

oauth  
Internet-Draft  
Intended status: Informational  
Expires: 29 January 2026

A. Tulshibagwale  
SGNL  
G. Fletcher  
Practical Identity LLC  
P. Kasselmann  
SPIRL  
28 July 2025

Transaction Tokens  
draft-ietf-oauth-transaction-tokens-06

## 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 programmatic 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 29 January 2026.

## Copyright Notice

Copyright (c) 2025 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 . . . . .	4
2. Overview . . . . .	4
2.1. What are Transaction Tokens? . . . . .	5
2.2. Creating Txn-Tokens . . . . .	5
2.2.1. Initial Creation . . . . .	5
2.2.2. Replacement Txn-Tokens . . . . .	5
2.3. Txn-Token Lifetime . . . . .	6
2.4. Benefits of Txn-Tokens . . . . .	6
2.5. Txn-Token Issuance and Usage Flows . . . . .	6
2.5.1. Basic Flow . . . . .	6
2.5.2. Replacement Txn-Token Flow . . . . .	8
3. Notational Conventions . . . . .	9
4. Terminology . . . . .	9
5. Txn-Token Format . . . . .	10
5.1. JWT Header . . . . .	10
5.2. JWT Body Claims . . . . .	10
5.2.1. Purpose claim . . . . .	11
5.2.2. Requester Context . . . . .	12
5.2.3. Transaction Context . . . . .	12
5.2.4. Example . . . . .	13
6. Txn-Token Service . . . . .	14
7. Requesting Txn-Tokens . . . . .	14
7.1. Txn-Token Request . . . . .	15
7.2. Subject Token Types . . . . .	16
7.2.1. Self-Signed Subject Token Type . . . . .	17

7.2.2. Unsigned JSON Object Subject Token Type . . . . .	17
7.3. Txn-Token Request Processing . . . . .	18
7.4. Txn-Token Response . . . . .	19
7.5. Creating Replacement Txn-Tokens . . . . .	20
7.5.1. Txn-Token Service Responsibilities . . . . .	20
7.5.2. Replacement Txn-Token Request . . . . .	20
7.5.3. Replacement Txn-Token Response . . . . .	21
7.6. Mutual Authentication of the Txn-Token Request . . . . .	21
8. Using Txn-Tokens . . . . .	22
8.1. Txn-Token HTTP Header . . . . .	22
9. Security Considerations . . . . .	22
9.1. Txn-Token Lifetime . . . . .	22
9.2. Access Tokens . . . . .	23
9.3. Subject Token Types . . . . .	23
9.4. Client Authentication . . . . .	23
9.5. Replacement Tokens . . . . .	23
9.6. Scope and Purpose processing . . . . .	23
9.7. Identifying Call Chains . . . . .	24
9.8. Transaction Token Service Discovery . . . . .	24
9.9. Workload Configuration Protection . . . . .	24
9.10. Transaction Token Service Authentication . . . . .	24
9.11. Transaction Token Service Key Rotation . . . . .	25
9.12. Transaction Tokens Are Not Authentication Credentials . . . . .	25
10. Privacy Considerations . . . . .	25
10.1. Obfuscation of Personal Information . . . . .	25
10.2. Logging . . . . .	25
11. IANA Considerations . . . . .	26
11.1. OAuth URI Subregistry Contents . . . . .	26
11.2. JWT Claims Registry Contents . . . . .	26
11.3. IANA Media Type Registration Contents . . . . .	27
11.4. HTTP Header . . . . .	28
12. References . . . . .	29
12.1. Normative References . . . . .	29
12.2. Informative References . . . . .	30
Acknowledgements . . . . .	31
Document History . . . . .	31
Since Draft 05 . . . . .	31
Since Draft 04 . . . . .	31
Contributors . . . . .	31
Authors' Addresses . . . . .	32

## 1. Introduction

Modern computing architectures often use multiple independently running components called workloads. In many cases, external invocations through externally visible 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 (external or internal) being present
- \* Arbitrary user impersonation
- \* Parameter modification or augmentation

The results of these actions are unauthorized access to resources.

## 2. Overview

Transaction Tokens (Txn-Token) are a means to mitigate damage 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. Transaction 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 computed 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 without the presence of an intentionally invoked transaction.

