Network Working Group G. D. Marco
Internet-Draft Dipartimento per la trasformazione digitale
Intended status: Informational O. Steele
Expires: 20 October 2024 Transmute
F. Marino
Istituto Poligrafico e Zecca dello Stato
18 April 2024

OAuth Status Attestations
draft-demarco-oauth-status-attestations-01

Abstract

Status Attestation is a signed object that demonstrates the validity status of a digital credential. These attestations are periodically provided to holders, who can present these to Verifiers along with the corresponding digital credentials. The approach outlined in this document makes the verifiers able to check the non-revocation of a digital credential without requiring to query any third-party entities.

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://peppelinux.github.io/draft-demarco-status-attestations/draft-demarco-status-attestations.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-demarco-oauth-status-attestations/.

Source for this draft and an issue tracker can be found at https://github.com/peppelinux/draft-demarco-status-attestations.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Status Attestations ensure the integrity and trustworthiness of digital credentials, whether in JSON Web Tokens (JWT) or CBOR Web Tokens (CWT) format, certifying their validity and non-revocation status. They function similarly to OCSP Stapling, allowing wallet instances to present time-stamped attestations from the Credential Issuer. The approach defined in this specification allows the verification of credentials against any revocation, without direct queries to the issuer, enhancing privacy, reducing latency, and enabling offline verification. Essential for offline scenarios, Status Attestations validate digital credentials' validity, balancing scalability, security, and privacy without internet connectivity.

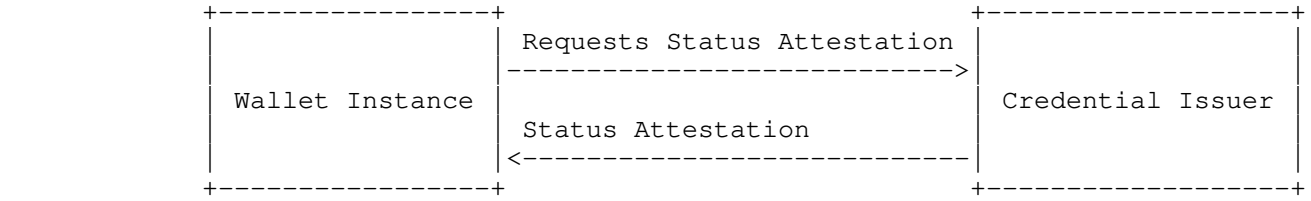


Figure 1: Status Attestation Issuance Flow

This figure illustrates the process by which a Wallet Instance requests a Status Attestation from the Credential Issuer and subsequently receives it.

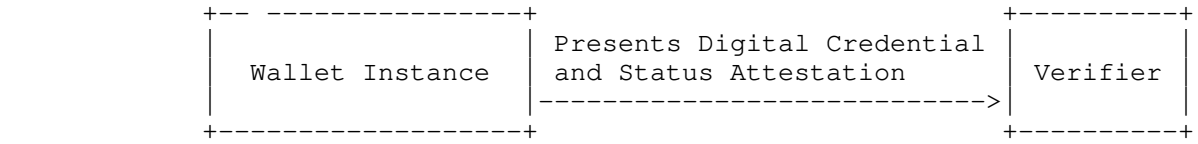


Figure 2: Status Attestation Presentation Flow

The Status Attestation is presented along with its digital credential, to prove the non-revocation status of a digital credential to a Verifier.

2. Conventions and Definitions

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.

3. Terminology

This specification uses the terms "End-User", "Entity" as defined by OpenID Connect Core [@OpenID.Core], the term "JSON Web Token (JWT)" defined by JSON Web Token (JWT) [RFC7519].

Digital Credential: A set of one or more claims about a subject made by a Credential Issuer.

Credential Issuer: Entity that is responsible for the issuance of the Digital Credentials. The Issuer is responsible for the lifecycle of their issued Digital Credentials and their validity status.

Verifier: Entity that relies on the validity of the Digital Credentials presented to it. This Entity, also known as a Relying Party, verifies the authenticity and validity of the Digital Credentials, including their revocation status, before accepting them.

Wallet Instance: The digital Wallet in control of a User, also known as Wallet or Holder. It securely stores the User’s Digital Credentials. It can present Digital Credentials to Verifiers and request Status Attestations from Issuers under the control of the User.

4. Rationale

OAuth Status Lists [@!I-D.looker-oauth-jwt-cwt-status-list] are suitable for specific scenarios, especially when the Verifier needs to verify the status of a Digital Credential at a later time after the User has presented the Digital Credential.

However, there are cases where the Verifier only needs to check the revocation status of a Digital Credential at the time of presentation, or situations where the Verifier should not be allowed to check the status of a Digital Credential over time due to some privacy constraints, in compliance with national privacy regulations.

For instance, consider a scenario under the European Union's General Data Protection Regulation (GDPR), where a Verifier's repeated access to a Status List to check the revocation status of a Digital Credential could be deemed as excessive monitoring of the End-User's activities. This could potentially infringe upon the End-User's right to privacy, as outlined in Article 8 of the European Convention on Human Rights, by creating a detailed profile of the End-User's interactions and credential usage without explicit consent for such continuous surveillance.

In scenarios where the Verifier, Credential Issuer, and OAuth Status List Provider are all part of the same domain or operate within a context where a high level of trust exists between them and the End-User, the OAuth Status List is the optimal solution; while there might be other cases where the OAuth Status List facilitates the exposure to the following privacy risks:

* An OAuth Status List Provider might know the association between a specific list and a Credential Issuer, especially if the latter issues a single type of Digital Credential. This could inadvertently reveal the Status List Provider which list corresponds to which Digital Credential.

* A Verifier retrieves an OAuth Status List by establishing a TCP/IP connection with an OAuth Status List Provider. This allows the OAuth Status List Provider to obtain the IP address of the Verifier and potentially link it to a specific Digital Credential type and Credential Issuer associated with that OAuth Status List. A malicious OAuth Status List Provider could use internet diagnostic tools, such as Whois or GeoIP lookup, to gather additional information about the Verifier. This could inadvertently disclose to the OAuth Status List Provider which Digital Credential the requester is using and from which Credential Issuer, information that should remain confidential.

Status Attestations differ significantly from OAuth Status Lists in several ways:

1. *Privacy*: Status Attestations are designed to be privacy-preserving. They do not require the Verifier to gather any additional information from third-party entities, thus preventing potential privacy leaks.

2. *Static Verification*: Status Attestations are designed to be statically provided to Verifiers by Wallet Instance. Once a Status Attestation is issued, it can be verified without any further communication with the Credential Issuer or any other party.

3. *Digital Credentials Formats*: Status Attestations are agnostic from the Digital Credential format to which they are bound.

4. *Trust Model*: Status Attestations operate under a model where the Verifier trusts the Credential Issuer to provide accurate status information, while the OAuth Status Lists operate under a model where the Verifier trusts the Status List Provider to maintain an accurate and up-to-date list of statuses.

5. *Offline flow*: OAuth Status List can be accessed by a Verifier when an internet connection is present. At the same time, OAuth Status List defines how to provide a static Status List Token, to be included within a Digital Credential. This requires the Wallet Instance to acquire a new Digital Credential for a specific presentation. Even if similar to the OAuth Status List Token, the Status Attestations enable the User to persistently use their preexistent Digital Credentials, as long as the linked Status Attestation is available and presented to the Verifier, and not expired.

5. Requirements

The general requirements for the implementation of Status Attestation are listed in this section. The Status Attestation:

* MUST be presented in conjunction with the Digital Credential. The Status Attestation MUST be timestamped with its issuance datetime, always referring to a previous period to the presentation time.

* MUST contain the expiration datetime after which the Digital Credential MUST NOT be considered valid anymore. The expiration datetime MUST be superior to the issuance datetime.

* enables offline use cases as it MUST be validated using a cryptographic signature and the cryptographic public key of the Credential Issuer.

Please note: in this specification the examples and the normative properties of Attestations are reported in accordance with the JWT standard, while for the purposes of this specification any Digital Credential or Attestation format may be used, as long as all attributes and requirements defined in this specification are satisfied, even using equivalent names or values.

6. Proof of Possession of a Credential

The concept of Proof of Possession (PoP) of a Credential within the framework of the Status Attestation specification encompasses a broader perspective than merely possessing the digital bytes of the Credential. It involves demonstrating rightful control or ownership over the Credential, which can manifest in various forms depending on the technology employed and the nature of the digital Credential itself. For instance, a Credential could be presented visually (de-visu) with a personal portrait serving as a binding element.

While this specification does not prescribe any additional methods for the proof of possession of the Credential, it aims to offer guidance for concrete implementations utilizing common proof of possession mechanisms. This includes, but is not limited to:

1. Having the digital representation of the credential (the bytes).

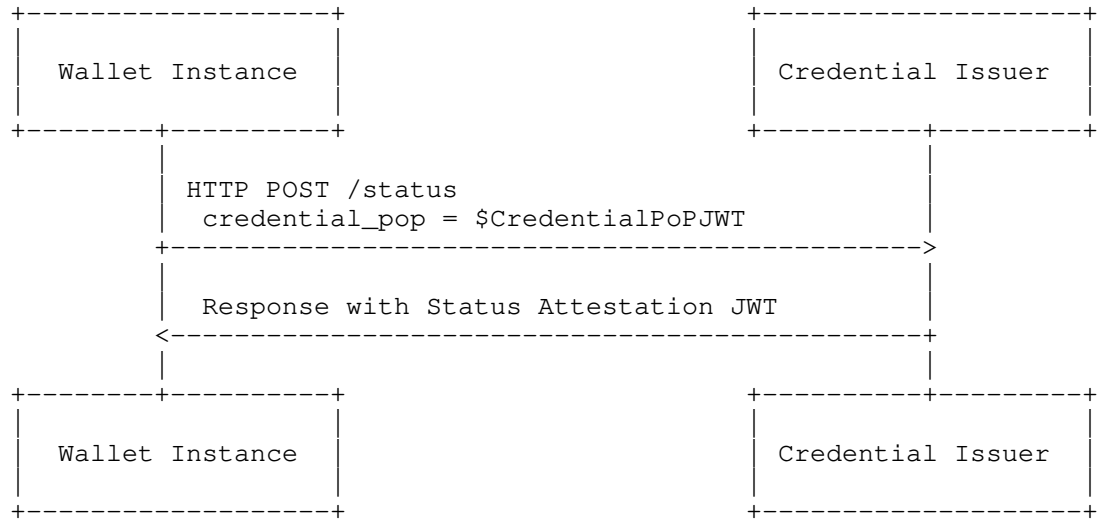
2. Controlling a private key that corresponds to a public key associated with the Credential, often indicated within the Credential's cnf (confirmation) claim or through a similar mechanism.

