Live Verify: Verifying Printed and Displayed Text Against Issuer Domains by Canonicalized Hashing*

Paul Hammant

Draft of August 29, 2026

Abstract

Documents that circulate as paper, PDF, or on-screen text routinely outlive any channel that could vouch for them: a determination letter is photographed, a license is framed on a wall, a bank statement is forwarded. Live Verify is a deliberately low-fidelity bridge from such artifacts back to the party that issued them. A document carries a human-readable `verify:` line naming the issuer's domain; a verifier's client canonicalizes the visible text, hashes it with SHA-256, and performs an HTTPS GET for that hash at the issuer's domain. A 200 response with an affirming status means the issuer, *now*, stands behind exactly that text. The design trades cryptographic evidence for radical deployability: there are no signatures, no key ceremonies, and nothing to install on the document side beyond one printable line. We specify the canonicalization pipeline that makes camera-captured and clipboard-captured text hash identically across implementations, publish a conformance corpus of hash-pinned vectors that three implementations pass, and give an honest account of the trust model's limits: verification is a live confirmation, not transferable proof; it is deniable by deletion; and public hash endpoints act as confirmation oracles for guessable documents, which constrains the class of documents the scheme should carry.

1 Introduction

Institutions already know how to publish facts on their own domains, and the web's public-key infrastructure already binds those domains to organizational identity well enough for banking. What is missing is a bridge from a *document in hand* — paper, screenshot, forwarded PDF — to that issuer-controlled surface. Existing bridges either carry opaque payloads (QR codes, NFC chips) that a human cannot audit and a printer cannot always reproduce, or demand heavyweight adoption (digital signatures, verifiable credentials) from issuers whose documents are still fundamentally text.

Live Verify's premise is that the document's own visible text can be the payload. The scheme adds one line:

```
verify:example.com/path
```

A verifying client captures the text above that line by camera or clipboard, canonicalizes it (§3.2), hashes it, and asks the named domain whether it stands behind that hash. The answer is rendered against the *authority domain* — the domain printed on the document — never against whichever host happened to serve bytes.

*Draft. Reference implementations, the normative specification, and the conformance corpus are at `https://github.com/live-verify/live-verify`.

Contributions.

1. A protocol for text-first document verification whose entire issuer-side deployment is static file hosting (§3).
2. A canonicalization pipeline reconciling OCR capture and digital clipboard capture into one hash space, with the order of operations specified normatively (§3.2).
3. A published, append-only conformance corpus in which each vector's filename *is* its pinned SHA-256, and cross-implementation results over it (§5).
4. An explicit negative result offered as a design position: what a live-lookup scheme *cannot* provide (non-repudiation, transferable proof, resistance to deletion), and why naming those limits is a precondition for deploying it safely (§6).

2 Related Work

Domain-anchored transparency. Certificate Transparency [3] demonstrated that append-only, publicly auditable logs can discipline an existing trust ecosystem without redesigning it. Live Verify shares the instinct of anchoring trust in operational domain control rather than new key material, but deliberately omits the log: the issuer's live endpoint is the (revocable, deniable) source of truth, with third-party witnessing left as an optional layer.

Hash-prefix privacy. Have I Been Pwned's k-anonymity range queries [1] showed that a hash-lookup service can avoid learning which exact record was queried. Live Verify's baseline lookup is a full-hash GET and therefore leaks the queried hash to the issuer; adopting prefix range queries is designed but unimplemented (§6).

Verifiable credentials. The W3C VC model [7] carries cryptographic proof inside the credential, achieving non-repudiation and offline verification at the cost of issuer-side signing infrastructure and holder-side wallets. Live Verify occupies the opposite corner of the design space: nothing travels with the document but text.

Provenance grading. The Isnād–Rijāl framework [5] grades chains of transmission in multi-agent knowledge systems, importing classical hadith methodology. Its concerns are complementary: Live Verify authenticates a leaf artifact against its issuer, while isnād-style grading evaluates the chain through which derived knowledge traveled. The overlap and its limits are examined in the repository's comparative documents.

3 Protocol Design

3.1 The verify line

The verify line is the last content line of a claim, in ABNF [2]:

```
verify-line = scheme ":" verify-target
scheme      = "verify" / "vfy"
```

Recognition is tolerant (case-insensitive, whitespace around the colon, bottom-up scan) because the line must survive OCR. A verify line immediately preceded by `>` is *quoted* — a claim reproduced inside another claim — and is inert; OCR artifact cleanup deliberately does not strip `>` so that quotation survives the camera path.

3.2 Canonicalization

Camera capture and clipboard capture of the same document must reach the same bytes. A conforming implementation applies, in order: (1) OCR artifact cleanup, camera path only — stripping border characters and scan debris; (2) Unicode canonical composition (NFC) [6], since precomposed and decomposed encodings of the same glyph otherwise hash differently; (3) issuer-declared single-character folds; (4) issuer-declared OCR regex rules, treated as untrusted input and guarded; (5) a fixed punctuation fold table (curly quotes, en/em dashes, no-break space, ellipsis); (6) line processing — trim, collapse internal whitespace, drop blank lines, rejoin with LF, no trailing newline. The hash is SHA-256 [4] over the UTF-8 encoding of the result, rendered as 64 lowercase hex characters.