## 2.1. 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.

## 2.2. Creating Txn-Tokens

### 2.2.1. Initial 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. This workload then performs an OAuth 2.0 Token Exchange [RFC8693] to obtain a Txn-Token. To do this, it invokes a special Token Service (the Txn-Token Service) and provides context that is sufficient for it to generate a Txn-Token. The context information provided to the Txn-Token Service MAY include:

- \* The external authorization token (e.g., the OAuth access token)
- \* An internally generated JWT representing the subject of the request
- \* Parameters that are required to be bound for the duration of this call
- \* Additional context, such as the incoming IP address, User Agent information, or other context that can help the Txn-Token Service to issue the Txn-Token

The Txn-Token Service 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. Subsequent workloads may obtain Txn-Tokens on their own.

If the requesting service does not have an inbound token that it can use in its request to the Txn-Token Service, it generates a self-signed JWT and passes that in the request in place of the external authorization token.

### 2.2.2. Replacement Txn-Tokens

A service within a call chain may choose to replace the Txn-Token. This can typically happen if the service wants to add to the context of the current Txn-Token

To get a replacement Txn-Token, a service will request a new Txn-Token from the Txn-Token Service and provide the current Txn-Token and other parameters in the request. The Txn-Token service must be careful about the types of replacement requests it supports to avoid undermining the entire value of Txn-Tokens.

### 2.3. Txn-Token Lifetime

Txn-Tokens are expected to be short-lived (order of minutes, e.g., 5 minutes), and as a result MAY be used only for the expected duration of an external invocation. Except in the case where the request is made using a self-signed JWT, if the token or other credential presented to the Txn-Token service when requesting a Txn-Token has an expiration time, then the Txn-Token Service 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, it can use a separate mechanism to perform the external invocation, but the resulting Txn-Token is still short-lived.

### 2.4. Benefits of Txn-Tokens

Txn-Tokens help prevent spurious invocations by ensuring that a workload receiving an invocation can independently verify the user or workload on whose behalf an external call was made and any context relevant to the processing of the call.

### 2.5. Txn-Token Issuance and Usage Flows

#### 2.5.1. Basic Flow

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

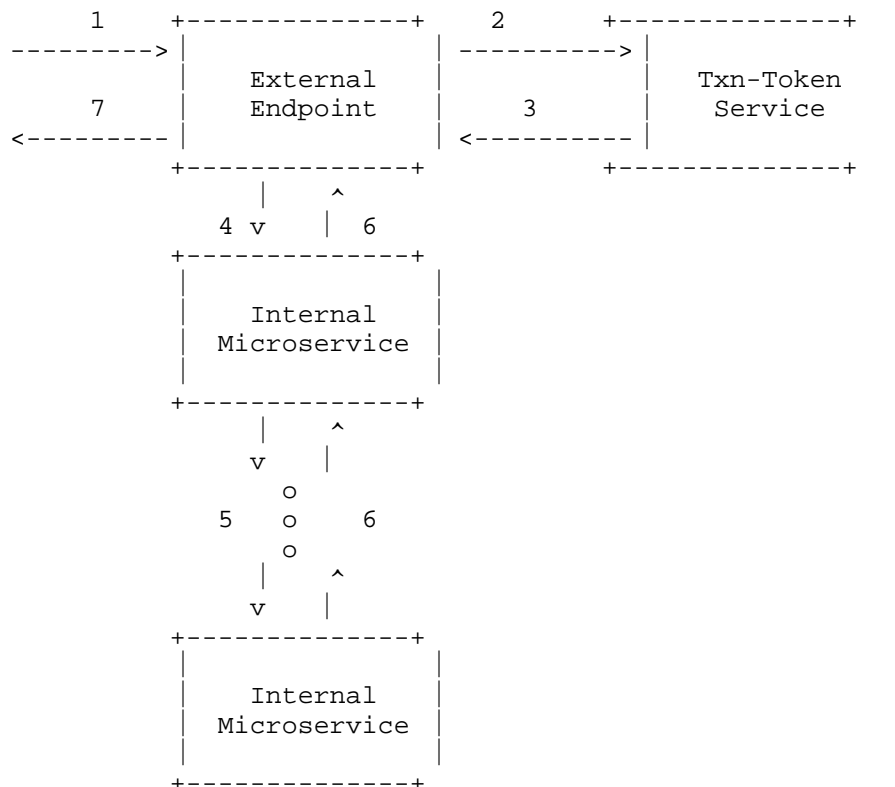


Figure 1: Basic Transaction Tokens Architecture

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 Txn-Token Service
3. Txn-Token Service 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 microservice and provides the Txn-Token as authorization
5. Subsequent calls to other internal microservices use the same Txn-Token to authorize calls
6. Responses are provided to callers based on successful authorization by the invoked microservices

