Crypto Forum
Internet-Draft
Intended status: Informational
Expires: 3 September 2026

X. Ma
M. Honda
University of Edinburgh
C. Perkins
University of Glasgow
2 March 2026

Looma: Low-Latency Post-Quantum Authentication for TLS 1.3 in Datacenters
draft-ma-cfrg-looma-00

Abstract

Post-quantum (PQ) authentication in TLS 1.3 can add tens to hundreds of microseconds of handshake processing time. In datacenters, where mutual authentication is mandatory, this authentication cost becomes a dominant contributor to end-to-end request latency, particularly when connections are short-lived and handshake rates are high.

This document specifies Looma, an online/offline authentication architecture integrated into the TLS 1.3 handshake. Looma replaces the on-path, per-handshake PQ signature with a fast, one-time signature over the TLS transcript and moves expensive work (including the multi-use PQ signature) to an asynchronous background plane. Looma includes a fallback strategy to preserve correct authentication when the verifier does not have the peer’s one-time verification key cached.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Crypto Forum Research Group mailing list (cfrg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/cfrg.

Source for this draft and an issue tracker can be found at https://github.com/uoenoplab/looma-id.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 3 September 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

2. Introduction

TLS 1.3 authentication relies on digital signatures over the handshake transcript in the CertificateVerify message [RFC8446]. For post-quantum migration, widely deployed candidates (e.g., Dilithium and Falcon) incur higher signing costs than classical signatures.

In cloud environments these costs become a serious deployment barrier. Modern cloud applications are built from microservices and serverless functions. Each inter-service RPC triggers a fresh TLS handshake. Containers and pods are frequently created and destroyed, rendering session resumption ineffective. Service-mesh sidecars (Istio, Linkerd, etc.) add extra mTLS hops along every path. The resulting handshake rate is orders of magnitude higher than on the public Internet.

Datacenter networks are engineered for sub-50 s fabric latency; therefore the dominant delay is host cryptographic processing. Mutual authentication (mTLS) is mandatory inside the trust boundary to prevent unauthorized service-to-service access. In mTLS both endpoints sign and verify, doubling the authentication cost compared with server-only TLS. When post-quantum signatures are used, authentication can consume 5470 % of total handshake latency (see [LoomaNDSS26] for measurements).

Existing accelerations do not close this gap:

* Protocol optimisations (KEMTLS) replaces signatures with KEMs, reducing the bandwidth cost during handshake, which is especially helpful in the Internet where RTT dominates the latency. However, the on-path KEM-based authentication time cost is still tens of microseconds. Besides, for mTLS case, KEMTLS requires extra round-trip time.

* Cryptographic optimisations focus primarily on PQ key exchange; authentication remains the bottleneck for mTLS.

Looma targets this setting by splitting PQ authentication into:

* *Foreground plane (on-path)*: performs per-handshake fast signing and verification.

* *Background plane (off-path)*: precomputes and distributes one-time verification (i.e., public) keys authenticated by a long-term PQ signature.

The Looma design and evaluation appear in [LoomaNDSS26], and its proof-of-concept implementation is available at https://github.com/uoenoplab/looma.

This document is an initial draft submitted for discussion and to allow review of the use of cryptographic techniques by CFRG and the broader protocol design by the TLS community. The authors welcome feedback on both the use of post quantum cryptographic primitives, on their integration into TLS, and more broadly on the design goals

2.1. Goals

Looma is designed to:

* Reduce the on-path computational cost of PQTLS 1.3 authentication.

* Preserve the authentication semantics of PQTLS 1.3 (identity bound to certificate keys).

* Minimize additional trust in auxiliary infrastructure (e.g., key distribution service).

2.2. Non-goals

This document does not specify:

* A new certificate format. Looma uses existing X.509 certificates and CA workflows.

* A new key exchange. Looma is orthogonal to ECDHE, hybrid key exchange, or KEM-based approaches.

