Stream: Internet Engineering Task Force (IETF)
RFC: 9770
Category: Standards Track
Published: May 2025
ISSN: 2070-1721
Authors: M. Tiloca, *RISE AB*; F. Palombini, *Ericsson AB*; S. Echeverria, *CMU SEI*; G. Lewis, *CMU SEI*

RFC 9770
Notification of Revoked Access Tokens in the Authentication and Authorization for Constrained Environments (ACE) Framework

Abstract

This document specifies a method of the Authentication and Authorization for Constrained Environments (ACE) framework, which allows an authorization server to notify clients and resource servers (i.e., registered devices) about revoked access tokens. As specified in this document, the method allows clients and resource servers (RSs) to access a Token Revocation List (TRL) on the authorization server by using the Constrained Application Protocol (CoAP), with the possible additional use of resource observation. Resulting (unsolicited) notifications of revoked access tokens complement alternative approaches such as token introspection, while not requiring additional endpoints on clients and RSs.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9770.

Copyright Notice

Table of Contents

1. Introduction

Authentication and Authorization for Constrained Environments (ACE) [RFC9200] is a framework that enforces access control on Internet of Things (IoT) devices acting as Resource Servers (RSs). In order to use ACE, both clients and RSs have to register with an Authorization Server (AS) and become registered devices. Once registered, a client can send a request to the AS to obtain an access token for an RS. For a client to access the RS, the client must present the issued access token at the RS, which then validates it before storing it (see Section 5.10.1.1 of [RFC9200]).

Even though access tokens have expiration times, there are circumstances by which an access token may need to be revoked before its expiration time, such as when:

1. a registered device has been compromised or is suspected of being compromised;
2. a registered device is decommissioned;
3. there has been a change in the ACE profile for a registered device;
4. there has been a change in access policies for a registered device; or
5. there has been a change in the outcome of policy evaluation for a registered device (e.g., if policy assessment depends on dynamic conditions in the execution environment, the user context, or the resource utilization).

As discussed in Section 6.1 of [RFC9200], only client-initiated revocation is currently specified [RFC7009] for OAuth 2.0 [RFC6749], based on the assumption that access tokens in OAuth are issued with a relatively short lifetime. However, this is not expected to be the case for constrained, intermittently connected devices that need access tokens with relatively long lifetimes.

This document specifies a method for allowing registered devices to access and possibly subscribe to a Token Revocation List (TRL) on the AS in order to obtain updated information about pertaining access tokens that were revoked prior to their expiration. As specified in this document, the registered devices use the Constrained Application Protocol (CoAP) [RFC7252] to communicate with the AS and with one another and can subscribe to the TRL on the AS by using resource observation for CoAP [RFC7641]. Underlying protocols other than CoAP are not prohibited from being supported in the future, if they are defined to be used in the ACE framework.

Unlike in the case of token introspection (see Section 5.9 of [RFC9200]), a registered device does not provide an owned access token to the AS for inquiring about its current state. Instead, registered devices simply obtain updated information about pertaining access tokens that were revoked prior to their expiration as efficiently identified by corresponding hash values.

The benefits of this method are that it complements token introspection and does not require the registered devices to support any additional endpoints (see Section 1.1). The only additional requirements for registered devices are a request/response interaction with the AS to access and possibly subscribe to the TRL (see Section 2) and the lightweight computation of hash values to use as access token identifiers (see Section 4).

The process by which access tokens are declared revoked is out of the scope of this document. The method by which the AS determines or is notified of revoked access tokens, according to which the AS consequently updates the TRL as specified in this document, is also out of scope.

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts described in the ACE framework [RFC9200], as well as with terms and concepts related to CBOR Web Tokens (CWTs) [RFC8392] and JSON Web Tokens (JWTs) [RFC7519].

The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes client, RS, and authorization server (AS).

Readers are also expected to be familiar with the terms and concepts related to the Concise Data Definition Language (CDDL) [RFC8610], Concise Binary Object Representation (CBOR) [RFC8949], JSON [RFC8259], CBOR Object Signing and Encryption (COSE) [RFC9052], CoAP [RFC7252], CoAP Observe [RFC7641], and the use of hash functions to name objects as defined in [RFC6920].

Note that the term "endpoint" is used here following its OAuth definition [RFC6749], aimed at denoting resources such as /token and /introspect at the AS, and /authz-info at the RS. The CoAP definition, which is "[a]n entity participating in the CoAP protocol" [RFC7252], is not used in this document.

This specification also uses the following terminology:

Token hash: identifier of an access token, in binary format encoding. The token hash has no relation to other access token identifiers possibly used, such as the 'cti' (CWT ID) claim of CBOR Web Tokens (CWTs) [RFC8392].

Token Revocation List (TRL): a collection of token hashes such that the corresponding access tokens have been revoked but are not expired yet.

TRL endpoint: an endpoint at the AS with a TRL as its representation. The default name of the TRL endpoint in a url-path is '/revoke/trl'. Implementations are not required to use this name and can define their own instead.

Registered device: a device registered at the AS, i.e., as a client, an RS, or both. A registered device acts as a requester towards the TRL endpoint.

Administrator: an entity that is authorized to get full access to the TRL at the AS and that acts as a requester towards the TRL endpoint. An administrator is not necessarily a registered device as defined above, i.e., a client requesting access tokens or an RS consuming access tokens.

An administrator might also be authorized to perform further administrative operations at the AS, e.g., through a dedicated admin interface that is out of the scope of this document. By considering the token hashes retrieved from the TRL together with other information obtained from the AS, the administrator becomes able to derive additional information, e.g., the fact that accesses have been revoked for specific registered devices.

Pertaining access token:

- With reference to an administrator, an access token issued by the AS.
- With reference to a registered device, an access token intended to be owned by that device. An access token pertains to a client if the AS has issued the access token for that client following its request. An access token pertains to an RS if the AS has issued the access token to be consumed by that RS.

Token hash pertaining to a requester: a token hash corresponding to an access token pertaining to that requester, i.e., an administrator or a registered device.

TRL update pertaining to a requester: an update to the TRL through which token hashes pertaining to that requester have been added to or removed from the TRL.

Full query: a type of query to the TRL where the AS returns the token hashes of the revoked access tokens currently in the TRL and pertaining to the requester. Further details are specified in Sections 6 and 7.

Diff query: a type of query to the TRL where the AS returns a list of diff entries, each related to one update that occurred to the TRL and containing a set of token hashes pertaining to the requester. Further details are specified in Sections 6 and 8.

See Section 4 for further terminology used throughout that section.

Examples throughout this document are expressed in CBOR diagnostic notation as defined in Section 8 of [RFC8949] and Appendix G of [RFC8610]. Diagnostic notation comments are often used to provide a textual representation of the numeric parameter names and values.

In the CBOR diagnostic notation used in this document, constructs of the form e'SOME_NAME' are replaced by the value assigned to SOME_NAME in the CDDL model shown in Figure 15 of Appendix D. For example, {e'full_set': [], e'cursor': 3} stands for {0: [], 2: 3}.

Note to RFC Editor: Please delete the paragraph immediately preceding this note. Also, in the CBOR diagnostic notation used in this document, please replace the constructs of the form e'SOME_NAME' with the value assigned to SOME_NAME in the CDDL model shown in Figure 15 of Appendix D. Finally, please delete this note.

2. Protocol Overview

This protocol defines how a CoAP-based AS informs clients and RSs, i.e., registered devices, about pertaining revoked access tokens. How the relationship between a registered device and the AS is established is out of the scope of this specification.

At a high level, the steps of this protocol are as follows:

1. Upon startup, the AS creates a single TRL accessible through the TRL endpoint. At any point in time, the TRL represents the list of all revoked access tokens issued by the AS that are not expired yet.
2. When a device registers at the AS, it also receives the url-path to the TRL endpoint.

   At any time after the registration procedure is finished, the registered device can send a GET request to the TRL endpoint at the AS. When doing so, it can request the following: the current list of pertaining revoked access tokens (see Section 7) or the most recent updates that occurred over the list of pertaining revoked access tokens (see Section 8).

   In particular, the registered device can rely on Observation for CoAP [RFC7641]. In such a case, the GET request sent to the TRL endpoint includes the CoAP Observe Option set to 0 (register), i.e., it is an Observation Request. By doing so, the registered device effectively subscribes to the TRL, as the device is interested in receiving notifications about the TRL's update. Upon receiving the Observation Request, the AS adds the registered device to the list of observers of the TRL endpoint.
3. When an access token is revoked, the AS adds the corresponding token hash to the TRL. Also, when a revoked access token eventually expires, the AS removes the corresponding token hash from the TRL.

In either case, after updating the TRL, the AS sends Observe notifications as per [RFC7641]. That is, an Observe notification is sent to each registered device that is subscribed to the TRL and to which the access token pertains.

Depending on the specific subscription established through the Observation Request, the notification provides either the current updated list of revoked access tokens in the subset of the TRL pertaining to that device (see Section 7) or the most recent TRL updates that occurred over that list of pertaining revoked access tokens (see Section 8).

Further Observe notifications may be sent, consistent with ongoing additional observations of the TRL endpoint.

4. An administrator can access and subscribe to the TRL like a registered device while getting the content of the whole TRL (see Section 7) or the most recent updates that occurred to the whole TRL (see Section 8).

Figure 1 shows a high-level overview of the service provided by this protocol. For the sake of simplicity, the example shown in the figure considers the simultaneous revocation of the three access tokens t1, t2, and t3 whose corresponding token hashes are th1, th2, and th3, respectively. Consequently, the AS adds the three token hashes to the TRL at once and sends Observe notifications to one administrator and four registered devices. Each dotted line associated with a pair of registered devices indicates the access token that they both own.

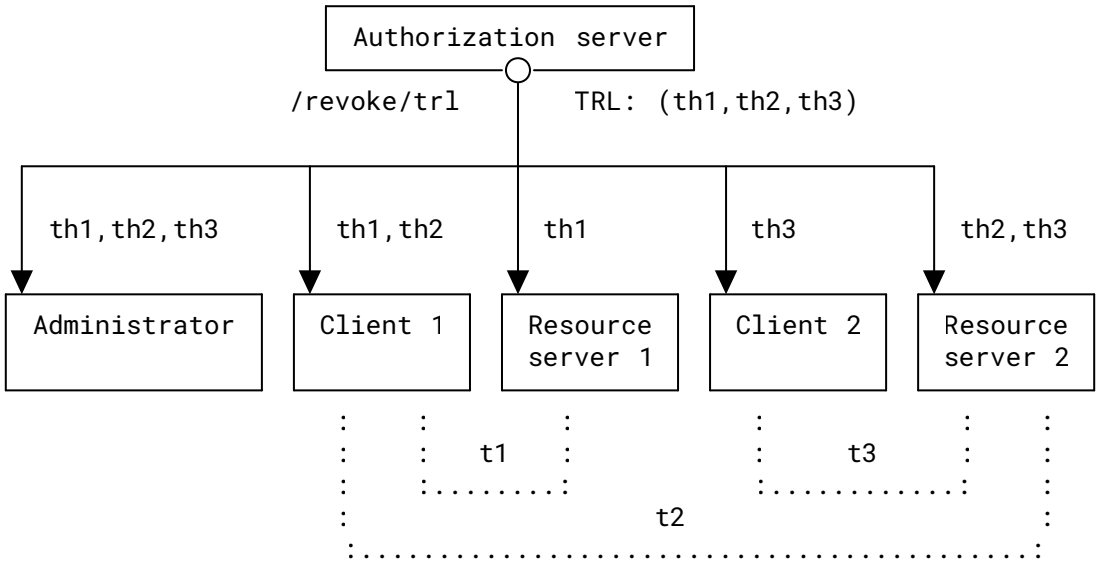

Figure 1: Protocol Overview

Appendix C provides examples of the protocol flow and message exchanges between the AS and a registered device.

3. Issuing of Access Tokens at the AS

An AS that supports the method defined in this document **MUST** adhere to the following rules when issuing an access token:

- All the intended header parameters in the access token **MUST** be specified within integrity-protected fields.
- If the access token is a CWT, the following applies:
  - Any "unprotected" field **MUST** be empty, i.e., its value **MUST** be encoded as the empty CBOR map (0xa0). This applies to the top-level "unprotected" field of the COSE object used for the CWT, the "unprotected" field of each element of the "signatures" array, and the "unprotected" field of each element of any "recipients" array (see Sections 2, 3, 4, 5, and 6 of [RFC9052]).
  - Consistent with the specific COSE object used for the CWT, the corresponding tagged structure in the set COSE_Tagged_Message **MUST** be used (see Section 2 of [RFC9052]). That is, the CBOR array that encodes the CWT **MUST** be tagged by using the COSE CBOR tag corresponding to the used COSE object. Table 1 in Section 2 of [RFC9052] specifies the tag numbers in question.

    In turn, the resulting tagged data item **MUST** be tagged by using the CWT CBOR tag with tag number 61 (see Section 6 of [RFC8392]). After that, the resulting data item **MUST NOT** be further tagged.

    Encoding of the tag numbers **MUST** be done using definite lengths, and the length of the encoded tag numbers **MUST** be the minimum possible length. This means that tag number 16 is encoded as 0xd0 and not as 0xd810.

    The example in Figure 2 shows a CWT that uses the COSE object COSE_Encrypt0 (see Section 5.2 of [RFC9052]).

- If, like for JWTs [RFC7519], the access token relies on a JSON object for encoding its claims, the following applies:

  Consistent with the ACE framework [RFC9200], this document specifically considers JWTs, which are always represented using the JSON Web Signature (JWS) Compact Serialization from [RFC7515] or the JSON Web Encryption (JWE) Compact Serialization from [RFC7516]. Consequently, all the header parameters are specified within integrity-protected fields.

  In case alternative access tokens were used, the following applies:

  - If the access token uses the JWS Compact Serialization from [RFC7515], it **MUST NOT** include the JWS Unprotected Header.
  - If the access token uses the JWE Compact Serialization from [RFC7516], it **MUST NOT** include the JWE Shared Unprotected Header and it **MUST NOT** include the "header" member in any of the elements of the "recipients" array.

```
/ CWT CBOR tag / 61(
  / COSE_Encrypt0 CBOR tag / 16(
    / COSE_Encrypt0 object / [
      / protected /   h'a3010a044c53796d6d65747269633132
                        38054d99a0d7846e762c49ffe8a63e0b',
      / unprotected / {},
      / ciphertext /  h'b918a11fd81e438b7f973d9e2e119bcb
                        22424ba0f38a80f27562f400ee1d0d6c
                        0fdb559c02421fd384fc2ebe22d70713
                        78b0ea7428fff157444d45f7e6afcda1
                        aae5f6495830c58627087fc5b4974f31
                        9a8707a635dd643b'
    ]
  )
)
```

Figure 2: Example of CWT Using COSE_Encrypt0

Section 14.6 discusses how adhering to the rules above neutralizes an attack against the RS where an active adversary can induce the RS to compute a token hash different from the correct one.

