

oauth  
Internet-Draft  
Intended status: Standards Track  
Expires: 28 July 2026

A. Tulshibagwale  
SGNL  
G. Fletcher  
Practical Identity LLC  
P. Kasselmann  
Defakto Security  
24 January 2026

Transaction Tokens  
draft-ietf-oauth-transaction-tokens-07

## Abstract

Transaction Tokens (Txn-Tokens) are designed to maintain and propagate user identity and authorization context across workloads within a trusted domain during the processing of external requests, such as API calls. They ensure that this context is preserved throughout the call chain, even when new transactions are initiated internally, thereby enhancing security and consistency in complex, multi-service architectures.

## About This Document

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

The latest revision of this draft can be found at <https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-oauth-transaction-tokens/>.

Source for this draft and an issue tracker can be found at <https://github.com/oauth-wg/oauth-transaction-tokens>.

## 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 28 July 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. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Overview . . . . .	4
3. Notational Conventions . . . . .	4
4. Terminology . . . . .	5
5. What are Transaction Tokens? . . . . .	5
5.1. Authorization Context . . . . .	5
6. Creating Txn-Tokens . . . . .	6
6.1. Creation . . . . .	6
7. Txn-Token Lifetime . . . . .	7
8. Benefits of Txn-Tokens . . . . .	7
9. Txn-Token Issuance and Usage Flows . . . . .	7
9.1. Basic Flow . . . . .	7
9.2. Internally Initiated Txn-Token Flow . . . . .	9
10. Txn-Token Format . . . . .	10
10.1. JWT Header . . . . .	10
10.2. JWT Body Claims . . . . .	10
10.2.1. Scope claim . . . . .	11
10.2.2. Requester Context . . . . .	12
10.2.3. Transaction Context . . . . .	12
10.2.4. Example . . . . .	13
11. Txn-Token Service (TTS) . . . . .	14
12. Requesting Txn-Tokens . . . . .	14
12.1. Txn-Token Request . . . . .	14
12.2. Subject Token Types . . . . .	16
12.2.1. Self-Signed Subject Token Type . . . . .	16

12.2.2. Unsigned JSON Object Subject Token Type . . . . .	17
12.3. Txn-Token Request Processing . . . . .	17
12.4. Txn-Token Response . . . . .	18
12.5. Mutual Authentication of the Txn-Token Request . . . . .	19
13. Using Txn-Tokens . . . . .	20
13.1. Txn-Token HTTP Header . . . . .	20
13.2. Txn-Token Validation . . . . .	20
14. Security Considerations . . . . .	20
14.1. Txn-Token Lifetime . . . . .	20
14.2. Access Tokens . . . . .	21
14.3. Subject Token Types . . . . .	21
14.4. Use of 'actor_token' . . . . .	21
14.5. Scope Processing . . . . .	21
14.6. Unique Transaction Identifier . . . . .	21
14.7. TTS Discovery . . . . .	22
14.8. Workload Configuration Protection . . . . .	22
14.9. TTS Authentication . . . . .	22
14.10. TTS Key Rotation . . . . .	22
14.11. Transaction Tokens Are Not Authentication Credentials .	23
14.12. Replacing Transaction Tokens . . . . .	23
14.12.1. TTS Responsibilities . . . . .	23
15. Privacy Considerations . . . . .	24
15.1. Obfuscation of Personal Information . . . . .	24
15.2. Logging . . . . .	24
16. IANA Considerations . . . . .	24
16.1. OAuth URI Subregistry Contents . . . . .	24
16.2. JWT Claims Registry Contents . . . . .	25
16.3. IANA Media Type Registration Contents . . . . .	25
16.4. HTTP Header . . . . .	26
17. References . . . . .	27
17.1. Normative References . . . . .	27
17.2. Informative References . . . . .	29
Acknowledgements . . . . .	30
Document History . . . . .	30
Since Draft 06 . . . . .	30
Since Draft 05 . . . . .	32
Since Draft 04 . . . . .	32
Authors' Addresses . . . . .	32

## 1. Introduction

Modern computing architectures often use multiple independently running components called workloads. In many cases, external invocations through interfaces such as APIs result in a number of internal workloads being invoked in order to process the external invocation. These workloads often run in virtually or physically isolated networks. These networks and the workloads running within their perimeter may be compromised by attackers through software

supply chain, privileged user compromise or other attacks. Workloads compromised through external attacks, malicious insiders or software errors can cause any or all of the following unauthorized actions:

- \* Invocations of workloads in the network without any explicit transaction invocation being present.
- \* Arbitrary user impersonation.
- \* Parameter modification or augmentation.
- \* Theft of tokens, such as OAuth access tokens, used to call external interfaces and passed to internal workloads to convey authorization context.

The results of these actions are unauthorized access to resources.

## 2. Overview

Transaction Tokens (Txn-Tokens) reduce the risks from such attacks or spurious invocations. A valid Txn-Token indicates a valid transaction invocation. Note that while many transactions are initiated via an external event (e.g. internet facing API invocation) other transactions are initiated from within the trusted domain. Txn-Tokens apply to both externally triggered and internally invoked transactions and ensure that the user's identity or a workload that made the request is preserved throughout subsequent workload invocations. They preserve any context such as:

- \* Parameters of the original call
- \* Environmental factors, such as IP address of the original caller
- \* Any context that needs to be preserved in the call chain. This includes information that was not in the original request to the external endpoint.

Cryptographically protected Txn-Tokens ensure that downstream workloads cannot make unauthorized modifications to such information, and cannot make spurious calls.

## 3. Notational Conventions

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.

#### 4. Terminology

**Workload:** A running instance of software executing for a specific purpose. Examples of workloads include containerized microservices, monolithic services and infrastructure services such as managed databases.

**Trust Domain:** A logical grouping of systems that share a common set of security controls and policies. In practice this may include a virtually or physically separated network, which contains two or more workloads. The workloads within a Trust Domain may be invoked only through published interfaces.

**External Endpoint:** A published interface to a Trust Domain that results in the invocation of a workload within the Trust Domain. In practice, the external endpoint may be accessed through a gateway service as described in the WIMSE architecture [I-D.ietf-wimse-arch].

**Call Chain:** A sequence of invocations that results from the invocation of an external endpoint.

**Transaction Token (Txn-Token):** A signed JWT with a short lifetime, providing immutable information about the user or workload, certain parameters of the call, and specific contextual attributes of the call. The Txn-Token is used to authorize subsequent calls in the call chain.

