IMPLEMENTATION DRAFT

FIDO UAF Authenticator Metadata Statements v1.0

FIDO Alliance Implementation Draft 22 November 2014

This version:
https://fidoalliance.org/specs/fido-uaf-authnr-metadata-v1.0-id-20141122.html
Previous version:
https://fidoalliance.org/specs/fido-uaf-authnr-metadata-v1.0-rd-20140209.pdf
Editors:
Brad Hill, PayPal, Inc.
Davit Baghdasaryan, Nok Nok Labs, Inc.
John Kemp, FIDO Alliance

Abstract

FIDO authenticators may have many different form factors, characteristics and capabilities. This document defines a standard means to describe the relevant pieces of information about an authenticator in order to interoperate with it, or to make risk-based policy decisions about transactions involving a particular authenticator.

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.

Table of Contents

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.

DOM APIs are described using the ECMAScript [ECMA-262] bindings for WebIDL [WebIDL-ED].

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.

The FIDO family of protocols enable simpler and more secure online authentication utilizing a wide variety of different devices in a competitive marketplace. Much of the complexity behind this variety is hidden from Relying Party applications, but in order to accomplish the goals of FIDO, Relying Parties must have some means of discovering and verifying various characteristics of authenticators. Relying Parties can learn a subset of verifiable information for authenticators certified by the FIDO Alliance with an Authenticator Metadata statement. The URL to access that Metadata statement is provided by the Metadata TOC file accessible through the Metadata Service [UAFMetadataService].

For definitions of terms, please refer to the FIDO Glossary [FIDOGlossary].

2.1 Scope

This document describes the format of and information contained in *Authenticator Metadata* statements. For a definitive list of possible values for the various types of information, refer to the FIDO Registry of Predefined Values [UAFRegistry].

The description of the processes and methods by which authenticator metadata statements are distributed and the methods how these statements can be verified are described in the UAF Metadata Service Specification [UAFMetadataService].

2.2 Audience

The intended audience for this document includes:

- FIDO authenticator vendors who wish to produce metadata statements for their products.
- FIDO server implementers who need to consume metadata statements to verify characteristics of authenticators and attestation statements, make proper algorithm choices for protocol messages, create policy statements or tailor various other modes of operation to authenticator-specific characteristics.
- FIDO relying parties who wish to
  - create custom policy statements about which authenticators they will accept
  - risk score authenticators based on their characteristics
  - verify attested authenticator IDs for cross-referencing with third party metadata