4. Token Hash

This section specifies how token hashes are computed.

First, Section 4.1 provides the motivation for the used construction.

Building on that, the value used as input to compute a token hash is defined in Section 4.2 for the client and the AS and in Section 4.3 for the RS. Finally, Section 4.4 defines how such an input is used for computing the token hash.

The process outlined below refers to the base64url encoding and decoding without padding (see Section 5 of [RFC4648]) and denotes as "binary representation" of a text string the corresponding UTF-8 encoding [RFC3629], which is the implied charset used in JSON (see Section 8.1 of [RFC8259]).

Consistent with Section 3.4 of [RFC8949], the term "tag" is used for the entire CBOR data item consisting of both a tag number and the tag content: the tag content is the CBOR data item that is being tagged.

Also, "tagged access token" is used to denote nested CBOR tags (possibly a single one), with the innermost tag content being a CWT.

4.1. Motivation for the Used Construction

An access token can have one among different formats. The most expected formats are CWT [RFC8392] and JWT [RFC7519], with the former being the default format to use in the ACE framework (see Section 3 of [RFC9200]). While access tokens are opaque to clients, an RS is aware of whether access tokens that are issued for it to consume are either CWTs or JWTs.

4.1.1. Issuing of the Access Token to the Client

There are two possible encodings that the AS can use for the AS-to-Client response (see Section 5.8.2 of [RFC9200]) where the issued access token is included and provided to the requester client. The RS may not be aware of which encoding is used for that response to that particular requester client.

- One method of encoding relies on CBOR, which is required if CoAP is used (see Section 5 of [RFC9200]) and is recommended otherwise (see Section 3 of [RFC9200]). That is, the AS-to-Client response has media-type "application/ace+cbor".

  This implies that, within the CBOR map specified as message payload, the 'access_token' parameter is a CBOR data item of type CBOR byte string and with a value of BYTES. In particular:

  - If the access token is a CWT, then BYTES is the binary representation of the CWT (i.e., of the CBOR array that encodes the untagged CWT) or of a tagged access token with the CWT as the innermost tag content.
  - If the access token is a JWT, then BYTES is the binary representation of the JWT (i.e., of the text string that encodes the JWT).

- An alternative method of encoding relies on JSON. That is, the AS-to-Client response has media-type "application/ace+json".

  This implies that, within the JSON object specified as message payload, the 'access_token' parameter has as a value a text string TEXT. In particular:

  - If the access token is a JWT, then TEXT is the text string that encodes the JWT.
  - If the access token is a CWT, then TEXT is the base64url-encoded text string of BYTES, which is the binary representation of the CWT (i.e., of the CBOR array that encodes the untagged CWT) or of a tagged access token with the CWT as the innermost tag content.

4.1.2. Provisioning of Access Tokens to the RS

In accordance with the used transport profile of ACE (e.g., [RFC9202], [RFC9203], [RFC9431]), the RS receives a piece of token-related information hereafter denoted as TOKEN_INFO.

In particular:

- If the AS-to-Client response was encoded in CBOR, then TOKEN_INFO is the value of the CBOR byte string conveyed by the 'access_token' parameter of that response. That is, TOKEN_INFO is the binary representation of the access token.

- If the AS-to-Client response was encoded in JSON and the access token is a JWT, then TOKEN_INFO is the binary representation of the text string conveyed by the 'access_token' parameter of that response. That is, TOKEN_INFO is the binary representation of the access token.
- If the AS-to-Client response was encoded in JSON and the access token is a CWT, then TOKEN_INFO is the binary representation of the base64url-encoded text string that encodes the binary representation of the access token. That is, TOKEN_INFO is the binary representation of the base64url-encoded text string conveyed by the 'access_token' parameter.

The following overviews how the above specifically applies to the existing transport profiles of ACE:

- The access token can be uploaded to the RS by means of a POST request to the /authz-info endpoint (see Section 5.10.1 of [RFC9200]), using a CoAP Content-Format or HTTP media-type that reflects the format of the access token, if available (e.g., "application/cwt" for CWTs), or "application/octet-stream" otherwise. When doing so (e.g., like in [RFC9202]), TOKEN_INFO is the payload of the POST request.
- The access token can be uploaded to the RS by means of a POST request to the /authz-info endpoint, using the media-type "application/ace+cbor". When doing so (e.g., like in [RFC9203]), TOKEN_INFO is the value of the CBOR byte string conveyed by the 'access_token' parameter, within the CBOR map specified as payload of the POST request.
- The access token can be uploaded to the RS during a DTLS session establishment, e.g., like it is defined in Section 3.2.2 of [RFC9202]. When doing so, TOKEN_INFO is the value of the 'psk_identity' field of the ClientKeyExchange message (when using DTLS 1.2 [RFC6347]) or of the 'identity' field of a PSKIdentity, within the PreSharedKeyExtension of a ClientHello message (when using DTLS 1.3 [RFC9147]).
- The access token can be uploaded to the RS within the MQTT CONNECT packet, e.g., like it is defined in Section 2.2.4.1 of [RFC9431]. When doing so, TOKEN_INFO is specified within the 'Authentication Data' field of the MQTT CONNECT packet, following the property identifier 22 (0x16) and the token length.

Note that, if the access token is a CWT, it is specifically tagged as defined in Section 3.

4.1.3. Design Rationale

Considering the possible variants discussed above, it must always be ensured that the same HASH_INPUT value is used as input for generating the token hash of a given access token, by the AS that has issued the access token and by the registered devices to which the access token pertains (both client and RS).

This is achieved by building HASH_INPUT according to the content of the 'access_token' parameter in the AS-to-Client responses because that is what the AS, the client, and the RS are all able to see.

4.2. Hash Input on the Client and the AS

The client and the AS consider the content of the 'access_token' parameter in the AS-to-Client response, in which the access token is included and provided to the requester client. Note that, if the access token is a CWT, it is specifically tagged as defined in Section 3.

The following defines how the client and the AS determine the HASH_INPUT value to use as input for computing the token hash of the conveyed access token, depending on the AS-to-Client response being encoded in CBOR (see Section 4.2.1) or in JSON (see Section 4.2.2).

Once the HASH_INPUT is determined, the client and the AS use it to compute the token hash of the conveyed access token as defined in Section 4.4.

4.2.1. AS-to-Client Response Encoded in CBOR

If the AS-to-Client response is encoded in CBOR, then HASH_INPUT is defined as follows:

- BYTES denotes the value of the CBOR byte string conveyed in the 'access_token' parameter.

  With reference to the example in Figure 3, BYTES is the bytes {0xd8, 0x3d, 0xd0, ..., 0x64, 0x3b}.

  Note that BYTES is the binary representation of the tagged access token if this is a CWT (as per Section 3) or of the access token if this is a JWT.

- HASH_INPUT_TEXT is the base64url-encoded text string that encodes BYTES.
- HASH_INPUT is the binary representation of HASH_INPUT_TEXT.

```
Header: Created (Code=2.01)
Content-Format: application/ace+cbor
Max-Age: 85800
Payload:
{
   / access_token / 1 : h'd83dd0835820a3010a044c53796d6d
                          6574726963313238054d99a0d7846e
                          762c49ffe8a63e0ba05858b918a11f
                          d81e438b7f973d9e2e119bcb22424b
                          a0f38a80f27562f400ee1d0d6c0fdb
                          559c02421fd384fc2ebe22d7071378
                          b0ea7428fff157444d45f7e6afcda1
                          aae5f6495830c58627087fc5b4974f
                          319a8707a635dd643b',
   / token_type /  34 : 2 / PoP /,
   / expires_in /   2 : 86400,
   / ace_profile / 38 : 1 / coap_dtls /,
   / (remainder of the response omitted for brevity) /
}
```

Figure 3: Example of AS-to-Client CoAP Response Using CBOR

4.2.2. AS-to-Client Response Encoded in JSON

If the AS-to-Client response is encoded in JSON, then HASH_INPUT is the binary representation of the text string conveyed by the 'access_token' parameter.

With reference to the example in Figure 4, HASH_INPUT is the binary representation of "eyJh...YFiA". When showing the access token, Figure 4 uses line breaks for display purposes only.

Note that:

- If the access token is a JWT, then HASH_INPUT is the binary representation of the JWT.
- If the access token is a CWT, then HASH_INPUT is the binary representation of a base64url-encoded text string, which encodes the binary representation of a tagged access token with the CWT as the innermost tag content (as per Section 3).

```
HTTP/1.1 200 OK
Content-Type: application/ace+json
Cache-Control: no-store
Pragma: no-cache
Payload:
{
   "access_token" : "eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJB
                     MTI4Q0JDLUhTMjU2In0.
                     QR1Owv2ug2WyPBnbQrRARTeEk9kDO2w8
                     qDcjiHnSJflSdv1iNqhWXaKH4MqAkQtM
                     oNfABIPJaZm0HaA415sv3aeuBWnD8J-U
                     i7Ah6cWafs3ZwwFKDFUUsWHSK-IPKxLG
                     TkND09XyjORj_CHAgOPJ-Sd8ONQRnJvW
                     n_hXV1BNMHzUjPyYwEsRhDhzjAD26ima
                     sOTsgruobpYGoQcXUwFDn7moXPRfDE8-
                     NoQX7N7ZYMmpUDkR-Cx9obNGwJQ3nM52
                     YCitxoQVPzjbl7WBuB7AohdBoZOdZ24W
                     lN1lVIeh8v1K4krB8xgKvRU8kgFrEn_a
                     1rZgN5TiysnmzTROF869lQ.
                     AxY8DCtDaGlsbGljb3RoZQ.
                     MKOle7UQrG6nSxTLX6Mqwt0orbHvAKeW
                     nDYvpIAeZ72deHxz3roJDXQyhxx0wKaM
                     HDjUEOKIwrtkHthpqEanSBNYHZgmNOV7
                     sln1Eu9g3J8.
                     fiK51VwhsxJ-siBMR-YFiA",
   "token_type"   : "pop",
   "expires_in"   : 86400,
   "ace_profile"  : "1"
}
```

Figure 4: Example of AS-to-Client HTTP Response Using JSON

4.3. HASH_INPUT on the RS

The following defines how the RS determines the HASH_INPUT value to use as input for computing the token hash of an access token, depending on the RS using either CWTs (see Section 4.3.1) or JWTs (see Section 4.3.2).

4.3.1. Access Tokens as CWTs

If the RS expects access tokens to be CWTs, then the RS performs the following steps:

1. The RS receives the token-related information TOKEN_INFO, in accordance with what is specified by the used profile of ACE (see Section 4.1.2).
2. The RS assumes that the client received the access token in an AS-to-Client response encoded in CBOR (see Section 4.2.1). Hence, the RS assumes TOKEN_INFO to be the binary representation of the tagged access token with the CWT as the innermost tag content (as per Section 3).
3. The RS verifies the access token as per Section 5.10.1.1 of [RFC9200]. If the verification fails, then the RS does not discard the access token yet; instead, it moves to step 4.

   Otherwise, the RS stores the access token and computes the corresponding token hash as defined in Section 4.4. In particular, the RS considers HASH_INPUT_TEXT as the base64url-encoded text string that encodes TOKEN_INFO. Then, HASH_INPUT is the binary representation of HASH_INPUT_TEXT.

   After that, the RS stores the computed token hash as associated with the access token; then, it terminates this algorithm.
4. The RS assumes that the client received the access token in an AS-to-Client response encoded in JSON (see Section 4.2.2). Hence, the RS assumes TOKEN_INFO to be the binary representation of HASH_INPUT_TEXT. In turn, HASH_INPUT_TEXT is the base64url-encoded text string that encodes the binary representation of the tagged access token with the CWT as the innermost tag content (as per Section 3).
5. The RS performs the base64url decoding of HASH_INPUT_TEXT and considers the result to be the binary representation of the tagged access token.
6. The RS verifies the access token as per Section 5.10.1.1 of [RFC9200]. If the verification fails, then the RS terminates this algorithm.

   Otherwise, the RS stores the access token and computes the corresponding token hash as defined in Section 4.4. In particular, HASH_INPUT is TOKEN_INFO.

   After that, the RS stores the computed token hash as associated with the access token.

4.3.2. Access Tokens as JWTs

If the RS expects access tokens to be JWTs, then the RS performs the following steps:

1. The RS receives the token-related information TOKEN_INFO, in accordance with what is specified by the used profile of ACE (see Section 4.1.2).

2. The RS verifies the access token as per Section 5.10.1.1 of [RFC9200]. If the verification fails, then the RS terminates this algorithm. Otherwise, the RS stores the access token.
3. The RS computes a first token hash associated with the access token as defined in Section 4.4.

   In particular, the RS assumes that the client received the access token in an AS-to-Client response encoded in JSON (see Section 4.2.2). Hence, HASH_INPUT is TOKEN_INFO.

   After that, the RS stores the computed token hash as associated with the access token.
4. The RS computes a second token hash associated with the access token as defined in Section 4.4.

   In particular, the RS assumes that the client received the access token in an AS-to-Client response encoded in CBOR (see Section 4.2.1). Hence, HASH_INPUT is the binary representation of HASH_INPUT_TEXT, which, in turn, is the base64url-encoded text string that encodes TOKEN_INFO.

   After that, the RS stores the computed token hash as associated with the access token.

The RS skips step 3 only if it is certain that all its pertaining access tokens are provided to any client by means of AS-to-Client responses encoded as CBOR messages. Otherwise, the RS **MUST** perform step 3.

The RS skips step 4 only if it is certain that all its pertaining access tokens are provided to any client by means of AS-to-Client responses encoded as JSON messages. Otherwise, the RS **MUST** perform step 4.

If the RS performs both steps 3 and 4 above, then the RS **MUST** store, maintain, and rely on both token hashes as associated with the access token, consistent with what is specified in Section 11.1.

Section 14.7 discusses how computing and storing both token hashes neutralizes an attack against the RS, where a dishonest client can induce the RS to compute a token hash different from the correct one.

4.4. Computing the Token Hash

Once HASH_INPUT is determined as defined in Sections 4.2 and 4.3, a hash value of HASH_INPUT is generated as per Section 6 of [RFC6920]. The resulting output in binary format is used as the token hash. Note that the used binary format embeds the identifier of the used hash function in the first byte of the computed token hash.

The specific hash function used **MUST** be collision resistant on byte strings and **MUST** be selected from the "Named Information Hash Algorithm Registry" [IANA.Hash.Algorithms]. Consistent with the compliance requirements in Section 2 of [RFC6920], the hash function sha-256 as specified in [SHA-256] is mandatory to implement.