The essence of requiring control over the private key and its demonstration through a cryptographic operation (e.g., signing a challenge or a token) is to ensure that the entity in possession of the Credential can execute actions exclusively reserved for the legitimate subject. The dual-layered approach of requiring both possession of the Credential and control over the corresponding private key indeed reinforces the security and integrity of the status attestation process. It also ensures that the entity requesting a Status Attestation is indeed the same entity to which the Credential was originally issued, affirming the authenticity and rightful possession of the Credential.

7. Status Attestation Request

The Credential Issuer provides the Wallet Instance with a Status Attestation, which is bound to a Digital Credential. This allows the Wallet Instance to present it, along with the Digital Credential itself, to a Verifier as proof of the Digital Credential's non-revocation status.

The following diagram shows the Wallet Instance requesting a Status Attestation to a Credential Issuer, related to a specific Credential issued by the same Credential Issuer.



The Wallet Instance sends the Status Attestation request to the Credential Issuer. The request MUST contain the base64url hash value of the Digital Credential, for which the Status Attestation is requested, and enveloped in a signed object as proof of possession. The proof of possession MUST be signed with the private key corresponding to the public key attested by the Credential Issuer and contained within the Digital Credential.

```
POST /status HTTP/1.1
Host: issuer.example.org
Content-Type: application/x-www-form-urlencoded

credential_pop=$CredentialPoPJWT
```

To validate that the Wallet Instance is entitled to request its Status Attestation, the following requirements MUST be satisfied:

* The Credential Issuer MUST verify the signature of the credential_pop object using the public key contained in the Digital Credential;

* the Credential Issuer MUST verify that it is the legitimate Issuer.

The technical and details about the credential_pop object are defined in the next section.

7.1. Status Attestation Request Errors

In cases where a Status Attestation request is made for a Digital Credential that does not exist, has expired, been revoked, or is in any way invalid, or if the HTTP Request is compromised by missing or incorrect parameters, the Credential Issuer is required to respond with an HTTP Response. This response should have a status code of 400 and use application/json as the content type, including the following parameters:

* error, REQUIRED. The value must be assigned one of the error types as specified in the OAuth 2.0 RFC Section 5.2 (https://tools.ietf.org/html/rfc6749#section-5.2);

* error_description, OPTIONAL. Text in human-readable form that offers more details to clarify the nature of the error encountered (for instance, changes in some attributes, reasons for revocation, other).

Below a non-normative example of an HTTP Response with an error.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8

{
  "error":"invalid_request"
  "error_description": "The signature of credential_pop JWT is not valid"
}
```

7.2. Digital Credential Proof of Possession

The Wallet Instance that holds a Digital Credential, when requests a Status Attestation, MUST demonstrate the proof of possession of the Digital Credential to the Credential Issuer.

The proof of possession is made by enclosing the Digital Credential in an object (JWT) signed with the private key that its related public key is referenced in the Digital Credential.

Below is a non-normative example of a Credential proof of possession with the JWT headers and payload are represented without applying signature and encoding, for better readability:

```
{
    "alg": "ES256",
    "typ": "status-attestation-request+jwt",
    "kid": $CREDENTIAL-CNF-JWKID
}
.
{
    "iss": "0b434530-e151-4c40-98b7-74c75a5ef760",
    "aud": "https://issuer.example.org/status-attestation-endpoint",
    "iat": 1698744039,
    "exp": 1698834139,
    "jti": "6f204f7e-e453-4dfd-814e-9d155319408c",
    "credential_hash": $Issuer-Signed-JWT-Hash
    "credential_hash_alg": "sha-256",
}
```

When the JWT format is used, the JWT MUST contain the parameters defined in the following table.

JOSE Header	Description	Reference
typ	It MUST be set to status-attestation-request+jwt	[RFC7516] Section 4.1.1
alg	A digital signature algorithm identifier such as per IANA "JSON Web Signature and Encryption Algorithms" registry. It MUST NOT be set to none or any symmetric algorithm (MAC) identifier.	[RFC7516] Section 4.1.1
kid	Unique identifier of the JWK used for the signature of the Status Attestation Request, it MUST match the one contained in the Credential cnf.jwk.	[RFC7515]

Table 1

JOSE Payload	Description	Reference
iss	Wallet identifier.	[RFC9126], [RFC7519]
aud	It MUST be set with the Credential Issuer Status Attestation endpoint URL as value that identify the intended audience	[RFC9126], [RFC7519]
exp	UNIX Timestamp with the expiration time of the JWT.	[RFC9126], [RFC7519]
iat	UNIX Timestamp with the time of JWT issuance.	[RFC9126], [RFC7519]
jti	Unique identifier for the JWT.	[RFC7519] Section 4.1.7
credential_hash	Hash value of the Digital Credential the Status Attestation is bound to.	this specification
credential_hash_alg	The Algorithm used of hashing the Digital Credential to which the Status Attestation is bound. The value SHOULD be set to sha-256.	this specification

Table 2

8. Status Attestation

When a Status Attestation is requested to a Credential Issuer, the Issuer checks the status of the Digital Credential and creates a Status Attestation bound to it.

If the Digital Credential is valid, the Credential Issuer creates a new Status Attestation, which a non-normative example is given below.

```
{
    "alg": "ES256",
    "typ": "status-attestation+jwt",
    "kid": $ISSUER-JWKID
}
.
{
    "iss": "https://issuer.example.org",
    "iat": 1504699136,
    "exp": 1504700136,
    "credential_hash": $CREDENTIAL-HASH,
    "credential_hash_alg": "sha-256",
    "cnf": {
        "jwk": {...}
    }
}
```

The Status Attestation MUST contain the following claims when the JWT format is used.

JOSE Header	Description	Reference
alg	A digital signature algorithm identifier such as per IANA "JSON Web Signature and Encryption Algorithms" registry. It MUST NOT be set to none or to a symmetric algorithm (MAC) identifier.	[RFC7515], [RFC7517]
typ	It MUST be set to status-attestation+jwt.	[RFC7515], [RFC7517] and this specification
kid	Unique identifier of the Issuer JWK.	[RFC7515]

Table 3

JOSE Payload	Description	Reference
iss	It MUST be set to the identifier of the Issuer.	[RFC9126], [RFC7519]
iat	UNIX Timestamp with the time of the Status Attestation issuance.	[RFC9126], [RFC7519]
exp	UNIX Timestamp with the expiry time of the Status Attestation.	[RFC9126], [RFC7519]
credential_hash	Hash value of the Digital Credential the Status Attestation is bound to.	this specification
credential_hash_alg	The Algorithm used of hashing the Digital Credential to which the Status Attestation is bound. The value SHOULD be set to sha-256.	this specification
cnf	JSON object containing the cryptographic key binding. The cnf.jwk value MUST match with the one provided within the related Digital Credential.	[RFC7800] Section 3.1

Table 4

9. Status Attestation Response

If the Status Attestation is requested for a non-existent, expired, revoked or invalid Digital Credential, the Credential Issuer MUST respond with an HTTP Response with the status code set to 404.

If the Digital Credential is valid, the Credential Issuer MUST return an HTTP status code of 201 (Created), with the content type set to application/json. The response MUST include a JSON object with a member named status_attestation, which contains the Status Attestation for the Wallet Instance, as illustrated in the following non-normative example:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "status_attestation": "eyJhbGciOiJFUzI1Ni ...",
}
```

10. Credential Issuers Supporting Status Attestations

This section outlines how Credential Issuers support Status Attestations, detailing the necessary metadata and practices to integrate into their systems.

10.1. Credential Issuer Metadata

The Credential Issuers that uses the Status Attestations MUST include in their OpenID4VCI [@!OpenID.VCI] metadata the claims:

* status_attestation_endpoint. REQUIRED. It MUST be an HTTPs URL indicating the endpoint where the Wallet Instances can request Status Attestations.

* credential_hash_alg_supported. REQUIRED. The supported Algorithm used by the Wallet Instance to hash the Digital Credential for which the Status Attestation is requested, using one of the hash algorithms listed in the IANA - Named Information Hash Algorithm Registry (https://www.iana.org/assignments/named-information/named-information.xhtml#hash-alg).

10.2. Issued Digital Credentials

The Credential Issuers that uses the Status Attestations SHOULD include in the issued Digital Credentials the object status with the JSON member status_attestation set to a JSON Object containing the following member:

* credential_hash_alg. REQUIRED. The Algorithm used of hashing the Digital Credential to which the Status Attestation is bound, using one of the hash algorithms listed in the IANA - Named Information Hash Algorithm Registry (https://www.iana.org/assignments/named-information/named-information.xhtml#hash-alg). Among the hash algorithms, sha-256 is recommended and SHOULD be implemented by all systems.

The non-normative example of an unsecured payload of an SD-JWT VC is shown below.

```
{
 "vct": "https://credentials.example.com/identity_credential",
 "given_name": "John",
 "family_name": "Doe",
 "email": "johndoe@example.com",
 "phone_number": "+1-202-555-0101",
 "address": {
   "street_address": "123 Main St",
   "locality": "Anytown",
   "region": "Anystate",
   "country": "US"
 },
 "birthdate": "1940-01-01",
 "is_over_18": true,
 "is_over_21": true,
 "is_over_65": true,
 "status": {
    "status_attestation": {
        "credential_hash_alg": "sha-256",
    }
 }
}
```

10.2.1. Credential Issuer Implementation Considerations

When the Digital Credential is issued, the Credential Issuer SHOULD calculate the hash value using the algorithm specified in status.status_attestation.credential_hash_alg and store this information in its database. This practice enhances efficiency by allowing the Credential Issuer to quickly compare the requested ‘credential_hash with the pre-calculated one, when processing Status Attestation requests made by Holders.

11. Presenting Status Attestations

The Wallet Instance that provides the Status Attestations using [@OpenID4VP], SHOULD include in the vp_token JSON array, as defined in [@OpenID4VP], the Status Attestation along with the related Digital Credential.

The Verifier that receives a Digital Credential supporting the Status Attestation, SHOULD:

* Decode and validate the Digital Credential;

* check the presence of status.status_attestation in the Digital Credential. If true, the Verifier SHOULD:

  - produce the hash of the Digital Credential using the hashing algorithm configured in status.status_attestation.credential_hash_alg;

  - decode all the Status Attestations provided in the presentation, by matching the JWS Header parameter typ set to status-attestation+jwt and looking for the credential_hash value that matches with the hash produced at the previous point;

  - evaluate the validity of the Status Attestation.

