FIDO UAF Authenticator Metadata Service v1.0

FIDO Alliance Implementation Draft 22 November 2014

This version:

Editor:
Rolf Lindemann, Nok Nok Labs, Inc.

Contributors:
Brad Hill, PayPal, Inc.
Davit Baghdasaryan, Nok Nok Labs, Inc.

Copyright © 2013-2014 FIDO Alliance. All Rights Reserved.

Abstract

The FIDO UAF 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
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 MST 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.

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.

The FIDO UAF specification defines Authenticator Metadata statements [UAFAuthnrMetadata].

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 its AAID, supported authentication and registration assertion schemes, and key protection flags.

These characteristics can be used when defining policies [UAFProtocol] 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 UAF 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 AAIDs relevant to the relying party.

Individual metadata statements will be downloaded from the URL specified in the entry of the 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 UAF 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 {
  required AAID aaid;
  required DOMString hash;
  required DOMString url;
  required StatusReport[] statusReports;
  required DOMString timeOfLastStatusChange;
};
```

3.1.1.1 Dictionary MetadataTOCPayloadEntry Members

**aaid** of type **required AAID**
The AAID of the authenticator this metadata TOC payload entry relates to. See [UAFProtocol] for the definition of the AAID structure.

**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 [UAFAuthnrMetadata]. 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). This URL must point to the base64url encoding of the UTF-8 representation of the JSON encoded Metadata Statement as defined in [UAFAuthnrMetadata].

```
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.