3.3 Lookup and affirmation

The client GETs `https://host[/path]/{hash}` — HTTPS unconditionally, with a parsed-hostname (never substring) exception for `localhost` development. A claim is verified iff the response is HTTP 200 and its status affirms, either as JSON `status:"verified"` or via an issuer-declared response-type whose class is `affirming`. Responses never echo document content: the verifier already holds the document, and an endpoint that repeats content is a data leak.

3.4 Endorsement

An issuer's metadata may name an endorser. The endorsement commits to a hash of the issuer's *entire* metadata file, so any change to the issuer's self-description invalidates it; clients walk at most three links and render unendorsed domains as self-verified with an explicit caution.

4 Trust Model

Verification here means: *the issuer's domain, at this moment, affirms this exact text.* That is a live confirmation with the strength — and only the strength — of WebPKI domain authentication plus the issuer's operational integrity. The repository's security document treats each threat in turn (query privacy, transport and DNS posture, issuer-supplied normalization, spoofing); §6 summarizes what survives honest scrutiny.

5 Evaluation

5.1 Cross-implementation hash determinism

The conformance corpus currently contains 17 vectors: 8 text vectors (each filename a pinned SHA-256) and 9 image vectors exercising the full camera pipeline on Android (ML Kit) and iOS (Vision). The reference JavaScript implementation passes 8/8 text vectors; the table in

Appendix A is *generated by executing the implementation against the corpus at paper build time*, so the numbers cannot drift from the code. Vectors are discriminating by design — the NFC vector's body is decomposed Unicode, and an implementation that skips canonical composition fails it.

5.2 OCR field failure study [PLANNED]

Which characters do production OCR stacks actually confuse on real documents (e-ink, laser print, screens), and does the issuer-fold mechanism cover the observed failure classes? This study is planned; anecdotal evidence from development (e.g. e-ink capture) motivates it but is not evidence.

5.3 Confirmation-oracle entropy analysis [PLANNED]

For which document classes is the plaintext guessable enough that a public hash endpoint becomes a confirmation oracle? A quantified entropy analysis across the use-case corpus is planned; the security document currently handles this qualitatively (high-entropy fields, optional salt).

6 Limitations

Live Verify's verification is not evidence. There is no signature, hence no non-repudiation: an issuer can delete a hash and deny ever having affirmed it. Confirmation is not transferable: a verifier cannot prove to a third party that verification succeeded. The full-hash lookup leaks query patterns to issuers; k-anonymity range queries [1] and OPRF-based lookups are designed but not implemented. An HTTP 404 is both "retry after network trouble" and "the issuer denies this document"; treating a security signal as retryable trains verifiers past it, and client design must separate the two. These properties bound the scheme's honest use: it complements, and does not replace, signature-bearing schemes where evidence must outlive the issuer's cooperation.

7 Conclusion

Live Verify makes one narrow bet: for documents that are already text, the cheapest verification bridge that institutions will actually deploy is a printable line, a canonicalization discipline, and static file hosting on a domain they already defend. The specification, three reference implementations, and a hash-pinned conformance corpus are published; the two planned studies above are the empirical work that remains.

References

[1] Junade Ali. Validating leaked passwords with k-anonymity. Cloudflare blog; the range-query protocol deployed by Have I Been Pwned, 2018.

[2] Dave Crocker and Paul Overell. Augmented BNF for syntax specifications: ABNF. RFC 5234, IETF, 2008.

[3] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. RFC 6962, IETF, 2013.

[4] National Institute of Standards and Technology. Secure hash standard (SHS). FIPS PUB 180-4, 2015.

[5] Ali Zahid Raja. Grading the narrators: An isnad-rijal framework for claim-level provenance in multi-agent knowledge systems. arXiv:2607.24117, 2026.

[6] The Unicode Consortium. Unicode standard annex #15: Unicode normalization forms. UAX #15.

[7] W3C. Verifiable credentials data model v2.0. W3C Recommendation, 2025.

A Conformance vectors (generated)

Vector (description)	**Pinned hash (prefix)**	**JS**
Bachelor of Thaumatology training page certificate	`1cddfbb2adfa`	✓
Curly double quotes normalized to straight quotes	`30c6d7a77f52`	✓
NFC canonicalization - body uses decomposed Unicode (base letters + combining accents), curly quotes, ellipsis and em-dash; implementations must apply NFC then the standard folds before hashing, or this vector fails	`4970a05e322a`	✓
En-dash and em-dash normalized to hyphen	`50093137b7cc`	✓
Master of Applied Anthropics training page certificate	`6725b845dcdf`	✓
Simple text - baseline test	`a591a6d40bf4`	✓
Document-specific character normalization (diacritics)	`ea9837801631`	✓
Whitespace normalization - leading, trailing, multiple spaces, blank lines	`eb4108396033`	✓