Please note: The importance of checking the revocation status of Digital Credentials as a 'SHOULD' rather than a 'MUST' for a Verifier who gets Status Attestation for the Digital Credential stems from the fact that the decision of a Verifier to check the revocation status of Digital Credentials is not absolute and can be influenced by numerous variables. Consider as an example the case of age-over x; even if it has expired, it may still perform its intended purpose. As a result, the expiration status alone does not render it invalid. The adaptability recognizes that the need to verify revocation status may not always coincide with the actual usability of a Digital Credential, allowing Verifiers to examine and make educated conclusions based on a variety of scenarios.

12. Security Considerations

TODO Security

13. Privacy Considerations

In the design and implementation of Status Attestations, particular attention has been paid to privacy considerations to ensure that the system is respectful of user privacy and compliant with relevant regulations.

13.1. Privacy Consideration: Status Attestation Request Opacity

The request for a status attestation does not transmit the digital credential for which the status is being attested. Instead, it includes a proof of possession (PoP) of the credential that is only interpretable by the credential issuer who issued the digital credential for which the status attestation is requested. This PoP can be achieved through a cryptographic signature using the public key contained within the digital credential over the request. This method is essential for preventing the potential for fraudulent requests intended to mislead or disclose sensitive information to unintended parties. By separating the digital credential from the status attestation request, the system ensures that the request does not inadvertently disclose any information about the digital credential or its holder. This strategy significantly enhances the privacy and security of the system by preventing the attestation process from being used to collect information about digital credentials or their holders through deceptive requests.

13.2. Privacy Consideration: Opacity of Status Attestation Content

An important privacy consideration is how the status attestation is structured to ensure it does not reveal any information about the user or the holder of the digital credential. The status attestation is crafted to prove only the vital information needed to verify the current state of a digital credential, moving beyond simple revocation or suspension checks. This is done by focusing the attestation content on the credential’s present condition and the method for its verification, rather than on the identity of the credential’s holder. This approach is key in keeping the user’s anonymity intact, making sure that the status attestation can be applied in various verification situations without risking the privacy of the people involved.

13.3. Unlinkability and Reusability of Status Attestations

Status Attestations are designed to uphold privacy by allowing verifiers to operate independently, without the need for interaction or information disclosure to third-party entities or other verifiers. This design is pivotal in ensuring unlinkability between verifiers, where actions taken by one verifier cannot be correlated or linked to actions taken by another. Verifiers can directly validate the status of a digital credential through the Status Attestation, eliminating the need for external communication. This mechanism is key in protecting the privacy of individuals whose credentials are being verified, as it significantly reduces the risk of tracking or profiling based on verification activities across various services.

While Status Attestations facilitate unlinkability, they are not inherently "single use." The specification accommodates the batch issuance of multiple status attestations, which can be single-use. However, particularly for offline interactions, a single attestation may be utilized by numerous verifiers. This flexibility ensures that Status Attestations can support a wide range of verification scenarios, from one-time validations to repeated checks by different entities, without compromising the privacy or security of the credential holder.

13.4. Untrackability by Digital Credential Issuers and the "Phone Home" Problem

A fundamental aspect of the privacy-preserving attributes of Status Attestations is their ability to address the "phone home" problem, which is the prevention of tracking by digital credential issuers. Traditional models often require verifiers to query a central status list or contact the issuer directly, a process that can inadvertently allow issuers to track when and where a digital credential is verified. Status Attestations, however, encapsulate all necessary verification information within the attestation itself. This design choice ensures that credential issuers are unable to monitor the verification activities of their issued digital credentials, thereby significantly enhancing the privacy of the credential holder. By removing the need for real-time communication with the issuer for status checks, Status Attestations effectively prevent the issuer from tracking verification activities, further reinforcing the system’s dedication to protecting user privacy.

13.5. Minimization of Data Exposure

The Status Attestations are designed around the data minimization principle. Data minimization ensures that only the necessary information required for the scope of attesting the non revocation status of the digital credential. This minimizes the exposure of potentially sensitive data.

13.6. Resistance to Enumeration Attacks

The design of Status Attestations incorporates measures to resist enumeration attacks, where an adversary attempts to gather information by systematically verifying different combinations of data. By implementing robust cryptographic techniques and limiting the information contained in status attestations, the system reduces the feasibility of such attacks. This consideration is vital for safeguarding the privacy of the credential holders and for ensuring the integrity of the verification process.

Status Attestations are based on a privacy-by-design approach, reflecting a deliberate effort to balance security and privacy needs in the Digital Credential ecosystem.

14. IANA Considerations

14.1. JSON Web Token Claims Registration

This specification requests registration of the following Claims in the IANA "JSON Web Token Claims" registry [@IANA.JWT] established by [RFC7519].

* Claim Name: credential_format
* Claim Description: The Digital Credential format the Status Attestation is bound to.
* Change Controller: IETF
* Specification Document(s): [[(#digital-credential-proof-of-possession) of this specification]]

* Claim Name: credential
* Claim Description: The Digital Credential the Status Attestation is bound to.

* Change Controller: IETF

* Specification Document(s): [[(#digital-credential-proof-of-possession) of this specification]]

* Claim Name: credential_hash

* Claim Description: Hash value of the Digital Credential the Status Attestation is bound to.

* Change Controller: IETF

* Specification Document(s): [[(#status-attestation) of this specification]]

* Claim Name: credential_hash_alg

* Claim Description: The Algorithm used of hashing the Digital Credential to which the Status Attestation is bound.

* Change Controller: IETF

* Specification Document(s): [[(#status-attestation) of this specification]]

14.2. Media Type Registration

This section requests registration of the following media types [@RFC2046] in the "Media Types" registry [@IANA.MediaTypes] in the manner described in [@RFC6838].

To indicate that the content is an JWT-based Status List:

* Type name: application

* Subtype name: status-attestation-request+jwt

* Required parameters: n/a

* Optional parameters: n/a

* Encoding considerations: binary; A JWT-based Status Attestation Request object is a JWT; JWT values are encoded as a series of base64url-encoded values (some of which may be the empty string) separated by period ('.') characters.

* Security considerations: See (#Security) of [[this specification]]

* Interoperability considerations: n/a

* Published specification: [[this specification]]

* Applications that use this media type: Applications using [[this specification]] for updated status information of tokens

* Fragment identifier considerations: n/a

* Additional information:

  - File extension(s): n/a

  - Macintosh file type code(s): n/a

* Person & email address to contact for further information: Giuseppe De Marco, gi.demarco@innovazione.gov.it

* Intended usage: COMMON

* Restrictions on usage: none

* Author: Giuseppe De Marco, gi.demarco@innovazione.gov.it

* Change controller: IETF

* Provisional registration? No

To indicate that the content is an CWT-based Status List:

* Type name: application

* Subtype name: status-attestation+jwt

* Required parameters: n/a

* Optional parameters: n/a

* Encoding considerations: binary

* Security considerations: See (#Security) of [[this specification]]

* Interoperability considerations: n/a

* Published specification: [[this specification]]

* Applications that use this media type: Applications using [[this specification]] for status attestation of tokens and Digital Credentials

* Fragment identifier considerations: n/a

* Additional information:

  - File extension(s): n/a

  - Macintosh file type code(s): n/a

* Person & email address to contact for further information: Giuseppe De Marco, gi.demarco@innovazione.gov.it

* Intended usage: COMMON

* Restrictions on usage: none

* Author: Giuseppe De Marco, gi.demarco@innovazione.gov.it

* Change controller: IETF

* Provisional registration? No

15. 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>.

[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>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/rfc/rfc7516>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[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>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/rfc/rfc7800>.

[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>.

[RFC9126] Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", RFC 9126, DOI 10.17487/RFC9126, September 2021, <https://www.rfc-editor.org/rfc/rfc9126>.

Appendix A. Acknowledgments

We would like to thank:

* Paul Bastien

* Emanuele De Cupis

* Riccardo Iaconelli

* Victor Näslund

* Giada Sciarretta

* Amir Sharif

Appendix B. Document History

TODO changelog.

Authors' Addresses

Giuseppe De Marco
Dipartimento per la trasformazione digitale
Email: gi.demarco@innovazione.gov.it

Orie Steele
Transmute
Email: orie@transmute.industries

Francesco Marino
Istituto Poligrafico e Zecca dello Stato
Email: fa.marino@ipzs.it

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 5 September 2024

T. Looker
MATTR
P. Bastian

C. Bormann

4 March 2024

Token Status List
draft-ietf-oauth-status-list-02

Abstract

This specification defines status list data structures and processing rules for representing the status of tokens secured by JSON Object Signing and Encryption (JOSE) or CBOR Object Signing and Encryption(COSE), such as JSON Web Tokens (JWTs), CBOR Web Tokens (CWTs) and ISO mdoc. The status list token data structures themselves are also represented as JWTs or CWTs.

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://vcstuff.github.io/draft-ietf-oauth-status-list/draft-ietf-oauth-status-list.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-oauth-status-list/.

Source for this draft and an issue tracker can be found at https://github.com/vcstuff/draft-ietf-oauth-status-list.

This Internet-Draft will expire on 5 September 2024.

Table of Contents

1. Introduction

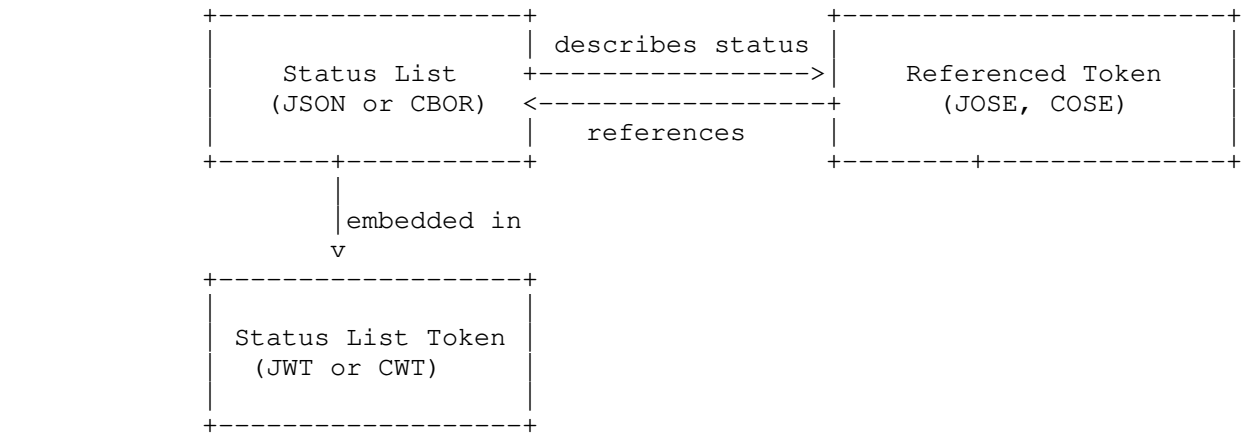
Token formats secured by JOSE [IANA.JOSE] or COSE [RFC9052], such as JSON Web Tokens (JWTs) [RFC7519], CBOR Web Tokens (CWTs) [RFC8392] and ISO mdoc [ISO.mdoc], have vast possible applications. Some of these applications can involve issuing a token whereby certain semantics about the token can change over time, which are important to be able to communicate to relying parties in an interoperable manner, such as whether the token is considered invalidated or suspended by its issuer.

