



IMPLEMENTATION DRAFT

## FIDO Metadata Service

FIDO Alliance Implementation Draft 02 February 2017

### This version:

<https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-metadata-service-v1.1-id-20170202.html>

### Previous version:

<https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-metadata-service-v1.0-ps-20141208.html>

### Editor:

[Rolf Lindemann](#), [Nok Nok Labs, Inc.](#)

### Contributors:

[Brad Hill](#), [PayPal, Inc.](#)  
Davit Baghdasaryan, [Nok Nok Labs, Inc.](#)

Copyright © 2013-2017 [FIDO Alliance](#) All Rights Reserved.

## Abstract

The FIDO Authenticator Metadata Specification defines so-called "Authenticator Metadata" statements. The metadata statements contain the "Trust Anchor" required to validate the attestation object, and they also describe several other important characteristics of the authenticator.

The metadata service described in this document defines a baseline method for relying parties to access the latest metadata statements.

## Status of This Document

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current FIDO Alliance publications and the latest revision of this technical report can be found in the [FIDO Alliance specifications index](#) at <https://www.fidoalliance.org/specifications/>.*

This document was published by the [FIDO Alliance](#) as a Implementation Draft. This document is intended to become a FIDO Alliance Proposed Standard. If you wish to make comments regarding this document, please [Contact Us](#). All comments are welcome.

**This Implementation Draft Specification has been prepared by FIDO Alliance, Inc.** Permission is hereby granted to use the Specification solely for the purpose of implementing the Specification. No rights are granted to prepare derivative works of this Specification. Entities seeking permission to reproduce portions of this Specification for other uses must contact the FIDO Alliance to determine whether an appropriate license for such use is available.

Implementation of certain elements of this Specification may require licenses under third party intellectual property rights, including without limitation, patent rights. The FIDO Alliance, Inc. and its Members and any other contributors to the Specification are not, and shall not be held, responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

THIS FIDO ALLIANCE SPECIFICATION IS PROVIDED "AS IS" AND WITHOUT ANY WARRANTY OF ANY KIND, INCLUDING, WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTY OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Table of Contents

- 1. [Notation](#)
  - 1.1 [Key Words](#)

- 2. [Overview](#)
  - 2.1 [Scope](#)
  - 2.2 [Detailed Architecture](#)
- 3. [Metadata Service Details](#)
  - 3.1 [Metadata TOC Format](#)
    - 3.1.1 [Metadata TOC Payload Entry dictionary](#)
      - 3.1.1.1 [Dictionary `MetadataTOCPayloadEntry` Members](#)
    - 3.1.2 [StatusReport dictionary](#)
      - 3.1.2.1 [Dictionary `StatusReport` Members](#)
    - 3.1.3 [AuthenticatorStatus enum](#)
    - 3.1.4 [RogueListEntry dictionary](#)
      - 3.1.4.1 [Dictionary `RogueListEntry` Members](#)
    - 3.1.5 [Metadata TOC Payload dictionary](#)
      - 3.1.5.1 [Dictionary `MetadataTOCPayload` Members](#)
    - 3.1.6 [Metadata TOC](#)
      - 3.1.6.1 [Examples](#)
    - 3.1.7 [Metadata TOC object processing rules](#)
- 4. [Considerations](#)
- A. [References](#)
  - A.1 [Normative references](#)
  - A.2 [Informative references](#)

## 1. Notation

Type names, attribute names and element names are written as `code`.

String literals are enclosed in “”, e.g. “UAF-TLV”.

In formulas we use “|” to denote byte wise concatenation operations.

The notation `base64url(byte[8..64])` reads as 8-64 bytes of data encoded in base64url, "Base 64 Encoding with URL and Filename Safe Alphabet" [[RFC4648](#)] *without padding*.

Following [[WebIDL-ED](#)], dictionary members are optional unless they are explicitly marked as `required`.

WebIDL dictionary members **must not** have a value of null.

Unless otherwise specified, if a WebIDL dictionary member is DOMString, it **must not** be empty.

Unless otherwise specified, if a WebIDL dictionary member is a List, it **must not** be an empty list.

UAF specific terminology used in this document is defined in [[FIDOGlossary](#)].

All diagrams, examples, notes in this specification are non-normative.

### NOTE

Note: Certain dictionary members need to be present in order to comply with FIDO requirements. Such members are marked in the WebIDL definitions found in this document, as `required`. The keyword `required` has been introduced by [[WebIDL-ED](#)], which is a work-in-progress. If you are using a WebIDL parser which implements [[WebIDL](#)], then you may remove the keyword `required` from your WebIDL and use other means to ensure those fields are present.

## 1.1 Key Words

The key words “`must`”, “`must not`”, “`required`”, “`shall`”, “`shall not`”, “`should`”, “`should not`”, “`recommended`”, “`may`”, and “`optional`” in this document are to be interpreted as described in [[RFC2119](#)].

## 2. Overview

*This section is non-normative.*

[[FIDOMetadataStatement](#)] defines authenticator metadata statements.

These metadata statements contain the trust anchor required to verify the attestation object (more specifically the `KeyRegistrationData` object), and they also describe several other important characteristics of the authenticator, including supported authentication and registration assertion schemes, and key protection flags.