7. External client is provided a response to the external invocation

#### 2.5.2. Replacement Txn-Token Flow

An intermediate service may decide to obtain a replacement Txn-Token from the Txn-Token service. That flow is described below in Figure 2

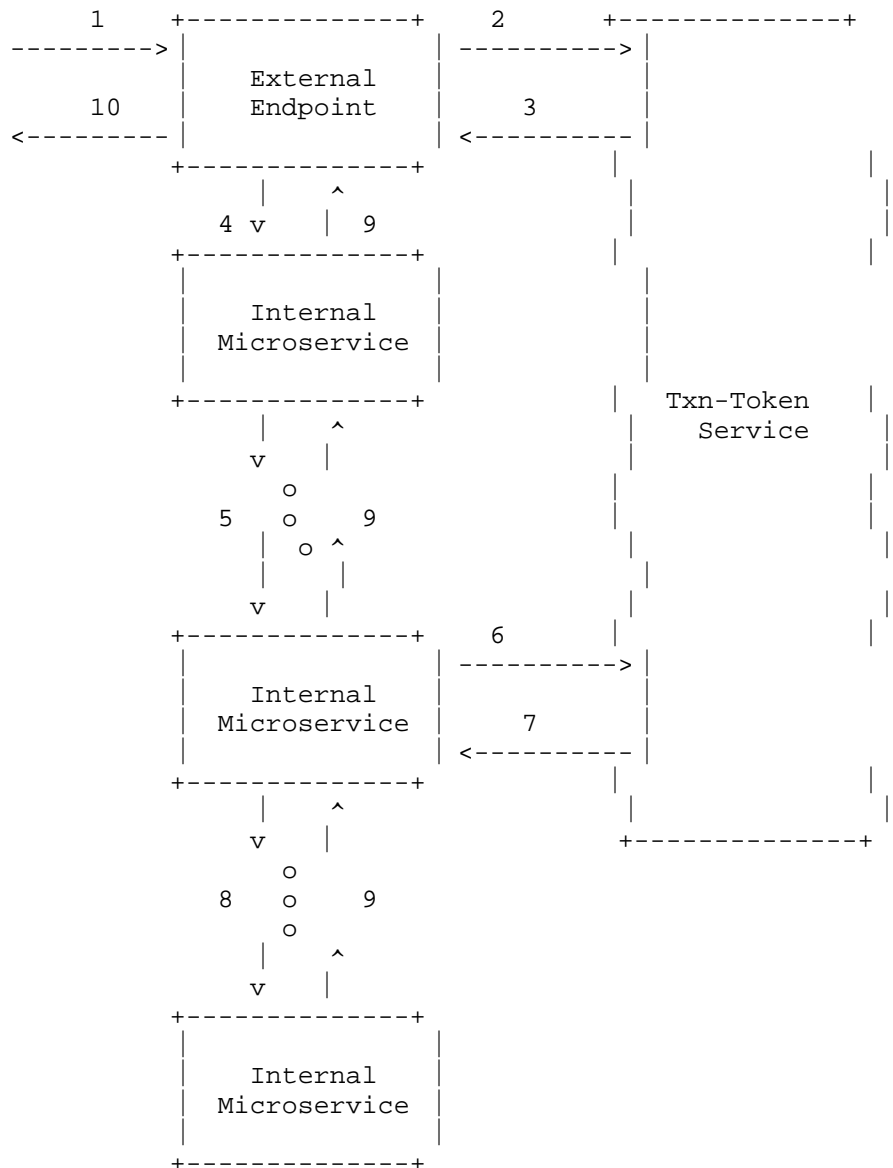




Figure 2: Replacement Txn-Token Flow

In the diagram above, steps 1-5 are the same as in Section 2.5.1

6. An intermediate service determines that it needs to obtain a Replacement Txn-Token. It requests a Replacement Txn-Token from the Txn-Token Service. It passes the incoming Txn-Token in the request, along with any additional context it needs to send the Txn-Token Service.
7. The Txn-Token Service responds with a replacement Txn-Token
8. The service that requested the Replacement Txn-Token uses that Txn-Token for downstream call authorization
9. Responses are provided to callers based on successful authorization by the invoked microservices
10. External client is provided a response to the external invocation

### 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:** An independent computational unit that can autonomously receive and process invocations, and can generate invocations of other workloads. Examples of workloads include containerized microservices, monolithic services and infrastructure services such as managed databases.