This document defines a Status List and its representations in JSON and CBOR formats that describe the individual statuses of multiple Referenced Tokens, which themselves are JWTs or CWTs. The statuses of all Referenced Tokens are conveyed via a bit array in the Status List. Each Referenced Token is allocated an index during issuance that represents its position within this bit array. The value of the bit(s) at this index correspond to the Referenced Token's status. A Status List may either be provided by an endpoint or be signed and

embedded into a Status List Token, whereas this document defines its representations in JWT and CWT. Status Lists may be composed for expressing a range of Status Types. This document defines basic Status Types for the most common use cases as well as an extensibility mechanism for custom Status Types. The document also defines how an issuer of a Referenced Token references a Status List (Token).

An example for the usage of a Status List is to manage the status of issued access tokens as defined in section 1.4 of [RFC6749]. Token Introspection [RFC7662] defines another way to determine the status of an issued access token, but it requires the party trying to validate an access tokens status to directly contact the token issuer, whereas the mechanism defined in this specification does not have this limitation.

Another possible use case for the Status List is to express the status of verifiable credentials (Referenced Tokens) issued by an Issuer in the Issuer-Holder-Verifier model [SD-JWT.VC]. The following diagram depicts the basic conceptual relationship.



1.1. Rationale

Revocation mechanisms are an essential part for most identity ecosystems. In the past, revocation of X.509 TLS certificates has been proven difficult. Traditional certificate revocation lists (CRLs) have limited scalability; Online Certificate Status Protocol (OCSP) has additional privacy risks, since the client is leaking the requested website to a third party. OCSP stapling is addressing some of these problems at the cost of less up-to-date data. Modern approaches use accumulator-based revocation registries and Zero-Knowledge-Proofs to accommodate for this privacy gap, but face

scalability issues again.

This specification seeks to find a balance between scalability, security, and privacy by minimizing the status information to mere bits (often a single bit) and compressing the resulting binary data. Thereby, a Status List may contain statuses of many thousands or millions Referenced Tokens while remaining as small as possible. Placing large amounts of Referenced Tokens into the same list also enables herd privacy relative to the Issuer.

This specification establishes the IANA "Status Mechanism Methods" registry for status mechanism and registers the members defined by this specification. Other specifications can register other members used for status retrieval.

1.2. Design Considerations

The decisions taken in this specification aim to achieve the following design goals:

* the specification shall favor a simple and easy to understand concept
* the specification shall be easy, fast and secure to implement in all major programming languages
* the specification shall be optimized to support the most common use cases and avoid unnecessary complexity of corner cases
* the Status List shall scale up to millions of tokens to support large scale government or enterprise use cases
* the Status List shall enable caching policies and offline support
* the specification shall support JSON and CBOR based tokens
* the specification shall not specify key resolution or trust frameworks
* the specification shall design an extension point to convey information about the status of a token that can be re-used by other mechanisms

2. Conventions and Definitions

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.

3. Terminology

Issuer: An entity that issues the Referenced Token and provides the status information of the Referenced Token by serving a Status List Token on a public endpoint.

Relying Party: An entity that relies on the Status List to validate the status of the Referenced Token. Also known as Verifier.

Status List: An object in JSON or CBOR representation containing a bit array that lists the statuses of many Referenced Tokens.

Status List Token: A token in JWT or CWT representation that contains a cryptographically secured Status List.

Referenced Token: A cryptographically secured data structure which contains a reference to a Status List or Status List Token. It is RECOMMENDED to use JSON [RFC8259] or CBOR [RFC8949] for representation of the token and secure it using JSON Object Signing as defined in [RFC7515] or CBOR Object Signing and Encryption as defined in [RFC9052]. The information from the contained Status List may give a Relying Party additional information about up-to-date status of the Referenced Token.

4. Status List

A Status List is a byte array that contains the statuses of many Referenced Tokens represented by one or multiple bits. A common representation of a Status List is composed by the following algorithm:

1. Each status of a Referenced Token MUST be represented with a bit-size of 1,2,4, or 8. Therefore up to 2,4,16, or 256 statuses for a Referenced Token are possible, depending on the bit-size. This limitation is intended to limit bit manipulation necessary to a single byte for every operation and thus keeping implementations simpler and less error prone.

2. The overall Status List is encoded as a byte array. Depending on the bit-size, each byte corresponds to 8/(#bit-size) statuses (8,4,2, or 1). The status of each Referenced Token is identified using the index that maps to one or more specific bits within the byte array. The index starts counting at 0 and ends with "size" - 1 (being the last valid entry). The bits within an array are counted from least significant bit "0" to the most significant bit ("7"). All bits of the byte array at a particular index are set to a status value.

3. The byte array is compressed using DEFLATE [RFC1951] with the ZLIB [RFC1950] data format. Implementations are RECOMMENDED to use the highest compression level available.

The following example illustrates a Status List that represents the statuses of 16 Referenced Tokens, requiring 16 bits (2 bytes) for the uncompressed byte array:

```
status[0] = 1
status[1] = 0
status[2] = 0
status[3] = 1
status[4] = 1
status[5] = 1
status[6] = 0
status[7] = 1
status[8] = 1
status[9] = 1
status[10] = 0
status[11] = 0
status[12] = 0
status[13] = 1
status[14] = 0
status[15] = 1
```

These bits are concatenated:

```
byte             0                   1               2
bit       7 6 5 4 3 2 1 0    7 6 5 4 3 2 1 0    7
         +-+-+-+-+-+-+-+-+  +-+-+-+-+-+-+-+-+  +-+...
values   |1|0|1|1|1|0|0|1|  |1|0|1|0|0|0|1|1|  |0|...
         +-+-+-+-+-+-+-+-+  +-+-+-+-+-+-+-+-+  +-+...
index     7 6 5 4 3 2 1 0   15   ...  10 9 8   23
         \_________________/  \________________/
                0xB9                0xA3
```

4.1. Status List in JSON Format

This section defines the structure for a JSON-encoded Status List:

* status_list: REQUIRED. JSON Object that contains a Status List. The object contains exactly two claims:
  - bits: REQUIRED. JSON Integer specifying the number of bits per Referenced Token in the Status List (lst). The allowed values for bits are 1,2,4 and 8.
  - lst: REQUIRED. JSON String that contains the status values for all the Referenced Tokens it conveys statuses for. The value MUST be the base64url-encoded (as defined in Section 2 of [RFC7515]) Status List as specified in Section 4.

The following example illustrates the JSON representation of the Status List:

```
byte_array = [0xb9, 0xa3]
encoded:
{
  "bits": 1,
  "lst": "eNrbuRgAAhcBXQ"
}
```

4.2. Status List in CBOR Format

This section defines the structure for a CBOR-encoded Status List:

* The StatusList structure is a map (Major Type 5) and defines the following entries:
  - bits: REQUIRED. Unsigned int (Major Type 0) that contains the number of bits per Referenced Token in the Status List. The allowed values for bits are 1, 2, 4 and 8.
  - lst: REQUIRED. Byte string (Major Type 2) that contains the Status List as specified in Section 4.1.

The following example illustrates the CBOR representation of the Status List:

```
byte_array = [0xb9, 0xa3]
encoded:
a2646269747301636c73744a78dadbb918000217015d
```

The following is the CBOR diagnostic output of the example above:

```
a2                              # map(2)
  64                            #   string(4)
    62697473                    #     "bits"
  01                            #   uint(1)
  63                            #   string(3)
    6c7374                      #     "lst"
  4a                            #   bytes(10)
    78dadbb918000217015d        #     "xÚÛ¹\x18\x00\x02\x17\x01]"
```

5. Status List Token

A Status List Token embeds the Status List into a token that is cryptographically signed and protects the integrity of the Status List. This allows for the Status List Token to be hosted by third parties or be transferred for offline use cases.

This section specifies Status List Tokens in JSON Web Token (JWT) and CBOR Web Token (CWT) format.

5.1. Status List Token in JWT Format

The Status List Token MUST be encoded as a "JSON Web Token (JWT)" according to [RFC7519].

The following content applies to the JWT Header:

* typ: REQUIRED. The JWT type MUST be statuslist+jwt.

The following content applies to the JWT Claims Set:

* iss: REQUIRED when also present in the Referenced Token. The iss (issuer) claim MUST specify a unique string identifier for the entity that issued the Status List Token. In the absence of an application profile specifying otherwise, compliant applications MUST compare issuer values using the Simple String Comparison method defined in Section 6.2.1 of [RFC3986]. The value MUST be equal to that of the iss claim contained within the Referenced Token.

* sub: REQUIRED. The sub (subject) claim MUST specify a unique string identifier for the Status List Token. The value MUST be equal to that of the uri claim contained in the status_list claim of the Referenced Token.

* iat: REQUIRED. The iat (issued at) claim MUST specify the time at which the Status List Token was issued.

* exp: OPTIONAL. The exp (expiration time) claim, if present, MUST specify the time at which the Status List Token is considered expired by its issuer.

* ttl: OPTIONAL. The ttl (time to live) claim, if present, MUST specify the maximum amount of time, in seconds, that the Status List Token can be cached by a consumer before a fresh copy SHOULD be retrieved. The value of the claim MUST be a positive number.

* status_list: REQUIRED. The status_list (status list) claim MUST specify the Status List conforming to the rules outlined in Section 4.1.

The following additional rules apply:

1. The JWT MAY contain other claims.

2. The JWT MUST be digitally signed using an asymmetric cryptographic algorithm. Relying parties MUST reject the JWT if it is using a Message Authentication Code (MAC) algorithm. Relying parties MUST reject JWTs with an invalid signature.

3. Relying parties MUST reject JWTs that are not valid in all other respects per "JSON Web Token (JWT)" [RFC7519].

4. Application of additional restrictions and policy are at the discretion of the verifying party.

The following is a non-normative example for a Status List Token in JWT format:

```
{
  "alg": "ES256",
  "kid": "12",
  "typ": "statuslist+jwt"
}
.
{
  "exp": 2291720170,
  "iat": 1686920170,
  "iss": "https://example.com",
  "status_list": {
    "bits": 1,
    "lst": "eNrbuRgAAhcBXQ"
  },
  "sub": "https://example.com/statuslists/1"
}
```

5.2. Status List Token in CWT Format

The Status List Token MUST be encoded as a "CBOR Web Token (CWT)" according to [RFC8392].

