OAuth
Internet-Draft
Intended status: Standards Track
Expires: 20 September 2026

R. Krishnan
JPMorgan Chase & Co
A. Prasad
Oracle
D. Lopez
Telefonica
S. Addepalli
Aryaka

19 March 2026

TLS-Session-Bound Access Tokens for OAuth 2.0
draft-mw-oauth-tls-session-bound-tokens-00

Abstract

This document defines a mechanism for binding OAuth 2.0 access tokens to a specific mutual TLS (mTLS) session. The binding is achieved through a per-request proof token that incorporates the TLS Exporter value {{!RFC5705}} derived from the current session and an access token hash, signed by the client's private key corresponding to its mTLS certificate. This mechanism prevents stolen bearer tokens from being replayed on a different TLS connection. While applicable to any OAuth 2.0 access token presented over mTLS, this specification is primarily motivated by the Token Exchange protocol {{!RFC8693}}, where multi-hop delegation chains in autonomous, agent-driven architectures create elevated replay risk.

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 20 September 2026.

Table of Contents

1. Introduction

1.1. The Bearer Token Replay Problem

OAuth 2.0 access tokens are typically *bearer tokens*: any party in possession of the token can use it to access protected resources, regardless of the presenter's identity or the communication channel. This is a known risk addressed by the OAuth 2.0 Security Best Current Practice {{!I-D.ietf-oauth-security-topics}}.

The Token Exchange protocol {{!RFC8693}} amplifies this risk by enabling chained delegation across service boundaries. Each exchange produces a new bearer token, and a compromise at any point in the chain exposes downstream tokens.

Existing mitigations address parts of this problem:

* *RFC 8705 (mTLS Certificate-Bound Tokens)* {{!RFC8705}}: Binds the token to the client's X.509 certificate thumbprint. However, the binding is to the _certificate identity_, not the _TLS connection_. If the same certificate is used across connections, or if the certificate and token are both exfiltrated, the token remains replayable.
* *RFC 9449 (DPoP)* {{!RFC9449}}: Provides application-layer proof-of-possession using ephemeral, application-managed keys. Applicable to both public and confidential clients, but binds to the key, not to the TLS channel.
* *Token Binding (RFC 8471-8473)*: Proposed direct TLS session binding but required a new TLS extension, was never specified for Token Exchange, encountered adoption barriers in browsers and TLS 1.3 transitions, and was ultimately abandoned.

None of these mechanisms provide *TLS-connection-level binding* for OAuth 2.0 access tokens in mTLS environments.

1.2. The Agentic AI Amplifier

The rise of autonomous AI agents dramatically amplifies the bearer token replay risk. Unlike traditional OAuth flows where a human initiates discrete requests, agentic AI systems:

* *Autonomously chain API calls*: An agent may invoke hundreds of API calls across multiple services without human intervention, each potentially involving RFC 8693 token exchanges.
* *Perform multi-hop delegation*: Agent A exchanges its token for a delegated token to call Agent B, which exchanges again for Agent C. Each hop produces a new bearer token. A single compromise anywhere in this chain exposes all downstream tokens.
* *Run for extended periods*: Agents operate autonomously for hours or days, providing a broad window for token interception and replay.
* *Generate opaque traffic*: Agent-to-agent API calls are fully automated. A replayed token produces legitimate-looking traffic that is extremely difficult to distinguish from genuine requests—there is no human in the loop to detect anomalies.
* *Are susceptible to prompt injection*: A compromised or prompt-injected LLM agent can exfiltrate bearer tokens via tool calls, side channels, or log leakage. These tokens are immediately usable from any connection.

These characteristics make bearer token replay a *first-order threat* in agentic AI architectures. This document addresses this gap by binding access tokens to the mTLS connection on which they are presented.