* Session resumption behavior. PSK resumption handshakes do not use CertificateVerify and are out of scope.

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the following terms:

* *Endpoint*: a PQTLS client or server participating in the handshake.

* *Long-term signing key (LT key)*: a multi-use PQ key pair (e.g., Dilithium-2). The LT public key is bound to the endpoint identity via an X.509 certificate.

* *One-time signature (OTS)*: a signature scheme intended to sign at most one message per key pair. Looma uses WOTS+.

* *WOTS+ key pair*: an OTS signing key SK_ots and verification key PK_ots.

* *KeyDist*: a repository service used to publish OTS verification keys authenticated under the LT key. KeyDist is not trusted for integrity; endpoints verify LT signatures.

* *Cache hit / miss*: whether the verifier has a validated PK_ots for the peer at handshake time.

4. Overview

Looma adapts the Even-Goldreich-Micali online/offline paradigm to TLS 1.3 authentication:

1. Offline: an endpoint generates a batch of WOTS+ key pairs and authenticates them using a long-term PQ signature over a Merkle tree root;

2. Online: during the TLS handshake, the endpoint signs the TLS CertificateVerify input using a fresh WOTS+ key SK_ots. The verifier validates the WOTS+ signature using a cached PK_ots or falls back to an authenticated Merkle proof.

Since this online/offline paradigm only works when the verifier has the right verification key, the fallback strategies are required when there is no validate key at the server side. Two fallback deployment modes are supported:

* *Dual-signature mode*: each handshake carries enough data for verification without any prior cached verification key (higher bandwidth, simple behavior logic).

* *Hybrid mode*: the server advertises a compact hint indicating which client keys are cached; the client includes fallback material only on cache miss (lower bandwidth).

5. Cryptographic Building Blocks

5.1. Long-term PQ signature

Looma assumes a non-hash-based secure multi-use signature scheme (e.g., Dilithium-2, Falcon-512). The LT public key is authenticated in the usual TLS way via X.509 certificate validation. It will also be authenticated during OTS verification key distribution.

This document describes one instantiation aligned with [LoomaNDSS26]: Dilithium-2 as the LT signature scheme.

5.2. WOTS+ (Winternitz One-Time Signature Plus)

Looma uses WOTS+ as the OTS. WOTS+ keys are one-time: an endpoint MUST NOT sign more than one distinct CertificateVerify input with the same SK_ots.

5.2.1. Parameters

To avoid ambiguity, this document defines a single mandatory parameter set (other sets MAY be specified by future documents).

The mandatory set matches the Looma implementation evaluated in [LoomaNDSS26]:

* n = 32 bytes (hash output length).

* Winternitz parameter w = 4.

* = 133 chains (derived by WOTS+ encoding for n=32, w=4).

The hash function used inside WOTS+ is denoted H_ots. This document does not require a specific H_ots at the protocol level; however, the reference implementation uses Haraka (and a SIMD variant) for performance as described in [LoomaNDSS26].

5.2.2. Nonce

The WOTS+ signature is computed over the TLS CertificateVerify input (see Section 4.4.3 of [RFC8446]) and an associated nonce r. The nonce serves as domain separation and helps avoid accidental cross-protocol reuse. The nonce MUST be unique per WOTS+ signing key.

6. Looma Authentication Architecture

6.1. Key Provisioning and Authentication (Background Plane)

This section defines the data produced off-path and consumed by the handshake.

6.1.1. Key records

An endpoint periodically produces a *key record* containing a batch of OTS verification keys. A key record MUST include:

* owner_id: an identifier for the endpoint identity that is consistent with the endpoint certificate identity (see Section 6.1.1.1).

* epoch: a monotonically increasing identifier scoped to (owner_id, verifier_group_id).

* verifier_group_id: a locally defined label identifying which peer set this batch targets.

* valid_from, valid_to: validity interval for the OTS keys in this record.

* merkle_root: root of the Merkle tree over OTS public keys.

* sig_lt: LT signature authenticating the key record metadata and merkle_root.