The following content applies to the CWT protected header:

* 16 TBD (type): REQUIRED. The type of the CWT MUST be statuslist+cwt as defined in [CWT.typ].

The following content applies to the CWT Claims Set:

* 1 (issuer): REQUIRED. Same definition as iss claim in Section 5.1.

* 2 (subject): REQUIRED. Same definition as sub claim in Section 5.1.

* 6 (issued at): REQUIRED. Same definition as iat claim in Section 5.1.

* 4 (expiration time): OPTIONAL. Same definition as exp claim in Section 5.1.

* 65534 (status list): REQUIRED. The status list claim MUST specify the Status List conforming to the rules outlined in Section 4.2.

The following additional rules apply:

1. The CWT MAY contain other claims.

2. The CWT MUST be digitally signed using an asymmetric cryptographic algorithm. Relying parties MUST reject the CWT if it is using a Message Authentication Code (MAC) algorithm. Relying parties MUST reject CWTs with an invalid signature.

3. Relying parties MUST reject CWTs that are not valid in all other respects per "CBOR Web Token (CWT)" [RFC8392].

4. Application of additional restrictions and policy are at the discretion of the verifying party.

The following is a non-normative example for a Status List Token in CWT format (not including the type header yet):

```
d28453a20126106e7374617475736c6973742b637774a1044231325860a502782168
747470733a2f2f6578616d706c652e636f6d2f7374617475736c697374732f310173
68747470733a2f2f6578616d706c652e636f6d061a648c5bea041a8898dfea19fffe
56a2646269747301636c73744a78dadbb918000217015d58400f2ca3772e10b09d5d
6ed56461f7cba1a816c6234072d1bb693db277048e5db5a4e64444492a9b781d6c7a
c9714db99cc7aadb3812ec90cab7794170bab5b473
```

The following is the CBOR diagnostic output of the example above:

```
d2                              # tag(18)
  84                            #   array(4)
    53                          #     bytes(19)
      a20126106e7374617475736c  #       "¢\x01&\x10nstatusl"
      6973742b637774            #       "ist+cwt"
    a1                          #     map(1)
      04                        #       uint(4)
      42                        #       bytes(2)
        3132                    #         "12"
    58 60                       #     bytes(96)
      a502782168747470733a2f2f  #       "¥\x02x!https://"
      6578616d706c652e636f6d2f  #       "example.com/"
      7374617475736c697374732f  #       "statuslists/"
      31017368747470733a2f2f65  #       "1\x01shttps://e"
      78616d706c652e636f6d061a  #       "xample.com\x06\x1a"
      648c3fca041a8898c3ca19ff  #       "d\x8c?Ê\x04\x1a\x88\x98ÃÊ\x19ÿ"
      fe56a2646269747301636c73  #       "þV¢dbits\x01cls"
      744a78dadbb918000217015d  #       "tJxÚÛ¹\x18\x00\x02\x17\x01]"
    58 40                       #     bytes(64)
      3fd60a6d10eb4b4131f1f6c1  #       "?Ö\x0am\x10ëKA1ñöÁ"
      2fb365ae27b969e8e8df0b4f  #       "/³e®'¹ièèß\x0bO"
      4029815b679cb1051c1c9eb3  #       "@)\x81[g\x9c±\x05\x1c\x1c\x9e³"
      6aa72f6f17bcfdb5ed443bdf  #       "j§/o\x17¼ýµíD;ß"
      c2339568ab42949169b413e7  #       "Â3\x95h«B\x94\x91i´\x13ç"
      02ae1e6a                  #       "\x02®\x1ej"
```

6. Referenced Token

6.1. Status Claim

By including a "status" claim in a Referenced Token, the Issuer is referencing a mechanism to retrieve status information about this Referenced Token. The claim contains members used to reference to a status list as defined in this specification. Other members of the "status" object may be defined by other specifications. This is analogous to "cnf" claim in Section 3.1 of [RFC7800] in which different authenticity confirmation methods can be included.

6.2. Referenced Token in JWT Format

The Referenced Token MUST be encoded as a "JSON Web Token (JWT)" according to [RFC7519].

The following content applies to the JWT Claims Set:

* iss: REQUIRED when also present in the Status List Token. The iss (issuer) claim MUST specify a unique string identifier for the entity that issued the Referenced Token. In the absence of an application profile specifying otherwise, compliant applications MUST compare issuer values using the Simple String Comparison method defined in Section 6.2.1 of [RFC3986]. The value MUST be equal to that of the iss claim contained within the referenced Status List Token.

* status: REQUIRED. The status (status) claim MUST specify a JSON Object that contains at least one reference to a status mechanism.

  - status_list: REQUIRED when the status list mechanism defined in this specification is used. It contains a reference to a Status List or Status List Token. The object contains exactly two claims:

    o idx: REQUIRED. The idx (index) claim MUST specify an Integer that represents the index to check for status information in the Status List for the current Referenced Token. The value of idx MUST be a non-negative number, containing a value of zero or greater.

    o uri: REQUIRED. The uri (URI) claim MUST specify a String value that identifies the Status List or Status List Token containing the status information for the Referenced Token. The value of uri MUST be a URI conforming to [RFC3986].

Application of additional restrictions and policy are at the discretion of the verifying party.

The following is a non-normative example for a decoded header and payload of a Referenced Token:

```
{
  "alg": "ES256",
  "kid": "11"
}
.
{
  "iss": "https://example.com",
  "status": {
    "status_list": {
      "idx": 0,
      "uri": "https://example.com/statuslists/1"
    }
  }
}
```

6.3. Referenced Token in CWT Format

The Referenced Token MUST be encoded as a "COSE Web Token (CWT)" object according to [RFC8392].

The following content applies to the CWT Claims Set:

* 1 (issuer): REQUIRED. Same definition as iss claim in Section 6.2.

* 65535 (status): REQUIRED. The status claim is encoded as a Status CBOR structure and MUST include at least one data item that refers to a status mechanism. Each data item in the Status CBOR structure comprises a key-value pair, where the key must be a CBOR text string (Major Type 3) specifying the identifier of the status mechanism, and the corresponding value defines its contents. This specification defines the following data items:

  - status_list (status list): REQUIRED when the status list mechanism defined in this specification is used. It has the same definition as the status_list claim in Section 6.2 but MUST be encoded as a StatusListInfo CBOR structure with the following fields:

    o idx: REQUIRED. Same definition as idx claim in Section 6.2.

    o uri: REQUIRED. Same definition as uri claim in Section 6.2.

Application of additional restrictions and policy are at the discretion of the verifying party.

The following is a non-normative example for a decoded payload of a Referenced Token:

```
18(
    [
      / protected / << {
        / alg / 1: -7 / ES256 /
      } >>,
      / unprotected / {
        / kid / 4: h'3132' / '13' /
      },
      / payload / << {
        / iss    / 1: "https://example.com",
        / status / 65535: {
          "status_list": {
            "idx": "0",
            "uri": "https://example.com/statuslists/1"
          }
        }
      } >>,
      / signature / h'...'
    ]
  )
```

6.4. Referenced Token in other COSE/CBOR Format

The Referenced Token MUST be encoded as a COSE_Sign1 or COSE_Sign CBOR structure as defined in "CBOR Object Signing and Encryption (COSE)" [RFC9052].

It is required to encode the status mechanisms referred to in the Referenced Token using the Status CBOR structure defined in Section 6.3.

It is RECOMMENDED to use status for the label of the field that contains the Status CBOR structure.

Application of additional restrictions and policy are at the discretion of the verifying party.

The following is a non-normative example for a decoded payload of a Referenced Token:

TBD: example

7. Status Types

This document defines potential statuses of Referenced Tokens as Status Type values. If the Status List contains more than one bit per token (as defined by "bits" in the Status List), then the whole value of bits MUST describe one value. A Status List can not represent multiple statuses per Referenced Token.

The registry in this document describes the basic Status Type values required for the most common use cases. Additional values may defined for particular use cases.

7.1. Status Types Values

A status describes the state, mode, condition or stage of an entity that is described by the Status List. Status Types MUST be numeric values between 0 and 255. Status types described by this specification comprise:

* 0x00 - "VALID" - The status of the Token is valid, correct or legal.

* 0x01 - "INVALID" - The status of the Token is revoked, annulled, taken back, recalled or cancelled. This state is irreversible.

* 0x02 - "SUSPENDED" - The status of the Token is temporarily invalid, hanging, debarred from privilege. This state is reversible.

The issuer of the Status List MUST choose an adequate bits (bit size) to be able to describe the required Status Types for the application.

The processing rules for JWT or CWT precede any evaluation of a Referenced Token's status. For example, if a token is evaluated as being expired through the "exp" (Expiration Time) but also has a status of 0x00 ("VALID"), the token is considered expired.

8. Verification and Processing

8.1. Status List Request

To obtain the Status List or Status List Token, the Relying Party MUST send a HTTP GET request to the Status List Endpoint. Communication with the Status List Endpoint MUST utilize TLS. Which version(s) should be implemented will vary over time. A TLS server certificate check MUST be performed as defined in Section 5 and 6 of [RFC6125].

The Relying Party SHOULD send the following Accept-Header to indicate the requested response type:

* "application/statuslist+json" for Status List in JSON format
* "application/statuslist+jwt" for Status List in JWT format
* "application/statuslist+cbor" for Status List in CBOR format
* "application/statuslist+cwt" for Status List in CWT format

If the Relying Party does not send an Accept Header, the response type is assumed to be known implicit or out-of-band.

8.2. Status List Response

In the successful response, the Status List Provider MUST use the following content-type:

* "application/statuslist+json" for Status List in JSON format
* "application/statuslist+jwt" for Status List in JWT format
* "application/statuslist+cbor" for Status List in CBOR format
* "application/statuslist+cwt" for Status List in CWT format

In the case of "application/statuslist+json", the response MUST be of type JSON and follow the rules of Section 4.1. In the case of "application/statuslist+jwt", the response MUST be of type JWT and follow the rules of Section 5.1. In the case of "application/statuslist+cbor", the response MUST be of type CBOR and follow the rules of Section 4.2. In the case of "application/statuslist+cwt", the response MUST be of type CWT and follow the rules of Section 5.2.

The HTTP response SHOULD use gzip Content-Encoding as defined in [RFC9110].