**Transaction Token Service (TTS):** A special service within the Trust Domain that issues Txn-Tokens to requesting workloads. Each Trust Domain using Txn-Tokens MUST have exactly one logical TTS.

#### 5. What are Transaction Tokens?

Txn-Tokens are short-lived, signed JWTs [RFC7519] that assert the identity of a user or a workload and assert an authorization context. The authorization context provides information expected to remain constant during the execution of a call chain as it passes through multiple workloads.

##### 5.1. Authorization Context

Authorization context includes information used for authorization, accounting and auditing purposes and often contains information about the request being made. A key aspect of the authorization context is the intent or purpose of the transaction, which should be as narrowly defined as possible for the given deployment. A narrowly scoped transaction token reduces the attack surface of captured and replayed

transaction tokens.

## 6. Creating Txn-Tokens

### 6.1. Creation

Txn-Tokens are typically created when a workload is invoked using an endpoint that is externally visible, and is authorized using a separate mechanism, such as an OAuth [RFC6749] access token. The externally visible endpoint exchanges the external authorization token for a transaction token before sending it to the workload. The transaction token may be obtained through a local interface, or it may be requested from a remote server.

If the transaction token request is made via HTTP to a remote server, it MUST use [RFC8693] as described in this specification. To do this, it invokes a special Token Service (the Transaction Token Service (TTS)) and provides context that is sufficient for it to generate a Txn-Token.

The context information provided to the TTS MUST include: \* the identification of the trust domain in which the issued Txn-Token is valid \* a token which identifies the subject of the Txn-Token (e.g. an OAuth access token, a self-issued JWT, etc) \* the desired authorisation scope for the issued Txn-Token

Additional contextual information MAY be provided such as: \* Parameters that are required to be integrity protected for the duration of this call \* Additional context, such as the incoming IP address, User Agent information, or other context that can help the TTS to issue the Txn-Token

The TTS responds to a successful invocation by generating a Txn-Token. The calling workload then uses the Txn-Token to authorize its calls to subsequent workloads.

If the requesting service does not have an inbound token that it can use in its request to the TTS, it generates a self-signed JWT and passes that in the request in place of the external authorization token. This can be the case when the external authentication does not use an access token or when the requesting service is initiating a scheduled internal request on for itself or on behalf of a user of the system.

## 7. Txn-Token Lifetime

Txn-Tokens are expected to be short-lived (on the order of minutes or less), and as a result **MUST** be used only for the expected duration of an external or internal invocation. If the token or other credential (e.g. self-signed JWT) presented to the TTS when requesting a Txn-Token has an expiration time, then the TTS **MUST NOT** issue a Txn-Token if the expiration time has passed. The lifetime of the Txn-Token itself **MAY** exceed the expiration time of the presented token. The expectation is that since Txn-Tokens are short lived and are authorizing a specific transaction, extending beyond the lifetime of the presented expiration time is not a security risk. If a long-running process such as a batch or offline task is involved, the mechanism used to perform the external or internal invocation still results in a short-lived Txn-Token.

## 8. Benefits of Txn-Tokens

Txn-Tokens prevent unauthorized or unintended invocations by allowing a workload to independently verify the identity of the user or workload that initiated an external call, as well as any contextual information relevant to processing that call.

## 9. Txn-Token Issuance and Usage Flows

### 9.1. Basic Flow

Figure 1 shows the basic flow of how Txn-Tokens are used in a multi-workload environment.

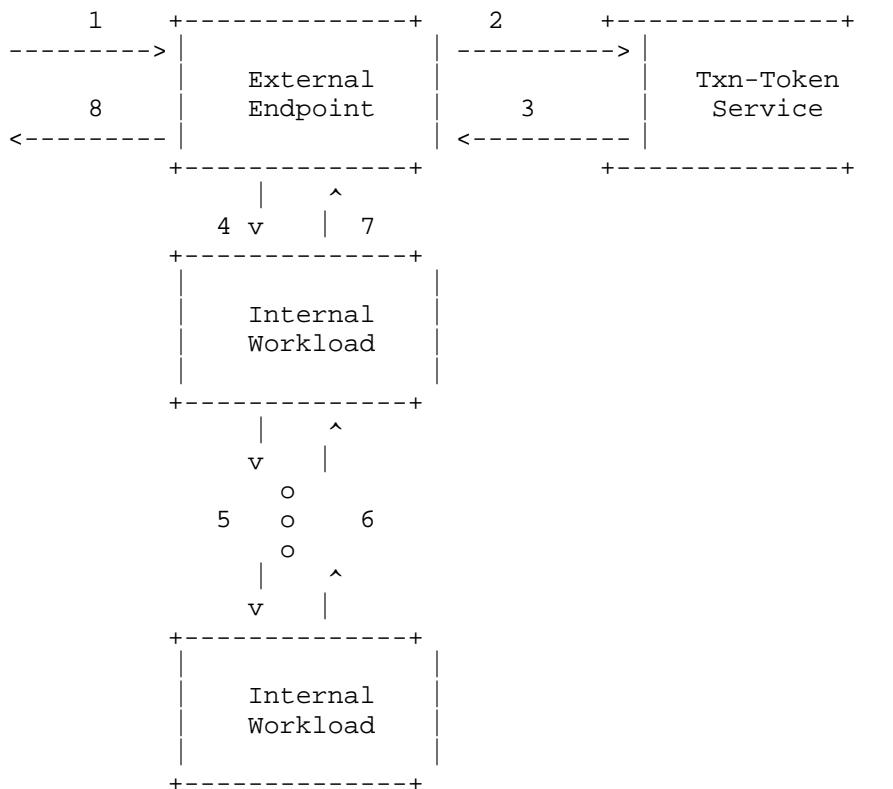


Figure 1: Basic Transaction Tokens Architecture

1. External endpoint is invoked using conventional authorization mechanism such as an OAuth 2.0 Access token
2. External endpoint provides context and incoming authorization (e.g., access token) to the TTS
3. TTS mints a Txn-Token that provides immutable context for the transaction and returns it to the requester
4. The external endpoint initiates a call to an internal workloads and provides the Txn-Token as authorization
5. Subsequent calls to other internal workloads use the same Txn-Token to authorize calls
6. Responses are provided to calling workloads based on successful authorization by the invoked workloads



7. Response provided to external endpoint based on successful authorization by the invoked workload
8. External client is provided a response to the external invocation

## 9.2. Internally Initiated Txn-Token Flow

An internal workload may need to initiate a transaction not on the basis of a current external request, but as part of a scheduled task or in reaction to a specific condition. The transaction may be requested on behalf of the identity of the requesting workload or as an impersonation on behalf of a specific user chosen based on information accessible to the workload.