These characteristics can be used when defining policies about which authenticators are acceptable for registration or authentication.

The metadata service described in this document defines a baseline method for relying parties to access the latest metadata statements.

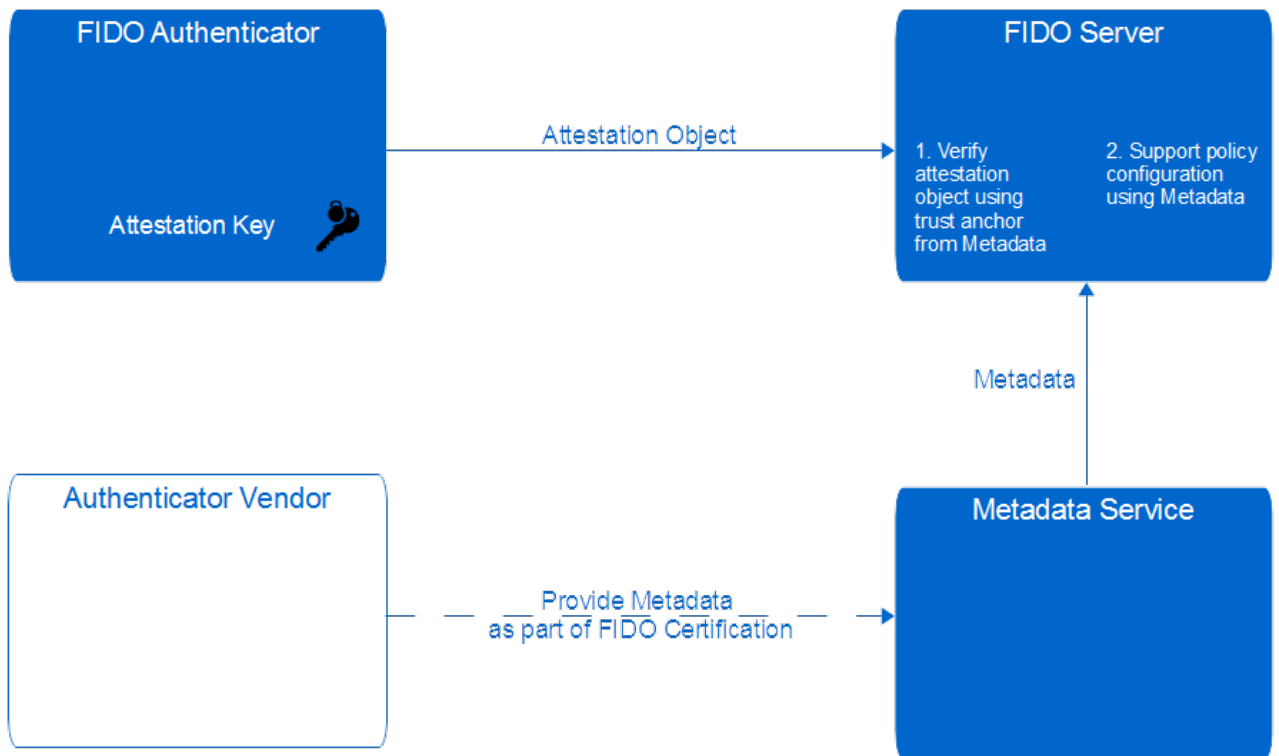


Fig. 1 FIDO Metadata Service Architecture Overview

## 2.1 Scope

This document describes the FIDO Metadata Service architecture in detail and it defines the structure and interface to access this service. It also defines the flow of the metadata related messages and presents the rationale behind the design choices.

## 2.2 Detailed Architecture

The metadata "table-of-contents" (TOC) file contains a list of metadata statements related to the authenticators known to the FIDO Alliance (FIDO Authenticators).

The FIDO Server downloads the metadata TOC file from a well-known FIDO URL and caches it locally.

The FIDO Server verifies the integrity and authenticity of this metadata TOC file using the digital signature. It then iterates through the individual entries and loads the metadata statements related to authenticator AIDs relevant to the relying party.

Individual metadata statements will be downloaded from the URL specified in the entry of the metadata TOC file, and may be cached by the FIDO Server as required.

The integrity of the metadata statements will be verified by the FIDO Server using the hash value included in the related entry of the metadata TOC file.