8.3. Caching

If caching is required (e.g., to enable the use of alternative mechanisms for hosting, like Content Delivery Networks), the control of the caching mechanism SHOULD be implemented using the standard HTTP Cache-Control as defined in [RFC9111].

8.4. Validation Rules

TBD

9. Further Examples

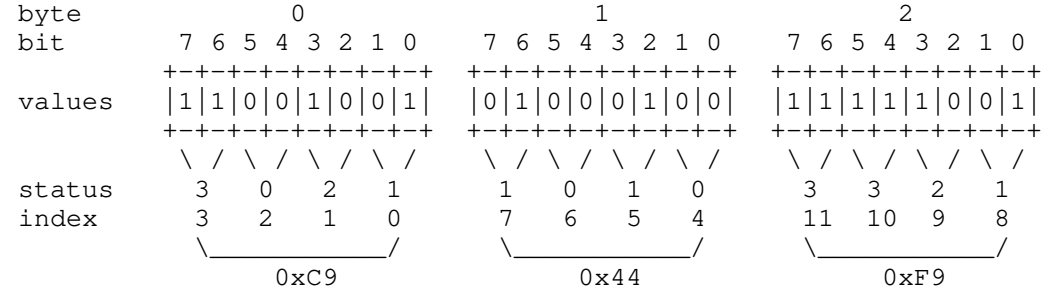
9.1. Status List Token with 2-Bit Status Values in JWT format

In this example, the Status List additionally includes the Status Type "SUSPENDED". As the Status Type value for "SUSPENDED" is 0x02 and does not fit into 1 bit, the "bits" is required to be 2.

This example Status List represents the status of 12 Referenced Tokens, requiring 24 bits (3 bytes) of status.

```
status[0] = 1
status[1] = 2
status[2] = 0
status[3] = 3
status[4] = 0
status[5] = 1
status[6] = 0
status[7] = 1
status[8] = 1
status[9] = 2
status[10] = 3
status[11] = 3
```

These bits are concatenated:



Resulting in the byte array and compressed/base64url encoded status list:

```
byte_array = [0xc9, 0x44, 0xf9]
encoded:
{
  "bits": 2,
  "lst": "eNo76fITAAPfAgc"
}
```

10. Security Considerations

10.1. Correct decoding and parsing of the encoded status list

TODO elaborate on risks of incorrect parsing/decoding leading to erroneous status data

10.2. Cached and Stale status lists

When consumers or verifiers of the Status List fetch the data, they need to be aware of its up-to-date status. The 'ttl' (time-to-live) claim in the Status List Token provides one mechanism for setting a maximum cache time for the fetched data. This property permits distribution of a status list to a CDN or other distribution mechanism while giving guidance to consumers of the status list on how often they need to fetch a fresh copy of the status list even if that status list is not expired.

10.3. Authorized access to the Status List

TODO elaborate on authorization mechanisms preventing misuse and profiling as described in privacy section

10.4. History

TODO elaborate on status list only providing the up-to date/latest status, no historical data, may be provided by the underlying hosting architecture

11. Privacy Considerations

11.1. Limiting issuers observability of token verification

The main privacy consideration for a Status List, especially in the context of the Issuer-Holder-Verifier model [SD-JWT.VC], is to prevent the Issuer from tracking the usage of the Referenced Token when the status is being checked. If an Issuer offers status information by referencing a specific token, this would enable him to create a profile for the issued token by correlating the date and identity of Relying Parties, that are requesting the status.

The Status List approaches these privacy implications by integrating the status information of many Referenced Tokens into the same list. Therefore, the Issuer does not learn for which Referenced Token the Relying Party is requesting the Status List. The privacy of the Holder is protected by the anonymity within the set of Referenced Tokens in the Status List, also called herd privacy. This limits the possibilities of tracking by the Issuer.

The herd privacy is depending on the number of entities within the Status List called its size. A larger size results in better privacy but also impacts the performance as more data has to be transferred to read the Status List.

11.2. Malicious Issuers

A malicious Issuer could bypass the privacy benefits of the herd privacy by generating a unique Status List for every Referenced Token. By these means, he could maintain a mapping between Referenced Tokens and Status Lists and thus track the usage of Referenced Tokens by utilizing this mapping for the incoming requests. This malicious behaviour could be detected by Relying Parties that request large amounts of Referenced Tokens by comparing the number of different Status Lists and their sizes.

11.3. Unobservability of Relying Parties

Once the Relying Party receives the Referenced Token, this enables him to request the Status List to validate its status through the provided uri parameter and look up the corresponding index. However, the Relying Party may persistently store the uri and index of the Referenced Token to request the Status List again at a later time. By doing so regularly, the Relying Party may create a profile of the Referenced Token’s validity status. This behaviour may be intended as a feature, e.g. for a KYC process that requires regular validity checks, but might also be abused in cases where this is not intended and unknown to the Holder, e.g. profiling the suspension of a driving license or checking the employment status of an employee credential.

This behaviour could be mitigated by: - adding authorization rules to the Status List, see Section 10.3. - regular re-issuance of the Referenced Token, see Section 12.1.

11.4. Unlinkability

Colluding Issuers and a Relying Parties have the possibility to link two transactions, as the tuple of uri and index inside the Referenced Token are unique and therefore traceable data. By comparing the status claims of received Referenced Tokens, two colluding Relying Parties could determine that they have interacted with the same user or an Issuer could trace the usage of its issued Referenced Token by colluding with various Relying Parties. It is therefore recommended to use Status Lists for Referenced Token formats that have similar unlinkability properties.

To avoid privacy risks for colluding Relying Parties, it is RECOMMENDED that Issuers use batch issuance to issue multiple tokens, see Section 12.1.

To avoid further correlatable information by the values of uri and index, Issuers are RECOMMENDED to:

* choose non-sequential, pseudo-random or random indices
* use decoy or dead entries to obfuscate the real number of Referenced Tokens within a Status List
* choose to deploy and utilize multiple Status Lists simultaneously

11.5. Third Party Hosting

TODO elaborate on increased privacy if the status list is hosted by a third party instead of the issuer reducing tracking possibilities TODO evaluate definition of Status List Provider? An entity that hosts the Status List as a resource for potential Relying Parties. The Status List Provider may be the issuer of the Status List but may also be outsourced to a trusted third party.

12. Implementation Considerations

12.1. Token Lifecycle

The lifetime of a Status List (and the Status List Token) depends on the lifetime of its Referenced Tokens. Once all Referenced Tokens are expired, the Issuer may stop serving the Status List (and the Status List Token).

Referenced Tokens may be regularly re-issued to increase security or to mitigate linkability and prevent tracking by Relying Parties. In this case, every Referenced Token MUST have a fresh Status List entry.

Referenced Tokens may also be issued in batches, such that Holders can use individual tokens for every transaction. In this case, every Referenced Token MUST have a dedicated Status List entry. Revoking batch issued Referenced Tokens might reveal this correlation later on.

13. IANA Considerations

13.1. JSON Web Token Claims Registration

This specification requests registration of the following Claims in the IANA "JSON Web Token Claims" registry [IANA.JWT] established by [RFC7519].

13.1.1. Registry Contents

* Claim Name: status
* Claim Description: Reference to a status or validity mechanism containing up-to-date status information on the JWT.
* Change Controller: IETF
* Specification Document(s): Section 6.1 of this specification
* Claim Name: status_list
* Claim Description: A status list containing up-to-date status information on multiple other JWTs encoded as a bitarray.
* Change Controller: IETF
* Specification Document(s): Section 5.1 of this specification
* Claim Name: ttl
* Claim Description: Time to Live
* Change Controller: IETF
* Specification Document(s): Section 5.1 of this specification

13.2. JWT Status Mechanism Methods Registry

This specification establishes the IANA "Status Mechanism Methods" registry for JWT "status" member values. The registry records the status mechanism method member and a reference to the specification that defines it.

13.2.1. Registration Template

Status Method Value:

> The name requested (e.g., "status_list"). The name is case sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Status Method Description:

> Brief description of the status mechanism method.

Change Controller:

> For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):

> Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

13.2.2. Initial Registry Contents

* Status Method Value: status_list
* Status Method Description: A status list containing up-to-date status information on multiple other JWTs encoded as a bitarray.
* Change Controller: IETF
* Specification Document(s): Section 6.2 of this specification

13.3. CBOR Web Token Claims Registration

This specification requests registration of the following Claims in the IANA "CBOR Web Token (CWT) Claims" registry [IANA.CWT] established by [RFC8392].

13.3.1. Registry Contents

* Claim Name: status

* Claim Description: Reference to a status or validity mechanism containing up-to-date status information on the CWT.

* Change Controller: IETF

* Specification Document(s): Section 6.1 of this specification

* Claim Name: status_list

* Claim Description: A status list containing up-to-date status information on multiple other CWTs encoded as a bitarray.

* Change Controller: IETF

* Specification Document(s): Section 5.2 of this specification

13.4. CWT Status Mechanism Methods Registry

This specification establishes the IANA "Status Mechanism Methods" registry for CWT "status" member values. The registry records the status mechanism method member and a reference to the specification that defines it.

13.4.1. Registration Template

Status Method Value:

The name requested (e.g., "status_list"). The name is case sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Status Method Description:

Brief description of the status mechanism method.

Change Controller:

For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):

Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

13.4.2. Initial Registry Contents

* Status Method Value: status_list
* Status Method Description: A status list containing up-to-date status information on multiple other CWTs encoded as a bitarray.
* Change Controller: IETF
* Specification Document(s): Section 6.3 of this specification

13.5. Media Type Registration

This section requests registration of the following media types [RFC2046] in the "Media Types" registry [IANA.MediaTypes] in the manner described in [RFC6838].

To indicate that the content is an JSON-based Status List:

* Type name: application
* Subtype name: statuslist+json
* Required parameters: n/a
* Optional parameters: n/a
* Encoding considerations: binary; A JSON-based Status List is a JSON Object.
* Security considerations: See (#Security) of [this specification]
* Interoperability considerations: n/a
* Published specification: [this specification]
* Applications that use this media type: Applications using [this specification] for updated status information of tokens
* Fragment identifier considerations: n/a
* Additional information:
  - File extension(s): n/a
  - Macintosh file type code(s): n/a

* Person & email address to contact for further information: Paul Bastian, paul.bastian@posteo.de

* Intended usage: COMMON

* Restrictions on usage: none

* Author: Paul Bastian, paul.bastian@posteo.de

* Change controller: IETF

* Provisional registration? No

To indicate that the content is an JWT-based Status List:

* Type name: application

* Subtype name: statuslist+jwt

* Required parameters: n/a

* Optional parameters: n/a

* Encoding considerations: binary; A JWT-based Status List is a JWT; JWT values are encoded as a series of base64url-encoded values (some of which may be the empty string) separated by period ('.') characters.