1.3. Conventions and 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:

TLS Exporter Value: A value derived from the TLS handshake using the mechanism defined in {{!RFC5705}} (for TLS 1.2) or Section 7.5 of {{!RFC8446}} (for TLS 1.3). The exporter value is unique to the specific TLS connection and is available to both endpoints.

Session-Binding Proof: A signed JWT presented alongside the access token that cryptographically binds the token to the current mTLS session via the TLS Exporter value.

Token Exchange: The protocol defined in {{!RFC8693}} for exchanging

one security token for another at an authorization server.

2. TLS Session Binding for Access Tokens

2.1. Overview

This specification defines a proof-of-possession mechanism that binds OAuth 2.0 access tokens to the mTLS connection on which they are presented. While applicable to any OAuth 2.0 access token, it is primarily designed for tokens issued via the Token Exchange protocol {{!RFC8693}}, where multi-hop delegation creates elevated replay risk. The mechanism operates as follows:

1. The client and resource server establish an mTLS connection. Both sides derive a TLS Exporter value unique to this connection.
2. When presenting an access token, the client constructs a *Session-Binding Proof*: a JWT containing the hash of the access token, the TLS Exporter value, and the HTTP method and URI of the request.
3. The client signs this JWT with the private key corresponding to its mTLS client certificate.
4. The resource server verifies the proof by checking the signature against the client certificate’s public key, confirming the exporter value matches the current connection, verifying the issuance time, and confirming the token hash matches the presented access token.

A token that requires session binding includes a confirmation method claim (tls_exp) containing the TLS Exporter label, which signals to the resource server that the Session-Binding Proof MUST be presented and verified.

2.2. TLS Exporter Derivation

Both the client and resource server MUST derive the TLS Exporter value using the following parameters:

* *Label*: EXPORTER-oauth-tls-session-bound
* *Context*: Empty (zero-length)
* *Length*: 32 octets

For TLS 1.3, the exporter is derived as specified in Section 7.5 of {{!RFC8446}}. For TLS 1.2, the exporter is derived as specified in {{!RFC5705}}. TLS 1.3 is RECOMMENDED because TLS 1.2 abbreviated handshakes (session resumption) may reuse the same master secret across connections, weakening the binding.

Note: By binding to the TLS Exporter rather than the application traffic keys, the binding remains valid across TLS 1.3 KeyUpdate operations. Standard key rotation refreshes traffic keys but does not change the exporter master secret, avoiding unnecessary re-proof cycles while maintaining strong connection binding.

2.3. Session-Binding Proof Format

The Session-Binding Proof is a JWT with the following structure:

2.3.1. Header

```
{
  "typ": "tls-binding-proof+jwt",
  "alg": "ES256",
  "x5t#S256": "<base64url SHA-256 thumbprint of client certificate>"
}
```

The alg value MUST match the key type of the client's mTLS certificate. The x5t#S256 value MUST be the base64url-encoded SHA-256 thumbprint of the DER-encoded client certificate, as defined in {{!RFC8705}}.

2.3.2. Payload

```
{
  "jti": "<unique identifier>",
  "ath": "<base64url SHA-256 hash of the access token>",
  "ekm": "<base64url TLS Exporter value>",
  "iat": 1710820000,
  "htm": "POST",
  "htu": "/api/resource"
}
```

The payload claims are defined as follows:

jti: REQUIRED. A unique identifier for the proof to prevent replay. The value MUST be unique per proof token and SHOULD be a UUID or equivalent.

ath: REQUIRED. The base64url-encoded SHA-256 hash of the ASCII encoding of the associated access token value.

ekm: REQUIRED. The base64url-encoded TLS Exporter value derived as specified in Section 2.2.

iat: REQUIRED. The time at which the proof was issued, as a NumericDate (seconds since the Unix epoch). The resource server MUST verify that this value is within an acceptable clock skew window.

htm: REQUIRED. The HTTP method of the request to which the proof is

attached (e.g., "GET", "POST").
htu: REQUIRED. The HTTP target URI of the request to which the proof is attached, without query and fragment parts.