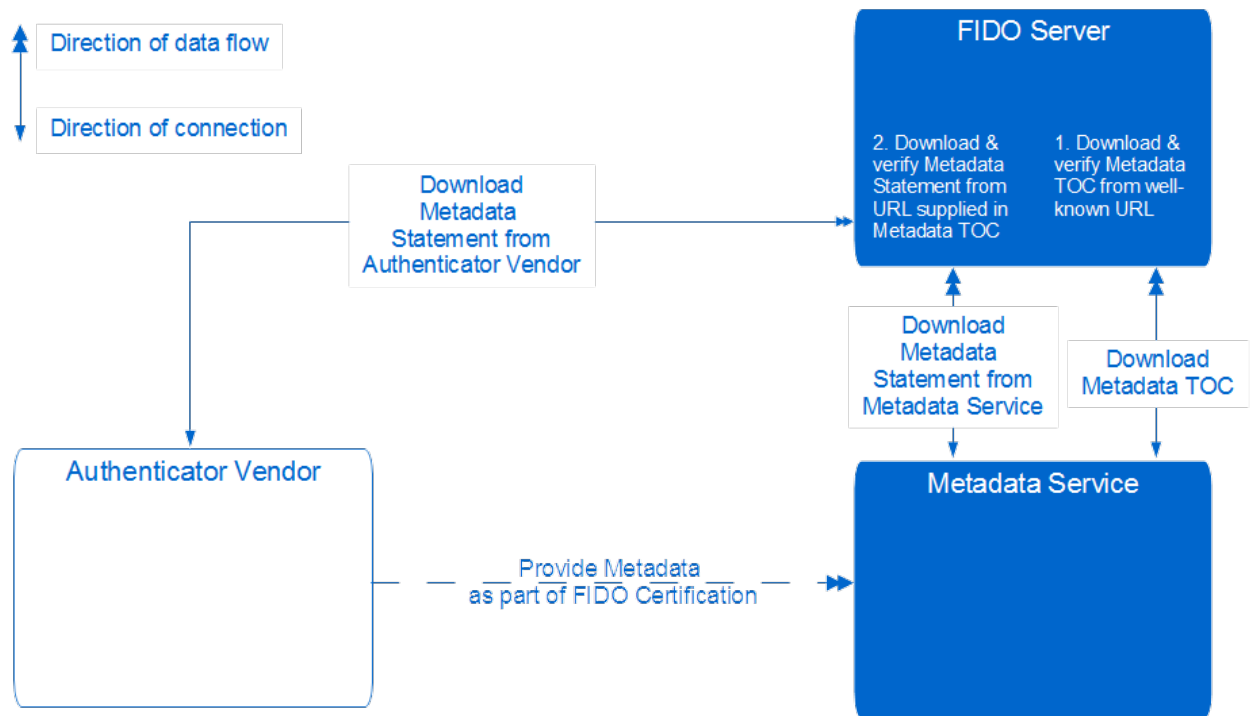


Fig. 2 FIDO Metadata Service Architecture

#### NOTE

The single arrow indicates the direction of the network connection, the double arrow indicates the direction of the data flow.

#### NOTE

The metadata TOC file is freely accessible at a well-known URL published by the FIDO Alliance.

#### NOTE

The relying party decides how frequently the metadata service is accessed to check for metadata TOC updates.

### 3. Metadata Service Details

*This section is normative.*

#### NOTE

The relying party can decide whether it wants to use the metadata service and whether or not it wants to accept certain authenticators for registration or authentication.

The relying party could also obtain metadata directly from authenticator vendors or other trusted sources.

#### 3.1 Metadata TOC Format

#### NOTE

The metadata service makes the metadata TOC object (see [Metadata TOC](#)) accessible to FIDO Servers. This object is a "table-of-contents" for metadata, as it includes the AAID, the download URL and the hash value of the individual metadata statements. The TOC object contains one signature.

### 3.1.1 Metadata TOC Payload Entry dictionary

Represents the MetadataTOCPayloadEntry

#### WebIDL

```
dictionary MetadataTOCPayloadEntry {  
  AAID aaid;  
  AAGUID aaguid;  
  DOMString[] attestationCertificateKeyIdentifiers;  
  required DOMString hash;  
  required DOMString url;  
  required StatusReport[] statusReports;  
  required DOMString timeOfLastStatusChange;  
  DOMString rogueListURL;  
  DOMString rogueListHash;  
};
```

#### 3.1.1.1 Dictionary *MetadataTOCPayloadEntry* Members

##### **aaid** of type **AAID**

The AAID of the authenticator this metadata TOC payload entry relates to. See [UAFProtocol] for the definition of the AAID structure. This field **must** be set if the authenticator implements FIDO UAF.

##### NOTE

FIDO UAF authenticators support AAID, but they don't support AAGUID.

##### **aaguid** of type **AAGUID**

The Authenticator Attestation GUID. See [FIDOKeyAttestation] for the definition of the AAGUID structure. This field **must** be set if the authenticator implements FIDO 2.

##### NOTE

FIDO 2 authenticators support AAGUID, but they don't support AAID.

##### **attestationCertificateKeyIdentifiers** of type array of **DOMString**

A list of the attestation certificate public key identifiers encoded as hex string. This value **must** be calculated according to method 1 for computing the keyIdentifier as defined in [RFC5280] section 4.2.1.2. The hex string **must not** contain any non-hex characters (e.g. spaces). All hex letters **must** be lower case. This field **must** be set if neither **aaid** nor **aaguid** are set. Setting this field implies that the attestation certificate(s) are dedicated to a single authenticator model.

##### NOTE

FIDO U2F authenticators do not support AAID nor AAGUID, but they use attestation certificates dedicated to a single authenticator model.

##### **hash** of type **required DOMString**

`base64url(string[1..512])`

The hash value computed over the base64url encoding of the UTF-8 representation of the JSON encoded metadata statement available at **url** and as defined in [FIDOMetadataStatement]. The hash algorithm related to the signature algorithm specified in the JWTHeader (see [Metadata TOC]) **must** be used.