The AS specifies the used hash function to registered devices during their registration procedure (see Section 10).

5. Token Revocation List (TRL)

Upon startup, the AS creates a single Token Revocation List (TRL) encoded as a CBOR array.

Each element of the array is a CBOR byte string, whose value is the token hash of an access token. The CBOR array **MUST** be treated as a set, i.e., the order of its elements has no meaning.

The TRL is initialized as empty, i.e., its initial content **MUST** be the empty CBOR array. The TRL is accessible through the TRL endpoint at the AS.

5.1. Update of the TRL

The AS updates the TRL in the following two cases:

- When a non-expired access token is revoked, the token hash of the access token is added to the TRL. That is, a CBOR byte string with the token hash as its value is added to the CBOR array encoding the TRL.
- When a revoked access token expires, the token hash of the access token is removed from the TRL. That is, the CBOR byte string with the token hash as its value is removed from the CBOR array encoding the TRL.

The AS **MAY** perform a single update to the TRL such that one or more token hashes are added or removed at once. For example, this can be the case if multiple access tokens are revoked or expire at the same time or within an acceptably narrow time frame.

6. The TRL Endpoint

Consistent with Section 6.5 of [RFC9200], all communications between the AS and a requester interacting with the TRL endpoint at the AS **MUST** be encrypted, as well as integrity and replay protected. Furthermore, responses from the AS to the requester **MUST** be bound to the corresponding requests.

Following a request to the TRL endpoint, the corresponding success response messages sent by the AS use Content-Format "application/ace-trl+cbor". Their payload is formatted as a CBOR map, and the CBOR values used to abbreviate the parameters included therein are defined in Section 12.

The AS **MUST** implement measures to prevent access to the TRL endpoint by entities other than registered devices and authorized administrators (see Section 10).

The TRL endpoint supports only the GET method, and allows two types of queries of the TRL:

1. Full query: the AS returns the token hashes of the revoked access tokens currently in the TRL and pertaining to the requester.

   The AS **MUST** support this type of query. The processing of a full query and the related response format are defined in Section 7.

2. Diff query: the AS returns a list of diff entries. Each diff entry is related to one update that occurred to the TRL, and it contains a set of token hashes pertaining to the requester. In particular, all such token hashes were added to the TRL or removed from the TRL at the update related to the diff entry in question.

   The AS **MAY** support this type of query. In such a case, the AS maintains the history of updates to the TRL as defined in Section 6.2. The processing of a diff query and the related response format are defined in Section 8.

If it supports diff queries, the AS **MAY** additionally support its "Cursor" extension, which has two benefits:

1. The AS can avoid excessively long messages when several diff entries have to be transferred by delivering several diff query responses, each containing one adjacent subset of diff entries at a time.
2. A requester can retrieve diff entries associated with TRL updates that, even if not the most recent ones, occurred after a TRL update associated with a diff entry indicated as a reference point.

If it supports the "Cursor" extension, the AS stores additional information when maintaining the history of updates to the TRL as defined in Section 6.2.1. Also, the processing of full query requests and diff query requests, as well as the related response format, are further extended as defined in Section 9.

Appendix B provides an aggregated overview of the local supportive parameters that the AS internally uses at its TRL endpoint when supporting diff queries and the "Cursor" extension.

6.1. Error Responses with Problem Details

Some error responses from the TRL endpoint at the AS can convey error-specific information according to the problem-details format defined in [RFC9290]. Such error responses **MUST** have Content-Format set to "application/concise-problem-details+cbor". The payload of these error responses **MUST** be a CBOR map specifying a Concise Problem Details data item (see Section 2 of [RFC9290]). The CBOR map is formatted as follows:

- It **MUST** include the Custom Problem Detail entry 'ace-trl-error' registered in Section 15.3 of this document. This entry is formatted as a CBOR map, which includes the following fields:
  - The 'error-id' field **MUST** be present. The map key used for this field is the CBOR unsigned integer with a value of 0. The value of this field is a CBOR integer specifying the error that occurred at the AS. This value is taken from the 'Value' column of the "ACE Token Revocation List Errors" registry defined in Section 15.5 of this document.
  - The 'cursor' field **MAY** be present. The map key used for this field is the CBOR unsigned integer with a value of 1. The value of this field is a CBOR unsigned integer or the CBOR simple value `null` (0xf6). The use of this field is defined in Section 6.3.

  The CDDL notation [RFC8610] of the 'ace-trl-error' entry is given below:

```
ace-trl-error = {
    0: int,         ; error-id
  ? 1: uint / null ; cursor
}
```

- It **MAY** include further Standard Problem Detail entries or Custom Problem Detail entries (see [RFC9290]).

  In particular, it can include the Standard Problem Detail entry 'detail' (map key -2), whose value is a CBOR text string that specifies a human-readable diagnostic description of the error that occurred at the AS. The diagnostic text is intended for software engineers as well as for device and network operators in order to aid in debugging and provide context for possible intervention. The diagnostic message **SHOULD** be logged by the AS. The 'detail' entry is unlikely to be relevant in an unattended setup where human intervention is not expected.

An example of an error response using the problem-details format is shown in Figure 5.

```
Header: Bad Request (Code=4.00)
Content-Format: application/concise-problem-details+cbor
Payload:
{
  / title /     -1: "Invalid parameter value",
  / detail /    -2: "Invalid value for 'cursor': -53",
  / ace-trl-error / e'ace-trl-error': {
    / error-id / 0: 0 / "Invalid parameter value" /,
    / cursor /   1: 42
  }
}
```

Figure 5: Example of Error Response with Problem Details

The problem-details format in general and the Custom Problem Detail entry 'ace-trl-error' in particular are **OPTIONAL** to support for registered devices. A registered device supporting the entry 'ace-trl-error' and that is able to understand the specified error may use that information to determine what actions to take next.

6.2. Supporting Diff Queries

If the AS supports diff queries, it is able to transfer a list of diff entries, each of which is related to one update that occurred to the TRL (see Section 6). That is, when replying to a diff query performed by a requester, the AS specifies the diff entries related to the most recent TRL updates pertaining to the requester.

The following defines how the AS builds and maintains an ordered list of diff entries, for each registered device and administrator, hereafter referred to as "requesters". In particular, a requester's diff entry associated with a TRL update contains a set of token hashes pertaining to that requester, each of which was added to the TRL or removed from the TRL at that update.

The AS defines the single constant positive integer MAX_N >= 1. For each requester, the AS maintains an updated collection of maximum MAX_N series items, each of which is a diff entry. For each requester, the AS **MUST** keep track of the MAX_N most recent TRL updates pertaining to the requester. If the AS supports diff queries, the AS **MUST** provide requesters with the value of MAX_N upon their registration (see Section 10).

The series of items in the update collection **MUST** be strictly chronologically ordered. That is, at any point in time, the first series item would be the one least recently added to the update collection and still retained by the AS; the last series item would be the one most recently added to the update collection. The particular method used to achieve this is implementation specific.

Each time the TRL changes, the AS performs the following operations for each requester:

1. The AS considers the subset of the TRL pertaining to that requester. If the TRL subset is not affected by this TRL update, the AS stops the processing for that requester. Otherwise, the AS moves to step 2.
2. The AS creates two trl_patch sets of token hashes, i.e., one "removed" set and one "added" set, as related to this TRL update.
3. The AS fills the two sets with the token hashes of the removed and added access tokens, respectively, from/to the TRL subset considered at step 1.
4. The AS creates a new series item that includes the two sets from step 3.
5. If the update collection associated with the requester currently includes MAX_N series items, the AS **MUST** delete the oldest series item in the update collection.
6. The AS adds the series item to the update collection associated with the requester as the last (most recent) entry.

6.2.1. Supporting the "Cursor" Extension

If it supports the "Cursor" extension for diff queries, the AS also performs the following actions:

The AS defines the single constant unsigned integer MAX_INDEX <= ((2^64) - 1). The value of MAX_INDEX is **REQUIRED** to be at least (MAX_N - 1) and is **RECOMMENDED** to be at least ((2^32) - 1). MAX_INDEX **SHOULD** be orders of magnitude greater than MAX_N.

The following applies separately for each requester's update collection:

- Each series item X in the update collection is also associated with an unsigned integer 'index', whose minimum value is 0 and whose maximum value is MAX_INDEX. The first series item ever added to the update collection **MUST** have an 'index' with a value of 0.

  If i_X is the value of 'index' associated with a series item X, then the following series item Y will take 'index' with a value of i_Y = (i_X + 1) % (MAX_INDEX + 1). That is, after having added a series item whose associated 'index' has a value of MAX_INDEX, the next added series item will result in a wraparound of the 'index' value; thus, it will take an 'index' with a value of 0.

For example, assuming MAX_N = 3, the values of 'index' in the update collection chronologically evolve as follows, as new series items are added and old series items are deleted:

- ...
- (i_A = MAX_INDEX - 2, i_B = MAX_INDEX - 1, i_C = MAX_INDEX)
- (i_B = MAX_INDEX - 1, i_C = MAX_INDEX, i_D = 0)
- (i_C = MAX_INDEX, i_D = 0, i_E = 1)
- (i_D = 0, i_E = 1, i_F = 2)
- ...

- The unsigned integer 'last_index' is also defined, with minimum value 0 and maximum value MAX_INDEX.

  If the update collection is empty (i.e., no series items have been added yet), the value of 'last_index' is not defined. If the update collection is not empty, 'last_index' has the value of 'index' currently associated with the last series item in the update collection.

  That is, after having added V series items to the update collection, the last and most recently added series item has an 'index' with a value of 'last_index' = (V - 1) % (MAX_INDEX + 1).

  As long as a wraparound of the 'index' value has not occurred, the value of 'last_index' is the absolute counter of series items added to that update collection, minus 1.

When processing a diff query using the "Cursor" extension, the values of 'index' are used as cursor information, as defined in Section 9.2.

For each requester's update collection, the AS also defines a constant positive integer MAX_DIFF_BATCH <= MAX_N, whose value specifies the maximum number of diff entries to be included in a single diff query response. The specific value **MAY** depend on the specific registered device or administrator associated with the update collection in question. If supporting the "Cursor" extension, the AS **MUST** provide registered devices and administrators with the corresponding value of MAX_DIFF_BATCH upon their registration (see Section 10).

6.3. Query Parameters

A GET request to the TRL endpoint can include the following query parameters. The AS **MUST** silently ignore unknown query parameters.

- 'diff': if included, perform a diff query of the TRL (see Section 8). Its value **MUST** be either:
  - the integer 0, indicating that a (notification) response should include as many diff entries as the AS can provide in the response; or
  - a positive integer strictly greater than 0, indicating the maximum number of diff entries that a (notification) response should include.

  If the AS does not support diff queries, it ignores the 'diff' query parameter when present in the GET request and proceeds like when processing a full query of the TRL (see Section 7).

Otherwise, the AS **MUST** return a 4.00 (Bad Request) response in case the 'diff' query parameter of the GET request specifies a value that is neither 0 nor a positive integer, irrespective of the presence of the 'cursor' query parameter and its value (see below). The response **MUST** have Content-Format set to "application/concise-problem-details+cbor", and its payload is formatted as defined in Section 6.1. Within the Custom Problem Detail entry 'ace-trl-error', the value of the 'error-id' field **MUST** be set to 0 ("Invalid parameter value"), and the 'cursor' field **MUST NOT** be present.

- 'cursor': if included, perform a diff query of the TRL together with the "Cursor" extension, as defined in Section 9.2. Its value **MUST** be either 0 or a positive integer. If the 'cursor' query parameter is included, then the 'diff' query parameter **MUST** also be included.

  If included, the 'cursor' query parameter specifies an unsigned integer value that was provided by the AS in a previous response from the TRL endpoint (see Sections 9.1, 9.2.2, and 9.2.3).

  If the AS does not support the "Cursor" extension, it ignores the 'cursor' query parameter when present in the GET request. In such a case, the AS proceeds as specified elsewhere in this document, that is:

  1. it performs a diff query of the TRL (see Section 8), if it supports diff queries and the 'diff' query parameter is present in the GET request; or
  2. it performs a full query of the TRL (see Section 7).

  If the AS supports both diff queries and the "Cursor" extension, and the GET request specifies the 'cursor' query parameter, then the AS **MUST** return a 4.00 (Bad Request) response in case any of the conditions below holds.

  The 4.00 (Bad Request) response **MUST** have Content-Format set to "application/concise-problem-details+cbor", and its payload is formatted as defined in Section 6.1.

  - The GET request does not specify the 'diff' query parameter, irrespective of the value of the 'cursor' query parameter.

    Within the Custom Problem Detail entry 'ace-trl-error', the value of the 'error-id' field **MUST** be set to 1 ("Invalid set of parameters"), and the 'cursor' field **MUST NOT** be present.

  - The 'cursor' query parameter has a value that is neither 0 nor a positive integer; otherwise, it has a value strictly greater than MAX_INDEX (see Section 6.2.1).

    Within the Custom Problem Detail entry 'ace-trl-error', the value of the 'error-id' field **MUST** be set to 0 ("Invalid parameter value"). The entry 'ace-trl-error' **MUST** include the 'cursor' field, whose value is either:

    - the CBOR simple value `null` (0xf6), if the update collection associated with the requester is empty; or
    - the corresponding current value of 'last_index'.

  - All of the following hold: the update collection associated with the requester is not empty; no wraparound of the 'index' value has occurred; and the 'cursor' query parameter has a value strictly greater than the current 'last_index' on the update collection (see Section 6.2.1).

Within the Custom Problem Detail entry 'ace-trl-error', the value of the 'error-id' field **MUST** be set to 2 ("Out of bound cursor value"), and the 'cursor' field **MUST NOT** be present.

7. Full Query of the TRL

In order to produce a (notification) response to a GET request asking for a full query of the TRL, the AS performs the following actions:

1. From the TRL, the AS builds a set of HASHES such that:
   - If the requester is a registered device, HASHES specifies the token hashes currently in the TRL and associated with the access tokens pertaining to that registered device. The AS can always use the authenticated identity of the registered device to perform the necessary filtering on the TRL content.
   - If the requester is an administrator, HASHES specifies all the token hashes currently in the TRL.
2. The AS sends a 2.05 (Content) response to the requester. The response **MUST** have Content-Format set to "application/ace-trl+cbor". The payload of the response is a CBOR map, which **MUST** be formatted as follows.
   - The 'full_set' parameter **MUST** be included and **MUST** encode a CBOR array 'full_set_value'. Each element of 'full_set_value' is a CBOR byte string with a value of one of the token hashes from the set HASHES. If the set HASHES is empty, the 'full_set' parameter specifies the empty CBOR array.