**Trust Domain:** A collection of systems, applications, or workloads that share a common security policy. 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.

**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.

Authorization Context: A JSON object containing a set of claims that represent the immutable context of a call chain.

Transaction Token Service (Txn-Token Service): 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 Txn-Token Service.

## 5. 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:

### 5.1. 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

### 5.2. 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 or it is desired that the Txn-Token Service signs with unique keys.

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

aud: REQUIRED This claim, defined in [RFC7519], identifies the Trust Domain in which the Txn-Token is valid. This identifier MUST uniquely identify the Trust Domain to prevent the Txn-Token from being accepted outside it's current Trust Domain.

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 A unique identifier for the subject within the context of the aud Trust Domain. Unlike OpenID Connect, the sub claim is NOT associated with the iss claim.

purp: REQUIRED A String defining the purpose or intent of this transaction.

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.

#### 5.2.1. Purpose claim

The purp claim captures the exact purpose of this particular transaction. This is often much narrower than a scope value issued to an external client. This is due to the fact that in most cases, the authorization model within the Trust Domain is quite different than the authorization model used with clients external to the Trust Domain. To that end, it is intentional to separate the concept of scope (often fairly coarse-grained) used with external clients from the purpose of the transaction used within the Trust Domain. How a given deployment represents the authorization model within the Trust Domain is out of scope for this specification.

### 5.2.2. Requester Context

The Txn-Token SHOULD contain an rctx claim. This MAY include the IP address information of the originating user, 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 Txn-Token Service 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 robotic process that requested the Transaction
- \* authn The authentication method used to identify the requester. Its value is a StringOrURI that uniquely identifies the method used.
- \* req\_wl The requesting workload. A StringOrURI that uniquely identifies the computational entity that requested the Txn-Token. This entity MUST be within the Trust Domain of the Txn-Token. If a replacement Txn-Token has been requested, then this claim will be an array of StringOrURIs representing the different workloads that have requested Txn-Tokens as part of the transaction processing.

### 5.2.3. 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 Txn-Token Service 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:

```
"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"
  }
}
```

#### 5.2.3.1. Requesting Workload Identifier

It is useful to be able to track the set of workloads that have requested a Txn-Token. The `req_wl` claim allows for tracking this information even through requests for a replacement Txn-Token. By default, the `req_wl` is a `StringOrURI` representing the original workload entity that requested the Txn-Token. However, if a workload within the path of servicing the transaction requests a replacement Txn-Token, then the Transaction Token Service will append the new requesting workload as a subsequent array element in the `req_wl` claim. This provides a "pathing" mechanism to track which services have requested replacement Txn-Tokens. If there is only a single value the `req_wl` will be a `StringOrURI`. If there is more than a single value, then `req_wl` will be represented by an array of `StringOrURIs`.

```
{
  "rctx": {
    "req_ip": "69.151.72.123", // env context of external call
    "authn": "urn:ietf:rfc:6749", // env context of the external call
    "req_wl": [ "apigateway.trust-domain.example", "workload3.trust-domain.example"
  ]
}
```