2.3 Architecture

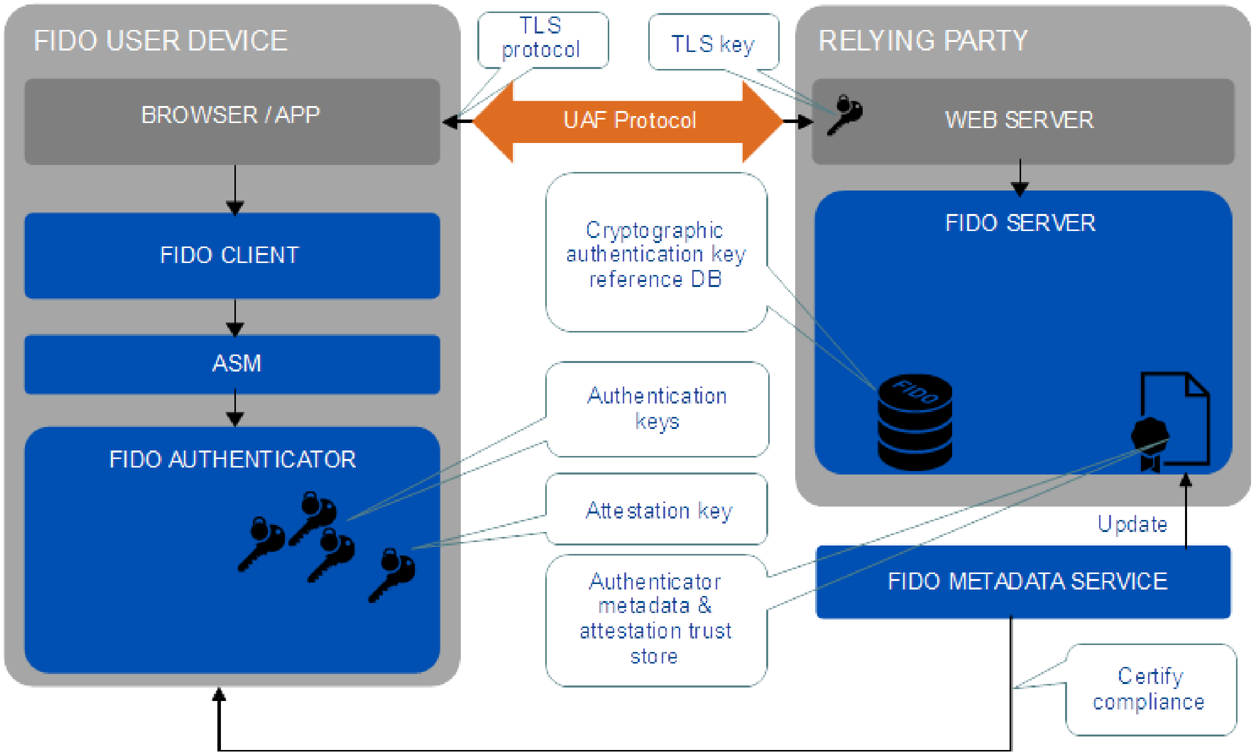

Fig. 1 The UAF Architecture

Authenticator metadata statements are used directly by the FIDO server at a relying party, but the information contained in the authoritative statement is used in several other places. How a server obtains these metadata statements is described in [UAFMetadataService].

The workflow around an authenticator metadata statement is as follows:

1. The authenticator vendor produces a metadata statement describing the characteristics of an authenticator.
2. The metadata statement is submitted to the FIDO Aliance as part of the FIDO certification process. The FIDO Alliance distributes the metadata as described in [UAFMetadataService].
3. A FIDO relying party configures its registration policy to allow authenticators matching certain characteristics to be registered.
4. The FIDO server sends a registration challenge message containing this policy statement.
5. The FIDO UAF Client receives the policy statement as part of the challenge message. It queries available authenticators for their self-reported characteristics and (with the user's input) selects an authenticator that matches the policy, to be registered.
6. The client processes and sends a registration response message to the server. This message contains the AAID for the authenticator and, optionally, a signature made with the private key corresponding to the public key in the authenticator's attestation certificate.
7. The FIDO Server looks up the metadata statement for the authenticator using the authenticator's AAID. If the metadata statement lists an attestation certificate(s), it verifies that an attestation signature is present, and made with the private key corresponding to either (a) one of the certificates listed in this metadata statement or (b) corrsponding to the public key in a certificate that *chains* to one of the issuer certificates listed in the authenticator's metadata statement.
8. The FIDO Server next verifies that the authenticator meets the originally supplied registration policy based on its authoritative metadata statement. This prevents a faulty, modified, or compromised FIDO UAF Client from registering authenticators that are out of policy.
9. *Optionally*, a FIDO Server may, with input from the Relying Party, assign a risk or trust score to the authenticator, based on its metadata, including elements not selected for by the stated policy.
10. *Optionally*, a FIDO Server may cross-reference the attested AAID of the authenticator with other metadata databases published by third parties. Such third-party metadata might, for example, inform the FIDO Server if an authenticator has achieved certifications relevant to certain markets or industry verticals, or whether it meets application-specific regulatory requirements.

3. Types

This section is normative.

3.1 CodeAccuracyDescriptor dictionary

The CodeAccuracyDescriptor describes the relevant accuracy/complexity aspects of passcode user verification methods.

> NOTE
>
> One example of such a method is the use of 4 digit PIN codes for mobile phone SIM card unlock.
>
> We are using the numeral system base (radix) and minLen, instead of the number of potential combinations since there is sufficient evidence [iPhonePasscodes] [MoreTopWorstPasswords] that users don't select their code evenly distributed at random. So software might take into account the various probability distributions for different bases. This essentially means that in practice, passcodes are not as secure as they could be if randomly chosen.

WebIDL

```
dictionary CodeAccuracyDescriptor {
    required unsigned short base;
    required unsigned short minLength;
    unsigned short          maxRetries;
    unsigned short          blockSlowdown;
};
```

3.1.1 Dictionary CodeAccuracyDescriptor Members

base of type required unsigned short
: The numeric system base (radix) of the code, e.g. 10 in the case of decimal digits.

minLength of type required unsigned short
: The minimum number of digits of the given base required for that code, e.g. 4 in the case of 4 digits.

maxRetries of type unsigned short
: Maximum number of false attempts before the authenticator will block this method (at least for some time). 0 means it will never block.

blockSlowdown of type unsigned short
: Enforced minimum number of seconds wait time after blocking (e.g. due to forced reboot or similar). 0 means this user verification method will be blocked, either permanently or until an alternative user verification method method succeeded. All alternative user verification methods must be specified appropriately in the Metadata in userVerificationDetails.