##### NOTE

This method of base64url encoding the UTF-8 representation is also used by JWT [JWT] to avoid encoding ambiguities.

##### **url** of type **required DOMString**

Uniform resource locator (URL) of the encoded metadata statement for this authenticator model (identified by its AAID, AAGUID or attestationCertificateKeyIdentifier). This URL **must** point to the base64url encoding of the UTF-8 representation of the JSON encoded metadata statement as defined in [FIDOMetadataStatement].

`encodedMetadataStatement = base64url(utf8(JSONMetadataStatement))`

## NOTE

This method of the base64url encoding the UTF-8 representation is also used by JWT [JWT] to avoid encoding ambiguities.

**statusReports** of type array of [required StatusReport](#)  
An array of status reports applicable to this authenticator.

**timeOfLastStatusChange** of type [required DOMString](#)  
ISO-8601 formatted date since when the status report array was set to the current value.

**rogueListURL** of type [DOMString](#)  
URL of a list of rogue (i.e. untrusted) individual authenticators.

**rogueListHash** of type [DOMString](#)  
`base64url(string[1..512])`

The hash value computed over the Base64url encoding of the UTF-8 representation of the JSON encoded rogueList available at [rogueListURL](#) (with type `rogueListEntry[]`). The hash algorithm related to the signature algorithm specified in the JWTHeader (see [Metadata TOC](#)) **must** be used.

This hash value **must** be present and non-empty whenever [rogueListURL](#) is present.

## NOTE

This method of base64url-encoding the UTF-8 representation is also used by JWT [JWT] to avoid encoding ambiguities.

### EXAMPLE 1: UAF Metadata TOC Payload

```
{ "no": 1234, "nextUpdate": "2014-03-31",
  "entries": [
    { "aaid": "1234#5678",
      "hash": "90da8da6de23248abb34da0d4861f4b30a793e198a8d5baa7f98f260db71acd4",
      "url": "https://fidoalliance.org/metadata/1234%x23abcd",
      "rogueListHash": "b5079cf40fd7ed174c645cc04df1e72b7f1229590585d16df62dd20b9541c6b5",
      "rogueListURL": "https://fidoalliance.org/metadata/1234%x23abcd.r1",
      "statusReports": [
        { status: "FIDO_CERTIFIED", effectiveDate: "2014-01-04" }
      ],
      "timeOfLastStatusChange": "2014-01-04"
    },
    { "attestationCertificateKeyIdentifiers": ["7c0903708b87115b0b422def3138c3c864e44573"],
      "hash": "785d16df640fd7b50ed174cb5645cc0f1e72b7f19cf22959052dd20b9541c64d",
      "url": "https://authnr-vendor-a.com/metadata/9876%x234321",
      "statusReports": [
        { status: "FIDO_CERTIFIED", effectiveDate: "2014-01-07"},
        { status: "UPDATE_AVAILABLE", effectiveDate: "2014-02-19",
          url: "https://example.com/update1234" }
      ],
      "timeOfLastStatusChange": "2014-02-19"
    }
  ]
}
```

## NOTE

The character `#` is a reserved character and not allowed in URLs [RFC3986]. As a consequence it has been replaced by its hex value `%x23`.

The authenticator vendors can decide to let the metadata service publish its metadata statements or to publish metadata statements themselves. Authenticator vendors can restrict access to the metadata statements they publish themselves.

### 3.1.2 StatusReport dictionary

## NOTE

Contains an `AuthenticatorStatus` and additional data associated with it, if any.

New `StatusReport` entries will be added to report known issues present in firmware updates.

The latest `StatusReport` entry **must** reflect the "current" status. For example, if the latest entry has status `USER_VERIFICATION_BYPASS`, then it is recommended assuming an increased risk associated with all authenticators of this AAID; if the latest entry has status `UPDATE_AVAILABLE`, then the update is intended to address at least all previous issues *reported* in this `StatusReport` dictionary.

## WebIDL

```
dictionary StatusReport {  
  required AuthenticatorStatus status;  
  DOMString effectiveDate;  
  DOMString certificate;  
  DOMString url;  
};
```

### 3.1.2.1 Dictionary `StatusReport` Members

**status** of type `required AuthenticatorStatus`

Status of the authenticator. Additional fields **may** be set depending on this value.

**effectiveDate** of type `DOMString`

ISO-8601 formatted date since when the status code was set, if applicable. If no date is given, the status is assumed to be effective while present.

**certificate** of type `DOMString`

Base64-encoded [[RFC4648](#)] (not base64url!) DER [[ITU-X690-2008](#)] PKIX certificate value related to the current status, if applicable.

#### NOTE

As an example, this could be an Attestation Root Certificate (see [[FIDOMetadataStatement](#)]) related to a set of compromised authenticators (`ATTESTATION_KEY_COMPROMISE`).

**url** of type `DOMString`

HTTPS URL where additional information may be found related to the current status, if applicable.