#### 5.2.4. Example

The figure below Figure 4 shows a non-normative example of the JWT body of a Txn-Token:

```

{
  "iat": 1686536226,
  "aud": "trust-domain.example",
  "exp": 1686536586,
  "txn": "97053963-771d-49cc-a4e3-20aad399c312",
  "sub": "d084sdrt234fsaw34tr23t",
  "rctx": {
    "req_ip": "69.151.72.123", // env context of external call
    "authn": "urn:ietf:rfc:6749", // env context of the external call
    "req_wl": "apigateway.trust-domain.example" // the internal entity that requested
the Txn-Token
  },
  "purp" : "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

## 6. Txn-Token Service

A Txn-Token Service implements a profile of the OAuth 2.0 Token Exchange [RFC8693] endpoint that can respond to Txn-Token issuance requests. This profile of the OAuth 2.0 Token Exchange [RFC8693] specification MUST be used to obtain Txn-Tokens. The unique properties of the Txn-Token requests and responses are described below. The Txn-Token Service MAY optionally support other OAuth 2.0 endpoints and features, but that is not a requirement for it to be a Txn-Token Service.

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

## 7. Requesting Txn-Tokens

A workload requests a Txn-Token from a Transaction Token Service using a profile of the OAuth 2.0 Token Exchange [RFC8693]. 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 Transaction Token Service 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.

### 7.1. Txn-Token Request

A workload requesting a Txn-Token must provide the Transaction Token Service 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 represent the subject of the transaction. This MAY be:
  - An inbound token received by 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 Txn-Token Service

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 MAY be present in a Txn-Token Request:

- \* `request_context` OPTIONAL. This parameter contains a base64url encoded JSON object which represents the context of this transaction. The parameter SHOULD be present and how the Transaction Token Service uses this parameter is out of scope for this specification.
- \* `request_details` OPTIONAL. This parameter contains a base64url encoded JSON object which represents additional details of the transaction that MUST remain immutable throughout the processing of the transaction by multiple workloads. The Transaction Token Service uses this information to construct the `tctx` claim.

The requesting workload MUST authenticate its identity to the Transaction Token Service. The exact client authentication mechanism used is outside the scope of this specification.

The figure below Figure 5 shows a non-normative example of a Txn-Token Request.