3.2 BiometricAccuracyDescriptor dictionary

The BiometricAccuracyDescriptor describes relevant accuracy/complexity aspects in the case of a biometric user verification method.

> NOTE
>
> The *False Acceptance Rate* (FAR) and *False Rejection Rate* (FRR) values typically are interdependent via the *Receiver Operator Characteristic* (ROC) curve.
>
> The *False Artefact Acceptance Rate* (FAAR) value reflects the capability of detecting presentation attacks, such as the detection of rubber finger presentation.
>
> The FAR, FRR, and FAAR values given here must reflect the actual configuration of the authenticators (as opposed to being theoretical best case values).

At least one of the values must be set. If the vendor doesn't want to specify such values, then VerificationMethodDescriptor.baDesc must be omitted.

WebIDL

```
dictionary BiometricAccuracyDescriptor {
    double         FAR;
    double         FRR;
    double         EER;
    double         FAAR;
    unsigned short maxReferenceDataSets;
```

```
    unsigned short maxRetries;
    unsigned short blockSlowdown;
};
```

3.2.1 Dictionary `BiometricAccuracyDescriptor` Members

`FAR` of type double
: The false acceptance rate [ISO19795-1] for a single reference data set, i.e. the percentage of non-matching data sets that are accepted as valid ones. For example a FAR of `0.1%` would be encoded as `0.001`.

> NOTE
>
> The resulting FAR when all reference data sets are used is `maxReferenceDataSets * FAR`.
>
> The false acceptance rate is relevant for the security. Lower false acceptance rates mean better security.
>
> Only the live captured subjects are covered by this value - not the presentation of artefacts.

`FRR` of type double
: The false rejection rate for a single reference data set, i.e. the percentage of presented valid data sets that lead to a (false) non-acceptance. For example a FRR of `0.1%` would be encoded as `0.001`.

> NOTE
>
> The false rejection rate is relevant for the convenience. Lower false acceptance rates mean better convenience.

`EER` of type double
: The equal error rate for a single reference data set.

`FAAR` of type double
: The false artefact acceptance rate [ISO30107-1], i.e. the percentage of artefacts that are incorrectly accepted by the system. For example a FAAR of `0.1%` would be encoded as `0.001`.

> NOTE
>
> The false artefact acceptance rate is relevant for the security of the system. Lower false artefact acceptance rates imply better security.

`maxReferenceDataSets` of type unsigned short
: Maximum number of alternative reference data sets, e.g. 3 if the user is allowed to enroll 3 different fingers to a fingerprint based authenticator.

`maxRetries` of type unsigned short
: Maximum number of false attempts before the authenticator will block this method (at least for some time). 0 means it will never block.

`blockSlowdown` of type unsigned short
: Enforced minimum number of seconds wait time after blocking (e.g. due to forced reboot or similar). 0 means that this user verification method will be blocked either permanently or until an alternative user verification method succeeded. All alternative user verification methods must be specified appropriately in the metadata in `userVerificationDetails`.

3.3 PatternAccuracyDescriptor dictionary

The `PatternAccuracyDescriptor` describes relevant accuracy/complexity aspects in the case that a pattern is used as the user verification method.

> NOTE
>
> One example of such a pattern is the 3x3 dot matrix as used in Android [AndroidUnlockPattern]

screen unlock. The minComplexity would be 1624 in that case, based on the user choosing a 4-digit PIN, the minimum allowed for this mechanism.

WebIDL

```
dictionary PatternAccuracyDescriptor {
    required unsigned long minComplexity;
    unsigned short         maxRetries;
    unsigned short         blockSlowdown;
};
```

3.3.1 Dictionary PatternAccuracyDescriptor Members

minComplexity of type required unsigned long
: Number of possible patterns (having the minimum length) out of which exactly one would be the right one, i.e. 1/probability in the case of equal distribution.

maxRetries of type unsigned short
: Maximum number of false attempts before the authenticator will block authentication using this method (at least temporarily). 0 means it will never block.

blockSlowdown of type unsigned short
: Enforced minimum number of seconds wait time after blocking (due to forced reboot or similar mechanism). 0 means this user verification method will be blocked, either permanently or until an alternative user verification method method succeeded. All alternative user verification methods must be specified appropriately in the metadata under userVerificationDetails.