     The CBOR array **MUST** be treated as a set, i.e., the order of its elements has no meaning.
   - The 'cursor' parameter **MUST** be included if the AS supports both diff queries and the related "Cursor" extension (see Sections 6.2 and 6.2.1). Its value is set as specified in Section 9.1 and provides the requester with information for performing a follow-up diff query using the "Cursor" extension (see Section 9.2).

     If the AS does not support both diff queries and the "Cursor" extension, this parameter **MUST NOT** be included. In case the requester does not support both diff queries and the "Cursor" extension, it **MUST** silently ignore the 'cursor' parameter if present.

Figure 6 provides the CDDL definition [RFC8610] of the CBOR array 'full_set_value' specified in the response from the AS as the value of the 'full_set' parameter.

```
token_hash = bytes
full_set_value = [* token_hash]
```

Figure 6: CDDL Definition of 'full_set_value'

Figure 7 shows an example response from the AS following a full query request to the TRL endpoint. In this example, the AS does not support diff queries nor the "Cursor" extension; hence the 'cursor' parameter is not included in the payload of the response. Also, full token hashes are omitted for brevity.

```
Header: Content (Code=2.05)
Content-Format: application/ace-trl+cbor
Payload:
{
   e'full_set' : [
     h'01fa51cc...4819', / elided for brevity /
     h'01748190...223d'  / elided for brevity /
   ]
}
```

Figure 7: Example of Response Following a Full Query Request to the TRL Endpoint

8. Diff Query of the TRL

In order to produce a (notification) response to a GET request asking for a diff query of the TRL, the AS performs the following actions:

Note that, if the AS supports both diff queries and the related "Cursor" extension, steps 3 and 4 defined below are extended as defined in Section 9.2.

1. The AS defines the positive integer NUM as follows: if the value N specified in the 'diff' query parameter in the GET request is equal to 0 or greater than the predefined positive integer MAX_N (see Section 6.2), then NUM takes the value of MAX_N. Otherwise, NUM takes N.
2. The AS determines U = min(NUM, SIZE), where SIZE <= MAX_N. In particular, SIZE is the number of diff entries currently stored in the requester's update collection.
3. The AS prepares U diff entries. If U is equal to 0 (e.g., because SIZE is equal to 0 at step 2), then no diff entries are prepared.

   The prepared diff entries are related to the U most recent TRL updates pertaining to the requester, as maintained in the update collection for that requester (see Section 6.2). In particular, the first diff entry refers to the most recent of such updates, the second diff entry refers to the second from last of such updates, and so on.

   Each diff entry is a CBOR array 'diff_entry', which includes the following two elements:

   a. A trl_patch set of token hashes encoded as a CBOR array 'removed'. Each element of the array is a CBOR byte string, whose value is the token hash of an access token such that it pertains to the requester and was removed from the TRL during the update associated with the diff entry.
   b. A trl_patch set of token hashes encoded as a CBOR array 'added'. Each element of the array is a CBOR byte string, whose value is the token hash of an access token such that it pertains to the requester and was added to the TRL during the update associated with the diff entry.

   The CBOR arrays 'removed' and 'added' **MUST** be treated as sets, i.e., the order of their elements has no meaning.

4. The AS prepares a 2.05 (Content) response for the requester. The response **MUST** have Content-Format set to "application/ace-trl+cbor". The payload of the response is a CBOR map, which **MUST** be formatted as follows:

   - The 'diff_set' parameter **MUST** be present and **MUST** encode a CBOR array 'diff_set_value' of U elements. Each element of 'diff_set_value' specifies one of the CBOR arrays 'diff_entry' prepared above as a diff entry. Note that U might have a value of 0; in this case, 'diff_set_value' is the empty CBOR array.

     Within 'diff_set_value', any 'diff_entry' CBOR arrays **MUST** be sorted to reflect the corresponding updates to the TRL in reverse chronological order. That is, the first 'diff_entry' element of 'diff_set_value' relates to the most recent TRL update pertaining to the requester. The second 'diff_entry' element relates to the second-to-last most recent TRL update pertaining to the requester, and so on.

   - The 'cursor' parameter and the 'more' parameter **MUST** be included if the AS supports both diff queries and the related "Cursor" extension (see Section 6.2.1). Their values are set as specified in Section 9.2 and provide the requester with information for performing a follow-up query of the TRL (see Section 9.2).

     In case the AS supports diff queries but not the "Cursor" extension, these parameters **MUST NOT** be included, and the AS **MUST** silently ignore the 'cursor' parameter and the 'more' parameter if present.

Figure 8 provides the CDDL definition [RFC8610] of the CBOR array 'diff_set_value' specified in the response from the AS, as the value of the 'diff_set' parameter.

```
token_hash = bytes
trl_patch = [* token_hash]
diff_entry = [removed: trl_patch, added: trl_patch]
diff_set_value = [* diff_entry]
```

Figure 8: CDDL Definition of 'diff_set_value'

Figure 9 shows an example response from the AS following a diff query request to the TRL endpoint, where U = 3 diff entries are specified. In this example, the AS does not support the "Cursor" extension; hence, the 'cursor' parameter and the 'more' parameter are not included in the payload of the response. Also, full token hashes are omitted for brevity.

```
Header: Content (Code=2.05)
Content-Format: application/ace-trl+cbor
Payload:
{
   e'diff_set' : [
     [
       [ h'01fa51cc...0f6a', / elided for brevity /
         h'01748190...8bce'  / elided for brevity /
       ],
       [ h'01cdf1ca...563d', / elided for brevity /
         h'01be41a6...a057'  / elided for brevity /
       ]
     ],
     [
       [ h'0144dd12...77bc', / elided for brevity /
         h'01231fff...a2ce'  / elided for brevity /
       ],
       []
     ],
     [
       [],
       [ h'01ca986f...ffc1', / elided for brevity /
         h'01fe1a2b...def0'  / elided for brevity /
       ]
     ]
   ]
}
```

Figure 9: Example of Response Following a Diff Query Request to the TRL Endpoint

Appendix A discusses how performing a diff query of the TRL is, in fact, a usage example of the Series Transfer Pattern defined in [STP].

9. Response Messages when Using the "Cursor" Extension

If the AS supports both diff queries and the "Cursor" extension, it composes a response to a full query request or diff query request as defined in Sections 9.1 and 9.2, respectively.

The exact format of the response depends on:

- the request being a full query or diff query request,
- the presence of the 'diff' and 'cursor' query parameters and their values in the diff query request, and
- the current status of the update collection associated with the requester.

Error handling and the possible resulting error responses are as defined in Section 6.3.

9.1. Response to Full Query

When processing a full query request to the TRL endpoint, the AS composes a response as defined in Section 7.

In particular, the 'cursor' parameter included in the CBOR map carried in the response payload specifies either the CBOR simple value `null` (0xf6) or a CBOR unsigned integer.

The 'cursor' parameter **MUST** encode the CBOR simple value `null` (0xf6) in case there are currently no TRL updates pertaining to the requester, i.e., the update collection for that requester is empty. This is the case from when the requester registers at the AS until the first update pertaining to that requester occurs to the TRL.

Otherwise, the 'cursor' parameter **MUST** encode a CBOR unsigned integer. The unsigned integer **MUST** take the 'index' value of the last series item in the update collection associated with the requester (see Section 6.2.1) as corresponding to the most recent TRL update pertaining to the requester. In fact, such a value is the current value of 'last_index' for the update collection associated with the requester.

9.2. Response to Diff Query

When processing a diff query request to the TRL endpoint, the AS composes a response as defined in the following subsections.

9.2.1. Empty Collection

If the update collection associated with the requester has no elements, the AS returns a 2.05 (Content) response. The response **MUST** have Content-Format set to "application/ace-trl+cbor", and its payload **MUST** be a CBOR map formatted as follows:

- The 'diff_set' parameter **MUST** be included and **MUST** encode the empty CBOR array.
- The 'cursor' parameter **MUST** be included and **MUST** encode the CBOR simple value `null` (0xf6).
- The 'more' parameter **MUST** be included and **MUST** encode the CBOR simple value `false` (0xf4).

Note that the above applies when the update collection associated with the requester has no elements, regardless of whether or not the 'cursor' query parameter is included in the diff query request and irrespective of the specified unsigned integer value if present.

9.2.2. Cursor Not Specified in the Diff Query Request

If the update collection associated with the requester is not empty and the diff query request does not include the 'cursor' query parameter, the AS performs the actions defined in Section 8, with the following differences:

- At step 3, the AS considers the value MAX_DIFF_BATCH (see Section 6.2.1) and prepares L = min(U, MAX_DIFF_BATCH) diff entries.

  If U <= MAX_DIFF_BATCH, the prepared diff entries are the last series items in the update collection associated with the requester, corresponding to the L most recent TRL updates pertaining to the requester.

If U > MAX_DIFF_BATCH, the prepared diff entries are the eldest of the last U series items in the update collection associated with the requester, as corresponding to the first L of the U most recent TRL updates pertaining to the requester.

- At step 4, the CBOR map to carry in the payload of the 2.05 (Content) response **MUST** be formatted as follows:
  - The 'diff_set' parameter **MUST** be present and **MUST** encode a CBOR array 'diff_set_value' of L elements. Each element of 'diff_set_value' specifies one of the CBOR arrays 'diff_entry' prepared as a diff entry.
  - The 'cursor' parameter **MUST** be present and **MUST** encode a CBOR unsigned integer. The unsigned integer **MUST** take the 'index' value of the series item of the update collection included as first diff entry in the 'diff_set_value' CBOR array, which is specified by the 'diff_set' parameter. That is, the 'cursor' parameter takes the 'index' value of the series item in the update collection corresponding to the most recent TRL update pertaining to the requester and returned in this diff query response.

    Note that the 'cursor' parameter takes the same 'index' value of the last series item in the update collection when U <= MAX_DIFF_BATCH.
  - The 'more' parameter **MUST** be present. The parameter **MUST** encode the CBOR simple value `false` (0xf4) if U <= MAX_DIFF_BATCH; otherwise, it **MUST** encode the CBOR simple value `true` (0xf5).

If the 'more' parameter in the payload of the received 2.05 (Content) response has a value of `true`, the requester can send a follow-up diff query request including the 'cursor' query parameter with the same value of the 'cursor' parameter specified in this diff query response. As defined in Section 9.2.3, this would result in the AS transferring the following subset of series items as diff entries, thus resuming from where interrupted in the previous transfer.

9.2.3. Cursor Specified in the Diff Query Request

If the update collection associated with the requester is not empty and the diff query request includes the 'cursor' query parameter with value P, the AS proceeds as follows, depending on which of the following two cases hold:

Case A: The series item X with 'index' having value P and the series item Y with 'index' having value (P + 1) % (MAX_INDEX + 1) are both not found in the update collection associated with the requester. This occurs when the item Y (and possibly further ones after it) has been previously removed from the update collection for that requester (see step 5 at Section 6.2).

In this case, the AS returns a 2.05 (Content) response. The response **MUST** have Content-Format set to "application/ace-trl+cbor", and its payload **MUST** be a CBOR map formatted as follows:

- The 'diff_set' parameter **MUST** be included and **MUST** encode the empty CBOR array.
- The 'cursor' parameter **MUST** be included and **MUST** encode the CBOR simple value `null` (0xf6).

- The 'more' parameter **MUST** be included and **MUST** encode the CBOR simple value `true` (0xf5).

With the combination ('cursor', 'more') = (`null`, `true`), the AS is indicating that the update collection is, in fact, not empty, but that one or more series items have been lost due to their removal. These include the item with 'index' value (P + 1) % (MAX_INDEX + 1) that the requester wished to obtain as the first one following the specified reference point with 'index' value P.

When receiving this diff query response, the requester **SHOULD** send a new full query request to the AS. A successful response provides the requester with the full current pertaining subset of the TRL as well as a valid value of the 'cursor' parameter (see Section 9.1) to be, possibly, used as query parameter in a following diff query request.

Case B: The series item X with 'index' having value P is found in the update collection associated with the requester or the series item X is not found and the series item Y with 'index' having value (P + 1) % (MAX_INDEX + 1) is found in the update collection associated with the requester.

In this case, the AS performs the actions defined in Section 8 with the following differences:

- At step 3, the AS considers the value MAX_DIFF_BATCH (see Section 6.2.1) and prepares L = min(SUB_U, MAX_DIFF_BATCH) diff entries, where SUB_U = min(NUM, SUB_SIZE) and SUB_SIZE is the number of series items in the update collection starting from and including the series item added immediately after X. If L is equal to 0 (e.g., because SUB_U is equal to 0), then no diff entries are prepared.

  If SUB_U <= MAX_DIFF_BATCH, the prepared diff entries are the last series items in the update collection associated with the requester, corresponding to the L most recent TRL updates pertaining to the requester.

  If SUB_U > MAX_DIFF_BATCH, the prepared diff entries are the eldest of the last SUB_U series items in the update collection associated with the requester, corresponding to the first L of the SUB_U most recent TRL updates pertaining to the requester.

- At step 4, the CBOR map to carry in the payload of the 2.05 (Content) response **MUST** be formatted as follows:
  - The 'diff_set' parameter **MUST** be present and **MUST** encode a CBOR array 'diff_set_value' of L elements. Each element of 'diff_set_value' specifies one of the CBOR arrays 'diff_entry' prepared as a diff entry. Note that L might have value 0, in which case 'diff_set_value' is the empty CBOR array.
  - The 'cursor' parameter **MUST** be present and **MUST** encode a CBOR unsigned integer. In particular:
    - If L is equal to 0, i.e., the series item X is the last one in the update collection, then the 'cursor' parameter **MUST** take the same 'index' value of the last series item in the update collection. In fact, such a value is the current value of 'last_index' for the update collection.

- If L is different than 0, then the 'cursor' parameter **MUST** take the 'index' value of the series element of the update collection included as first diff entry in the 'diff_set' CBOR array. That is, the 'cursor' parameter takes the 'index' value of the series item in the update collection corresponding to the most recent TRL update pertaining to the requester and returned in this diff query response.

  Note that the 'cursor' parameter takes the same 'index' value of the last series item in the update collection when SUB_U <= MAX_DIFF_BATCH.

- The 'more' parameter **MUST** be present. The parameter **MUST** encode the CBOR simple value `false` (0xf4) if SUB_U <= MAX_DIFF_BATCH; otherwise, it **MUST** encode the CBOR simple value `true` (0xf5).

If the 'more' parameter in the payload of the received 2.05 (Content) response has value `true`, the requester can send a follow-up diff query request including the 'cursor' query parameter with the same value of the 'cursor' parameter specified in this diff query response. This would result in the AS transferring the following subset of series items as diff entries, thus, resuming from where interrupted in the previous transfer.