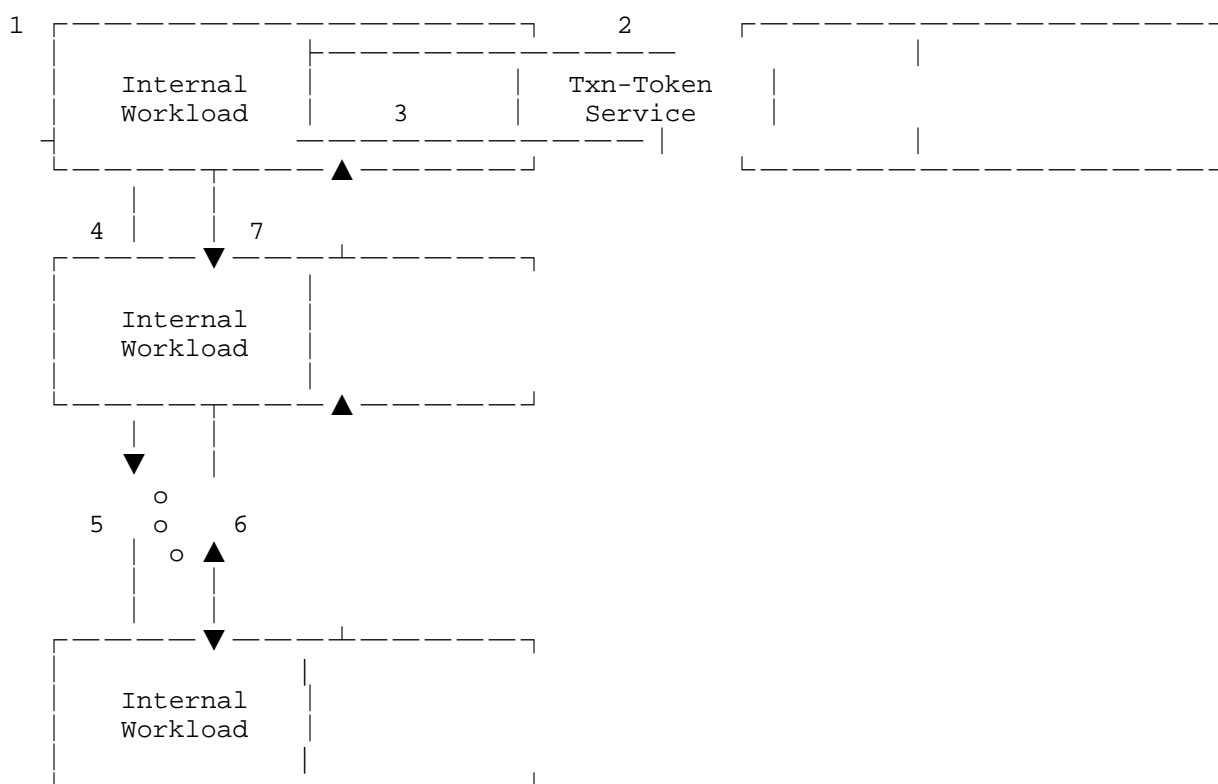


Figure 2: Internally Initiated Txn-Token Flow

In the diagram above, steps 5-6 are the same as in Section 9.1.

1. A microservice determines that it needs to initiate a request on behalf of a user in response to a scheduled timer or other trigger.
2. The internal microservice authenticates to the token service and makes a request for a Txn-Token. The request contains information about the transaction along with optional additional authorization credentials
3. TTS authorizes the requester and then mints a Txn-Token that provides immutable context for the transaction and returns it to the requester
4. The originating microservice then contacts another internal microservice and provides the Txn-Token as authorization

## 10. Txn-Token Format

A Txn-Token is a JSON Web Token [RFC7519] protected by a JSON Web Signature [RFC7515]. The following describes the required values in a Txn-Token:

### 10.1. JWT Header

In the JWT Header:

- \* The typ Header Parameter MUST be present and MUST have the value txntoken+jwt.
- \* Key rotation of the signing key SHOULD be supported through the use of a kid Header Parameter.

Figure 3 is a non-normative example of the JWT Header of a Txn-Token

```
{
  "typ": "txntoken+jwt",
  "alg": "RS256",
  "kid": "identifier-to-key"
}
```

Figure 3: Example: Txn-Token Header

### 10.2. JWT Body Claims

The transaction token body follows the JWT format and includes existing JWT claims as well as defines new claims. These claims are described below:

iss: OPTIONAL The iss claim as defined in [RFC7519] is not required as Txn-Tokens are bound to a single Trust Domain as defined by the aud claim and often the signing keys are known. The iss claim MUST be used in cases where the signing keys are not predetermined.

iat: REQUIRED The issued at time of the Txn-Token as defined in [RFC7519]

aud: REQUIRED This claim, defined in [RFC7519], MUST identify the Trust Domain in which the Txn-Token is valid. A Txn-Token MUST NOT be accepted outside the Trust Domain identified by the aud claim.

exp: REQUIRED Expiry time of the Txn-Token as defined in [RFC7519]

txn: REQUIRED A unique transaction identifier as defined in Section 2.2 of [RFC8417].

sub: REQUIRED This claim represents the principal of the transaction as defined by Section 4.1.2 of [RFC7519]. The value MUST be unique within the context of the aud Trust Domain. Note: Unlike OpenID Connect, the sub claim is NOT associated with the iss claim.

scope: REQUIRED The scope claim is defined in Section 4.2 of [RFC8693]. Note that the value of this claim is determined by the TTS and is not required to match the requested scope nor the scope in any supplied external token.

tctx: OPTIONAL A JSON object that contains values that remain immutable throughout the call chain.

rctx: OPTIONAL A JSON object that describes the environmental context of the requested transaction.

req\_wl: REQUIRED. A StringOrURI value that identifies the workload that requested the Txn-Token.

#### 10.2.1. Scope claim

The scope claim captures, as narrowly as possible, the purpose of this particular transaction. The values used for this claim are defined by the TTS as representative of the authorization model defined by the Trust Domain. The value may be literally and semantically different from, and represent an intent narrower, than a scope value issued to an external client. How a given deployment represents the authorization model within the Trust Domain is at its

discretion and not prescribed by this specification.