3.4 VerificationMethodDescriptor dictionary

A descriptor for a specific *base user verification method* as implemented by the authenticator.

A base user verification method must be chosen from the list of those described in [UAFRegistry]

NOTE

In reality, several of the methods described above might be combined. For example, a fingerprint based user verification can be combined with an alternative password.

The specification of the related AccuracyDescriptor is optional, but recommended.

WebIDL

```
dictionary VerificationMethodDescriptor {
    required unsigned long      userVerification;
    CodeAccuracyDescriptor      caDesc;
    BiometricAccuracyDescriptor baDesc;
    PatternAccuracyDescriptor   paDesc;
};
```

3.4.1 Dictionary VerificationMethodDescriptor Members

userVerification of type required unsigned long
: a *single* USER_VERIFY constant (see [UAFRegistry]), **not a bit flag combination**. This value must be non-zero.

caDesc of type *CodeAccuracyDescriptor*
: May optionally be used in the case of method USER_VERIFY_PASSCODE.

baDesc of type *BiometricAccuracyDescriptor*
: May optionally be used in the case of method USER_VERIFY_FINGERPRINT, USER_VERIFY_VOICEPRINT, USER_VERIFY_FACEPRINT, USER_VERIFY_EYEPRINT, or USER_VERIFY_HANDPRINT.

paDesc of type *PatternAccuracyDescriptor*
: may optionally be used in case of method USER_VERIFY_PATTERN.

3.5 verificationMethodANDCombinations typedef

WebIDL

```
typedef VerificationMethodDescriptor[] VerificationMethodANDCombinations;
```

`VerificationMethodANDCombinations` must be non-empty. It is a list containing the list of base user verification methods which must be passed as part of a successful user verification.

This list will contain only a single entry if using a single user verification method is sufficient.

If this list contains multiple entries, then all of the listed user verification methods must be passed as part of the user verification process.

3.6 rgbPalletteEntry dictionary

The `rgbPalletteEntry` is an RGB three-sample tuple pallete entry

WebIDL

```
dictionary rgbPalletteEntry {
    required unsigned short r;
    required unsigned short g;
    required unsigned short b;
};
```

3.6.1 Dictionary `rgbPalletteEntry` Members

`r` of type required unsigned short
: Red channel sample value

`g` of type required unsigned short
: Green channel sample value

`b` of type required unsigned short
: Blue channel sample value

3.7 DisplayPNGCharacteristicsDescriptor dictionary

The DisplayPNGCharacteristicsDescriptor describes a PNG image characteristics as defined in the PNG [PNG] spec for IHDR (image header) and PLTE (palette table)

WebIDL

```
dictionary DisplayPNGCharacteristicsDescriptor {
    required unsigned long width;
    required unsigned long height;
    required octet         bitDepth;
    required octet         colorType;
    required octet         compression;
    required octet         filter;
    required octet         interlace;
    rgbPalletteEntry[]     plte;
};
```

3.7.1 Dictionary `DisplayPNGCharacteristicsDescriptor` Members

`width` of type required unsigned long
: image width

`height` of type required unsigned long
: image height

`bitDepth` of type required octet
: Bit depth - bits per sample or per palette index.

`colorType` of type required octet
: Color type defines the PNG image type.

`compression` of type required octet
: Compression method used to compress the image data.

`filter` of type required octet

Filter method is the preprocessing method applied to the image data before compression.

interlace of type required octet
Interlace method is the transmission order of the image data.

plte of type array of *rgbPalletteEntry*
1 to 256 palette entries

4. Metadata Keys

This section is normative.

WebIDL

```
dictionary MetadataStatement {
    required AAID                                       aaid;
    required DOMString                                  description;
    required unsigned short                             authenticatorVersion;
    required Version[]                                  upv;
    required DOMString                                  assertionScheme;
    required unsigned short                             authenticationAlgorithm;
    required unsigned short                             publicKeyAlgAndEncoding;
    required unsigned short[]                           attestationTypes;
    required VerificationMethodANDCombinations[]        userVerificationDetails;
    required unsigned short                             keyProtection;
    required unsigned short                             matcherProtection;
    required unsigned long                              attachmentHint;
    required boolean                                    isSecondFactorOnly;
    required unsigned short                             tcDisplay;
    DOMString                                           tcDisplayContentType;
    DisplayPNGCharacteristicsDescriptor[]               tcDisplayPNGCharacteristics;
    required DOMString[]                                attestationRootCertificates;
    required DOMString                                  icon;
};
```

4.1 Dictionary MetadataStatement Members

aaid of type required AAID
The Authenticator Attestation ID. See [UAFProtocol] for the definition of the AAID structure.

description of type required DOMString
A human-readable short description of the authenticator.