EXAMPLE 1: UAF Metadata TOC Payload

```
{ "no": 1234, "next-update": "2014-03-31",
  "entries": [ 
    { "aaid": "1234#5678",
      "hash": "90da8da6de23248abb34da0d4861f4b30a793e198a8d5b5a1f798f260db71ac4d",
      "url": "https://fidoalliance.org/metadata/1234%x23abcd",
      "statusReports": [ 
        { status: "FIDO_CERTIFIED", effectiveDate: "2014-01-04" }
      ],
      "timeOfLastStatusChange": "2014-01-04"
    },
    { "aaid": "9876#4321",
      "hash": "785d16df640fd7b50ed174c6b5645c0f1e72b7f19e22959052dd20b9541c64d",
      "url": "https://authnr-vendor-a.com/metadata/9876%x23421",
      "statusReports": [ 
        { status: "FIDO_CERTIFIED", effectiveDate: "2014-01-07"},
        { status: "UPDATE_AVAILABLE", effectiveDate: "2014-03-08",
          url: "https://example.com/update1234" }
      ],
      "timeOfLastStatusChange": "2014-02-19"
    }
  ]
}
```
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 [UAFAuthnrMetadata]) 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 {
    "FIDO_CERTIFIED",
    "NOT_FIDO_CERTIFIED",
    "USER_VERIFICATION_BYPASS",
    "ATTESTATION_KEY_COMPROMISE",
};
```
Enumeration description

<table>
<thead>
<tr>
<th>Enumerated Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIDO_CERTIFIED</td>
<td>This authenticator is FIDO certified.</td>
</tr>
<tr>
<td>NOT_FIDO_CERTIFIED</td>
<td>This authenticator is not FIDO certified.</td>
</tr>
<tr>
<td>USER_VERIFICATION_BYPASS</td>
<td>Indicates that malware is able to bypass the user verification. This means that the authenticator could be used without user's consent and potentially even without user's knowledge.</td>
</tr>
<tr>
<td>ATTESTATION_KEY_COMPROMISE</td>
<td>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.</td>
</tr>
<tr>
<td>USER_KEY_REMOTE_COMPROMISE</td>
<td>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.</td>
</tr>
<tr>
<td>USER_KEY_PHYSICAL_COMPROMISE</td>
<td>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.</td>
</tr>
<tr>
<td>UPDATE_AVAILABLE</td>
<td>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. When this code is used, then the field AuthenticatorVersion in the metadata Statement [UAFAuthnMetadata] must be updated, if the update fixes severe security issues, e.g. the ones reported by preceding StatusReport entries with status code USER_VERIFICATION_BYPASS, ATTESTATION_KEY_COMPROMISE, USER_KEY_REMOTE_COMPROMISE, USER_KEY_PHYSICAL_COMPROMISE, REVOKED.</td>
</tr>
<tr>
<td>REVOKED</td>
<td>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.</td>
</tr>
</tbody>
</table>

NOTE
Relying parties might want to inform users about available firmware updates.

3.1.4 Metadata TOC Payload Dictionary

Represents the MetadataTOCPayload

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

3.1.4.1 Dictionary MetadataTOCPayload Members

- **no** of type required Number
  The serial number of this UAF Metadata TOC Payload. Serial numbers must be consecutive and strictly monotonical, 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.

3.1.5 Metadata TOC

The metadata table of contents (TOC) is a JSON Web Token (see [JWT] and [JWS]). It consists of three elements:
The hash algorithm related to the signing algorithm specified in the JWT Header (e.g. SHA256 in the "ES256") must also be used to compute the hash of the metadata data (see section Metadata TOC Payload Entry Dictionary).

3.1.5.1 Examples

**EXAMPLE 2: Encoded Metadata Statement**

eyaAIuQFRCJCI6CICxKwMOJIS2zg1IA0KIAclXQOKROXNtXbpx2s5b29q0QvzyGlmaWtCGUlOIAi
TULjQ1BQUN8RU9qxDkQxPSbV73Ie1VH09M5nGc63QHH0qRX1CINNDC1U1TJhHzeEEW0C02Z050W
FPN6MO3eRq6G0JyNau0JCE2hShBcMmrkz222yMcL2tOTBH1U1R3KFRF9URUeEQt7FHI
UIVSUEF61mK2xo0maD0aKA7VFDOR3WNU6WVF4KOFvF1WVFh6Wnr6RTERF01UCU2M6BVFPqDn
3L05q600FTWPN6MO3eRq6NWFCF806NWFCF806NWFCF806NWFCF806NWFCF806NWFCF806NWFCF80
U0X06QWY5R7URU2G6JN0CUN0XK0CUN0XK0CUN0XK0CUN0XK0CUN0XK0CUN0XK0CUN0XK0CUN0XK0
UXd0ku0UTVHNek16IXa0GNTNkFR6xEUF0kQPv6t1XPne6QKvp2C01qTq1lZI1ElV1RFEtC8FZ
Vz6K3F5a12WYJ1uAw4WbGSsC61NGb6Vt12ZERF01UCU2M6BVFPqDn3T7g0Km6S8Kv595KJ1Kw
Wb0CAV0JbQ0UQ06X0JbQ0UQ06X0JbQ0UQ06X0JbQ0UQ06X0JbQ0UQ06X0JbQ0UQ06X0JbQ0UQ06X0JbQ0UQ
UG01QTmD84hJCMN01J0npF4e4as1eUovUF7S0KTFQ7VtS2EU3N0Q4Wd04nnW311spj
em0wU0K3J5R4tU0F311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W311spjU050W31
In order to produce the tbsPayload, we first need the base64url-encoded (without padding) JWT Header:
then we have to append a period ("."), and the base64url encoding of the EncodedMetadataTOCPayload (taken from the example in section Metadata TOC Format):
and finally we have to append another period ("."), and the base64url encoding of the signature.

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

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

NOTE
The line breaks are for display purposes only.

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

3.1.6 Metadata TOC object Processing Rules
The FIDO Server must follow these processing rules:
1. The FIDO Server must be able to download the latest metadata TOC object from the well-known URL, when appropriate. The `nextUpdate` field in the Metadata TOC specifies a date when the download should occur at latest.

2. If the `x5u` attribute is present in the JWT Header, then:
   1. The FIDO Server must verify that the URL specified by the `iss` 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).
   2. 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].
   3. The FIDO Server should ignore the file if the chain cannot be verified or if one of the chain certificates is revoked.

3. If the `x5u` attribute is missing, the Metadata TOC signing trust anchor is considered the TOC signing certificate chain.

4. 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 (`.n`) is less or equal to the number of the last Metadata TOC object cached locally.

5. Write the verified object to a local cache as required.

6. Iterate through the individual entries (of type `MetadataTOCPayloadEntry`). For each entry:
   1. Ignore the entry if the AAID is not relevant to the relying party (e.g. not acceptable by any policy)
   2. Download the metadata statement from the URL specified by the `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 identified by the `AAID` 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 AAIDs. 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 cryptographic their 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.

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


A.2 Informative references