#### 10.2.2. Requester Context

The Txn-Token SHOULD contain an rctx claim. This MAY include the IP address information of the originating requestor, as well as information about the computational entity that requested the Txn-Token and contextual attributes of the originating request itself.

The JSON value of the rctx claim MAY include any values the TTS determines are interesting to downstream services that rely on the Txn-Token. The following claims are defined so that if they are included, they have the following meaning:

- \* req\_ip The IP address of the requester. This MAY be the end-user or a process that initiated the Transaction.
- \* authn The authentication method used to identify the requester. Its value is a string that uniquely identifies the method used.

The following is a non-normative example of an rctx claim initiated by an external call:

```
{
  "rctx": {
    "req_ip": "69.151.72.123", // env context of external call
    "authn": "urn:ietf:rfc:6749" // env context of external call
  }
}
```

#### 10.2.3. Transaction Context

The Txn-Token SHOULD contain an tctx claim. The value of this claim is a JSON object that contains name/value pairs (wherein the value could itself be an object), which together assert the details that remain immutable through the call-chain where this Txn-Token is used.

Txn-Tokens are primarily used to assure identity and context for a transaction, and the content of this field is a critical part of that context.

Whereas the rctx field contains environmental values related to the request, the tctx field contains the actual authorization details that are determined by the TTS. These values are used by services using the Txn-Token to reliably obtain specific parameters needed to perform their work. The content of the tctx field is determined by the TTS and they may be computed internally or from parameters it receives from the service that requests the Txn-Token.

The following is a non-normative example of an tctx claim initiated by an external call:

```
"tctx": {
  "action": "BUY", // parameter of external call
  "ticker": "MSFT", // parameter of external call
  "quantity": "100", // parameter of external call
  "customer_type": { // computed value not present in external call
    "geo": "US",
    "level": "VIP"
  }
}
```

#### 10.2.4. Example

The figure below Figure 4 shows a non-normative example of the JWT body of a Txn-Token initiated by an external call:

```
{
  "iat": 1686536226,
  "aud": "trust-domain.example",
  "exp": 1686536586,
  "txn": "97053963-771d-49cc-a4e3-20aad399c312",
  "sub": "d084sdrt234fsaw34tr23t",
  "req_wl": "apigateway.trust-domain.example", // the internal entity that requested the Txn-Token
  "rctx": {
    "req_ip": "69.151.72.123", // env context of external call
    "authn": "urn:ietf:rfc:6749", // env context of the external call
  },
  "scope": "trade.stocks",
  "tctx": {
    "action": "BUY", // parameter of external call
    "ticker": "MSFT", // parameter of external call
    "quantity": "100", // parameter of external call
    "customer_type": { // computed value not present in external call
      "geo": "US",
      "level": "VIP"
    }
  }
}
```

Figure 4: Example: Txn-Token Body

A Txn-Token from an internal request would look much the same except the rctx and tctx field claims would be populated with information from the initiating workloads request.

## 11. Txn-Token Service (TTS)

A Txn-Token Service (TTS) uses [RFC8693] as described in this specification if it issues transaction tokens in response to HTTP requests. The unique properties of the Txn-Token requests and responses are described below. The TTS MAY optionally support other OAuth 2.0 endpoints and features, but that is not a requirement for it to be a TTS.

Each Trust Domain that uses Txn-Tokens MUST have exactly one logical TTS.

## 12. Requesting Txn-Tokens

A workload requests a Txn-Token from a TTS. If the transaction token request is made via HTTP to a remote server, it MUST use [RFC8693] as described in this specification. Txn-Tokens may be requested for both externally originating or internally originating requests. The profile describes how required and optional context can be provided to the TTS in order for the Txn-Token to be issued. The request to obtain a Txn-Token using this method is called a Txn-Token Request, and a successful response is called a Txn-Token Response. The Txn-Token profile of the OAuth 2.0 Token Exchange [RFC8693] is described below.

### 12.1. Txn-Token Request

A workload requesting a Txn-Token provides the TTS with proof of its identity (client authentication), the purpose of the Txn-Token and optionally any additional context relating to the transaction being performed. Most of these elements are provided by the OAuth 2.0 Token Exchange specification and the rest are defined as new parameters. Additionally, this profile defines a new token type URN `urn:ietf:params:oauth:token-type:txn_token` which is used by the requesting workload to identify that it is requesting the Txn-Token Response to contain a Txn-Token.

To request a Txn-Token the workload invokes the OAuth 2.0 [RFC6749] token endpoint with the following parameters:

- \* `grant_type` REQUIRED. The value MUST be set to `urn:ietf:params:oauth:grant-type:token-exchange`.
- \* `audience` REQUIRED. The value MUST be set to the Trust Domain name.

- \* `scope` REQUIRED. A space-delimited list of case-sensitive strings where the value(s) MUST represent the specific purpose or intent of the transaction.
- \* `requested_token_type` REQUIRED. The value MUST be `urn:ietf:params:oauth:token-type:txn_token`
- \* `subject_token` REQUIRED. The value MUST contain a token that represent the subject of the transaction. The manner in which the subject is represented in the `subject_token` depends on the `subject_token_type`. The `subject_token` MAY be:
  - An inbound token received by an external endpoint (e.g. an API Gateway)
  - A self-signed JWT constructed by a workload initiating a transaction
  - An unsigned JSON object constructed by a workload initiating a transaction
  - Any other format that is understood by the TTS

The type of the `subject_token` field is identified by `subject_token_type`.

- \* `subject_token_type` REQUIRED. The value MUST indicate the type of the token or value present in the `subject_token` parameter

The following additional parameters are RECOMMENDED to be present in a Txn-Token Request:

- \* `request_context` OPTIONAL. This parameter contains a JSON object which represents the context of this transaction.
- \* `request_details` OPTIONAL. This parameter contains a JSON object which contains additional details about the request. This could include API parameters, authorization criteria or other details the requester would like to pass to the TTS. The TTS uses this data along with other information at its disposal to construct the txct JSON object (if required).

All parameters are encoded using the "application/x-www-form-urlencoded" format per Appendix B of [RFC6749].