2.3.3. Signature

The JWT MUST be signed using the private key corresponding to the client's mTLS certificate. The signature algorithm MUST match the alg header parameter.

2.4. Token Confirmation Claim

An authorization server that supports TLS-session-bound access tokens MUST include a cnf (confirmation) claim in the issued access token (when the token is a JWT) or in the token introspection response. The cnf claim MUST contain:

```
{
  "cnf": {
    "x5t#S256": "<cert-thumbprint>",
    "tls_exp": "EXPORTER-oauth-tls-session-bound"
  }
}
```

x5t#S256: REQUIRED. The certificate thumbprint as defined in {{!RFC8705}}.
tls_exp: REQUIRED. A string value containing the TLS Exporter label that the client and resource server MUST use to derive the session-binding value. The presence of this claim signals that the resource server MUST require and verify a Session-Binding Proof for every request using this token. This follows the pattern of existing cnf members which carry key/binding material rather than boolean flags (see {{!RFC7800}}).

3. Protocol Flow

3.1. Token Issuance with Session Binding

The mechanism for requesting and issuing TLS-session-bound tokens is as follows:

1. The client authenticates to the authorization server using mTLS and requests a token. This MAY be a token exchange per {{!RFC8693}}, a client credentials grant, or any other OAuth 2.0 grant type.
2. The authorization server issues a new access token.

3. If the authorization server policy requires TLS session binding for this client, it includes the cnf claim with tls_exp set to the exporter label in the issued token.
4. The client receives the token and notes the tls_exp requirement.

3.2. Authorization Server Behavior

The authorization server determines whether to issue TLS-session-bound tokens based on per-client configuration. The following client registration metadata parameter is defined:

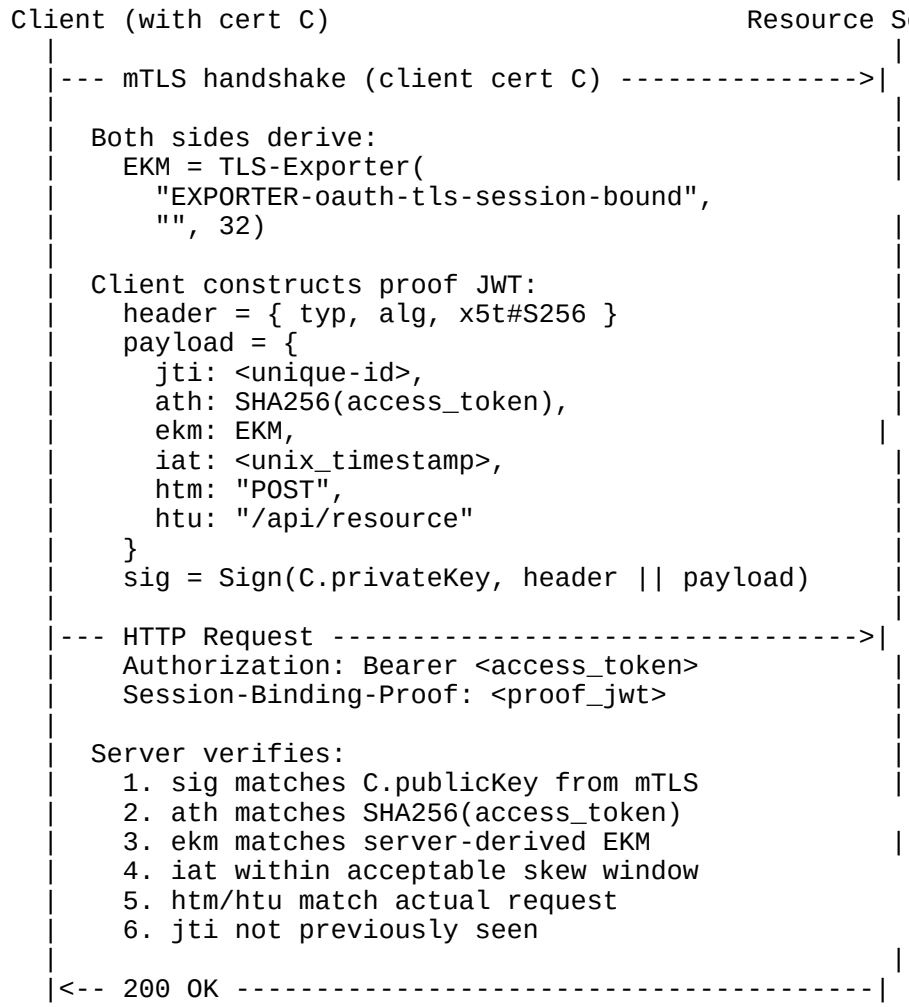
tls_session_bound_access_tokens: OPTIONAL. A boolean value indicating that the authorization server MUST issue TLS-session-bound access tokens for this client. When set to true, the authorization server includes the tls_exp confirmation method in all access tokens issued to this client. Defaults to false.