10. Registration at the Authorization Server

During the registration process at the AS, an administrator or a registered device receives the following information as part of the registration response:

- The url-path to the TRL endpoint at the AS.
- The hash function used to compute token hashes. This is specified by identifying an entry in the "Named Information Hash Algorithm Registry" [IANA.Hash.Algorithms]. The specific means for this is outside the scope of this document.
- A positive integer MAX_N, if the AS supports diff queries of the TRL (see Sections 6.2 and 8).
- A positive integer MAX_DIFF_BATCH, if the AS supports diff queries of the TRL as well as the related "Cursor" extension (see Sections 6.2.1 and 9).

Once the registration process is completed, the AS maintains the registration and related information until a possible deregistration occurs, hence, keeping track of active administrators and registered devices. The particular way to achieve this is implementation specific. In any case, such a mechanism to maintain registrations is enforced at the AS in order to ensure that requests sent by clients to the /token endpoint (see Section 5.8 of [RFC9200]) and by RSs to the /introspect endpoint (see Section 5.9 of [RFC9200]) are processed as intended.

When communicating with one another, the registered devices and the AS have to use a secure communication association and be mutually authenticated (see Section 5 of [RFC9200]).

In the same spirit, communications between the AS and an administrator **MUST** be ensured to be mutually authenticated, encrypted, and integrity protected as well as protected against message replay.

Before starting its registration process at the AS, an administrator has to establish such a secure communication association with the AS, if they do not share one already. In particular, mutual authentication is **REQUIRED** during the establishment of the secure association. To this end, the administrator and the AS can rely, e.g., on establishing a TLS or DTLS secure session with mutual authentication (see [RFC8446] and [RFC9147]) or an Object Security for Constrained RESTful Environments (OSCORE) Security Context [RFC8613] by running the authenticated key exchange protocol EDHOC [RFC9528].

When receiving authenticated requests from the administrator for accessing the TRL endpoint, the AS can always check whether the requester is authorized to take such a role, i.e., to access the content of the whole TRL.

To this end, the AS may rely on a local access control list or similar, which specifies the authentication credentials of trusted, authorized administrators. In particular, the AS verifies the requester to the TRL endpoint as an authorized administrator only if the access control list includes the same authentication credential used by the requester when establishing the mutually authenticated secure communication association with the AS.

Further details about the registration process at the AS are out of scope for this specification. Note that the registration process is also out of the scope of the ACE framework (see Section 5.5 of [RFC9200]).

11. Notification of Revoked Access Tokens

Once registered at the AS, the administrator or registered device can send a GET request to the TRL endpoint at the AS. The request can express the wish for a full query (see Section 7) or a diff query (see Section 8) of the TRL. Also, the request can include the CoAP Observe Option set to 0 (register) in order to start an observation of the TRL endpoint as per Section 3.1 of [RFC7641].

In case the request is successfully processed, the AS replies with a 2.05 (Content) response. In particular, if the AS supports diff queries but not the "Cursor" extension (see Sections 6.2 and 6.2.1), then the payload of the response is formatted as defined in Sections 7 or 8, in case the GET request has yielded the execution of a full query or of a diff query of the TRL, respectively. Instead, if the AS supports both diff queries and the related "Cursor" extension, then the payload of the response is formatted as defined in Section 9.

In case a requester does not receive a response from the TRL endpoint or it receives an error response from the TRL endpoint, the requester does not make any assumptions or draw any conclusions regarding the revocation or expiration of its pertaining access tokens. The requester **MAY** try again by sending a new request to the TRL endpoint.

When the TRL is updated (see Section 5.1), the AS sends Observe notifications to the observers whose pertaining subset of the TRL has changed. Observe notifications are sent as per Section 4.2 of [RFC7641]. If supported by the AS, an observer may configure the behavior according to which the AS sends those Observe notifications. To this end, a possible way relies on the conditional control attribute "c.pmax" defined in [CoRE-ATTRIBUTES], which can be included as

a "name=value" query parameter in an Observation Request. This ensures that no more than c.pmax seconds elapse between two consecutive notifications sent to that observer, regardless of whether or not the TRL has changed.

Following a first exchange with the AS, an administrator or a registered device can send additional GET requests to the TRL endpoint at any time, analogously to what is defined above. When doing so, the requester towards the TRL endpoint can perform a full query (see Section 7) or a diff query (see Section 8) of the TRL. In the latter case, the requester can additionally rely on the "Cursor" extension (see Sections 6.3 and 9.2).

As specified in Section 6.2, an AS supporting diff queries maintains an update collection of maximum MAX_N series items for each administrator or registered device, hereafter referred to as a "requester". In particular, if an update collection includes MAX_N series items, adding a further series item to that update collection results in deleting the oldest series item from that update collection.

From then on, the requester associated with the update collection will not be able to retrieve the deleted series item when sending a new diff query request to the TRL endpoint. If that series item reflected the revocation of an access token pertaining to the requester, then the requester will not learn about that when receiving the corresponding diff query response from the AS.

Sending a diff query request specifically as an Observation Request, and, thus, relying on Observe notifications, largely reduces the chances for a requester to miss updates that have occurred to its associated update collection. In turn, this relies on the requester successfully receiving the Observe notification responses from the TRL (see also Section 14.3).

In order to limit the amount of time during which the requester is unaware of pertaining access tokens that have been revoked but are not expired yet, a requester **SHOULD NOT** rely solely on diff query requests. In particular, a requester **SHOULD** also regularly send a full query request to the TRL endpoint according to a related application policy.

11.1. Handling of Revoked Access Tokens and Token Hashes

When receiving a response from the TRL endpoint, a registered device **MUST** expunge every stored access token associated with a token hash specified in the response. In case the registered device is an RS, it **MUST NOT** delete the stored token hash after having expunged the associated access token.

If an RS uses the method defined in this document with the AS that has issued an access token, then the RS **MUST NOT** accept and store that access token if any of the following holds.

- The token hash corresponding to the access token is among the currently stored ones.
- The access token is a CWT and any of the following holds:
  - The access token includes a non-empty "unprotected" field, i.e., the value of the field is not encoded as the empty CBOR map (0xa0). This applies to the top-level "unprotected" field of the COSE object used for the CWT, the "unprotected" field of each element of the "signatures" array, and the "unprotected" field of each element of any "recipients" array.

- The received CBOR data item that embodies the access token does not comply with what is defined in Section 3. This concerns:
  - the use of exactly two nested CBOR tags, where the outer tag is the CWT CBOR tag and the inner tag is one of the COSE CBOR tags;
  - the tag numbers encoded with the minimum possible length; and
  - the access token being the innermost tag content of the received CBOR data item.
- In the received CBOR data item that embodies the access token, the inner tag has a tag number that is not consistent with the actual COSE data item to process. For instance, the inner tag number is 16 (COSE_Encrypt0) but the CWT is actually a COSE_Sign data item.

- The access token relies on a JSON object for encoding its claims, but it is not a JWT [RFC7519] and any of the following holds:
  - The access token uses the JWS JSON Serialization from [RFC7515] and includes the JWS Unprotected Header.
  - The access token uses the JWE JSON Serialization from [RFC7516] and includes the JWE Shared Unprotected Header and/or includes the "header" member in any of the elements of the "recipients" array.

An RS **MUST** store the token hash th1 corresponding to an access token t1 until both the following conditions hold:

- The RS has received and seen t1, irrespective of having accepted and stored it.
- The RS has gained knowledge that t1 has expired. This can be achieved, e.g., through the following means:
  - A response from the TRL endpoint indicating that t1 has expired after its earlier revocation, i.e., the token hash th1 has been removed from the TRL. This can be indicated, for instance, in a response from the TRL endpoint following a diff query of the TRL (see Section 8).
  - The value of the 'exp' claim specified in t1 indicates that t1 has expired.
  - The locally determined expiration time for t1 has passed, based on the time at the RS when t1 was first accepted and on the value of its 'exi' claim.
  - The result of token introspection performed on t1 (see Section 5.9 of [RFC9200]), if supported by both the RS and the AS.

The RS **MUST NOT** delete the stored token hashes whose corresponding access tokens do not fulfill both the two conditions above, unless it becomes necessary due to memory limitations. In such a case, the RS **MUST** delete the earliest stored token hashes first.

Retaining the stored token hashes as specified above limits the impact from a (dishonest) client whose pertaining access token:

1. specifies the 'exi' claim,
2. is uploaded at the RS for the first time after it has been revoked and later expired, and

3. has the sequence number encoded in the 'cti' claim (for CWTs) or in the 'jti' claim (for JWTs) greater than the highest sequence number among the expired access tokens specifying the 'exi' claim for the RS (see Section 5.10.3 of [RFC9200]). That is, the RS would not accept such a revoked and expired access token as long as it stores the corresponding token hash.

This risk can be further limited. Specifically, if token introspection is implemented by both the RS and the AS, the RS can introspect the access token (see Section 5.9 of [RFC9200]) when receiving an access token that specifies the 'exi' claim and for which a corresponding token hash is not stored.

When, due to the stored and corresponding token hash th2, an access token t2 that includes the 'exi' claim is expunged or is not accepted upon its upload, the RS retrieves the sequence number sn2 encoded in the 'cti' claim (for CWTs) or in the 'jti' claim (for JWTs) (see Section 5.10.3 of [RFC9200]). Then, the RS stores sn2 as associated with th2. If expunging or not accepting t2 yields the deletion of th2, then the RS **MUST** associate sn2 with th2 before continuing with the deletion of th2.

When deleting any token hash, the RS checks whether the token hash is associated with a sequence number sn_th. In such a case, the RS checks whether sn_th is greater than the highest sequence number sn* among the expired access tokens specifying the 'exi' claim for the RS. If that is the case, sn* **MUST** take the value of sn_th.

By virtue of what is defined in Section 5.10.3 of [RFC9200], this ensures that, following the deletion of the token hash associated with an access token specifying the 'exi' claim and uploaded for the first time after it has been revoked and later expired, the RS will not accept the access token at that point in time or in the future.

12. ACE Token Revocation List Parameters

This specification defines a number of parameters that can be transported in the response from the TRL endpoint, when the response payload is a CBOR map. Note that such a response **MUST** use the Content-Format "application/ace-trl+cbor" defined in Section 15.2 of this specification.

The table below summarizes the parameters. For each of them, it specifies the value to use as CBOR key, i.e., as abbreviation in the key of the map pair for the parameter, instead of the parameter's name as a text string.

Name	CBOR Key	CBOR Type
full_set	0	array
diff_set	1	array
cursor	2	Null or unsigned integer

Name	CBOR Key	CBOR Type
more	3	True or False

Table 1: CBOR Abbreviations for the ACE Token Revocation List Parameters

13. ACE Token Revocation List Error Identifiers

This specification defines a number of values that the AS can use as error identifiers. These are used in error responses with Content-Format "application/concise-problem-details+cbor", as values of the 'error-id' field within the Custom Problem Detail entry 'ace-trl-error' (see Section 6.1).

Value	Description
0	Invalid parameter value
1	Invalid set of parameters
2	Out of bound cursor value

Table 2: ACE Token Revocation List Error Identifiers

14. Security Considerations

The protocol defined in this document inherits the security considerations from the ACE framework [RFC9200], the usage of CWTs from [RFC8392], the usage of JWTs from [RFC7519] and [RFC8725], the usage of CoAP Observe from [RFC7641], and computation of the token hashes from [RFC6920]. The following considerations also apply.

14.1. Content Retrieval from the TRL

The AS **MUST** ensure that each registered device can access and retrieve only its pertaining subset of the TRL. To this end, the AS can always perform the required filtering based on the authenticated identity of the registered device, i.e., a (non-public) identifier that the AS can securely relate to the registered device and the secure association that they use to communicate.

The AS **MUST** ensure that, other than registered devices accessing their own pertaining subset of the TRL, only authorized and authenticated administrators can access the content of the whole TRL (see Section 10).

Note that the TRL endpoint supports only the GET method (see Section 6). Therefore, as detailed in Sections 7 and 8, access to the TRL endpoint is performed only by means of protected and authenticated GET requests, which, by definition, are safe in the REST sense and do not alter the content of the TRL. That is, registered devices and administrators can perform exclusively read-only operations when accessing the TRL endpoint.

In the two circumstances described in Section 5.1, the content of the TRL can be updated only internally by the AS. Therefore, an adversary that is not in control of the AS cannot manipulate the content of the TRL, e.g., by removing a token hash and thereby fraudulently allowing a client to access protected resources in spite of a revoked access token or by adding a token hash and thereby fraudulently stopping a client from accessing protected resources in spite of an access token being still valid.

14.2. Size of the TRL

If many non-expired access tokens associated with a registered device are revoked, the pertaining subset of the TRL could grow to a size bigger than what the registered device is prepared to handle upon reception of a response from the TRL endpoint, especially if relying on a full query of the TRL (see Section 7).

This could be exploited by attackers to negatively affect the behavior of a registered device. Therefore, in order to help reduce the size of the TRL, the AS **SHOULD** refrain from issuing access tokens with an excessively long expiration time.

14.3. Communication Patterns

The communication about revoked access tokens presented in this specification is expected to especially rely on CoAP Observe notifications sent from the AS to a requester (i.e., an administrator or a registered device). The suppression of those notifications by an external attacker that has access to the network would prevent requesters from ever knowing that their pertaining access tokens have been revoked.

In order to avoid this, a requester **SHOULD NOT** rely solely on the CoAP Observe notifications. In particular, a requester **SHOULD** also regularly poll the AS for the most current information about revoked access tokens by sending GET requests to the TRL endpoint. Specific strategies and schedules for polling the AS are to be defined by a related application policy and by taking into account the expected operational and availability patterns adopted by the requester (e.g., in the interest of energy saving and other optimizations).

14.4. Request of New Access Tokens

If a client stores an access token that it still believes to be valid, and it accordingly attempts to access a protected resource at the RS, the client may receive an unprotected 4.01 (Unauthorized) response from the RS.

This can be due to a number of causes, for example:

- the access token has been revoked, the RS has become aware of it, and the RS has expunged the access token, but the client is not aware of this (yet).
- the access token is still valid, but an on-path active adversary might have injected a forged 4.01 (Unauthorized) response or the RS might have deleted the access token from its local storage due to its dedicated storage space being all consumed.

In either case, if the client believes that the access token is still valid, it **SHOULD NOT** immediately ask for a new access token to the AS upon receiving a 4.01 (Unauthorized) response from the RS. Instead, the client **SHOULD** send a request to the TRL endpoint at the AS. If the client gains knowledge that the access token is not valid anymore, the client expunges the access token and can ask for a new one. Otherwise, the client can try again to upload the same access token to the RS or request a new one.

14.5. Vulnerable Time Window at the RS

A client may attempt to access a protected resource at an RS after the access token allowing such an access has been revoked but before the RS is aware of the revocation.

In such a case, if the RS is still storing the access token, the client will be able to access the protected resource even though it should not. Such access is a security violation, even if the client is not attempting to be malicious.