* pk_list (OPTIONAL): list of OTS public keys as the distribution mechanism delivers full keys; this document assumes full keys are distributable but does not mandate KeyDist internal format.

Key record authentication:

* The producer MUST compute sig_lt = Sign_lt(H(record_fields), SK_lt), where record_fields includes the items above, in a canonical encoding.

* A consumer MUST verify sig_lt under the LT public key from the producer's validated certificate, before accepting any OTS keys derived from the record.

6.1.1.1. owner-id

owner_id is used as a stable index for caching the peer's OTS keys. owner_id MUST be bound to the peer identity authenticated by TLS certificate validation.

RECOMMENDED: owner_id = SHA-256(SPKI) where SPKI is the DER-encoded SubjectPublicKeyInfo of the LT public key in the certificate. Other constructions MAY be used if they provide a collision-resistant, stable binding to the LT key.

6.1.2. Merkle tree construction

An endpoint constructs a binary Merkle tree over the batch of OTS public keys. The leaf value for key i is:

leaf_i = H_leaf(encode(PK_ots_i))

where H_leaf is a collision-resistant hash and encode(PK_ots_i) is the canonical encoding of the OTS public key for the parameter set.

The tree root is merkle_root. The record MUST allow a verifier to validate an inclusion proof.

Implementation note (informative): [LoomaNDSS26] evaluates B = 1024 leaves and 32-byte hash values, yielding a proof of log2(B) * 32 bytes (320 bytes).

6.1.3. KeyDist behavior

KeyDist is a distribution component that stores and serves key records. This document treats KeyDist as a repository without trusting it:

* KeyDist MAY validate records before storing them, but consumers MUST NOT rely on KeyDist validation for security.

* Consumers MUST verify sig_lt and associated metadata locally.

* A compromised KeyDist can cause denial of service (e.g., withholding records), but MUST NOT enable signature forgery if endpoints follow this specification.

6.2. TLS 1.3 Integration (Foreground Plane)

This section specifies how Looma is carried in TLS 1.3 and how the CertificateVerify message is generated and verified.

6.2.1. Looma negotiation

Looma is negotiated using the TLS 1.3 signature_algorithms extension.

* Endpoints that support Looma MUST advertise Looma signature scheme in signature_algorithms.

* The server selects a Looma scheme in the usual TLS 1.3 way when producing CertificateVerify.

* Hybrid mode additionally uses a server-to-client hint extension (Section Section 7).

If Looma is not negotiated, endpoints MUST use standard TLS 1.3 authentication with the selected non-Looma signature scheme.

6.2.2. Looma signature format

Looma defines a signature wrapper LoomaSignature carried in the CertificateVerify.signature field.

All multi-byte integers are network byte order.

```
struct {
  uint8  looma_sig_type;    /* 0 = dual, 1 = hybrid_hit,
                             * 2 = hybrid_miss */
  opaque owner_id<0..255>;  /* as defined in Section {#owner-id} */
  opaque pk_id<0..255>;     /* identifies which OTS key is used */
  opaque nonce<0..255>;     /* per-signature nonce r */
  opaque wots_sig<0..2^16-1>;
} LoomaSignature;
```

pk_id identifies the one-time public key within the producer's current key record. The mapping of pk_id to a concrete key is deployment-specific, but pk_id MUST be sufficient for the verifier to locate the intended cached key or validate it via fallback data.

6.2.2.1. Dual-signature mode (looma_sig_type = 0)

In dual-signature mode, besides LoomaSignature, fallback MUST contain:

```
struct {
  opaque merkle_root<0..2^16-1>;
  opaque merkle_proof<0..2^16-1>;
  opaque sig_lt<0..2^16-1>;
} LoomaFallbackDual;
```

fallback is carried in the CertificateVerify extension filed. The verifier uses the peer certificate to obtain PK_lt and verify sig_lt over the authenticated root and associated record fields (the record fields MUST be reconstructible or transmitted as part of the proof bundle in a future version; see Section Section 10).