> NOTE
>
> This description should help an administrator configuring authenticator policies. This description might deviate from the description returned by the ASM for that authenticator.

authenticatorVersion of type required unsigned short
Earliest (i.e. lowest) trustworthy authenticatorVersion meeting the requirements specified in this metadata statement.

Adding new StatusReport entries with status UPDATE_AVAILABLE to the metadata TOC object [UAFMetadataService] must also change this authenticatorVersion 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.

It is recommended to assume increased risk if this version is higher (newer) than the firmware version present in an authenticator. For example, if a StatusReport entry with status USER_VERIFICATION_BYPASS or USER_KEY_REMOTE_COMPROMISE precedes the UPDATE_AVAILABLE entry, than any firmware version lower (older) than the one specified in the metadata statement is assumed to be vulnerable.

upv of type array of required Version
The UAF protocol version(s) supported by this authenticator. See [UAFProtocol] for the definition of the Version structure.

assertionScheme of type required DOMString
The assertion scheme supported by the Authenticator. Must be set to one of the enumerated Strings defined in the FIDO UAF Registry of Predefined Values [UAFRegistry].

authenticationAlgorithm of type required unsigned short
: The authentication algorithm supported by the authenticator. Must be set to one of the `UAF_ALG` constants defined in the FIDO UAF Registry of Predefined Values [UAFRegistry]. This value must be non-zero.

publicKeyAlgAndEncoding of type required unsigned short
: The public key format used by the authenticator during registration operations. Must be set to one of the `UAF_ALG_KEY` constants defined in the FIDO UAF Registry of Predefined Values [UAFRegistry]. Because this information is not present in APIs related to authenticator discovery or policy, a FIDO server must be prepared to accept and process any and all key representations defined for any public key algorithm it supports. This value must be non-zero.

attestationTypes of type array of required unsigned short
: The supported attestation type(s). (e.g. `TAG_ATTESTATION_BASIC_FULL`) See UAF Registry for more information [UAFRegistry].

userVerificationDetails of type array of required VerificationMethodANDCombinations
: A list *alternative* VerificationMethodANDCombinations. Each of these entries is one alternative user verification method. Each of these alternative user verification methods might itself be an "AND" combination of multiple modalities.

All effectively available alternative user verification methods must be properly specified here. A user verification method is considered effectively available if this method can be used to either:

- enroll new verification reference data to one of the user verification methods

  or

- unlock the UAuth key directly after successful user verification

keyProtection of type required unsigned short
: A 16-bit number representing the bit fields defined by the `KEY_PROTECTION` constants in the FIDO Registry of Predefined Values [UAFRegistry].

This value must be non-zero.

matcherProtection of type required unsigned short
: A 16-bit number representing the bit fields defined by the `MATCHER_PROTECTION` constants in the FIDO Registry of Predefined Values [UAFRegistry].

This value must be non-zero.

> NOTE
>
> If multiple matchers are implemented, then this value must reflect the *weakest* implementation of all matchers.

attachmentHint of type required unsigned long
: A 32-bit number representing the bit fields defined by the `ATTACHMENT_HINT` constants in the FIDO Registry of Predefined Values [UAFRegistry].

> NOTE
>
> The connection state and topology of an authenticator may be transient and cannot be relied on as authoritative by a relying party, but the metadata field should have all the bit flags set for the topologies possible for the authenticator. For example, an authenticator instantiated as a single-purpose hardware token that can communicate over bluetooth should set `ATTACHMENT_HINT_EXTERNAL` but not `ATTACHMENT_HINT_INTERNAL`.

isSecondFactorOnly of type required boolean
: Indicates if the authenticator is designed to be used only as a second factor, i.e. requiring some other authentication method as a first factor (e.g. username+password).

tcDisplay of type required unsigned short
: A 16-bit number representing the bit fields defined by the `TRANSACTION_CONFIRMATIOM_DISPLAY` constants in the FIDO Registry of Predefined Values [UAFRegistry].

This value must be 0, if transaction confirmation is not supported by the authenticator.

tcDisplayContentType of type DOMString
: Supported MIME content type [RFC2049] for the transaction confirmation display, such as `text/plain` or `image/png`.