#### NOTE

For example a link to a web page describing an available firmware update in the case of status `UPDATE_AVAILABLE`, or a link to a description of an identified issue in the case of status `USER_VERIFICATION_BYPASS`.

### 3.1.3 AuthenticatorStatus enum

This enumeration describes the status of an authenticator model as identified by its AAID and potentially some additional information (such as a specific attestation key).

## WebIDL

```
enum AuthenticatorStatus {  
  "NOT_FIDO_CERTIFIED",  
  "FIDO_CERTIFIED",  
  "USER_VERIFICATION_BYPASS",  
  "ATTESTATION_KEY_COMPROMISE",  
  "USER_KEY_REMOTE_COMPROMISE",  
  "USER_KEY_PHYSICAL_COMPROMISE",  
  "UPDATE_AVAILABLE",  
  "REVOKED",  
  "SELF_ASSERTION_SUBMITTED",  
  "FIDO_SECURITY_CERTIFIED_L1",  
  "FIDO_SECURITY_CERTIFIED_L2",  
  "FIDO_SECURITY_CERTIFIED_L3",  
  "FIDO_SECURITY_CERTIFIED_L4"  
};
```

## Enumeration description

<code>NOT_FIDO_CERTIFIED</code>	This authenticator is not FIDO certified - no functional and no security certification.
<code>FIDO_CERTIFIED</code>	This authenticator has passed FIDO functional certification.
<code>USER_VERIFICATION_BYPASS</code>	Indicates that malware is able to bypass the user verification. This means that the authenticator could be used without the user's consent and potentially even

	without the user's knowledge.
ATTESTATION_KEY_COMPROMISE	Indicates that an attestation key for this authenticator is known to be compromised. Additional data should be supplied, including the key identifier and the date of compromise, if known.
USER_KEY_REMOTE_COMPROMISE	This authenticator has identified weaknesses that allow registered keys to be compromised and should not be trusted. This would include both, e.g. weak entropy that causes predictable keys to be generated or side channels that allow keys or signatures to be forged, guessed or extracted.
USER_KEY_PHYSICAL_COMPROMISE	This authenticator has known weaknesses in its key protection mechanism(s) that allow user keys to be extracted by an adversary in physical possession of the device.
UPDATE_AVAILABLE	<p>A software or firmware update is available for the device. Additional data should be supplied including a URL where users can obtain an update and the date the update was published.</p> <p>When this code is used, then the field <code>authenticatorVersion</code> in the metadata Statement [FIDOMetadataStatement] <b>must</b> be updated, if the update fixes severe security issues, e.g. the ones reported by preceding StatusReport entries with status code <code>USER_VERIFICATION_BYPASS</code>, <code>ATTESTATION_KEY_COMPROMISE</code>, <code>USER_KEY_REMOTE_COMPROMISE</code>, <code>USER_KEY_PHYSICAL_COMPROMISE</code>, <code>REVOKED</code>.</p> <div style="border-left: 2px solid green; padding-left: 10px; margin-top: 10px;"> <p><b>NOTE</b></p> <p>Relying parties might want to inform users about available firmware updates.</p> </div>
REVOKED	The FIDO Alliance has determined that this authenticator should not be trusted for any reason, for example if it is known to be a fraudulent product or contain a deliberate backdoor.
SELF_ASSERTION_SUBMITTED	The authenticator vendor has completed and submitted the self-certification checklist to the FIDO Alliance. If this completed checklist is publicly available, the URL will be specified in <code>StatusReport.url</code> .
FIDO_SECURITY_CERTIFIED_L1	The authenticator has passed a sanctioned third party security validation according to FIDO level 1.
FIDO_SECURITY_CERTIFIED_L2	The authenticator has passed a sanctioned third party security validation according to FIDO level 2.
FIDO_SECURITY_CERTIFIED_L3	The authenticator has passed a sanctioned third party security validation according to FIDO level 3.
FIDO_SECURITY_CERTIFIED_L4	The authenticator has passed a sanctioned third party security validation according to FIDO level 4.

More values might be added in the future. FIDO Servers **must** silently ignore all unknown AuthenticatorStatus values.

### 3.1.4 RogueListEntry dictionary

#### NOTE

Contains a list of individual authenticators known to be rogue.

New `RogueListEntry` entries will be added to report new individual authenticators known to be rogue.

Old `RogueListEntry` entries will be removed if the individual authenticator is known to not be rogue any longer.

#### WebIDL

```
dictionary RogueListEntry {
    required DOMString sk;
    required DOMString date;
};
```

#### 3.1.4.1 Dictionary `RogueListEntry` Members

**sk** of type `required DOMString`

Base64url encoding of the rogue authenticator's secret key (sk value, see [FIDOEcdaaAlgorithm], section ECDAAttestation).



## NOTE

In order to revoke an individual authenticator, its secret key (sk) must be known.

**date** of type **required DOMString**

ISO-8601 formatted date since when this entry is effective.