The authorization server MAY also apply session binding based on:

* Token exchange policy: When the subject_token or actor_token in an RFC 8693 exchange is itself session-bound, the resulting token SHOULD also be session-bound to maintain the security property across the delegation chain.
* Resource server requirements: When a resource server’s metadata indicates that it requires session-bound tokens.
* Risk-based policy: When the requested scope, audience, or delegation depth exceeds a policy threshold.

3.3. Resource Access with Session-Binding Proof

The following diagram illustrates the complete flow:



When the client presents the access token to a resource server:

1. The client establishes an mTLS connection with the resource server.

2. The client derives the TLS Exporter value for the current session.

3. The client constructs a Session-Binding Proof JWT as specified in Section 2.3.

4. The client sends the HTTP request with:

* The access token in the Authorization header: Authorization: Bearer <access_token>
* The Session-Binding Proof in the Session-Binding-Proof header: Session-Binding-Proof: <proof_jwt>

5. The resource server performs the following verifications:

a. *Standard mTLS verification*: Verifies the client certificate as part of the TLS handshake. b. *Token validation*: Validates the access token (signature, expiration, audience, etc.). c. *Certificate binding*: Verifies that the x5t#S256 in the token's cnf claim matches the presented client certificate. d. *TLS binding required*: Checks that cnf.tls_exp is present and a Session-Binding Proof is present. e. *Proof signature*: Verifies the proof JWT signature against the public key in the client certificate. f. *Exporter match*: Derives the TLS Exporter value for the current session and confirms it matches the ekm claim in the proof. g. *Token hash*: Computes SHA-256 of the presented access token and confirms it matches the ath claim. h. *Timestamp*: Confirms iat is within the acceptable skew window. i. *Method and URI*: Confirms htm and htu match the actual request. j. *Uniqueness*: Confirms the jti has not been seen before within the token's validity period.

6. If all verifications succeed, the resource server processes the request. If any verification fails, the resource server MUST reject the request as specified in Section 3.5.

3.4. Token Introspection Considerations

When token introspection {{!RFC7662}} is used, the introspection response MUST include the cnf claim with the tls_exp field. This allows resource servers that do not have direct access to the token's claims (e.g., opaque tokens) to determine whether session binding is required.

3.5. Error Responses

When verification of the Session-Binding Proof fails, the resource server MUST respond with HTTP 401 and include a WWW-Authenticate header with the following error codes:

```
WWW-Authenticate: Bearer error="invalid_proof",
  error_description="description of failure"
```

The following error code values are defined:

invalid_proof: The Session-Binding Proof is missing, malformed, or

failed verification. This includes signature verification failure, exporter mismatch, expired iat, and htm/htu mismatch.

use_session_binding: The access token requires a Session-Binding Proof (the cnf.tls_exp claim is present) but no Session-Binding-Proof header was provided. This error signals to the client that it must construct and present a proof.