  This value must be present if transaction confirmation is supported, i.e. `tcDisplay` is non-zero.

tcDisplayPNGCharacteristics of type array of *DisplayPNGCharacteristicsDescriptor*
: A list of *alternative* DisplayPNGCharacteristicsDescriptor. Each of these entries is one alternative of supported image characteristics for displaying a PNG image.

  This list must be present if transaction confirmation is supported, i.e. `tcDisplay` is non-zero.

attestationRootCertificates of type array of required DOMString
: Each element of this array represents a PKIX [RFC5280] trust root X.509 certificate that is valid for this AAID. Multiple certificates might be used for different batches without distinct AAIDs. The array does not represent a certificate chain, but only the trust anchor of that chain.

  Each array element is a Base64-encoded (section 4 of [RFC4648]), DER-encoded [ITU-X690-2008] PKIX certificate value. Each element must be dedicated for authenticator attestation.

  > NOTE
  >
  > A certificate listed here is a trust root. It might be the actual certificate presented by the authenticator, or it might be an issuing authority certificate from the vendor that the actual certificate in the authenticator chains to.
  >
  > The attestation certificate itself and the ordered certificate chain is included in the registration assertion (see [UAFAuthnrCommands]).

  Either

  - the manufacturer attestation root certificate

    or

  - the root certificate related to a specific AAID

  must be specified included here.

  In the case (a), the root certificate might cover multiple authenticator types (i.e. multiple AAIDs). In this case, the AAID must be specified in the SubjectDN CommonName (oid 2.5.4.3) of the Attestation Certificate. In the case (b) it is not required to include the AAID in the SubjectDN CommonName of the attestation certificate, as the root certificate only covers a single AAID.

  In the case of surrogate basic attestation (see [UAFProtocol], section "Surrogate Basic Attestation"), no attestation root certificate is required/used. So this array must be empty in that case.

icon of type required DOMString
: A `data:` url [RFC2397] encoded PNG [PNG] icon for the Authenticator.

5. Metadata Statement Format

This section is non-normative.

> NORMATIVE
>
> A FIDO Authenticator Metadata Statement is a document containing a JSON encoded dictionary MetadataStatement.

Example of the metadata statement for an authenticator with:

- authenticatorVersion 2.
- Fingerprint based user verification with false acceptance rate of 0.001.
- Authenticator is embedded with the FIDO User device.
- The authentication keys are protected by TEE.
- The (fingerprint) matcher is implemented in TEE.

- The Transaction Confirmation Display is implemented in a TEE.
- The Transaction Confirmation Display supports display of "image/png" objects only.
- Display has a width of 320 and a height of 480 pixel. A bit depth of 16 bits per pixel offering True Color (=Color Type 2). The zlib compression method (0). It doesn't support filtering (i.e. filter type of=0) and no interlacing support (interlace method=0).
- The Authentiator can act as first factor or as second factor, i.e. isSecondFactorOnly = false.
- It supports the "UAFV1TLV" assertion scheme.
- It uses the `UAF_ALG_SIGN_ECDSA_SHA256_RAW` authentication algorithm.
- It uses the `UAF_ALG_KEY_ECC_X962_RAW` public key format (0x100=256 decimal).
- It only implements the `TAG_ATTESTATION_BASIC_FULL` method (0x3E07=15879 decimal).
- It implements UAF protocol version 1.0 only.