The figure below Figure 5 shows a non-normative example of a Txn-Token Request. Line breaks added for readability.

```

POST /txn-token-service/token_endpoint HTTP 1.1
Host: txn-token-service.trust-domain.example
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&requested_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Atxn-token
&audience=http%3A%2F%2Ftrust-domain.example
&scope=finance.watchlist.add
&subject_token=eyJhbGciOiJIUzI1NiIsImtpZC...kdXjwhw
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
&request_context=%7B%0A%20%20%20%20%20%20%22req_ip%22%3A%20%2269.151.72.123%22%2C%20%2
%2F%20env%20context%20of%20external%20call%0A%20%20%20%20%20%20%22authn%22%3A%20%22urn
%3Aietf%3Arfc%3A6749%22%2C%20%2F%2F%20env%20context%20of%20external%20call%0A%20%20%20
%20%20%20%22workload3.trust-domain.example%22%20%5D%0A%20%20%20%20%7D

```

F

Figure 5: Example: Txn-Token Request

## 12.2. Subject Token Types

The `subject_token_type` parameter value MUST be a URI [RFC3986]. It MAY be:

- \* Any one of the subject token types described in Section 3 of OAuth 2.0 Token Exchange [RFC8693] except the Refresh Token type (i.e., `urn:ietf:params:oauth:token-type:refresh_token`).
- \* A URN type name when the subject token is a self-signed JWT, as described below.
- \* A URN type name when the subject token is an unsigned JSON object, as described below.
- \* A custom URN agreed to between requesters and the TTS. The TTS MAY support other token formats, which MAY be specified in the `subject_token_type` parameter.

### 12.2.1. Self-Signed Subject Token Type

A requester MAY use a self-signed JWT as a `subject_token` value. In that case, the requester MUST set the `subject_token_type` value to: `urn:ietf:params:oauth:token-type:self_signed`. This self-signed JWT MUST contain the following claims:

- \* `iss`: The unique identifier of the requesting workload.
- \* `sub`: The subject for whom the Txn-Token is being requested. The TTS SHALL use this value in determining the `sub` value in the Txn-Token issued in the response to this request.



- \* `aud`: The unique identifier of the TTS. The TTS SHALL verify that this value matches its own unique identifier.
- \* `iat`: The time at which the self-signed JWT was created. Note that the TTS may reject self-signed tokens with an `iat` value that is unreasonably far in the past or future.
- \* `exp`: The expiration time for the JWT. Section 14.1 provides guidance on setting the expiry of a Txn-Token.

The self-signed JWT MAY contain other claims.

#### 12.2.2. Unsigned JSON Object Subject Token Type

A requester MAY use an unsigned JSON object as a `subject_token` value. In that case, the requester MUST set the `subject_token_type` value to: `urn:ietf:params:oauth:token-type:unsigned_json`. The JSON object in the subject token MUST contain the following fields:

- \* `sub`: The subject for whom the Txn-Token is being requested. The TTS SHALL use this value in determining the `sub` value in the Txn-Token issued in the response to this request.

The unsigned JSON object MAY contain other fields, and the TTS MAY consider them when generating the Txn-Token.

#### 12.3. Txn-Token Request Processing

When the TTS receives a Txn-Token Request it:

- \* MUST validate the requesting workload client authentication and determine if that workload is authorized to obtain the Txn-Tokens with the requested value(s). The authorization policy for determining such issuance is out of scope for this specification.
- \* Next, the TTS MUST validate the `subject_token`, including verifying the signature, if it is signed.
- \* The TTS determines the value to specify as the `sub` of the Txn-Token and MUST ensure the `sub` value is unique within the Trust Domain defined by the `aud` claim.
- \* The TTS MUST set the `iat` claim to the time of issuance of the Txn-Token.

- \* The TTS MUST set the aud claim to an identifier representing the Trust Domain of the TTS. If the TTS supports multiple Trust Domains, then it MUST determine the correct aud value for this request.
- \* The TTS MUST set the exp claim to the expiry time of the Txn-Token. The TTS MAY consider any exp value present in the subject\_token parameter of the Txn-Token Request in determining the exp value of the resulting Txn-Token.
- \* The TTS MUST set the txn claim to a unique ID specific to this transaction.
- \* The TTS MAY set the iss claim of the Txn-Token to a value defining the entity that signed the Txn-Token. This claim MUST be omitted if not set.
- \* The TTS MUST evaluate the value specified in the scope parameter of the request to determine the scope claim of the issued Txn-Token.
- \* If a request\_context parameter is present in the Txn-Token Request, the data SHOULD be added to the rctx object of the Txn-Token.
- \* If a request\_details parameter is present in the Txn-Token Request, then the TTS SHOULD propagate the data from the request\_details object into the claims in the tctx object as authorized by the TTS authorization policy for the requesting client.

The TTS MAY provide additional processing and verification that is outside the scope of this specification.

#### 12.4. Txn-Token Response

A successful response to a Txn-Token Request by a TTS is called a Txn-Token Response. The following values defined in [RFC8693] MUST be included in the Txn-Token Response:

- \* The token\_type value MUST be set to N\_A per guidance in OAuth 2.0 Token Exchange [RFC8693]
- \* The access\_token value MUST be the Txn-Token JWT
- \* The issued\_token\_type value MUST be set to urn:ietf:params:oauth:token-type:txn\_token

The Txn-Token Response MUST NOT include the refresh\_token value.

If the TTS responds with an error, the error response is as described in Section 5.2 of [RFC6749].

Figure 6 shows a non-normative example of a Txn-Token Response.