* Security considerations: See (#Security) of [this specification]

* Interoperability considerations: n/a

* Published specification: [this specification]

* Applications that use this media type: Applications using [this specification] for updated status information of tokens

* Fragment identifier considerations: n/a

* Additional information:

  - File extension(s): n/a

  - Macintosh file type code(s): n/a

* Person & email address to contact for further information: Paul Bastian, paul.bastian@posteo.de

* Intended usage: COMMON

* Restrictions on usage: none

* Author: Paul Bastian, paul.bastian@posteo.de

* Change controller: IETF

* Provisional registration? No

To indicate that the content is an CBOR-based Status List:

* Type name: application

* Subtype name: statuslist+cbor

* Required parameters: n/a

* Optional parameters: n/a

* Encoding considerations: binary; A CBOR-based Status List is a CBOR Object.

* Security considerations: See (#Security) of [this specification]

* Interoperability considerations: n/a

* Published specification: [this specification]

* Applications that use this media type: Applications using [this specification] for updated status information of tokens

* Fragment identifier considerations: n/a

* Additional information:

  - File extension(s): n/a

  - Macintosh file type code(s): n/a

* Person & email address to contact for further information: Paul Bastian, paul.bastian@posteo.de

* Intended usage: COMMON

* Restrictions on usage: none

* Author: Paul Bastian, paul.bastian@posteo.de

* Change controller: IETF

* Provisional registration? No

To indicate that the content is an CWT-based Status List:

* Type name: application

* Subtype name: statuslist+cwt

* Required parameters: n/a

* Optional parameters: n/a

* Encoding considerations: binary;

* Security considerations: See (#Security) of [this specification]

* Interoperability considerations: n/a

* Published specification: [this specification]

* Applications that use this media type: Applications using [this specification] for updated status information of tokens

* Fragment identifier considerations: n/a

* Additional information:

  - File extension(s): n/a

  - Macintosh file type code(s): n/a

* Person & email address to contact for further information: Paul Bastian, paul.bastian@posteo.de

* Intended usage: COMMON

* Restrictions on usage: none

* Author: Paul Bastian, paul.bastian@posteo.de

* Change controller: IETF

* Provisional registration? No

14. References

14.1. Normative References

[CWT.typ] Jones, M. B. and O. Steele, "COSE "typ" (type) Header Parameter", Work in Progress, Internet-Draft, draft-ietf-cose-typ-header-parameter-03, 26 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-typ-header-parameter-03>.

[IANA.CWT] IANA, "CBOR Web Token (CWT) Claims", n.d., <https://www.iana.org/assignments/cwt/cwt.xhtml>.

[IANA.JOSE] IANA, "JSON Object Signing and Encryption (JOSE)", n.d., <https://www.iana.org/assignments/jose/jose.xhtml>.

[IANA.JWT] IANA, "JSON Web Token Claims", n.d., <https://www.iana.org/assignments/jwt/jwt.xhtml>.

[IANA.MediaTypes] IANA, "Media Types", n.d., <https://www.iana.org/assignments/media-types/media-types.xhtml>.

[RFC1950] Deutsch, P. and J. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, DOI 10.17487/RFC1950, May 1996, <https://www.rfc-editor.org/rfc/rfc1950>.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <https://www.rfc-editor.org/rfc/rfc1951>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/rfc/rfc2046>.

[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>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/rfc/rfc6125>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[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>.

[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>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/rfc/rfc8392>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[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>.

14.2. Informative References

[ISO.mdoc] ISO/IEC JTC 1/SC 17, "ISO/IEC 18013-5:2021 ISO-compliant driving licence", n.d..

[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>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/rfc/rfc7662>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/rfc/rfc7800>.

[SD-JWT.VC] Terbu, O., Fett, D., and B. Campbell, "SD-JWT-based Verifiable Credentials (SD-JWT VC)", Work in Progress, Internet-Draft, draft-ietf-oauth-sd-jwt-vc-02, 27 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-sd-jwt-vc-02>.

Acknowledgments

We would like to thank Brian Campbell, Filip Skokan, Francesco Marino, Guiseppe De Marco, Kristina Yasuda, Michael B. Jones, Mike Prorock, Oliver Terbu, Orie Steele, Timo Glastra and Torsten Lodderstedt

for their valuable contributions, discussions and feedback to this specification.

Document History

-02

* add ttl claim to Status List Token to convey caching

* relax requirements on referenced token

* clarify Deflate / zlib compression
* make a reference to the Issuer-Holder-Verifier model of SD-JWT VC
* add COSE/CWT/CBOR encoding

-01

* Rename title of the draft
* add design consideration to the introduction
* Change status claim to in referenced token to allow re-use for other mechanisms
* Add IANA Registry for status mechanisms
* restructure the sections of this document
* add option to return an unsigned Status List
* Changing compression from gzip to zlib
* Change typo in Status List Token sub claim description
* Add access token as an example use-case

-00

* Initial draft after working group adoption
* update acknowledgments
* renamed Verifier to Relying Party
* added IANA consideration

[draft-ietf-oauth-status-list]

-01

* Applied editorial improvements suggested by Michael Jones.

-00

* Initial draft

Authors' Addresses

Tobias Looker
MATTR
Email: tobias.looker@mattr.global

Paul Bastian
Email: paul.bastian@posteo.de

Christian Bormann
Email: chris.bormann@gmx.de

Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: 22 September 2024

A. Parecki
Okta

21 March 2024

Global Token Revocation
draft-parecki-oauth-global-token-revocation-03

Abstract

Global Token Revocation enables parties such as a security incident management tool or an external Identity Provider to send a request to an Authorization Server to indicate that it should revoke all of the user's existing tokens and require that the user re-authenticates before issuing new tokens.

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.aaronpk.com/global-token-revocation/draft-parecki-oauth-global-token-revocation.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-parecki-oauth-global-token-revocation/.

Discussion of this document takes place on the Web Authorization Protocol Working Group mailing list (mailto:oauth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/oauth/. Subscribe at https://www.ietf.org/mailman/listinfo/oauth/.

Source for this draft and an issue tracker can be found at https://github.com/aaronpk/global-token-revocation.

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/.

This Internet-Draft will expire on 22 September 2024.

Table of Contents

1. Introduction

An OAuth Authorization Server issues tokens in response to a user authorizing a client. A party external to the OAuth Authorization Server may wish to instruct the Authorization Server to revoke all tokens belonging to a particular user, and prevent the server from issuing new tokens until the user re-authenticates.

For example, a security incident management tool may detect anomalous behaviour on a user's account, or if the user logged in through an enterprise Identity Provider, the Identity Provider may want to revoke all of a user's tokens in the event of a security incident or on the employee's termination.

This specification describes an API endpoint on an Authorization Server that can accept requests from external parties to revoke all tokens associated with a given user.

2. Conventions and Definitions

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.

2.1. Terminology

This specification uses the terms "Access Token", "Authorization Code", "Authorization Endpoint", "Authorization Server" (AS), "Client", "Client Authentication", "Client Identifier", "Client Secret", "End-User", "Grant Type", "Protected Resource", "Redirection URI", "Refresh Token", "Resource Owner", "Resource Server" (RS) and "Token Endpoint" defined by [RFC6749], and the terms "OpenID Provider" (OP) and "ID Token" defined by [OpenID].

This specification uses the term "Identity Provider" (IdP) to refer to the Authorization Server or OpenID Provider that is used for End-User authentication.

TODO: Replace RFC6749 references with OAuth 2.1

2.2. Roles

In a typical OAuth deployment, the OAuth client obtains tokens from the authorization server when a user logs in and authorizes the client. In many cases, the method by which a user logs in at the authorization server is through an external identity provider.

For example, a mobile chat application is an OAuth Client, and obtains tokens from its backend server which stores the chat messages. The mobile chat backend plays the OAuth roles of "Resource Server" and "Authorization Server".

In some cases, the user will log in to the Authorization Server using an external (e.g. enterprise) Identity Provider. In that case, when a user logs in to the chat application, the backend server may play the role of an OAuth client (or OpenID or SAML relying party) to the Identity Provider in a new authorization or authentication flow.

3. Token Revocation

A revocation request is a POST request to the Global Token Revocation endpoint, which starts the process of revoking all tokens for the identified subject.

3.1. Revocation Endpoint

The Global Token Revocation endpoint is a URL at the authorization server which accepts HTTP POST requests with parameters in the HTTP request message body using the application/json format. The Global Token Revocation endpoint URL MUST use the https scheme.

If the authorization server supports OAuth Server Metadata ([RFC8414]), the authorization server SHOULD include the URL of their Global Token Revocation endpoint in their authorization server metadata document using the global_token_revocation_endpoint parameter as defined in Section 5.

The authorization server MAY alternatively register the endpoint with tools that will use it.

3.2. Revocation Request

The request is a POST request with an application/json body containing a single property subject, the value of which is a Security Event Token Subject Identifier as defined in "Subject Identifiers for Security Event Tokens" [RFC9493].

In practice, this means the value of subject is a JSON object with a property format, and at least one additional property depending on the value of format.

The request MUST also be authenticated, the particular authentication method and means by which the authentication is established is out of scope of this specification, but may include OAuth 2.0 Bearer Token [RFC6750] or a JWT [RFC7523].

The following example requests that all tokens for a user identified by an email address be revoked:

```
POST /global-token-revocation
Host: example.com
Content-Type: application/json
Authorization: Bearer f5641763544a7b24b08e4f74045

{
  "sub_id": {
    "format": "email",
    "email": "user@example.com"
  }
}
```

If the user identifier at the authorization server is known by the system making the revocation request, the request can use the "Opaque Identifer" format to provide the user identifier:

```
POST /global-token-revocation
Host: example.com
Content-Type: application/json
Authorization: Bearer f5641763544a7b24b08e4f74045

{
  "sub_id": {
    "format": "opaque",
    "id": "e193177dfdc52e3dd03f78c"
  }
}
```

If it is expected that the authorization server knows about the user identifier at the IdP, the request can use the "Issuer and Subject Identifier" format:

```
POST /global-token-revocation
Host: example.com
Content-Type: application/json
Authorization: Bearer f5641763544a7b24b08e4f74045

{
  "sub_id": {
    "format": "iss_sub",
    "iss": "https://issuer.example.com/",
    "sub": "af19c476f1dc4470fa3d0d9a25"
  }
}
```

3.3. Revocation Expectations

Upon receiving a revocation request, authorizing the request, and validating the identified user, the Authorization Server:

* MUST revoke all active refresh tokens
* SHOULD invalidate all access tokens, although it is recognized that it might not be technically feasible to invalidate access tokens (see Section 4 below)
* MUST re-authenticate the user before issuing new access tokens or refresh tokens

3.4. Revocation Response

This specification indicates success and error conditions by using HTTP response codes, and does not define the response body format or content.