EXAMPLE 1: MetadataStatement


```
{ "aaid": "1234#5678",
  "description": "FIDO Alliance Sample UAF Authenticator",
  "authenticatorVersion": 2,
  "upv": [{ "major": 1, "minor": 0 }],
  "assertionScheme": "UAFV1TLV",
  "authenticationAlgorithm": 1,
  "publicKeyAlgAndEncoding": 256,
  "attestationTypes": [15879],
  "userVerificationDetails": [ [ { "userVerification": 2, "baDesc": { "FAR": 0.001 } } ] ],
  "keyProtection": 6,
  "matcherProtection": 2,
  "attachmentHint": 1,
  "isSecondFactorOnly": "false",
  "tcDisplay": 4,
  "tcDisplayContentType": ["image/png"],
  "tcDisplayPNGCharacteristics": [{"width": 320, "height": 480, "bitDepth": 16,
        "colorType": 2, "compression": 0, "filter": 0, "interlace": 0}],
  "attestationRootCertificates": [
"MIICPTCCAeOgAwIBAgIJAOuexvU3Oy2wMAoGCCqGSM49BAMCMHsxIDAeBgNVBAMM
F1NhbXBsZSBBdHRlc3RhdGlvbiBSb290MRYwFAYDVQQKDA1GSURPIEFsbGlhbmNl
MREwDwYDVQQLDAhVQUYgVFdHLDESMBAGA1UEBwwJUGFsbyBBbHRvMQswCQYDVQQI
DAJDQTELMAkGA1UEBhMCVVMwHhcNMTQwNjE4MTMzMzMyWhcNNDExMTAzMTMzMzMy
WjB7MSAwHgYDVQQDDBdTYW1wbGUgQXR0ZXN0YXRpb24gUm9vdDEWMBQGA1UECgwN
RklETyBBbGxpYW5jZTERMA8GA1UECwwIVUFGIFRXRywxEjAQBgNVBAcMCVBhbG8g
QWx0bzELMAkGA1UECAwCQ0ExCzAJBgNVBAYTAlVTMFkwEwYHKoZIzj0CAQYIKoZI
zj0DAQcDQgAEH8hv2D0HXa59/BmpQ7RZehL/FMGzFd1QBg9vAUpOZ3ajnuQ94PR7
aMzH33nUSBr8fHYDrqOBb58pxGqHJRyX/6NQME4wHQYDVR0OBBYEFPoHA3CLhxFb
C0It7zE4w8hk5EJ/MB8GA1UdIwQYMBaAFPoHA3CLhxFbC0It7zE4w8hk5EJ/MAwG
A1UdEwQFMAMBAf8wCgYIKoZIzj0EAwIDSAAwRQIhAJ06QSXt9ihIbEKYKIjsPkri
VdLIgtfsbDSu7ErJfzr4AiBqoYCZf0+zI55aQeAHjIzA9Xm63rruAxBZ9ps9z2XN
lQ=="],
  "icon": "data:image/png;base64,
iVBORw0KGgoAAAANSUhEUgAAAE8AAAAvCAYAAACiwJfcAAAAAXNSR0IArs4c6QAAAARnQU1BAACx
jwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAahSURBVGhD7Zr5bxRlGMf9KzTB8AM/YEhE2W7p
QZcWKKBclSpHATlELARE7kNECCA3FkWK0CKKSCFIsKBcgVCDWGNESdAYidwgggJBiRiMhFc/4wy8
884zu9NdlnGTfZJP2n3nO++88933fveBBx+PqCzJkTUvBbLmpUDWvBTImpcCSZvXLCdX9R05Sk19
bb5atf599fG+/erA541q47aP1LLVa9SIyVNUi8Ii8d5kGTsi30NFv7ai9n7QZPMwbdys2erU2XMq
Udy8+ZcaNmGimE8yXN3RUd3a18nF0fUlovZ+0CTzWpd2Vj+eOm1bEyy6Dx4i5pUMGWveo506q227
dtuWBIuffr6oWpV0FPNLhow1751Nm21LvPH3rVtWjfz66Lfql8tX7FRl9YFSXsmSseb9ceOGbYk7
MNUcGPg8ZsbMe9rfQUaaV/JMX9sqdzDCSvp0kZHmTZg9x7bLHcMnThb16eJ+mVfQq8yaUZQNG64i
XZ+0/kq6uOZFO0QtatdWKfXnRQ99Bj91R5OIFnk54jN0mkUiqlO3XDW+Ml+98mKB6tW7rWpZcPc+
0zg4tLrYlUc86E6eGDjIMubVpcusearfgIYGRk6brhZVr/JcHzooL7550jedLExopWcApi2ZUqhu
7JLvrVsQU81zkzOPeemMRYvVuQsX7PbiDQY5JvZonftK+1VY8H9utx530h0ob+jmRYqj6ouaYvEe
nW/WlYjp8cwbMm682tPwq1R4tj/2SH13IRJYl4moZvXpiSqDr7dXtQHxa/PK3/+BWsK1dTgHu6V
8tQJ3bwFkwpFrUOQ50s1r3levm8zZcq17+BBaw7K8lEK5qzkYeark9A8p7P3GzDK+nd3DQow+6UC
8SVN82iuv38im7NtaXtV1CVq6Rgw4pksmbdi3bu2De7YfaBBxcqfvqPrUjFQNTQ22lfdUVVT68rT
JKF5DnSmUjgdqg4mSS9pmsfDJR3G6ToH0iW9aV7LWLHYXKllTDt0LTAtkYIaamp1QjVv++uyGUxV
dJ0DNVXSm+b1qRxpl84ddfX1Lp1O/d69tsod0vs5hGre9xu8o+fpLR1cGhNTD6Z57C9KMWXefJdO
Z94bb9oqd1ROnS7qITTzHimMqivbO3g0DdVyk3WQBhBztK35YKNdOnc8O3acS6fDZFgKaXLsEJp5
rdrliBqp89cJcs/m7Tvs0rkjGfN4b0kPoZn3UJuIOrnZ22yP1fmvUx+O5gSqebV1m+zSuYNVhq7T
WbDiLVvljplLlop6CLXP+2qtvGLIL/1vimISdMBgzSoFZyu6Tqd+jzxgsPaV9BCqee/NjYk6v6lK
9cwiUc/STtf1HDpM3b592y7h3Thx5ozK69Hv5ozLpYWuAwaqS5cv26q7ceb8eb8efVYaRePP3iFU8zj1knSw
ZXHMmnCjY0Ogalo7UQfSCM3qQQr2H/XFP7ssXx45+Yl0fGOB8BBN4
XZzXDb2TXnPL9EwX2KA2k3XIr4XXvXqqXd7ucKp5MRNTWs4tl4KmfeBnE2ttncEbCqy1Gcc6DGw6
ZZQoGKxqRz6nXPzG7ys1ffWncvAQhsx4u3PWGE16DGwbAocYzsgHxrbcnwbY8ntlR4Vrf7kNp5ku
RRaDgHlUmvbcP6ZJuO5UVxMWNs6o1gd2aLk5lL0JR3JvOkaZdYnCMS6nCwqVuP3NhQWVzWDEwK2
HzbqeBrNhfvXqggjW0PVHxW3QmE8yGKfjh1F2YzHm0MWPOkn6E0WcXhdm0tUVaqvn4aFmlxkHiYu
qu8Wp4tR7ln4FQ8UAjYhpq4iWK65sL9xA53Ub6zMnwxGEZzevY7Cxp42Ej0sEwMvnYWlw2iRXfGp
lRqnDCUqqYxwf4rl2FUKsDqV5kIyG5DhJmkWRevuq0gF0BXdKCVQavG4kLQHvCkUKd0Nj7o3uFKv
8Gl5WYhyGPe4X2k3rPIZp0vLXHqDQ2KUC4VTGSGyaUkUsMxqJHX5rL7LSm7KJyfuUCDezG0Q7UsD
5qVA1rwUyJqXAlnzkiai/gHSD7RkTyihogAAAABJRU5ErkJggg=="
}
```