### EXAMPLE 2: RogueListEntry[] example

```
[
  { "sk": "30efa86aa6de25249acb35da0d4861f4b30a793e198a8d5baa7e96f240da51f3",
    "date": "2016-06-07" },
  { "sk": "93de8da6de23248abb34da0d4861f4b30a793e153a8d5bb27f98f260db71acd4",
    "date": "2016-06-09" },
]
```

## 3.1.5 Metadata TOC Payload dictionary

Represents the MetadataTOCPayload

### WebIDL

```
dictionary MetadataTOCPayload {
  required Number          no;
  required DOMString      nextUpdate;
  required MetadataTOCPayloadEntry[] entries;
};
```

### 3.1.5.1 Dictionary MetadataTOCPayload Members

**no** of type **required Number**

The serial number of this UAF Metadata TOC Payload. Serial numbers **must** be consecutive and strictly monotonic, i.e. the successor TOC will have a **no** value exactly incremented by one.

**nextUpdate** of type **required DOMString**

ISO-8601 formatted date when the next update will be provided at latest.

**entries** of type array of **required MetadataTOCPayloadEntry**

List of zero or more MetadataTOCPayloadEntry objects.

## 3.1.6 Metadata TOC

The metadata table of contents (TOC) is a JSON Web Token (see [JWT] and [JWS]).

It consists of three elements:

- The base64url encoding, without padding, of the UTF-8 encoded JWT Header (see example below),
- the base64url encoding, without padding, of the UTF-8 encoded UAF Metadata TOC Payload ( see example at the beginning of section [Metadata TOC Format](#)),
- and the base64url-encoded, also without padding, JWS Signature [JWS] computed over the to-be-signed payload, i.e.

```
tbsPayload = EncodedJWTHeader | "." | EncodedMetadataTOCPayload
```

All three elements of the TOC are concatenated by a period ("."):

```
MetadataTOC = EncodedJWTHeader | "." | EncodedMetadataTOCPayload | "." | EncodedJWSSignature
```

The hash algorithm related to the signing algorithm specified in the JWT Header (e.g. SHA256 in the case of "ES256") **must** also be used to compute the hash of the metadata statements (see section [Metadata TOC Payload Entry Dictionary](#)).

### 3.1.6.1 Examples

*This section is non-normative.*

### EXAMPLE 3: Encoded Metadata Statement

```
eyJ0b250b290Q2VydG1maWNhdGUioiAi
TU1JQ1BUQ0NBZU9nQXdJQkFnSUpBT3VleHVM095MndNQW9HQ0NzR1NNNDlCQU1DTUhzEElEQWVC
```



```
Ww0KICAgEgYwYWFpZCI6ICIXMjM0IzU2NzgiLCAgICAgImhhc2giOiAiOTBkYThkYTZkZTIz
MjQ4YWJiMzRkYTBkNDg2MmY0YjMwYTC5M2UxOThhOGQlYmFhN2Y5OGYyNjBkYjcxYWNkNCIsIA0K
ICAgICAgICAgICAgICAgInN0YXR1cyI6ICJmaWRvQ2VydGlmawVkIlg0KICAgICAgICAgICAgICAgICAgICAgICAgICAg
YXR1c0NoYW5nZSI6ICIlLA0KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
fSwnCiAgIHsgImFhaWQiOiAiOTg3NiM0MzIeIiwgdQogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
ZmQ3YjUwZGxNzRjYjU2NDVjYzBmMWU3MmI3ZjE5Y2YyMjk1OTA1MmRkMjBiOTU0MmM2NGQlLA0K
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
MzQzMjEiLA0KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
U3RhHvZQ2hhbmdlIjogIjIwMTQ0MTI0MTk1LA0KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
MjAxNC0wMS0wNyIgQ0KICBDbDQp9DQo
```

and finally we have to append another period (".") followed by the base64url-encoded signature.

### EXAMPLE 7: JWT

```
eyJ0eXAiOiJKV1QiLAoGImFfszYi6IkdVMTMjU2IiwKICJ4NXQjUzI1NiI6IjcyMzE5NjIyMTBkMjMjZ
M2VjOTkzYTc3YjRnZiIwZmZg50GFInzRjZGy5NzRmZjAyZDkZkZTNmMWVjN2NiOWR1Njg1fQ.
eyJhbGciOiAiAxiMjMOLCAibmV4dC1lcGRhdGUiOiAiMzE5Y2YyMjM0IiwgdQogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
Ww0KICAgEgYwYWFpZCI6ICIXMjM0IzU2NzgiLCAgICAgImhhc2giOiAiOTBkYThkYTZkZTIz
MjQ4YWJiMzRkYTBkNDg2MmY0YjMwYTC5M2UxOThhOGQlYmFhN2Y5OGYyNjBkYjcxYWNkNCIsIA0K
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
Y2Q4YWJiMzRkYTBkNDg2MmY0YjMwYTC5M2UxOThhOGQlYmFhN2Y5OGYyNjBkYjcxYWNkNCIsIA0K
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
YXR1c0NoYW5nZSI6ICIlLA0KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
fSwnCiAgIHsgImFhaWQiOiAiOTg3NiM0MzIeIiwgdQogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
ZmQ3YjUwZGxNzRjYjU2NDVjYzBmMWU3MmI3ZjE5Y2YyMjk1OTA1MmRkMjBiOTU0MmM2NGQlLA0K
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
MzQzMjEiLA0KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
U3RhHvZQ2hhbmdlIjogIjIwMTQ0MTI0MTk1LA0KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
MjAxNC0wMS0wNyIgQ0KICBDbDQp9DQo.
AP-qoJ3VPzj7L6lCE1UzHzjYQnszFQ8d2hJz51sPASgyABK5VXOFnAHZBTQRRkgwGqULy6PtTyUV
zKxM0HrvoyZq
```

### NOTE

The line breaks are for display purposes only.