In order to minimize such a risk, if an RS relies solely on polling through individual requests to the TRL endpoint to learn of revoked access tokens, the RS **SHOULD** implement an adequate trade-off between the polling frequency and the maximum length of the vulnerable time window.

14.6. Preventing Unnoticed Manipulation of Access Tokens

As defined in Section 3, issued access tokens **MUST NOT** rely on unprotected headers to specify information as header parameters. Also, when issued access tokens are CWTs, they **MUST** be tagged by using the COSE CBOR tag corresponding to the used COSE object. Further, the result **MUST** be tagged using the CWT CBOR tag; no further tagging is performed.

This ensures that the RS always computes the correct token hash corresponding to an access token, i.e., the same token hash computed by the AS and C for that access token.

By construction, the rules defined in Section 3 prevent an active adversary from successfully performing an attack against the RS, which would otherwise be possible in case the access token is uploaded to the RS over an unprotected communication channel.

In such an attack, the adversary intercepts the access token en route to the RS. Then, the adversary manipulates the access token in a way that is going to be unnoticed by the RS but without preventing the successful cryptographic validation of the access token at the RS. To this end, the adversary has two possible options:

- Adding and/or removing fields within the unprotected header(s) of the access token, as long as those fields do not play a role in the cryptographic validation of the access token.
- Specifically when the access token is a CWT, adding, removing, or manipulating possible CBOR tags enclosing the access token.

After that, the adversary sends the manipulated access token to the RS.

After having successfully validated the manipulated access token, the RS computes a corresponding token hash different from the one computed and stored by C and the AS. Finally, the RS stores the manipulated access token and the corresponding wrong token hash.

Later on, if the access token is revoked and the AS provides the RS with the corresponding correct token hash, the RS does not recognize the received token hash among the stored ones; therefore, it does not delete the revoked access token.

14.7. Two Token Hashes at the RS Using JWTs

Section 4.3.2 states that an RS using JWTs as access tokens has to compute and store two token hashes associated with the same access token. This is because, when using JWTs, the RS does not know for sure if the AS provided the access token to the client by means of an AS-to-Client response encoded in CBOR or in JSON.

Taking advantage of that, a dishonest client can attempt to perform an attack against the RS. That is, the client can first receive the JWT in an AS-to-Client response encoded in CBOR (JSON). Then, the client can upload the JWT to the RS in a way that makes the RS believe that the client instead received the JWT in an AS-to-Client response encoded in JSON (CBOR).

Consequently, the RS considers a HASH_INPUT different from the one considered by the AS and the client (see Section 4.2). Hence, the RS computes a token hash h' different from the token hash h computed by the AS and the client. It follows that, if the AS revokes the access token and advertises the right token hash h, then the RS will not learn about the access token revocation; thus, it will not delete the access token.

Fundamentally, this would happen because the HASH_INPUT used to compute the token hash of a JWT depends on whether the AS-to-Client response is encoded in CBOR or in JSON. This makes the RS vulnerable to the attack described above when JWTs are used as access tokens. However, this is not a problem if the access token is a CWT since the HASH_INPUT used to compute the token hash of a CWT does not depend on whether the AS-to-Client response is encoded in CBOR or in JSON.

While this asymmetry cannot be avoided altogether, the method defined for the AS and the client in Section 4.2 deliberately penalizes the case where the RS uses JWTs as access tokens. In such a case, the RS effectively neutralizes the attack described above by computing and storing two token hashes associated with the same access token (see Section 4.3.2).

Conversely, this design deliberately favors the case where the RS uses CWTs as access tokens, which is a preferable option for resource-constrained RSs as well as the default case in the ACE framework (see Section 3 of [RFC9200]). That is, if an RS uses CWTs as access tokens, then the RS is not exposed to the attack described above; thus, it safely computes and stores only one token hash per access token (see Section 4.3.1).

14.8. Additional Security Measures

By accessing the TRL at the AS, registered devices and administrators are able to learn that their pertaining access tokens have been revoked. However, they cannot learn the reason why, including when that reason is the compromise, misbehavior, or decommissioning of a registered device.

In fact, even the AS might not know that a registered device to which a revoked access token pertains has been specifically compromised, misbehaving, or decommissioned. At the same time, it might not be acceptable to only revoke the access tokens pertaining to such a registered device.

Therefore, in order to preserve the security of the system and application, the entity that authoritatively declares a registered device to be compromised, misbehaving, or decommissioned should also promptly trigger the execution of additional revocation processes as deemed appropriate. These include, for instance:

- The de-registration of the registered device from the AS so that the AS does not issue further access tokens pertaining to that device.
- If applicable, the revocation of the public authentication credential associated with the registered device (e.g., its public key certificate).

The methods by which these processes are triggered and carried out are out of the scope of this document.

15. IANA Considerations

The IANA actions for this document are described in the following subsections.

15.1. Media Type Registrations

IANA has registered the media type "application/ace-trl+cbor" for messages of the protocol defined in this document encoded in CBOR. This registration follows the procedures specified in [RFC6838].

Type name: application

Subtype name: ace-trl+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: Must be encoded as a CBOR map containing the protocol parameters defined in RFC 9770.

Security considerations: See Section 14 of this document.

Interoperability considerations: N/A

Published specification: RFC 9770

Applications that use this media type: The type is used by authorization servers, clients, and RSs that support the notification of revoked access tokens according to a Token Revocation List maintained by the authorization server as specified in RFC 9770.

Fragment identifier considerations: N/A

Additional information: N/A

Person & email address to contact for further information: ACE WG mailing list (ace@ietf.org) or IETF Applications and Real-Time Area (art@ietf.org)

Intended usage: COMMON

Restrictions on usage: None

Author/Change controller: IETF

15.2. CoAP Content-Formats Registry

IANA has added the following entry to the "CoAP Content-Formats" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Content Type: application/ace-trl+cbor
Content Coding: -
ID: 262
Reference: RFC 9770

15.3. Custom Problem Detail Keys Registry

IANA has registered the following entry in the "Custom Problem Detail Keys" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Key Value: 1
Name: ace-trl-error
Brief Description: Carry RFC 9770 problem details in a Concise Problem Details data item.
Change Controller: IETF

Reference: Section 6.1 of RFC 9770

15.4. ACE Token Revocation List Parameters Registry

IANA has established the "ACE Token Revocation List Parameters" registry within the "Authentication and Authorization for Constrained Environments (ACE)" registry group.

One of the following registration policies is used: "Standards Action with Expert Review", "Specification Required" per Section 4.6 of [RFC8126], or "Expert Review" per Section 4.5 of [RFC8126]. Expert Review guidelines are provided in Section 15.6.

All assignments according to "Standards Action with Expert Review" are made on a "Standards Action" basis per Section 4.9 of [RFC8126] with Expert Review additionally required per Section 4.5 of [RFC8126]. The procedure for early IANA allocation of Standards Track code points defined in [RFC7120] also applies. When such a procedure is used, IANA will ask the designated expert(s) to approve the early allocation before registration. In addition, WG chairs are encouraged to consult the expert(s) early during the process outlined in Section 3.1 of [RFC7120].

The columns of this registry are as follows:

- Name: This field contains a descriptive name that enables easier reference to the item. The name **MUST** be unique, and it is not used in the encoding.
- CBOR Key: This field contains the value used as CBOR map key of the item. The value **MUST** be unique. The value is an unsigned integer or a negative integer. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as "Standards Action With Expert Review". Integer values from -65536 to -257 and from 256 to 65535 are designated as "Specification Required". Integer values greater than 65535 are designated as "Expert Review". Integer values less than -65536 are marked as "Private Use".
- CBOR Type: This field contains the allowable CBOR data types for values of this item or a pointer to the registry that defines its type, when that depends on another item.
- Reference: This field contains a pointer to the public specification for the item.

This registry has been initially populated by the values in Section 12. The "Reference" column for all of these entries refers to this document.

15.5. ACE Token Revocation List Errors

IANA has established the "ACE Token Revocation List Errors" registry within the "Authentication and Authorization for Constrained Environments (ACE)" registry group.

One of the following registration policies is used: "Standards Action with Expert Review", "Specification Required" per Section 4.6 of [RFC8126], or "Expert Review" per Section 4.5 of [RFC8126]. Expert Review guidelines are provided in Section 15.6.

All assignments according to "Standards Action with Expert Review" are made on a "Standards Action" basis per Section 4.9 of [RFC8126] with Expert Review additionally required per Section 4.5 of [RFC8126]. The procedure for early IANA allocation of Standards Track code points

defined in [RFC7120] also applies. When such a procedure is used, IANA will ask the designated expert(s) to approve the early allocation before registration. In addition, WG chairs are encouraged to consult the expert(s) early during the process outlined in Section 3.1 of [RFC7120].

The columns of this registry are as follows:

- Value: The field contains the value to be used to identify the error. The value **MUST** be unique. The value is an unsigned integer or a negative integer. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as "Standards Action With Expert Review". Integer values from -65536 to -257 and from 256 to 65535 are designated as "Specification Required". Integer values greater than 65535 are designated as "Expert Review". Integer values less than -65536 are marked as "Private Use".
- Description: This field contains a brief description of the error.
- Reference: This field contains a pointer to the public specification defining the error, if one exists.

This registry has been initially populated by the values in Section 13. The "Reference" column for all of these entries refers to this document.

15.6. Expert Review Instructions

The IANA registries established by this document use "Standards Action with Expert Review", "Specification Required", or "Expert Review" registration procedures depending on the range of values for which an assignment is requested. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason, so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments. The zones tagged as Private Use are intended for testing purposes and closed environments. Code points in other ranges should not be assigned for testing.
- Specifications are required for the "Standards Action With Expert Review" range of point assignment. Specifications should exist for "Specification Required" ranges, but early assignment before a specification is available is considered to be permissible. For the "Expert Review" range of point assignment, specifications are recommended and are needed if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.
- Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for Standards Track documents does not mean that a Standards Track document cannot have points assigned outside of that range. The

length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

16. References

16.1. Normative References

[IANA.Hash.Algorithms] IANA, "Named Information Hash Algorithm Registry", <https://www.iana.org/assignments/named-information>.

[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/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[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/info/rfc6838>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <https://www.rfc-editor.org/info/rfc7120>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[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/info/rfc7515>.

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

[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/info/rfc7519>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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/info/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/info/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/info/rfc8392>.

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

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/info/rfc8725>.

[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/info/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/info/rfc9052>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

[RFC9200] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/info/rfc9200>.

[RFC9202] Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", RFC 9202, DOI 10.17487/RFC9202, August 2022, <https://www.rfc-editor.org/info/rfc9202>.

[RFC9203] Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "The Object Security for Constrained RESTful Environments (OSCORE) Profile of the Authentication and Authorization for Constrained Environments (ACE) Framework", RFC 9203, DOI 10.17487/RFC9203, August 2022, <https://www.rfc-editor.org/info/rfc9203>.

[RFC9290] Fossati, T. and C. Bormann, "Concise Problem Details for Constrained Application Protocol (CoAP) APIs", RFC 9290, DOI 10.17487/RFC9290, October 2022, <https://www.rfc-editor.org/info/rfc9290>.

[RFC9431] Sengul, C. and A. Kirby, "Message Queuing Telemetry Transport (MQTT) and Transport Layer Security (TLS) Profile of Authentication and Authorization for Constrained Environments (ACE) Framework", RFC 9431, DOI 10.17487/RFC9431, July 2023, <https://www.rfc-editor.org/info/rfc9431>.

[RFC9528] Selander, G., Preuß Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", RFC 9528, DOI 10.17487/RFC9528, March 2024, <https://www.rfc-editor.org/info/rfc9528>.

[SHA-256] NIST, "Secure Hash Standard", NIST FIPS PUB 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

16.2. Informative References

[CoRE-ATTRIBUTES] Silverajan, B., Koster, M., and A. Soloway, "Conditional Query Parameters for CoAP Observe", Work in Progress, Internet-Draft, draft-ietf-core-conditional-attributes-11, 16 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-conditional-attributes-11>.

[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/info/rfc7009>.

[STP] Bormann, C. and K. Hartke, "The Series Transfer Pattern (STP)", Work in Progress, Internet-Draft, draft-bormann-t2trg-stp-03, 7 April 2020, <https://datatracker.ietf.org/doc/html/draft-bormann-t2trg-stp-03>.

Appendix A. On Using the Series Transfer Pattern

Performing a diff query of the TRL as specified in Section 8 is, in fact, a usage example of the Series Transfer Pattern defined in [STP].

That is, a diff query enables the transfer of a series of diff entries with the AS specifying U <= MAX_N diff entries as related to the U most recent TRL updates pertaining to a requester, i.e., a registered device or an administrator.

When responding to a diff query request from a requester (see Section 8), 'diff_set' is a subset of the update collection associated with the requester where each 'diff_entry' record is a series item from that update collection. Note that 'diff_set' specifies the whole current update collection when the value of U is equal to SIZE, i.e., the current number of series items in the update collection.

The value N of the 'diff' query parameter in the GET request allows the requester and the AS to trade the amount of provided information with the latency of the information transfer.

Since the update collection associated with each requester includes up to MAX_N series items, the AS deletes the oldest series item when a new one is generated and added to the end of the update collection, due to a new TRL update pertaining to that requester (see Section 6.2). This addresses the question "When can the server decide to no longer retain older items?" raised in Section 3.2 of [STP].

Furthermore, performing a diff query of the TRL together with the "Cursor" extension, as specified in Section 9, in fact, relies on the "cursor" pattern of the STP (see Section 3.3 of [STP]).

Appendix B. Local Supportive Parameters of the TRL Endpoint

Table 3 provides an aggregated overview of the local supportive parameters that the AS internally uses at its TRL endpoint when supporting diff queries (see Section 6) and the "Cursor" extension (see Section 6.2.1).

Except for MAX_N defined in Section 6.2, all the other parameters are defined in Section 6.2.1 and are used only if the AS supports the "Cursor" extension.

For each parameter, the columns of the table specify the following information. Both a registered device and an administrator are referred to as "requester".

Name: The parameter name. A name with letters in uppercase denotes a parameter whose value does not change after its initialization.

Single instance: "Y" if there is a single parameter instance associated with the TRL or "N" if there is one parameter instance per update collection (i.e., per requester).

Description: A short description of the parameter.

Values: The unsigned integer values that the parameter can assume, where LB and UB denote the inclusive lower bound and upper bound, respectively.

