 2.4 with HTTP/2 (mpm_event + PHP-FPM), mod_rewrite, mod_headers
- MySQL 8, PHP 8.1, Node.js 20 LTS
- TLS via Let's Encrypt / Certbot — apex, www, and staging subdomains
- Cloudflare DNS (no proxy — real origin IP, no orange cloud), email routing
- Staging environment at staging.xrpltimestamp.com (Basic Auth, shared live DB)
- pm2 process manager for worker; systemd unit for boot persistence

5.4 Internationalisation

The frontend supports 8 languages (EN, IT, FR, DE, ES, PT, JA, KO) via a lazy-loaded i18n system. Language files in js/i18n/ set per-language objects to avoid conflicts on same-language re-selection. GeoIP (MaxMind GeoLite2) provides automatic locale detection on first visit.

6. Roadmap & Future Development

The following extensions are planned or under active design. They are presented in approximate priority order.

6.1 User accounts

The current service is stateless from the user's perspective: each timestamp is independent and identified only by the job_id and the printed certificate. A future account system will allow registered users to:

- Maintain a dashboard of all their timestamps linked to a single identity.
- Retrieve past certificates without having saved the original JSON receipt.
- Delegate stamping rights to sub-accounts or teams (relevant for enterprise compliance workflows).

Account authentication will use email + TOTP (no passwords stored in plaintext). OAuth 2.0 social login is under consideration as a convenience layer. All timestamps remain independently verifiable on-chain regardless of account state.

6.2 Escrow from any account (multi-party notarization)

Today the service uses a single operator-controlled signing account. A significant limitation is that the on-chain transaction is signed by the service, not by the user, meaning the chain of custody is: user → service → ledger.

The escrow extension will invert this model:

15. User creates an XRPL Escrow transaction from their own account, locking 1 XRP with a FinishAfter condition tied to the operator's signing key.
16. The operator account signs the AccountSet commitment transaction and, in the same ledger, releases the escrow.
17. If the operator fails to submit within N ledgers, the escrow expires and the user reclaims the 1 XRP automatically.

This design provides a cryptographic guarantee to the user: they cannot be charged without a corresponding on-chain commitment, and any failure automatically refunds them. It also makes the user's XRPL account appear directly in the notarization chain, strengthening legal provenance.

> **Design note**
>
> *Conditional escrow requires the user to have a funded XRPL account. For users without an existing account, a deposit flow (funded by the service with automatic later reclamation) is described in §6.4.*

6.3 Merkle batch notarization

High-volume use cases (legal discovery, audit trails, software releases) require stamping hundreds or thousands of files in a single operation. Batch mode will use the Merkle tree helper already present in worker/merkle.js:

- Client computes commitments $C_1 \ldots C_n$ for all files locally.
- Commitments are arranged as leaves of a binary Merkle tree; the root hash `Mroot` is sent to the API.
- A single AccountSet transaction anchors `Mroot` on-chain.
- Each file receives an individual Merkle proof (a $\log_2(n)$ path) that allows independent verification against `Mroot` without knowing the other files.

Batch mode reduces on-chain cost to 1 XRP per batch regardless of file count, and removes the bottleneck of one transaction per file under XRPL's per-account sequence number constraint.

6.4 Account delete reclamation — recovering the base reserve

Every funded XRPL account carries a base reserve (currently 1 XRP at network parameters). Users who create or fund a dedicated XRPL account for timestamping purposes may wish to reclaim that reserve once their work is complete. This is entirely under the user's control: the service does not touch funds held in user-owned accounts.

The reclamation flow charges a 0.2 XRP service fee, after which the user retains 0.799988 XRP available to recover from their XRPL account:

18. The user requests reclamation via the dashboard and pays the 0.2 XRP service charge.
19. The user ensures all pending timestamps on the account are complete and no escrows or offers remain open.
20. The user submits an `AccountDelete` transaction from their XRPL wallet, designating any destination address for the returned balance.
21. The XRP Ledger enforces that the account's sequence number must have advanced at least 256 beyond the current ledger sequence before deletion is permitted — a protocol-level anti-spam measure.
22. Upon successful deletion, the remaining balance (0.799988 XRP after the service charge, minus the on-chain AccountDelete fee) is transferred to the destination.