The signature in the example above was computed with the following ECDSA key

### EXAMPLE 8: ECDSA Key used for signature computation

```
x: d4166ba8843d1731813f46f1af32174b5c2f6013831fb16f12c9c0b18af3a9b4
y: 861bc2f803a2241f4939bd0d8ecd34e468e42f7fdccd424edb1c3ce7c4dd04e
d: 3744c426764f331f153e182d24f133190b6393cea480a8ee1c722fce161fe2d
```

## 3.1.7 Metadata TOC object processing rules

The FIDO Server **must** follow these processing rules:

- The FIDO Server **must** be able to download the latest metadata TOC object from the well-known URL, when appropriate. The `nextUpdate` field of the [Metadata TOC](#) specifies a date when the download **should** occur at latest.
- If the `x5u` attribute is present in the JWT Header, then:
  - The FIDO Server **must** verify that the URL specified by the `x5u` attribute has the same web-origin as the URL used to download the metadata TOC from. The FIDO Server **should** ignore the file if the web-origin differs (in order to prevent loading objects from arbitrary sites).
  - The FIDO Server **must** download the certificate (chain) from the URL specified by the `x5u` attribute [JWS]. The certificate chain **must** be verified to properly chain to the metadata TOC signing trust anchor according to [RFC5280]. All certificates in the chain **must** be checked for revocation according to [RFC5280].
  - The FIDO Server **should** ignore the file if the chain cannot be verified or if one of the chain certificates is revoked.
- If the `x5u` attribute is missing, the chain should be retrieved from the `x5c` attribute. If that attribute is missing as well, Metadata TOC signing trust anchor is considered the TOC signing certificate chain.
- Verify the signature of the Metadata TOC object using the TOC signing certificate chain (as determined by the steps above). The FIDO Server **should** ignore the file if the signature is invalid. It **should** also ignore the file if its number (`no`) is less or equal to the number of the last Metadata TOC object cached locally.
- Write the verified object to a local cache as required.
- Iterate through the individual entries (of type `MetadataTOCPayloadEntry`). For each entry:
  - Ignore the entry if the AAID, AAGUID or attestationCertificateKeyIdentifiers is not relevant to the relying party (e.g. not acceptable by any policy)
  - Download the metadata statement from the URL specified by the field `url`. Some authenticator vendors might require authentication in order to provide access to the data. Conforming FIDO Servers **should**

support the HTTP Basic, and HTTP Digest authentication schemes, as defined in [RFC2617].

3. Check whether the status report of the authenticator model has changed compared to the cached entry by looking at the fields `timeOfLastStatusChange` and `statusReport`. Update the status of the cached entry. It is up to the relying party to specify behavior for authenticators with status reports that indicate a lack of certification, or known security issues. However, the status `REVOKED` indicates significant security issues related to such authenticators.

#### NOTE

Authenticators with an unacceptable status should be marked accordingly. This information is required for building registration and authentication policies included in the registration request and the authentication request [UAFProtocol].

4. Compute the hash value of the (base64url encoding without padding of the UTF-8 encoded) metadata statement downloaded from the URL and verify the hash value to the hash specified in the field `hash` of the metadata TOC object. Ignore the downloaded metadata statement if the hash value doesn't match.
5. Update the cached metadata statement according to the downloaded one.

## 4. Considerations

*This section is non-normative.*

This section describes the key considerations for designing this metadata service.

**Need for Authenticator Metadata** When defining policies for acceptable authenticators, it is often better to describe the required authenticator characteristics in a generic way than to list individual authenticator AIDs. The metadata statements provide such information. Authenticator metadata also provides the trust anchor required to verify attestation objects.

The metadata service provides a standardized method to access such metadata statements.

**Integrity and Authenticity** Metadata statements include information relevant for the security. Some business verticals might even have the need to document authenticator policies and trust anchors used for verifying attestation objects for auditing purposes.

It is important to have a strong method to verify and proof integrity and authenticity and the freshness of metadata statements. We are using a single digital signature to protect the integrity and authenticity of the Metadata TOC object and we protect the integrity and authenticity of the individual metadata statements by including their cryptographic hash values into the Metadata TOC object. This allows for flexible distribution of the metadata statements and the Metadata TOC object using standard content distribution networks.

**Organizational Impact** Authenticator vendors can delegate the publication of metadata statements to the metadata service in its entirety. Even if authenticator vendors choose to publish metadata statements themselves, the effort is very limited as the metadata statement can be published like a normal document on a website. The FIDO Alliance has control over the FIDO certification process and receives the metadata as part of that process anyway. With this metadata service, the list of known authenticators needs to be updated, signed and published regularly. A single signature needs to be generated in order to protect the integrity and authenticity of the metadata TOC object.