```
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=eyJhbnBfYWRkcmVzcyI6IClzMjcuMC4wLjEiLCAiY2xpZW50IjogImlvYmlsZS1hcHAiLCAiY2xpZW50X3ZlcnNpb24iOiAidjExIiB9
```

Figure 5: Example: Txn-Token Request

## 7.2. 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 Txn-Token Service. The Txn-Token Service MAY support other token formats, which MAY be specified in the `subject_token_type` parameter.

#### 7.2.1. 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. The Txn-Token Service SHALL use this value in determining the `req_wl` value in the Txn-Token issued in response to this request.
- \* `sub`: The subject for whom the Txn-Token is being requested. The Txn-Token Service 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 Txn-Token Service. The Txn-Token Service SHALL verify that this value matches its own unique identifier.
- \* `iat`: The time at which the self-signed JWT was created. Note that the Txn-Token Service 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. This should be a very short duration (order of seconds) in order to prevent any abuse of the JWT.

The self-signed JWT MAY contain other claims.

#### 7.2.2. 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 value of the `subject_token` field MUST be the BASE64URL encoded value of the JSON object as described in Section 5 of [RFC6848]. The JSON object in the subject token MUST contain the following fields:

- \* **sub**: The subject for whom the Txn-Token is being requested. The Txn-Token Service SHALL use this value in determining the sub value in the Txn-Token issued in the response to this request.
- \* **exp**: The expiration time of the unsigned JSON object, which the TTS MAY use as input to determine the lifetime of the Txn-token.

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

### 7.3. Txn-Token Request Processing

When the Transaction Token Service 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 values. The authorization policy for determining such issuance is out of scope for this specification.

Next, the Transaction Token Service MUST validate the `subject_token` and determine the value to specify as the `sub` of the issued Txn-Token. The Txn-Token Service MUST ensure the `sub` value is unique within the Trust Domain defined by the `aud` claim.

The Transaction Token Service MUST set the `iat` claim to the time of issuance of the Txn-Token.

The Transaction Token Service MUST set the `aud` claim to an identifier representing the Trust Domain of the Transaction Token Service. If the Transaction Token Service supports multiple Trust Domains, then it MUST determine the correct `aud` value for this request.

The Transaction Token Service MUST set the `exp` claim to the expiry time of the Txn-Token. The Txn-Token Service 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 Transaction Token Service MUST set the `txn` claim to a unique ID specific to this transaction.

The Transaction Token Service 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 Transaction Token Service MUST evaluate the value specified in the `scope` parameter of the request to determine the `purp` 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. In addition, the Transaction Token Service SHOULD add the authenticated requesting workload identifier in the `rctx` object as the `req_wl` claim.

If a `request_details` parameter is present in the Txn-Token Request, then the Transaction Token Service SHOULD propagate the data from the `request_details` object into the claims in the `tctx` object as authorized by the Transaction Token Service authorization policy for the requesting client.

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

#### 7.4. Txn-Token Response

A successful response to a Txn-Token Request by a Transaction Token Service is called a Txn-Token Response. If the Transaction Token Service responds with an error, the error response is as described in Section 5.2 of [RFC6749]. 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 values `expires_in`, `refresh_token` and `scope`

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

### 7.5. Creating Replacement Txn-Tokens

A workload within a call chain may request the Transaction Token Service to replace a Txn-Token.

Workloads MAY request replacement Txn-Tokens in order to change (add to, remove or modify) the asserted values within a Txn-Token.

The values of the sub and aud claims MUST remain unchanged in a replacement Txn-Token. If the claim rctx is present in the original Txn-Token, then it MUST be present and unchanged in the replacement Txn-Token except for the req\_wl claim which MUST be updated to include the requesting workload identifier.

#### 7.5.1. Txn-Token Service Responsibilities

When issuing replacement Txn-Tokens, a Txn-Token Service:

- \* 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 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 field in the rctx claim of the Txn-Token
- \* MUST NOT issue replacement Txn-token with lifetime exceeding the lifetime of the originally presented token

#### 7.5.2. Replacement Txn-Token Request

To request a replacement Txn-Token, the requester makes a Txn-Token Request as described in Section 7.1 but includes the Txn-Token to be replaced as the value of the subject\_token parameter and sets the subject\_token\_type parameter to the value urn:ietf:params:oauth:token-type:txn\_token. The scope value in the replacement request, if different from that in the original Txn-Token, MUST NOT increase the authorization surface beyond that of the original Txn-Token.

### 7.5.3. Replacement Txn-Token Response

A successful response by the Txn-Token Service to a Replacement Txn-Token Request is a Txn-Token Response as described in Section 7.4

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

A workload and Transaction Token Service MUST perform mutual authentication.

A Txn-Token Service MUST ensure that it authenticates any workloads requesting Txn-Tokens. In order to do so:

- \* It MUST maintain a limited, pre-configured set of authorized workloads that MAY request Txn-Tokens.
- \* It MUST authenticate the requesting workload and confirm that it is included in the list of workloads authorized to request a transaction token.
- \* It SHOULD accept workload credentials such as JWTs or X.509 certificates which MAY be provisioned using mechanisms such as [SPIFFE] or other provisioning protocols.
- \* It SHOULD use X.509 credentials in conjunction with MTLS [RFC8446], or a JWT protected by TLS at the transport layer, to securely authenticate the requesting workload.
- \* It SHOULD NOT rely on insecure mechanisms, such as long-lived shared secrets to authenticate the requesting workloads.

The requesting workload MUST ensure that it authenticates the Transaction Token Service. In order to do so:

- \* It MUST have a pre-configured location for the Transaction Token Service.
- \* It SHOULD accept Transaction Token Service credentials such as JWTs or X.509 certificates which MAY be provisioned using mechanisms such as [SPIFFE] or other provisioning protocols.
- \* It SHOULD use X.509 credentials in conjunction with MTLS [RFC8446], or a JWT protected by TLS at the transport layer, to securely authenticate the Transaction Token Service.