The transaction is signed and submitted by the user's own wallet — no third-party custody of the reserve is required at any point.

> ⚠ *The XRPL AccountDelete fee is set by network consensus. Users should verify current fee levels before initiating deletion to confirm the net reclaimed amount.*

6.5 Verifier CLI and open-source SDK

To eliminate any remaining dependency on xrpltimestamp.com, a standalone verifier will be published:

- `xrplt verify <file> <certificate.json>` — full seven-step verification in one command.
- JavaScript/TypeScript SDK for embedding verification in third-party applications.

- Python library for integration with document management and legal tech systems.

All verification tooling will be released as open source (MIT license).

6.6 API key access for developers

A token-based API will allow developers to integrate timestamping into their own pipelines without managing XRPL infrastructure. Rate limits and billing will be per API key; commitments remain as private from the service as in the browser flow.

7. Economic Model

Service fee	1 XRP per timestamp (single file or Merkle batch root)
On-chain fee	12 drops (0.000012 XRP) per AccountSet transaction
Settlement time	≈ 4 seconds (one XRPL ledger close cycle)
Account delete fee	0.2 XRP service charge; user retains 0.799988 XRP for reclamation
Batch efficiency	n files for 1 XRP; Merkle proofs allow per-file independent verification

At 1 XRP per timestamp, the service is priced at a predictable, low-friction amount settled in approximately four seconds. The dominant on-chain cost is the 12-drop transaction fee — negligible relative to the service charge. For account reclamation (§6.4), a separate 0.2 XRP service charge applies, leaving the user with 0.799988 XRP to recover from their XRPL account.

8. Legal & Compliance Considerations

Blockchain timestamps are increasingly recognised in legal contexts. The eIDAS 2.0 regulation in the European Union explicitly addresses electronic timestamps. XRPL Timestamp does not itself constitute a qualified electronic timestamp authority under eIDAS; however, the cryptographic proof it produces can serve as corroborating evidence of prior existence in civil disputes.

Users operating in regulated industries (finance, healthcare, legal) should consult counsel regarding the evidentiary weight of XRPL-anchored proofs in their jurisdiction. The service provides the cryptographic infrastructure; the legal interpretation remains with the user.

Because the file never leaves the user's device, XRPL Timestamp does not process personal data on behalf of the user, simplifying GDPR compliance for the operator. IP addresses retained for fraud prevention fall under legitimate interest and are subject to the operator's privacy policy.

9. Security Considerations

9.1 Nonce quality

The nonce is generated using the Web Crypto API (crypto.getRandomValues), which provides cryptographically secure pseudorandom bytes in all modern browsers. A 128-bit nonce ensures that even if two users timestamp the same file, their commitments are statistically certain to differ.

9.2 Re-submission attacks

An adversary who intercepts a commitment cannot replay it for a different file: the commitment binds the file hash and nonce together. Changing either value invalidates the commitment.

9.3 Worker process hardening

The worker process runs as a non-root system user. The .env file containing the private key has 600 permissions. The worker does not expose any network interface; it communicates only with MySQL (loopback) and the XRPL WebSocket endpoint. Dependency updates are managed via npm audit in CI.

9.4 XRPL account management

The signing account does not hold significant XRP beyond the base reserve and a small operational buffer. Large XRP balances are held in a cold wallet and topped up manually. Future escrow flows will further reduce the hot-wallet surface.

10. Conclusion

XRPL Timestamp demonstrates that blockchain notarization can be simultaneously privacy-preserving, self-verifiable, and economically accessible. By computing the cryptographic commitment entirely in the browser, the service achieves a genuine zero-knowledge property with respect to file contents — not merely a policy claim.

The roadmap extensions — user accounts, multi-party escrow, Merkle batching, and reserve reclamation — address the natural requirements of higher-volume, higher-trust, and higher-automation use cases without compromising the core privacy model.

All future development will maintain the principle that a printed certificate remains self-sufficient: no internet connection, no dependency on xrpltimestamp.com, no trust in any third party is required to verify a proof after it has been anchored on the XRP Ledger.

xrpltimestamp.com
info@xrpltimestamp.com

Cryptographic proof of existence · Anchored on the XRP Ledger

White Paper v1.0 · April 2026