Name	Single instance	Description	Values
MAX_N	Y	Max number of series items in the update collection of each requester	LB = 1 If supporting the "Cursor" extension, then UB = MAX_INDEX+1
MAX_DIFF_BATCH	N	Max number of diff entries included in a diff query response when using the "Cursor" extension	LB = 1 UB = MAX_N
MAX_INDEX	Y	Max value of each instance of 'index'	LB = MAX_N-1 UB = (2^{64})-1
index	N	Value associated with a series item of an update collection	LB = 0 UB = MAX_INDEX
last_index	N	The 'index' value of the most recently added series item in an update collection	LB = 0 UB = MAX_INDEX

Table 3: Local Supportive Parameters of the TRL Endpoint

Appendix C. Interaction Examples

This section provides examples of interactions between an RS as a registered device and an AS. In the examples, all the access tokens issued by the AS are intended to be consumed by the considered RS.

The AS supports both full queries and diff queries of the TRL, as defined in Sections 7 and 8, respectively.

Registration is assumed to be done by the RS sending a POST request with an unspecified payload to the AS, which replies with a 2.01 (Created) response. The payload of the registration response is assumed to be a CBOR map, which, in turn, is assumed to include the following entries:

- a 'trl_path' parameter specifying the path of the TRL endpoint;

- a 'trl_hash' parameter specifying the "Hash Name String" of the hash function used to compute token hashes as defined in Section 4;
- a 'max_n' parameter specifying the value of MAX_N, i.e., the maximum number of series items that the AS retains in the update collection associated with a registered device (see Section 8);
- possible further parameters related to the registration process.

Furthermore, 'h(x)' refers to the hash function used to compute the token hashes, as defined in Section 4 of this specification and according to [RFC6920]. Assuming the usage of CWTs transported in AS-to-Client responses encoded in CBOR (see Section 4.2.1), 'bstr.h(t1)' and 'bstr.h(t2)' denote the CBOR byte strings, whose values are the token hashes of the access tokens t1 and t2, respectively.

C.1. Full Query with Observe

Figure 10 shows an interaction example considering a CoAP observation and a full query of the TRL.

In this example, the AS does not support the "Cursor" extension. Hence, the 'cursor' parameter is not included in the payload of the responses to a full query request.

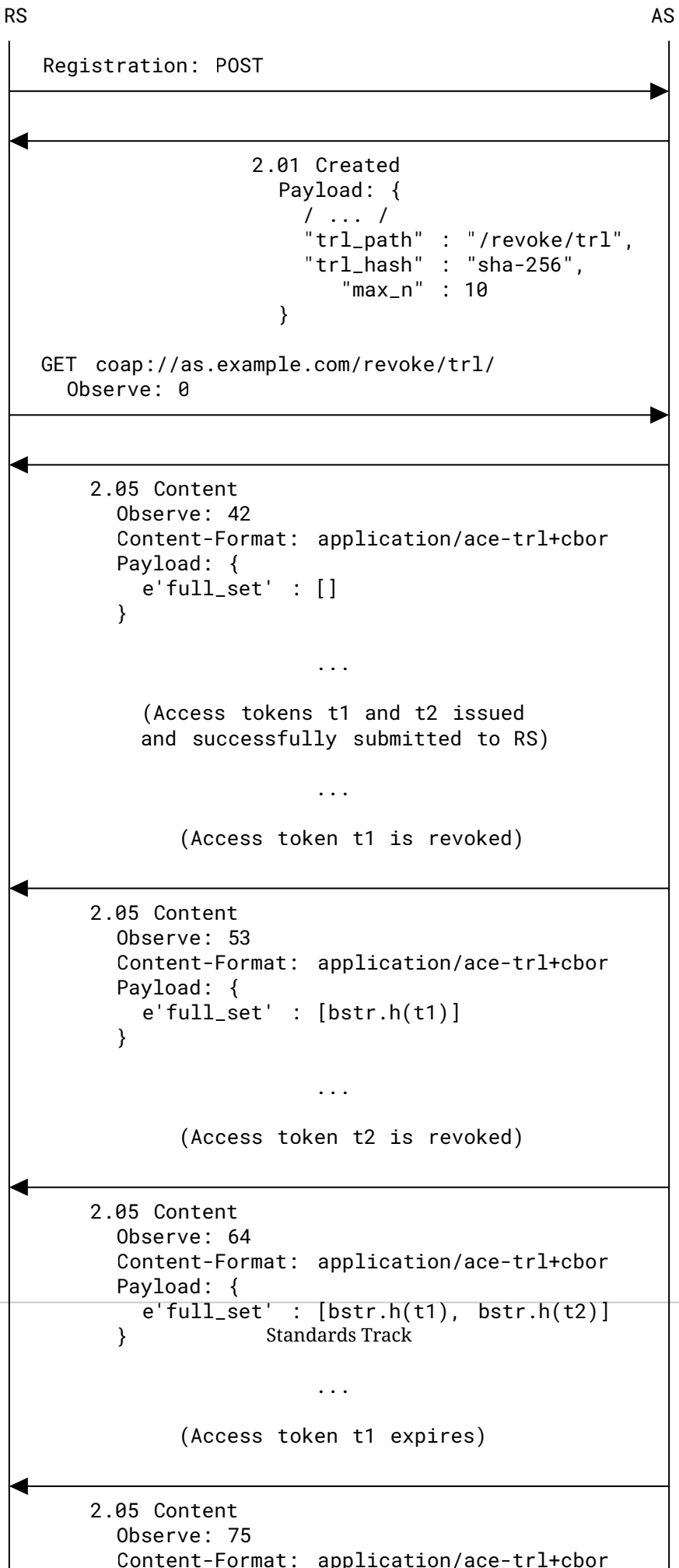
RS
AS
Registration: POST
2.01 Created
Payload: {
/ ... /
"trl_path" : "/revoke/trl",
"trl_hash" : "sha-256",
"max_n" : 10
}
GET coap://as.example.com/revoke/trl/
Observe: 0
2.05 Content
Observe: 42
Content-Format: application/ace-trl+cbor
Payload: {
e'full_set' : []
}
...
(Access tokens t1 and t2 issued
and successfully submitted to RS)
...
(Access token t1 is revoked)
2.05 Content
Observe: 53
Content-Format: application/ace-trl+cbor
Payload: {
e'full_set' : [bstr.h(t1)]
}
...
(Access token t2 is revoked)
2.05 Content
Observe: 64
Content-Format: application/ace-trl+cbor
Payload: {
e'full_set' : [bstr.h(t1), bstr.h(t2)]
}
...
(Access token t1 expires)
2.05 Content
Observe: 75
Content-Format: application/ace-trl+cbor

Figure 10: Interaction for Full Query with Observe

C.2. Diff Query with Observe

Figure 11 shows an interaction example of a CoAP observation and a diff query of the TRL.

The RS indicates N = 3 as the value of the 'diff' query parameter, i.e., as the maximum number of diff entries to be specified in a response from the AS.

In this example, the AS does not support the "Cursor" extension. Hence, the 'cursor' parameter and the 'more' parameter are not included in the payload of the responses to a diff query request.

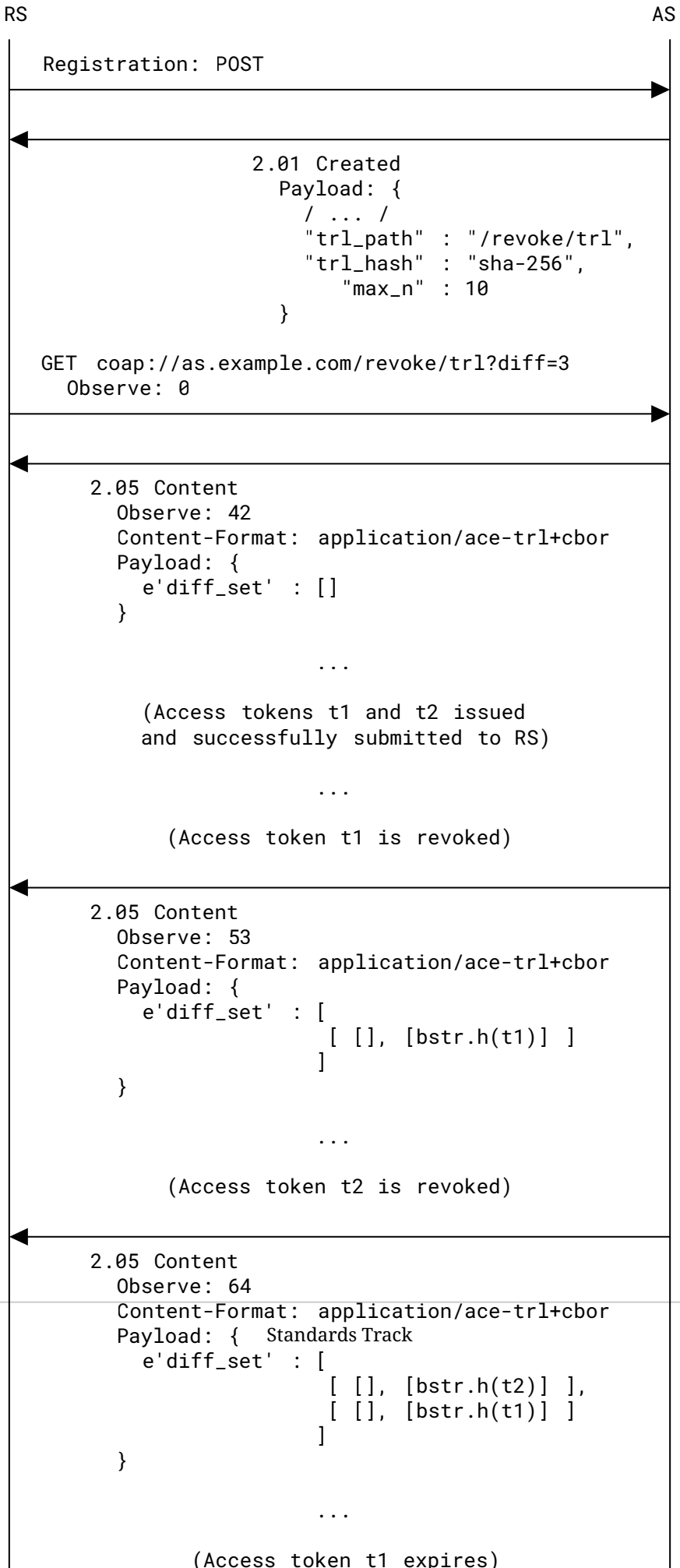
RS
AS
Registration: POST
2.01 Created
Payload: {
/ ... /
"trl_path" : "/revoke/trl",
"trl_hash" : "sha-256",
"max_n" : 10
}
GET coap://as.example.com/revoke/trl?diff=3
Observe: 0
2.05 Content
Observe: 42
Content-Format: application/ace-trl+cbor
Payload: {
e'diff_set' : []
}
...
(Access tokens t1 and t2 issued
and successfully submitted to RS)
...
(Access token t1 is revoked)
2.05 Content
Observe: 53
Content-Format: application/ace-trl+cbor
Payload: {
e'diff_set' : [
[[], [bstr.h(t1)]]
]
}
...
(Access token t2 is revoked)
2.05 Content
Observe: 64
Content-Format: application/ace-trl+cbor
Payload: {
e'diff_set' : [
[[], [bstr.h(t2)]],
[[], [bstr.h(t1)]]
]
}
...
(Access token t1 expires)

Figure 11: Interaction for Diff Query with Observe

C.3. Full Query with Observe and Diff Query

Figure 12 shows an interaction example of a CoAP observation and a full query of the TRL.

The example also shows one of the notifications from the AS getting lost in transmission; thus, it does not reach the RS.

When this happens, and after a waiting time defined by the application has elapsed, the RS sends a GET request with no Observe Option to the AS to perform a diff query of the TRL. The RS indicates N = 8 as the value of the 'diff' query parameter, i.e., as the maximum number of diff entries to be specified in a response from the AS.

In this example, the AS does not support the "Cursor" extension. Hence, the 'cursor' parameter is not included in the payload of the responses to a full query request. Also, the 'cursor' parameter and the 'more' parameter are not included in the payload of the responses to a diff query request.

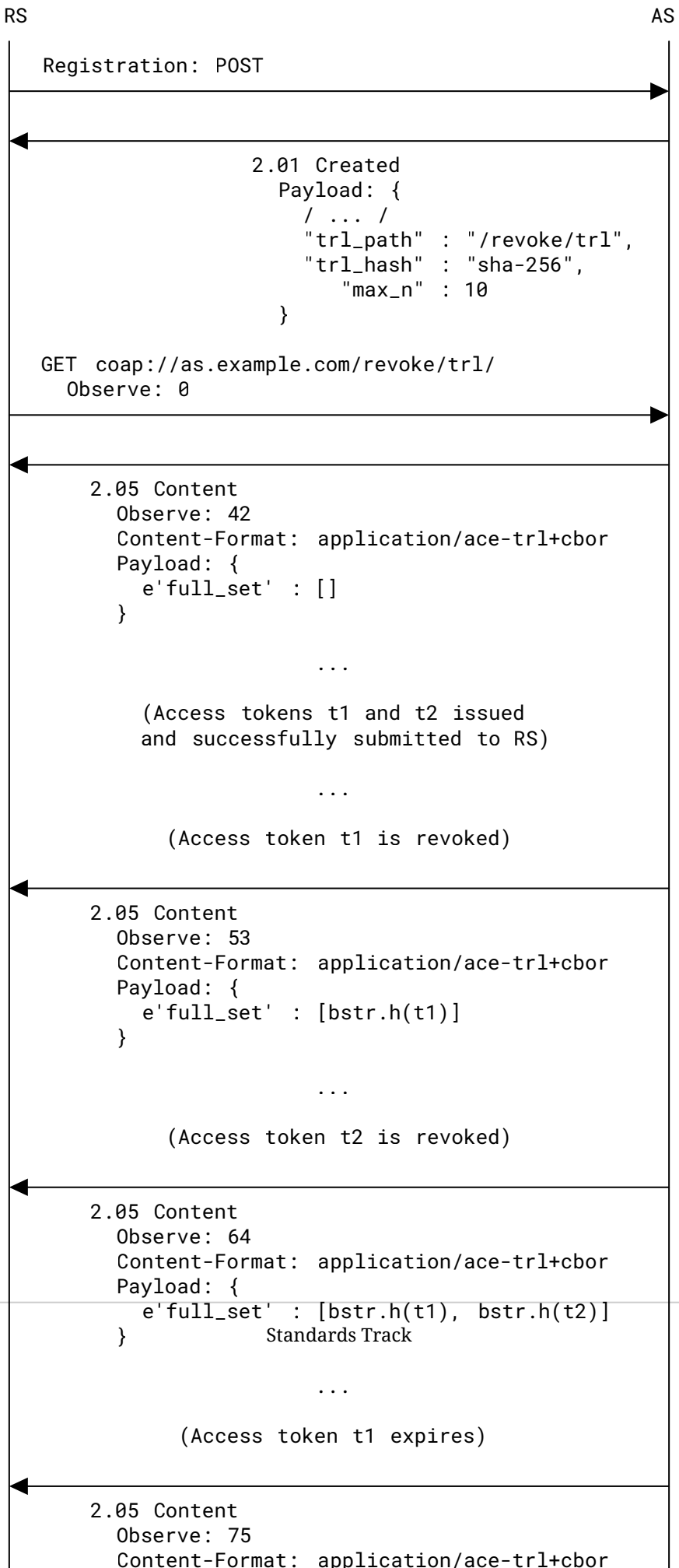
RS
AS
Registration: POST
2.01 Created
Payload: {
/ ... /
"trl_path" : "/revoke/trl",
"trl_hash" : "sha-256",
"max_n" : 10
}
GET coap://as.example.com/revoke/trl/
Observe: 0
2.05 Content
Observe: 42
Content-Format: application/ace-trl+cbor
Payload: {
e'full_set' : []
}
...
(Access tokens t1 and t2 issued
and successfully submitted to RS)
...
(Access token t1 is revoked)
2.05 Content
Observe: 53
Content-Format: application/ace-trl+cbor
Payload: {
e'full_set' : [bstr.h(t1)]
}
...
(Access token t2 is revoked)
2.05 Content
Observe: 64
Content-Format: application/ace-trl+cbor
Payload: {
e'full_set' : [bstr.h(t1), bstr.h(t2)]
}
...
(Access token t1 expires)
2.05 Content
Observe: 75
Content-Format: application/ace-trl+cbor

Figure 12: Interaction for Full Query with Observe and Diff Query

C.4. Diff Query with Observe and "Cursor" Extension

In this example, the AS supports the "Cursor" extension. Hence, the CBOR map conveyed as payload of the registration response additionally includes a "max_diff_batch" parameter. This specifies the value of MAX_DIFF_BATCH, i.e., the maximum number of diff entries that can be included in a response to a diff query request from this RS.