- \* It SHOULD NOT rely on insecure mechanisms, such as long-lived shared secrets to authenticate the Transaction Token Service.

## 8. 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 to communicate channel authorization.

### 8.1. 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. The value of this header MUST be the JWT that represents the Txn-Token.

## 9. Security Considerations

### 9.1. 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 Txn-Token service does not contain the `refresh_token` field. A Txn-Token cannot be issued by presenting a `refresh_token`.

The `expires_in` and `scope` fields of the OAuth 2.0 Token Exchange specification [RFC8693] are also not used in Txn-Token responses. The `expires_in` is not required since the issued token has an `exp` field, which indicates the token lifetime. The `scope` field is omitted from the response in favor of the `purp` claim in the Txn-Token.

## 9.2. 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.

## 9.3. 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 Txn-Token service.

## 9.4. Client Authentication

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

## 9.5. Replacement Tokens

Validation of a replacement Txn-Token, as well as any Txn-Token, is critical to the security of the entire transaction invocation sequence. Only Txn-Tokens issued by a trusted Transaction Token Service may be trusted, so verification of the Txn-Token signature is required. For replacement transaction tokens, not only must the JWT signature be verified but also the workload identity of the workload requesting the replacement Txn-Token.

## 9.6. Scope and Purpose processing

The authorization model within a Trust Domain boundary is most often 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 Transaction Token Service. The TTS MUST ensure that the requested purpose (scope) of the Txn-Token is equal or less than the scope(s) identified in the `subject_token`. This is also true of requesting a replacement Txn-Token. The TTS MUST ensure there is no unintentional increase in authorization scope.

### 9.7. Identifying Call Chains

A Txn-token typically represents the call-chain of workloads necessary to complete a logical function initiated by an external or internal workload. The txn claim in the Txn-token provides a unique identifier that when logged by the TTS and each subsequent workload can provide both discovery and auditability of successful and failed transactions. It is therefore strongly RECOMMENDED to use an identifier, unique within the Trust Domain, for the txn value.

### 9.8. Transaction Token Service Discovery

A workload may use various mechanisms to determine which Transaction Token Service 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 a Transaction Token Service 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.

### 9.9. Workload Configuration Protection

A workload may be configured to access more than one instance of a Transaction Token Service to ensure redundancy or reduce latency for transaction token requests. The workload configuration should be protected against unauthorized addition or removal of Transaction Token Service instances. An attacker may perform a denial of service attack or degrade the performance of a system by removing an instance of a Transaction Token Service from the workload configuration.

### 9.10. Transaction Token Service Authentication

A workload may accidentally send a transaction token request to a service that is not a Transaction Token Service, or an attacker may attempt to impersonate a Transaction Token Service 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 Transaction Token Service and only contact instances of the Transaction Token Service that is authorized to issue transaction tokens.

#### 9.11. Transaction Token Service Key Rotation

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

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

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

### 10. Privacy Considerations

#### 10.1. 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).

#### 10.2. 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 Txn-Token Service 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.

## 11. 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 5.2 in the IANA JSON Web Token Claims Registry defined in [RFC7519]. It also registers the Media Type [IANA.MediaType] "txntoken+jwt" as defined in the section Section 5.1.

### 11.1. OAuth URI Subregistry Contents

- \* URN: urn:ietf:params:oauth:token-type:txn\_token
  - Common Name: Transaction Token
  - Change Controller: IETF
  - Specification Document Section 7.1 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 7.2.1 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 7.2.2 of this specification

### 11.2. JWT Claims Registry Contents

- \* Claim Name: tctx
  - Claim Description: The transaction authorization details
  - Change Controller: IETF

- Specification Document: Section Section 5.2 of this specification
- \* Claim Name: rctx
  - Claim Description: The requester context
  - Change Controller: IETF
  - Specification Document: Section Section 5.2.2 of this specification
- \* Claim Name: purp
  - Claim Description: The purpose of the transaction
  - Change Controller: IETF
  - Specification Document: Section Section 5.2 of this specification

### 11.3. 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

#### 11.4. 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 8.1. 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:

- \* Field Name: Txn-Token
- \* Type:
- \* Status: permanent
- \* Specification Document: Section Section 8.1 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.

## 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>>.
- [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>>.
- [RFC6848] Winterbottom, J., Thomson, M., Barnes, R., Rosen, B., and R. George, "Specifying Civic Address Extensions in the Presence Information Data Format Location Object (PIDF-LO)", RFC 6848, DOI 10.17487/RFC6848, January 2013, <<https://www.rfc-editor.org/rfc/rfc6848>>.
- [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>>.
- [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>>.
- [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)>.

## 12.2. Informative References

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

## Acknowledgements

The authors would like to thank the contributors and the OAuth working group members who gave valuable input to this draft.

## Document History

[[ To be removed from final specification ]]

### 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 captialization (<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>)

## Contributors

Dr. Kelley W. Burgin, PhD.  
MITRE Corporation

Email: kburgin@mitre.org

Brian Campbell  
Ping Identity  
Email: bcampbell@pingidentity.com

Evan Gilman  
SPIRL  
Email: evan@spirl.com

Kai Lehmann  
l&l Mail & Media Development & Technology GmbH  
Email: kai.lehmann@lund1.de

Arndt Schwenkschuster  
Microsoft  
Email: arndts@microsoft.com

Hannes Tschofenig  
Arm Ltd.  
Email: Hannes.Tschofenig@arm.com

#### Authors' Addresses

Atul Tulshibagwale  
SGNL  
Email: atul@sgnl.ai

George Fletcher  
Practical Identity LLC  
Email: george@practicalidentity.com

Pieter Kasselmann  
SPIRL  
Email: pieter@spirl.com