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "token_type": "N_A",
  "issued_token_type": "urn:ietf:params:oauth:token-type:txn_token",
  "access_token": "eyJCI6IjllciJ9...Qedw6rx"
}
```

Figure 6: Example: Txn-Token Response

#### 12.5. Mutual Authentication of the Txn-Token Request

A TTS and requesting workload MUST authenticate each other (mutual authentication). Workloads MUST authenticate the TTS to ensure that they do not disclose sensitive information, such as OAuth access tokens, to a rogue TTS. A TTS MUST authenticate a workload to ensure the workload is authorized to request a transaction token.

Workloads SHOULD use the https scheme to secure the communication channel and authenticate the TTS. When using https, TLS certificates MUST be checked according to Section 4.3.4 of [RFC9110]. At the time of this writing, TLS version 1.3 [RFC8446] is the most recent version.

Workloads SHOULD authenticate to a Transaction Token Server using asymmetric (public-key based) methods or signed client authentication JWTs in accordance with [RFC7523].

Examples of public-key based authentication include those defined in OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens [RFC8705], Workload Authentication Using Mutual TLS [I-D.ietf-wimse-mutual-tls], WIMSE Workload-to-Workload Authentication with HTTP Signatures [I-D.ietf-wimse-http-signature] and WIMSE Workload Proof Token [I-D.ietf-wimse-wpt].

### 13. Using Txn-Tokens

Txn-Tokens need to be communicated between workloads that depend upon them to authorize the request. Such workloads will often present HTTP [RFC9110] interfaces for being invoked by other workloads. This section specifies the HTTP header the invoking workload MUST use to communicate the Txn-Token to the invoked workload, when the invoked workload presents an HTTP interface. Note that the standard HTTP Authorization header MUST NOT be used because that may be used by the workloads for other purposes.

#### 13.1. Txn-Token HTTP Header

A workload that invokes another workload using HTTP and needs to present a Txn-Token to the invoked workload MUST use the HTTP Header Txn-Token to communicate the Txn-Token in the HTTP Request. The value of this header MUST be exactly one Txn-Token.

#### 13.2. Txn-Token Validation

A workload that receives a Txn-Token MUST evaluate the token for validity before authorizing it for activities supported by the workload. To validate the Txn-Token, the workload MUST:

- \* validate the Txn-Token JWS signature
- \* verify the aud claim identifies the trust domain of the workload
- \* verify the Txn-Token is not expired

In addition, any outbound calls made by this workload MUST include the Txn-Token as it was received so that it is passed unmodified to any downstream workloads.

Once the Txn-Token is determined to be valid, the workload MAY use any of the data contained in the Txn-Token to determine if the Txn-Token authorizes the requested activity. How the workload determines this authorization is out of scope for this specification.

### 14. Security Considerations

#### 14.1. Txn-Token Lifetime

A Txn-Token is not resistant to replay attacks. A long-lived Txn-Token therefore represents a risk if it is stored in a file, discovered by an attacker, and then replayed. For this reason, a Txn-Token lifetime must be kept short, not exceeding the lifetime of a call-chain. Even for long-running "batch" jobs, a longer-lived access token should be used to initiate the request to the batch endpoint. It then obtains short-lived Txn-Tokens that may be used to authorize the call to downstream services in the call-chain.

Because Txn-Tokens are short-lived, the Txn-Token response from the TTS does not contain the `refresh_token` field. A Txn-Token cannot be issued by presenting a `refresh_token`.

#### 14.2. Access Tokens

When creating Txn-Tokens, the Txn-Token MUST NOT contain the Access Token presented to the external endpoint. If an Access Token is included in a Txn-Token, an attacker may extract the Access Token from the Txn-Token, and replay it to any Resource Server that can accept that Access Token. Txn-Token expiry does not protect against this attack since the Access Token may remain valid even after the Txn-Token has expired.

#### 14.3. Subject Token Types

A service requesting a Txn-Token SHOULD provide an incoming token if it has one that it used itself to authorize a caller, and if it directly correlates with the downstream call chain it needs the Txn-Token for. In the absence of an appropriate incoming token, the requesting service MAY use a self-signed JWT, an unsigned JSON object or any other format to represent the details of the requester to the TTS.

#### 14.4. Use of 'actor\_token'

If using the `actor_token` and `actor_token_type` parameters of the OAuth 2.0 Token Exchange specification [RFC8693], both parameters MUST be present in the request. The `actor_token` can authenticate the identity of the requesting workload.

#### 14.5. Scope Processing

The authorization model within a Trust Domain boundary may be quite different from the authorization model (e.g. OAuth scopes) used with clients external to the Trust Domain. This makes managing unintentional scope increase a critical aspect of the TTS. The TTS MUST ensure that the requested scope of the Txn-Token is equal or less than the scope(s) identified in the `subject_token`.

#### 14.6. Unique Transaction Identifier

A transaction token conveys user identity and authorization context across workloads in a call chain. The `txn` claim is a unique identifier that, when logged by the TTS and workloads, enables discovery and auditing of successful and failed transactions. The `txn` value SHOULD be unique within the Trust Domain.

#### 14.7. TTS Discovery

A workload may use various mechanisms to determine which instance of a TTS to interact with. Workloads **MUST** retrieve configuration information from a trusted source to minimize the risk of a threat actor providing malicious configuration data that points to an instance of a TTS under its control. Such a service could be used to collect Access Tokens sent as part of the Transaction Token Request message.

To mitigate this risk, workloads **SHOULD** authenticate the service providing the configuration information and verify the integrity of the configuration information. This ensures that no unauthorized entity can insert or alter configuration data. The workload **SHOULD** use Transport Layer Security (TLS) to authenticate the endpoint and secure the communication channel. Additionally, application-layer signatures or message authentication codes **MAY** be used to detect any tampering with the configuration information.

#### 14.8. Workload Configuration Protection

A deployment may include multiple instances of a TTS to improve resiliency, reduce latency and enhance scalability. A workload may be configured to access more than one instance of a TTS to ensure reliability or reduce latency for transaction token requests. The workload configuration should be protected against unauthorized addition or removal of TTS instances. An attacker may perform a denial of service attack or degrade the performance of a system by removing an instance of a TTS from the workload configuration.

#### 14.9. TTS Authentication

A workload may accidentally send a transaction token request to a service that is not a TTS, or an attacker may attempt to impersonate a TTS in order to gain access to transaction token requests which includes sensitive information like access tokens. To minimise the risk of leaking sensitive information like access tokens that are included in the transaction token request, the workload **MUST** ensure that it authenticates the TTS and only contact instances of the TTS that is authorized to issue transaction tokens.