The resource server SHOULD include an error_description parameter with a human-readable explanation of the specific verification failure.

4. Integration with Existing Mechanisms

4.1. Relationship to RFC 8705 (mTLS-Bound Tokens)

This specification extends RFC 8705 by adding session-level binding on top of certificate binding. The x5t#S256 claim from RFC 8705 is reused. Deployments MAY support both mechanisms simultaneously: RFC 8705 provides certificate binding, while this specification adds per-request session binding.

4.2. Relationship to RFC 9449 (DPoP)

DPoP and this specification address similar goals (proof-of-possession) but use different mechanisms and binding targets:

* *DPoP*: Applicable to both public and confidential clients. Particularly valuable for public clients that cannot use mTLS. Uses ephemeral, application-managed keys not bound to the TLS layer.
* *This specification*: Designed for confidential clients and workloads that already use mTLS. Reuses the existing mTLS certificate and adds TLS channel binding.

In environments where both mTLS and DPoP are available, this specification provides stronger security guarantees because it binds to both the client identity (certificate) and the transport session (exporter).

4.3. Relationship to WIMSE WIT/WPT

The WIMSE Workload Identity Token (WIT) and Workload Proof Token (WPT) defined in {{!I-D.ietf-wimse-s2s-protocol}} provide a similar proof-of-possession mechanism for workload-to-workload communication. This specification is compatible with WIMSE and can be used in conjunction with WIT/WPT when mTLS-based session binding is required for tokens obtained via RFC 8693 exchange.

4.4. Relationship to Transitive Attestation

The Transitive Attestation profile {{!I-D.draft-mw-wimse-transitive-attestation}} addresses a complementary problem: binding an identity to a verified execution environment ("Proof of Residency"). While this specification binds tokens to a TLS session to prevent network-level replay, Transitive Attestation binds identities to a hardware-rooted host to prevent credential export. In high-assurance deployments, both mechanisms MAY be combined: Transitive Attestation ensures the token is used from the correct host, and TLS session binding ensures it is used on the correct connection.

4.5. Relationship to RFC 8693 (Token Exchange)

This specification does not modify the token exchange protocol itself. The authorization server's token exchange endpoint continues to operate as specified in {{!RFC8693}}. The session binding is applied to the _resulting_ access token through the cnf claim. While this specification is applicable to any OAuth 2.0 access token, RFC 8693 Token Exchange is a primary motivator: each hop in a delegation chain produces a new bearer token, and session binding contains the blast radius of any single token compromise to the specific TLS connection on which it was issued.

5. TLS Proxy Considerations

5.1. Pass-Through mTLS

In deployments where TLS is terminated at the application server (pass-through mode), this specification works without modification. Both the client and server have direct access to the TLS Exporter value.

5.2. TLS Termination at Proxy

When a TLS-terminating proxy (e.g., a load balancer or API gateway) sits between the client and the resource server, the proxy MUST forward the following information to the backend:

1. The client certificate or its thumbprint, as specified in {{!RFC9440}}.
2. The TLS Exporter value derived from the client-to-proxy mTLS session.

A new HTTP header is defined for conveying the exporter value:

TLS-Exporter: <base64url-encoded exporter value>

The proxy MUST derive the exporter using the label and parameters specified in Section 2.2 from the client-facing TLS session, and forward it in this header over a trusted, integrity-protected connection to the backend.

The backend resource server MUST use the forwarded exporter value (instead of its own locally-derived value) when verifying the Session-Binding Proof.

Security Warning: The TLS-Exporter header contains security-sensitive material. The connection between the proxy and backend MUST be integrity-protected (e.g., via a separate mTLS connection or a trusted network). The backend MUST NOT accept this header from untrusted sources.