6.2.2.2. Hybrid-hit mode (looma_sig_type = 1)

In hybrid-hit mode, fallback MUST be empty. The verifier MUST have a cached, validated PK_ots for (owner_id, pk_id).

6.2.2.3. Hybrid-miss mode (looma_sig_type = 2)

In hybrid-miss mode, fallback is identical to LoomaFallbackDual.

6.2.3. Computing CertificateVerify

TLS 1.3 defines the signed content for CertificateVerify as:

* a context string,

* a separator,

* and the transcript hash (see Section 4.4.3 of [RFC8446]).

For Looma, the endpoint MUST compute the TLS-defined input bytes exactly as [RFC8446] specifies. Let that byte string be cv_input.

The endpoint then computes:

* wots_sig = WOTS.Sign(cv_input, SK_ots, nonce)

and constructs LoomaSignature per the negotiated mode.

The endpoint MUST ensure one-time use of SK_ots and MUST bind the WOTS+ signature to the selected mode, owner_id, pk_id, and nonce (e.g., by including these fields in cv_input via an inner hash) to prevent substitution attacks. One safe construction is:

wots_sig = WOTS.Sign(H_bind(owner_id || pk_id || nonce || cv_input), SK_ots)

where H_bind is collision-resistant. The reference design in [LoomaNDSS26] signs the transcript and carries (nonce, pk_id) alongside the signature; this document adopts explicit binding.

6.2.4. Verifying CertificateVerify

Given cv_input and a received LoomaSignature, the verifier proceeds as follows:

1. Determine the expected_PK_ots:
    * If in hybrid-hit mode, look up cached PK_ots using (owner_id, pk_id).
    * Otherwise, validate fallback data:
        - verify sig_lt under the peer certificate LT public key,
2. Verify wots_sig over the bound input described above and derived the computed_PK_ots.
3. Enforce one-time use:
    * If in hybrid-hit mode, check if computed_PK_ots is the same as expected_PK_ots, and delete cached expected_PK_ots if there is a match.
    * Otherwise, validate that the computed_PK_ots is included in merkle_root via merkle_proof.

The verifier MUST ensure that (owner_id, pk_id) is not accepted more than once within the applicable validity interval, unless the deployment defines a safe reuse policy (NOT RECOMMENDED).

If any check fails, the handshake MUST be aborted with an appropriate TLS alert.

7. Hybrid Mode Cache Hint Extension

In hybrid mode, the server sends a compact hint indicating which client keys are cached, so that the client can select looma signature type as hybrid_hit vs hybrid-miss.

This document defines a new TLS extension (type TBD) carried in the CertificateRequest message.

```
struct {
  uint8  version;
  opaque owner_id_s<0..255>;  /* server owner_id to scope the hint */
  opaque bloom<0..2^16-1>;    /* Bloom filter bitstring */
  uint8  k;                   /* number of hash probes */
} LoomaCacheHint;
```

The Bloom filter represents membership over client identifiers (e.g., client owner_id values) for which the server currently has a list of cached PK_otss. The exact element inserted MUST be specified by the deployment; RECOMMENDED is the client’s owner_id.

False positives are possible. If the server indicates membership but does not have the key, the server will reject hybrid-hit verification and the connection will fail. See Section Section 7.1.2 for recovery behavior.

7.1. Fallback and Error Handling

7.1.1. Cache miss behavior

If the verifier does not have the peer PK_ots cached:

* In dual-signature mode, the verifier uses the fallback bundle.

* In hybrid mode:

  - If the client receives a hint indicating miss, it MUST send hybrid-miss (with fallback).

  - If the client receives a hint indicating hit, it sends hybrid-hit (without fallback).

7.1.2. False positives and recovery

If a hybrid-hit handshake fails because the server cannot validate the signature due to missing cached key (e.g., Bloom false positive), endpoints SHOULD recover by retrying with a full (non-Looma) TLS 1.3 handshake using standard PQ authentication (e.g., Dilithium-2), or by retrying hybrid-miss if policy allows.