#### 14.10. TTS Key Rotation

The TTS may need to rotate signing keys. When doing so, it **MAY** adopt the key rotation practices in Section 10.1.1 of [OpenIdConnect].

#### 14.11. Transaction Tokens Are Not Authentication Credentials

A workload MUST NOT use a transaction token to authenticate itself to another workload, service or the TTS. Transaction tokens represents information relevant to authorization decisions and are not workload identity credentials. Authentication between the workload and the TTS is described in Section 12.5. The mechanisms used by workloads to authenticate to other workloads, services or system components is out of scope of this specification.

#### 14.12. Replacing Transaction Tokens

A service within a call chain may choose to replace the Txn-Token. This can typically happen if the service wants to change (add to, remove, or modify) the context of the current Txn-Token

To get a replacement Txn-Token, a service will request a new Txn-Token from the TTS and provide the current Txn-Token and other parameters in the request.

##### 14.12.1. TTS Responsibilities

A TTS MUST exercise caution when issuing replacement Txn-Tokens, since replacing Txn-Tokens with arbitrary values negates the primary purpose of having Txn-Tokens. When issuing replacement Txn-Tokens, a TTS:

- \* MAY enable modifications to asserted values that reduce the scope of permitted actions
- \* MAY enable additional asserted values
- \* MUST NOT enable modification to asserted values that expand the scope of permitted actions
- \* MUST NOT modify txn, sub, and aud values of the Txn-Token in the request
- \* MUST NOT remove any of the existing requesting workload identifiers from the req\_wl claim
- \* SHOULD NOT issue replacement Txn-token with lifetime exceeding the lifetime of the originally presented token
- \* MUST append the workload identifier of the workload requesting the replacement to the req\_wl claim using the character , as the separator between individual workload identifiers.

## 15. Privacy Considerations

### 15.1. Obfuscation of Personal Information

Some rctx and tctx claims may be considered personal information in some jurisdictions and if so their values need to be obfuscated. For example, originating IP address (req\_ip) is often considered personal information and in that case must be protected through some obfuscation method (e.g. salted SHA256).

### 15.2. Logging

Complete Txn-Tokens MUST NOT be logged verbatim. This is in order to prevent replay of tokens or leakage of PII or other sensitive information via log files. A hash of the Txn-Token may be logged to allow for correlation with the log files of the TTS that records issued tokens. Alternatively the JWS payload of a Txn-Token may be logged after the signature has been removed. If the Txn-Token contains PII, then care should be taken in logging the content of the Txn-Token so that the PII does not get logged.

## 16. IANA Considerations

This specification registers the following token type identifiers to the "OAuth URI" subregistry of the "OAuth Parameters" [IANA.OAuth.Parameters] registry. It also registers the following claims defined in Section 10.2 in the IANA JSON Web Token Claims Registry defined in [RFC7519]. It also registers the Media Type [IANA.MediaType] "txn-token+jwt" as defined in the section Section 10.1.

### 16.1. OAuth URI Subregistry Contents

- \* URN: urn:ietf:params:oauth:token-type:txn\_token
  - Common Name: Transaction Token
  - Change Controller: IETF
  - Specification Document Section 12.1 of this specification
- \* URN: urn:ietf:params:oauth:token-type:self\_signed
  - Common Name: Token type for Self-signed JWT
  - Change Controller: IETF



- Specification Document: Section Section 12.2.1 of this specification
- \* URN: urn:ietf:params:oauth:token-type:unsigned\_json
- Common Name: Token type for Unsigned JSON Object
- Change Controller: IETF
- Specification Document: Section Section 12.2.2 of this specification

#### 16.2. JWT Claims Registry Contents

- \* Claim Name: tctx
  - Claim Description: The transaction authorization details
  - Change Controller: IETF
  - Specification Document: Section Section 10.2 of this specification
- \* Claim Name: rctx
  - Claim Description: The requester context
  - Change Controller: IETF
  - Specification Document: Section Section 10.2 of this specification

#### 16.3. IANA Media Type Registration Contents

The following entry will be proposed using the IANA Media Type registration [IANA.MediaTypes] form.

- \* Type Name: application
- \* Subtype Name: txntoken+jwt
- \* Change Controller: IETF
- \* Required Parameters: N/A.
- \* Optional Parameters: N/A.
- \* Encoding Considerations: 7-bit text

\* Security Considerations:

1. The media type is used to identify JWTs that may be used as Transaction Tokens. It is a piece of data, and may not contain executable content.
2. Transaction Tokens are short-lived tokens used within a trusted environment, so there are no privacy considerations. Transaction Tokens are unmodifiable tokens, which need integrity protection.
3. The JWTs representing Transaction Tokens are signed, and therefore are integrity protected. A recipient of a Transaction Token must verify the signature on the Transaction Token before using it.
4. There are no additional security considerations specific to the use of JWTs as Transaction Tokens
5. The Transaction Tokens format does not require the use of links within the token. If links are used by specific instances of Transaction Tokens, then their interpretation is usage specific

\* Interoperability Considerations: Transaction Tokens inherit all interoperability properties of JWTs.

\* Published Specification: this document (when published)

\* Application Usage: Any application supporting the use of JWTs

\* Fragment Identifier Consideration: N/A.

\* Restrictions on Usage: Any application supporting the use of JWTs

\* Intended Usage: Common

\* Contact Person: Atul Tulshibagwale

#### 16.4. HTTP Header

The header name Txn-Token is proposed to be added to the HTTP Field Name Registry [IANA.HTTP.FieldNames] as an unstructured Header Field. This header is defined in the section Section 13.1. The following entry will be proposed in the HTTP Field Name Registry. Note that this is an unstructured field, therefore the value of the Type field is left empty as shown below:

- \* Field Name: Txn-Token
- \* Type:
- \* Status: permanent
- \* Specification Document: Section 13.1 of this document
- \* Comment: The Authorization header cannot be used for Txn-tokens because that may be used for service-to-service authorization, and the services may simultaneously require the use of Txn-tokens to convey detailed immutable information such as user identity and details of fine-grained authorization that are included in the Txn-token.

## 17. References

### 17.1. 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>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/rfc/rfc3986>>.
- [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>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/rfc/rfc6749>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/rfc/rfc7519>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/rfc/rfc7515>>.

- [RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <<https://www.rfc-editor.org/rfc/rfc7523>>.
- [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>>.