3.4.1. Successful Response

To indicate that the request was successful and revocation of the requested set of tokens has begun, the server returns an HTTP 204 response.

3.4.2. Error Response

The following HTTP response codes can be used to indicate various error conditions:

* *400 Bad Request*: The request was malformed, e.g. an unrecognized or unsupported type of subject identifier.
* *401 Unauthorized*: Authentication provided was invalid.

* *403 Forbidden*: Insufficient authorization, e.g. missing scopes.

* *404 User Not Found*: The user indicated by the subject identifier was not found.

* *422 Unable to Process Request*: Unable to log out the user.

4. Revocation of Access Tokens

OAuth 2.0 allows deployment flexibility with respect to the style of access tokens. The access tokens may be self-contained (e.g. [RFC9068]) so that a resource server needs no further interaction with an authorization server issuing these tokens to perform an authorization decision of the client requesting access to a protected resource. A system design may, however, instead use access tokens that are handles (also known as "reference tokens") referring to authorization data stored at the authorization server.

While these are not the only options, they illustrate the implications for revocation. In the latter case of reference tokens, the authorization server is able to revoke an access token by removing it from storage. In the former case, without storing tokens, it may be impossible to revoke tokens without taking additional measures. One such measure is to use [I-D.ietf-oauth-status-list] to maintain a distributed and easily-compressed list of token revocation statuses.

For this reason, revocation of access tokens is optional in this specification, since it may pose too significant of a burden for implementers. It is not required to revoke access tokens to be able to return a success code to the caller.

5. Authorization Server Metadata

The following authorization server metadata parameters [RFC8414] are introduced to signal the server’s capability and policy with respect to Global Token Revocation.

"global_token_revocation_endpoint": The URL of the authorization server’s global token revocation endpoint.

"global_token_revocation_endpoint_auth_methods_supported": OPTIONAL. JSON array containing a list of client authentication methods supported by this introspection endpoint. The valid client authentication method values are those registered in the IANA "OAuth Token Endpoint Authentication Methods" registry [IANA.oauth-parameters] or those registered in the IANA "OAuth Access Token Types" registry [IANA.oauth-parameters]. (These

values are and will remain distinct, due to Section 7.2.) If omitted, the set of supported authentication methods MUST be determined by other means.

6. Security Considerations

6.1. Authentication of Revocation Request

While Section 3.2 requires that the revocation request is an authenticated request, the specifics of the authentication are out of scope of this specification.

Since the revocation request ultimately has wide-reaching effects (a user is expected to be logged out of all devices), this presents a new Denial of Service attack vector. As such, the authentication used for this request SHOULD be narrowly scoped to avoid granting unnecessary privileges to the caller.

For example, if using OAuth Bearer Tokens, the token SHOULD be issued with a single scope that enables it to perform the revocation request, and no other type of token issued should include this scope.

If the authorization server is multi-tenant (supports multiple customers) through different identity providers, each identity provider SHOULD use its own scoped credential that is only authorized to revoke tokens for users within the same tenant.

6.2. Enumeration of User Accounts

Typically, an API that accepts a user identifier and returns different statuses depending on whether the user exists would provide an attack vector allowing enumeration of user accounts. This specification does require a "User Not Found" response, so would normally fall under this category. However, requests to the endpoint defined by this specification are required to be authenticated, so this is not considered a public endpoint.

If the tool making the request is compromised, and the attacker can impersonate the requests from this tool (either by coercing the tool to make the request, or by extracting the credentials), then the attacker would be able to enumerate user accounts. However, since the request is not just testing the presence of a user account, but actually revoking the tokens associated with the user if successful, this would likely be easily visible in any audit logs as many users' tokens would be revoked in a short period of time.

To mitigate some of the concerns of providing such a powerful API endpoint, the users that a particular client can request revocation for SHOULD be limited, and the authentication of the request SHOULD be used to scope the possible user revocation list to only users authorized to the client as described in Section 6.1.

For example, a multi-tenant identity provider that uses different signing keys for users associated with different tenants, can also use the same signing keys to authenticate revocation requests, such as creating a JWT to use as client authentication as described in [RFC7523]. This enables the authorization server receiving the request to only accept revocation requests for users that are associated with the particular tenant at the identity provider.

6.3. Malicious Authorization Server

From the point of view of an identity provider that supports integrations with multiple downstream applications, there is an opportunity for a downstream application to maliciously set up a Global Token Revocation endpoint to harvest user identifiers and authentication of the revocation requests.

Similarly as described in Section 6.1 above, each integration SHOULD be using separate authentication credentials, and each credential SHOULD be scoped as narrowly as possible, such that a malicious server that receives this authentication cannot replay it anywhere else to perform any actions on other systems.

7. IANA Considerations

7.1. OAuth Authorization Server Metadata

IANA has (TBD) registered the following values in the IANA "OAuth Authorization Server Metadata" registry of [IANA.oauth-parameters] established by [RFC8414].

Metadata Name: global_token_revocation_endpoint

Metadata Description: URL of the authorization server’s global token revocation endpoint.

Change Controller: IESG

Specification Document: Section X of [[this specification]]

Metadata Name:
global_token_revocation_endpoint_auth_methods_supported

Metadata Description: OPTIONAL. Indicates the list of client authentication methods supported by this endpoint.

Change Controller: IESG

Specification Document: Section X of [[this specification]]

8. References

8.1. Normative References

[IANA.oauth-parameters]
IANA, "OAuth Parameters", <http://www.iana.org/assignments/oauth-parameters>.

[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>.

[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>.

[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>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/rfc/rfc8414>.

[RFC9493] Backman, A., Ed., Scurtescu, M., and P. Jain, "Subject Identifiers for Security Event Tokens", RFC 9493, DOI 10.17487/RFC9493, December 2023, <https://www.rfc-editor.org/rfc/rfc9493>.

8.2. Informative References

[I-D.ietf-oauth-status-list]
Looker, T., Bastian, P., and C. Bormann, "Token Status List", Work in Progress, Internet-Draft, draft-ietf-oauth-status-list-02, 3 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-status-list-02>.

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

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/rfc/rfc6750>.

[RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <https://www.rfc-editor.org/rfc/rfc7009>.

[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>.

[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>.

Appendix A. Relationship to Related Specifications

A.1. RFC7009: Token Revocation

OAuth 2.0 Token Revocation [RFC7009] defines an endpoint for authorization servers that an OAuth client can use to notify the authorization server that a previously-obtained access or refresh token is no longer needed.

The request is made by the OAuth client. The input to the Token Revocation request is the token itself, as well as the client's own authentication credentials.

This differs from the Global Token Revocation endpoint which does not take a token as an input, but instead takes a user identifier as input. It is not called by OAuth clients, but is instead called by an external party such as a security monitoring tool or an identity provider that the user used to authenticate at the authorization server.

A.2. OpenID Connect Front-Channel Logout

OpenID Connect Front-Channel Logout (https://openid.net/specs/openid-connect-frontchannel-1_0.html) provides a mechanism for an OpenID Provider to log users out of Relying Parties by redirecting the user agent.

While the logout request is the same direction as this draft describes, this relies on the redirection of the user agent, so is only applicable when the user is actively interacting with the application in a web browser.

The Global Token Revocation request works regardless of whether the user is actively using the application, and is also applicable to non-web based applications.

A.3. OpenID Connect Back-Channel Logout

OpenID Connect Back-Channel Logout (https://openid.net/specs/openid-connect-backchannel-1_0.html) provides a mechanism for an OpenID Provider to log users out of a Relying Party by making a back-channel POST request containing the user identifier of the user to log out.

This is the most similar existing logout specification to Global Token Revocation. However, there are still a few key differences that make it insufficient for the use cases enabled by Global Token Revocation.

OpenID Connect Back-Channel Logout requires Relying Parties to clear state of any sessions for the user, but doesn't mention anything about access tokens. It also says that refresh tokens issued with the offline_access scope "SHOULD NOT be revoked". This is a concretely different outcome than is described by Global Token Revocation, which requires the revocation of all refresh tokens for the user regardless of whether the refresh token was issued with the offline_access scope.

Additionally, OpenID Connect Back-Channel Logout assumes that the Relying Party implements OpenID Connect, which creates implementation challenges to use it when the Relying Party actually integrates with the identity provider using other specifications such as SAML.

Global Token Revocation works regardless of the protocol that the user uses to authenticate, so works equally well with OpenID Connect and SAML integrations.

A.4. Shared Signals Framework

The Shared Signals Framework at the OpenID Foundation provides two specifications that have functionality related to session and token revocation.

Continuous Access Evaluation Profile (CAEP) (https://openid.net/specs/openid-caep-specification-1_0.html) defines several event types that can be sent between cooperating parties. In

particular, the "Session Revoked" event can be sent from an identity provider to an authorization server when the user's session at the identity provider was revoked. The main difference between this and the Global Token Revocation request is that the CAEP event is a signal that may or may not be acted upon by the receiver, whereas the Global Token Revocation request is a command that has a defined list of expected outcomes.

Risk Incident Sharing and Coordination (RISC) (https://openid.net/specs/openid-risc-profile-specification-1_0.html) defines events that have somewhat stronger defined meanings compared to CAEP. In particular, the "Account Disabled" event has clear meaning and strongly implies that a receiver should also disable the specified account. However, RISC also has a mechanism for a user to opt out of sending events for their account, so it does not provide the same level of assurance as a Global Token Revocation request.

Lastly, it is more complex to set up a receiver for CAEP and RISC events compared to a receiver for the Global Token Revocation request, so if the receiver is only interested in supporting the revocation use cases, it is much simpler to support the single POST request described in this draft.

Appendix B. Document History

((To be removed from the final specification))

-03

* Renamed property from subject to sub_id for consistency with JWT claim name defined in RFC9493

* Added reference to draft-ietf-oauth-status-list

* Added additional security considerations for authentication of the revocation request and malicious authorization servers

-02

* Added security consideration around enumeration of user accounts

* Added an appendix describing the differences between this and related logout specifications

-01

* Clarified revocation expectations

* Better definition of endpoint

* Added section defining endpoint in Authorization Server Metadata

-00

* Initial Draft

Acknowledgments

The authors would like to thank the following people for their contributions and reviews of this specification: Apoorva Deshpande, George Fletcher, Karl McGuinness, Mike Jones.

Author's Address

Aaron Parecki
Okta
Email: aaron@parecki.com
URI: https://aaronparecki.com