**Performance Impact** Metadata TOC objects and metadata statements can be cached by the FIDO Server.

The update policy can be specified by the relying party.

The metadata TOC object includes a date for the next scheduled update. As a result there is *no additional impact* to the FIDO Server during FIDO Authentication or FIDO Registration operations.

Updating the Metadata TOC object and metadata statements can be performed asynchronously. This reduces the availability requirements for the metadata service and the load for the FIDO Server.

The metadata TOC object itself is relatively small as it does not contain the individual metadata statements. So downloading the metadata TOC object does not generate excessive data traffic.

Individual metadata statements are expected to change less frequently than the metadata TOC object. Only the modified metadata statements need be downloaded by the FIDO Server.

**Non-public Metadata Statements** Some authenticator vendors might want to provide access to metadata statements only to their subscribed customers.

They can publish the metadata statements on access protected URLs. The access URL and the cryptographic hash of the metadata statement is included in the metadata TOC object.

**High Security Environments** Some high security environments might only trust internal policy authorities. FIDO Servers in such environments could be restricted to use metadata TOC objects from a proprietary trusted source only. The metadata service is the baseline for most relying parties.

**Extended Authenticator Information** Some relying parties might want additional information about authenticators

before accepting them. The policy configuration is under control of the relying party, so it is possible to only accept authenticators for which additional data is available and meets the requirements.

## A. References

### A.1 Normative references

#### [FIDOMetadataStatement]

B. Hill, D. Baghdasaryan, J. Kemp, *FIDO Metadata Statements v1.0*. FIDO Alliance Implementation Draft. URLs:  
HTML: <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-metadata-statements.html>  
PDF: <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-metadata-statements.pdf>

#### [JWS]

M. Jones; J. Bradley; N. Sakimura. *JSON Web Signature (JWS)*. May 2015. RFC. URL: <https://tools.ietf.org/html/rfc7515>

#### [JWT]

M. Jones; J. Bradley; N. Sakimura. *JSON Web Token (JWT)*. May 2015. RFC. URL: <https://tools.ietf.org/html/rfc7519>

#### [RFC4648]

S. Josefsson, *The Base16, Base32, and Base64 Data Encodings (RFC 4648)*, IETF, October 2006, URL: <http://www.ietf.org/rfc/rfc4648.txt>

#### [RFC5280]

D. Cooper, S. Santesson, s. Farrell, S.Boeyen, R. Housley, W. Polk; *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, IETF, May 2008, URL: <http://www.ietf.org/rfc/rfc5280.txt>

#### [WebIDL-ED]

Cameron McCormack, *Web IDL*, W3C. Editor's Draft 13 November 2014. URL: <http://heycam.github.io/webidl/>

### A.2 Informative references

#### [FIDOEcdaaAlgorithm]

R. Lindemann, J. Camenisch, M. Drijvers, A. Edgington, A. Lehmann, R. Urian, *FIDO ECDA A Algorithm*. FIDO Alliance Implementation Draft. URLs:  
HTML: <fido-ecdaa-v1.1-id-20170202.html>  
PDF: <fido-ecdaa-v1.1-id-20170202.pdf>.

#### [FIDOGlossary]

R. Lindemann, D. Baghdasaryan, B. Hill, J. Hodges, *FIDO Technical Glossary*. FIDO Alliance Implementation Draft. URLs:  
HTML: <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-glossary-v1.1-id-20170202.html>  
PDF: <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-glossary-v1.1-id-20170202.pdf>

#### [FIDOKeyAttestation]

*FIDO 2.0: Key attestation format*. URL: <https://fidoalliance.org/specs/fido-v2.0-ps-20150904/fido-key-attestation-v2.0-ps-20150904.html>

#### [ITU-X690-2008]

*X.690: Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), (T-REC-X.690-200811)*. International Telecommunications Union, November 2008 URL: <http://www.itu.int/rec/T-REC-X.690-200811-1/en>

#### [RFC2119]

S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels* March 1997. Best Current Practice. URL: <https://tools.ietf.org/html/rfc2119>

#### [RFC2617]

J. Franks; P. Hallam-Baker; J. Hostetler; S. Lawrence; P. Leach; A. Luotonen; L. Stewart *HTTP Authentication: Basic and Digest Access Authentication*. June 1999. Draft Standard. URL: <https://tools.ietf.org/html/rfc2617>

#### [RFC3986]

T. Berners-Lee; R. Fielding; L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax* January 2005. Internet Standard. URL: <https://tools.ietf.org/html/rfc3986>

#### [UAFProtocol]

R. Lindemann, D. Baghdasaryan, E. Tiffany, D. Balfanz, B. Hill, J. Hodges, *FIDO UAF Protocol Specification v1.0*. FIDO Alliance Proposed Standard. URLs:  
HTML: <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-protocol-v1.1-id-20170202.html>  
PDF: <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-protocol-v1.1-id-20170202.pdf>

#### [WebIDL]

Cameron McCormack; Boris Zbarsky; Tobie Langel. *Web IDL*. 15 December 2016. W3C Working Draft. URL: <https://www.w3.org/TR/WebIDL-1/>