6. Security Considerations

This section addresses security considerations in addition to those described in the OAuth 2.0 Security Best Current Practice {{!I-D.ietf-oauth-security-topics}}.

6.1. Addressed Threats

6.1.1. Cross-Connection Replay

The TLS Exporter value is cryptographically derived from the TLS handshake transcript and is unique per TLS connection. An attacker who intercepts a bearer token cannot replay it on a different TLS connection because the exporter value will not match.

6.1.2. Cross-Host Replay

The Session-Binding Proof is signed with the client’s mTLS private key. An attacker on a different host cannot produce a valid proof without possessing the private key.

6.1.3. Token Exfiltration via LLM Prompt Injection

Even if an AI agent’s bearer token is exfiltrated via prompt injection, tool-call side channels, or log leakage, the attacker cannot produce a valid Session-Binding Proof because they lack both the client’s private key and the ability to derive the TLS Exporter value from the legitimate session.

6.1.4. Multi-Hop Delegation Chain Compromise

Each hop in a delegation chain (A→B→C→D) uses a distinct mTLS session with a distinct exporter value. A token stolen at any point in the chain is bound to that specific session and cannot be replayed on a different hop.

6.2. Residual Risks

6.2.1. Intra-Session Replay

Within the same TLS session, an attacker with access to the channel (e.g., a compromised middleware component) could observe and replay requests. This risk is mitigated by:

* The jti claim, which provides per-proof uniqueness when the server maintains a replay cache.
* Short iat validity windows that limit the temporal scope of any replay.
* An OPTIONAL server-issued nonce mechanism for environments requiring stronger intra-session replay protection.

6.2.2. Compromised Private Key

If the client's mTLS private key is compromised, the attacker can produce valid proofs. This risk is mitigated by:

* Hardware-backed key storage (TPM, HSM, TEE).
* Short-lived certificates (e.g., SPIFFE SVIDs with 1-hour expiry).
* Transitive Attestation {{!I-D.draft-mw-wimse-transitive-attestation}} for binding identity to a verified execution context.

6.2.3. TLS Proxy Trust

The TLS-Exporter header introduces a trust dependency on the proxy. A compromised proxy could forge exporter values. Mitigations include mutual authentication between proxy and backend, and restricting the header to trusted network segments.

7. IANA Considerations

7.1. OAuth Token Confirmation Methods

This specification registers the following confirmation method in the IANA "OAuth Token Confirmation Methods" registry established by {{!RFC7800}}:

* *Confirmation Method Value*: tls_exp
* *Confirmation Method Description*: TLS Exporter Session Binding
* *Change Controller*: IETF
* *Reference*: [this document]

7.2. OAuth Dynamic Client Registration Metadata

This specification registers the following client metadata value:

* *Client Metadata Name*: tls_session_bound_access_tokens
* *Client Metadata Description*: Boolean indicating the client requires TLS-session-bound access tokens
* *Change Controller*: IETF
* *Reference*: [this document]

7.3. HTTP Header Fields

This specification registers the following HTTP header fields:

7.3.1. Session-Binding-Proof

* *Header Field Name*: Session-Binding-Proof
* *Status*: permanent
* *Reference*: [this document]

7.3.2. TLS-Exporter

* *Header Field Name*: TLS-Exporter
* *Status*: permanent
* *Reference*: [this document]

7.4. TLS Exporter Label

This specification registers the following TLS Exporter label in the IANA "TLS Exporter Labels" registry:

* *Value*: EXPORTER-oauth-tls-session-bound
* *DTLS-OK*: N
* *Recommended*: Y
* *Reference*: [this document]

Authors' Addresses

Ram Krishnan
JPMorgan Chase & Co
Email: ramkri123@gmail.com

A Prasad
Oracle
Email: a.prasad@oracle.com

Diego R. Lopez
Telefonica
Email: diego.r.lopez@telefonica.com

Srinivasa Addepalli
Aryaka
Email: srinivasa.addepalli@aryaka.com