Figure 13 shows an interaction example of a CoAP observation and a diff query of the TRL.

The RS specifies the 'diff' query parameter with a value of 3, i.e., the maximum number of diff entries to be specified in a response from the AS.

If the RS has not received a notification from the AS after a waiting time defined by the application, the RS sends a GET request with no Observe Option to the AS to perform a diff query of the TRL.

This is followed up by a further diff query request that specifies the 'cursor' query parameter. Note that the payload of the corresponding response differs from the payload of the response to the previous diff query request.

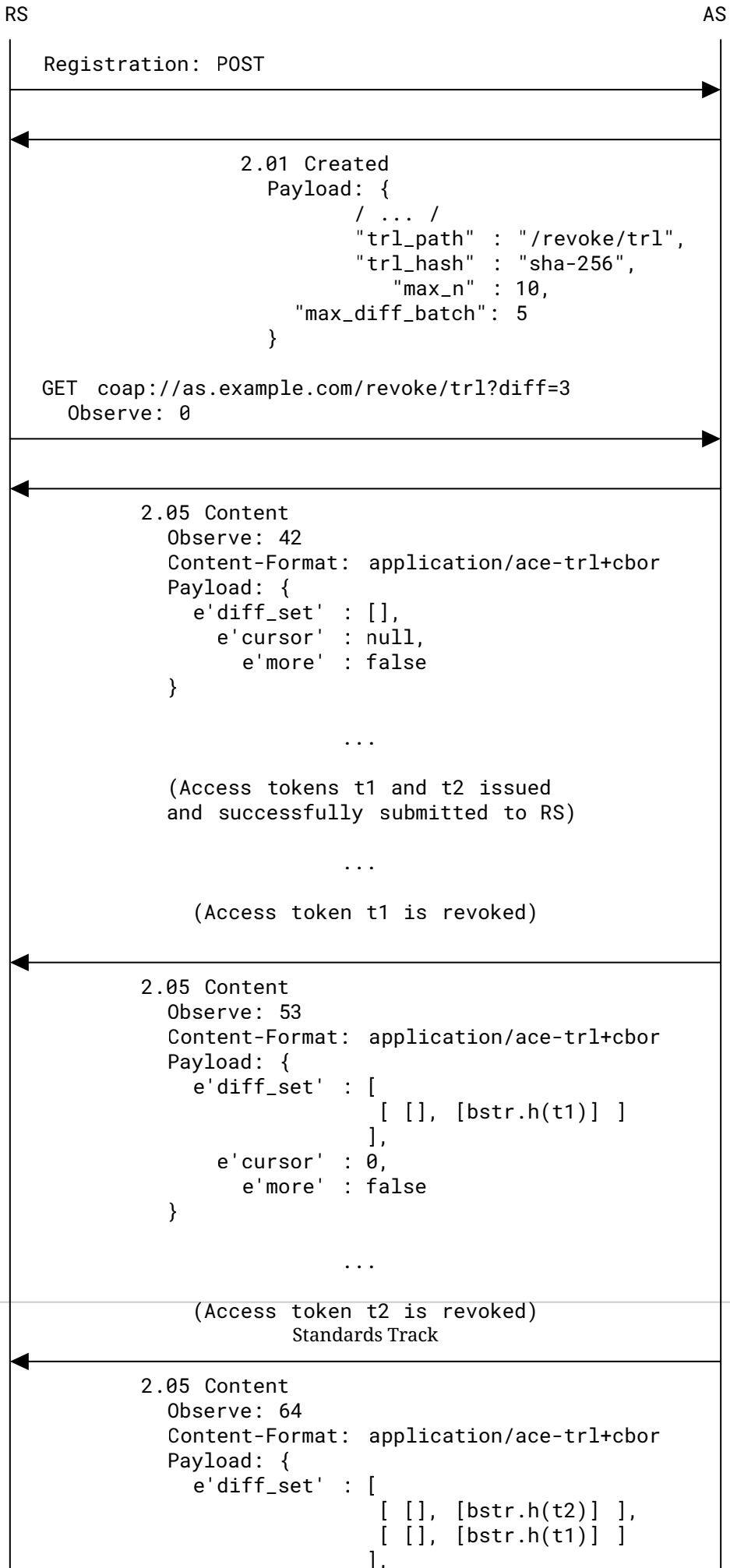
RS
AS
Registration: POST
2.01 Created
Payload: {
/ ... /
"trl_path" : "/revoke/trl",
"trl_hash" : "sha-256",
"max_n" : 10,
"max_diff_batch": 5
}
GET coap://as.example.com/revoke/trl?diff=3
Observe: 0
2.05 Content
Observe: 42
Content-Format: application/ace-trl+cbor
Payload: {
e'diff_set' : [],
e'cursor' : null,
e'more' : false
}
...
(Access tokens t1 and t2 issued
and successfully submitted to RS)
...
(Access token t1 is revoked)
2.05 Content
Observe: 53
Content-Format: application/ace-trl+cbor
Payload: {
e'diff_set' : [
[[], [bstr.h(t1)]]
],
e'cursor' : 0,
e'more' : false
}
...
(Access token t2 is revoked)
2.05 Content
Observe: 64
Content-Format: application/ace-trl+cbor
Payload: {
e'diff_set' : [
[[], [bstr.h(t2)]],
[[], [bstr.h(t1)]]
],

Figure 13: Interaction for Diff Query with Observe and "Cursor" Extension

C.5. Full Query with Observe and Diff Query with "Cursor" Extension

In this example, the AS supports the "Cursor" extension. Hence, the CBOR map conveyed as payload of the registration response additionally includes a "max_diff_batch" parameter. This specifies the value of MAX_DIFF_BATCH, i.e., the maximum number of diff entries that can be included in a response to a diff query request from this RS.

Figure 14 shows an interaction example of a CoAP observation and a full query of the TRL.

The example also shows some of the notifications from the AS getting lost in transmission; thus, they do not reach the RS.

When this happens, and after a waiting time defined by the application has elapsed, the RS sends a GET request with no Observe Option to the AS, to perform a diff query of the TRL. In particular, the RS specifies:

- The 'diff' query parameter with a value of 8, i.e., the maximum number of diff entries to be specified in a response from the AS.
- The 'cursor' query parameter with a value of 2, thus requesting from the update collection the series items following the one with the 'index' value equal to 2 (i.e., following the last series item that the RS successfully received in an earlier notification response).

The response from the AS conveys a first batch of MAX_DIFF_BATCH = 5 series items from the update collection corresponding to the RS. The AS indicates that further series items are actually available in the update collection by setting the 'more' parameter of the response to `true`. Also, the 'cursor' parameter of the response is set to 7, i.e., to the 'index' value of the most recent series item included in the response.

After that, the RS follows up with a further diff query request specifying the 'cursor' query parameter with a value of 7 in order to retrieve the next and last batch of series items from the update collection.

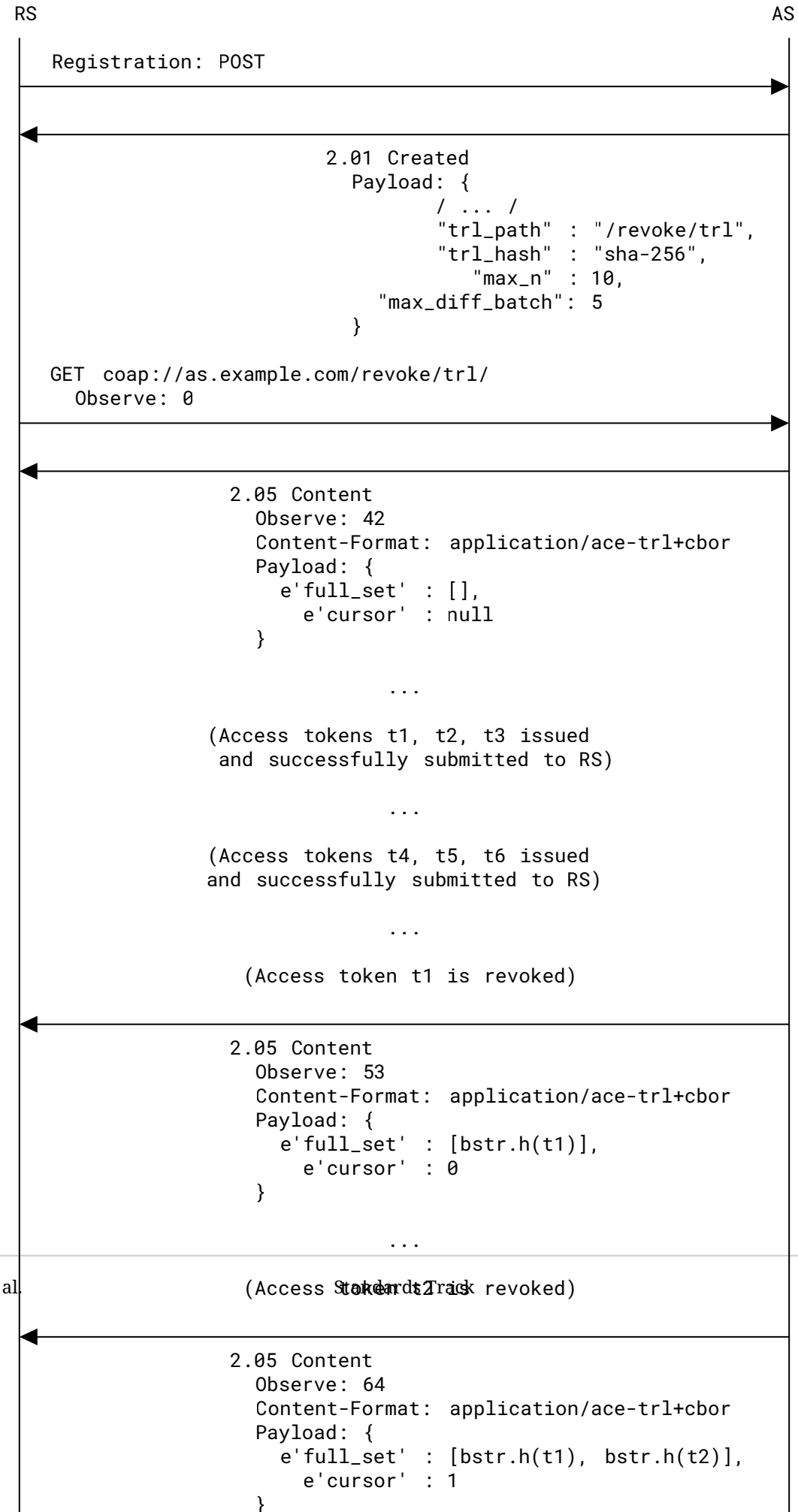
RS
AS
Registration: POST
2.01 Created
Payload: {
/ ... /
"trl_path" : "/revoke/trl",
"trl_hash" : "sha-256",
"max_n" : 10,
"max_diff_batch": 5
}
GET coap://as.example.com/revoke/trl/
Observe: 0
2.05 Content
Observe: 42
Content-Format: application/ace-trl+cbor
Payload: {
e'full_set' : [],
e'cursor' : null
}
...
(Access tokens t1, t2, t3 issued
and successfully submitted to RS)
...
(Access tokens t4, t5, t6 issued
and successfully submitted to RS)
...
(Access token t1 is revoked)
2.05 Content
Observe: 53
Content-Format: application/ace-trl+cbor
Payload: {
e'full_set' : [bstr.h(t1)],
e'cursor' : 0
}
...
(Access token t2 is revoked)
2.05 Content
Observe: 64
Content-Format: application/ace-trl+cbor
Payload: {
e'full_set' : [bstr.h(t1), bstr.h(t2)],
e'cursor' : 1
}

Figure 14: Interaction for Full Query with Observe and Diff Query with "Cursor" Extension

Appendix D. CDDL Model

```
full_set = 0
diff_set = 1
cursor = 2
more = 3

ace-trl-error = 1
```

Figure 15: CDDL Model

Acknowledgments

Ludwig Seitz contributed as a coauthor of initial versions of this document.

The authors sincerely thank Christian Amsüss, Carsten Bormann, Deb Cooley, Roman Danyliw, Dhruv Dhody, Rikard Höglund, Benjamin Kaduk, David Navarro, Joerg Ott, Marco Rasori, Michael Richardson, Kyle Rose, Zaheduzzaman Sarker, Jim Schaad, Göran Selander, Travis Spencer, Orie Steele, Éric Vyncke, Niklas Widell, Dale Worley, and Paul Wouters for their comments and feedback.

The work on this document has been partly supported by the Sweden's Innovation Agency VINNOVA and the Celtic-Next projects CRITISEC and CYPRESS; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Kista
Sweden
Email: marco.tiloca@ri.se

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
SE-16440 Kista
Sweden
Email: francesca.palombini@ericsson.com

Sebastian Echeverria
CMU SEI
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
United States of America
Email: secheverria@sei.cmu.edu

Grace Lewis
CMU SEI
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
United States of America
Email: glewis@sei.cmu.edu