- [RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <<https://www.rfc-editor.org/rfc/rfc8693>>.
- [RFC8417] Hunt, P., Ed., Jones, M., Denniss, W., and M. Ansari, "Security Event Token (SET)", RFC 8417, DOI 10.17487/RFC8417, July 2018, <<https://www.rfc-editor.org/rfc/rfc8417>>.
- [RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <<https://www.rfc-editor.org/rfc/rfc8705>>.
- [RFC9068] Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, DOI 10.17487/RFC9068, October 2021, <<https://www.rfc-editor.org/rfc/rfc9068>>.
- [RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <<https://www.rfc-editor.org/rfc/rfc9110>>.
- [RFC9111] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <<https://www.rfc-editor.org/rfc/rfc9111>>.
- [RFC9651] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 9651, DOI 10.17487/RFC9651, September 2024, <<https://www.rfc-editor.org/rfc/rfc9651>>.
- [IANA.HTTP.FieldNames]  
"HTTP Authentication Schemes", n.d.,  
<<https://www.iana.org/assignments/http-fields/>>.

[IANA.OAuth.Parameters]

IANA, "OAuth Parameters", n.d.,  
<<https://www.iana.org/assignments/oauth-parameters>>.

[IANA.MediaTypees]

IANA, "Media Types", n.d.,  
<<http://www.iana.org/assignments/media-types>>.

[OpenIdConnect]

Sakimura, N., Bradley, J., Jones, M., Medeiros, B. de.,  
and C. Mortimore, "OpenID Connect Core 1.0 incorporating  
errata set 2", November 2014,  
<[https://openid.net/specs/openid-connect-core-1\\_0.html](https://openid.net/specs/openid-connect-core-1_0.html)>.

## 17.2. Informative References

[SPIFFE] Cloud Native Computing Foundation, "Secure Production  
Identity Framework for Everyone", n.d.,  
<<https://spiffe.io/docs/latest/spiffe-about/overview/>>.

[I-D.ietf-wimse-arch]

Salowey, J. A., Rosomakho, Y., and H. Tschofenig,  
"Workload Identity in a Multi System Environment (WIMSE)  
Architecture", Work in Progress, Internet-Draft, draft-  
ietf-wimse-arch-06, 30 September 2025,  
<<https://datatracker.ietf.org/doc/html/draft-ietf-wimse-arch-06>>.

[I-D.ietf-wimse-mutual-tls]

Salowey, J. A. and Y. Rosomakho, "Workload Authentication  
Using Mutual TLS", Work in Progress, Internet-Draft,  
draft-ietf-wimse-mutual-tls-00, 2 November 2025,  
<<https://datatracker.ietf.org/doc/html/draft-ietf-wimse-mutual-tls-00>>.

[I-D.ietf-wimse-http-signature]

Salowey, J. A. and Y. Sheffer, "WIMSE Workload-to-Workload  
Authentication with HTTP Signatures", Work in Progress,  
Internet-Draft, draft-ietf-wimse-http-signature-01, 8  
January 2026, <<https://datatracker.ietf.org/doc/html/draft-ietf-wimse-http-signature-01>>.

[I-D.ietf-wimse-wpt]

Campbell, B. and A. Schwenkschuster, "WIMSE Workload Proof  
Token", Work in Progress, Internet-Draft, draft-ietf-  
wimse-wpt-00, 3 November 2025,  
<<https://datatracker.ietf.org/doc/html/draft-ietf-wimse-wpt-00>>.

## Acknowledgements

The authors would like to thank John Bradley, Kelley Burgin, Brian Campbell, Naveen CM, Andrii Deinega, Apoorva Deshpande, Daniel Fett, Evan Gilman, Joseph Heenan, Watson Ladd, Kai Lehmann, Jeff Lombardo, Dan Moore, Steinar Noem, Ashay Raut, Justin Richer, Joe Salowey, Dean Saxe, Arndt Schwenkschuster, Dag Sneeggen, Yaron Scheffer, Orie Steele, Dmitry Telegin, and Hannes Tschofenig for supporting, commenting, contributing and providing feedback on this specification.

## Document History

[[ To be removed from final specification ]] \* Remove contradiction in "request\_details" description and simpliffy normative language Clarify claim usage (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/228>).

## Since Draft 06

- \* Consistency in terms of expectations of input token (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/224>)
- \* Replace StringOrURI with string Relace StringOrURI with String (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/195>)
- \* Include token theft as a threat to be mitigated Consider information disclosure as a benefit (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/207>)
- \* Remove definition of Authorization Context Be more specific on Authorization Context (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/192>)
- \* Clarify text on use of empty parameter: <https://github.com/oauth-wg/oauth-transaction-tokens/issues/235>
- \* Clarify that workloads should ensure it is communicating with a legitimate instance of a transaction token service (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/233>)
- \* Clarify need to validate signature on subject\_token if it is signed.
- \* Clarify role of transaction tokens in call chain (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/203>)

- \* Revise normative language for enforcement of token expiry (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/210>)
- \* Remove exp field from unsigend token (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/201>)
- \* Change document category from informational to standards track (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/169>)
- \* Clarify that txnshould remain unchanged when included in a replacement transaction token.
- \* Editorial updates (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/204>)
- \* Removed the requirement to encode parameters in based64url format
- \* Rename the purpose claim to scope
- \* Enhanced the description of the sub claim addressing issue #225
- \* Removed references to replacing transaction tokens, and added a note in the Security Considerations to clarify replacement concerns.
- \* Editorial updates identified by Dan Moore (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/236>)
- \* Editorial comments from Joe Saloway (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/219>)
- \* Clarify request\_details (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/197>)
- \* Add normative language for processing Txn-Tokens (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/270>)
- \* Strengthen normative language for Txn-Token Requests (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/209>)
- \* Aligned with WIMSE terminology (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/213>)
- \* Updated Acknpwledgement section (<https://github.com/oauth-wg/oauth-transaction-tokens/issues/260>)

## Since Draft 05

- \* Strengthened prohibition on expanding TraT scope (<https://github.com/oauth-wg/oauth-transaction-tokens/pull/173>)
- \* Clarified that TraTs can exceed request token lifetime, but cannot use expired tokens in request (<https://github.com/oauth-wg/oauth-transaction-tokens/pull/170>)
- \* Improved abstract for clarity (<https://github.com/oauth-wg/oauth-transaction-tokens/pull/160>)
- \* Clarified that the HTTP header Txn-Token is unstructured (<https://github.com/oauth-wg/oauth-transaction-tokens/pull/176>)

## Since Draft 04

- \* Clarified Transaction Token Service discovery (<https://github.com/oauth-wg/oauth-transaction-tokens/pull/153>)
- \* Language improvements (<https://github.com/oauth-wg/oauth-transaction-tokens/pull/148>)
- \* Renamed azd claim to tctx claim (<https://github.com/oauth-wg/oauth-transaction-tokens/pull/150>)
- \* Fixed terminology capitalization (<https://github.com/oauth-wg/oauth-transaction-tokens/pull/151>)
- \* Added key rotation guidance (<https://github.com/oauth-wg/oauth-transaction-tokens/pull/156>)
- \* Clarified text around external vs internal invocation (<https://github.com/oauth-wg/oauth-transaction-tokens/pull/157>)

## Authors' Addresses

Atul Tulshibagwale  
SGNL  
Email: [atul@sgnl.ai](mailto:atul@sgnl.ai)

George Fletcher  
Practical Identity LLC  
Email: [george@practicalidentity.com](mailto:george@practicalidentity.com)



Pieter Kasselmann  
Defakto Security  
Email: pieter@defakto.security