This document does not define a new TLS alert. Implementations SHOULD use existing alerts such as handshake_failure or illegal_parameter to signal verification failure, consistent with [RFC8446] behavior.

8. Security Considerations

This section summarizes security arguments aligned with [LoomaNDSS26].

8.1. Signature unforgeability

An adversary that forges a Looma CertificateVerify must either:

* forge the LT PQ signature authenticating merkle_root, or
* forge a WOTS+ signature under an authenticated one-time public key, or
* break collision resistance / second-preimage resistance needed by the Merkle tree and WOTS+.

Assuming EUF-CMA security of the LT PQ signature and the security properties of WOTS+ and underlying hash functions, Looma provides transcript authentication comparable to the baseline TLS 1.3 signature scheme, with the additional requirement that one-time keys are not reused.

8.2. One-time key reuse

Reusing a WOTS+ signing key across two different cv_input values can enable forgeries. Therefore:

* Signers MUST enforce strict one-time use.
* Verifiers SHOULD track used (owner_id, pk_id) values within the key validity interval.
* Deployments MUST provision enough keys for the expected handshake rate and set key expiration to bound state.

8.3. Binding to certificate identity

Looma’s OTS keys MUST be bound to the identity authenticated by the certificate. This document achieves binding by (a) authenticating merkle_root under the LT key from the certificate, and (b) scoping caches by owner_id derived from that LT key.

8.4. KeyDist compromise

A malicious KeyDist can withhold, replay, or reorder key records, causing denial of service or performance degradation (e.g., more cache misses), but MUST NOT enable authentication forgery if endpoints perform local verification and enforce freshness/epoch rules.

Deployments SHOULD consider availability hardening (replication, multi-source fetching) and should ensure that stale records do not cause key reuse.

8.5. Bloom filter privacy and integrity

The cache hint can leak information about which client identifiers are cached by the server. Deployments concerned with this leakage SHOULD avoid hybrid mode or should scope hints to a small set of expected peers.

An attacker who can modify the hint on-path can influence whether the client sends fallback material. However, the hint is carried inside the TLS handshake and is integrity protected by the handshake transcript; modifying it will fail the handshake.

9. IANA Considerations

This document anticipates the need for the following registries:

* One or more entries in the TLS SignatureScheme registry for Looma signature schemes.

* One entry in the TLS ExtensionType registry for looma_cache_hint.

This draft does not yet request code points and uses "TBD" placeholders.

10. Open Issues and Future Work

* Canonical encoding of key records and exact contents covered by sig_lt.

* Precise definition of pk_id mapping and whether it is stable across epochs.

* Whether to standardize a specific H_ots for interoperability, or define a registry.

* Interaction with delegated credentials [RFC9345] and deployment in service meshes.

Besides those specific points, we seek feedback on the appropriate use of PQ cryptographic primitives from CFRG, and from the broader TLS community on TLS handshake integration.

11. Acknowledgements

The authors thank the CFRG community for feedback.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

12.2. Informative References

[LoomaNDSS26] Ma, X. and M. Honda, "Looma: A Low-Latency PQTLS Authentication Architecture for Cloud Applications", In Network and Distributed System Security (NDSS) Symposium, 2026, <https://dx.doi.org/10.14722/ndss.2026.240074>.

[RFC9345] Barnes, R., Iyengar, S., Sullivan, N., and E. Rescorla, "Delegated Credentials for TLS and DTLS", RFC 9345, DOI 10.17487/RFC9345, July 2023, <https://www.rfc-editor.org/rfc/rfc9345>.

Authors' Addresses

Xinshu Ma
University of Edinburgh
Email: x.ma@ed.ac.uk

Michio Honda
University of Edinburgh
Email: michio.honda@ed.ac.uk

Colin Perkins
University of Glasgow
Email: csp@csperkins.org