Example of an *User Verification Methods* entry for an authenticator with:

- Fingerprint based user verification method, with:
  - the ability for the user to enroll up to 5 fingers (reference data sets) with
    - a false acceptance rate of 1 in 50000 (0.002%) per finger. This results in a FAR of 0.01% (0.0001).
    - The fingerprint verification will be blocked after 5 unsuccessful attempts.
- A PIN code with a minimum length of 4 decimal digits has to be set-up as alternative verification method. Entering the PIN will be required to re-activate fingerprint based user verification after it has been blocked.

EXAMPLE 2: User Verification Methods Entry

```
[
  [ { "userVerification": 2, "baDesc": { "FAR": 0.00002, "maxReferenceDataSets": 5,
                            "maxRetries": 5, "blockSlowdown": 0} }],
  [ { "userVerification": 4, "caDesc": { "base": 10, "minLength": 4 } } ]
]
```

6. Additional Considerations

This section is non-normative.

6.1 Field updates and metadata

Metadata statements are intended to be stable once they have been published. When authenticators are updated in the field, such updates are expected to improve the authenticator security (for example, improve FRR or FAR). The `authenticatorVersion` must be updated if firmware updates fixing severe security issues (e.g. as reported previously) are available.

NOTE

The metadata statement is assumed to relate to all authenticators having the same AAID.

NOTE

The FIDO Server is recommended to assume increased risk if the `authenticatorVersion` specified in the metadata statement is newer (higher) than the one present in the authenticator.

NORMATIVE

Significant changes in authenticator functionality are not anticipated in firmware updates. For example, if an authenticator vendor wants to modify a PIN-based authenticator to use "Speaker Recognition" as a user verification method, the vendor would must assign a new AAID to this authenticator.

NORMATIVE

A single authenticator implementation could report itself as two "virtual" authenticators using different AAIDs. Such implementations must properly (i.e. according to the security characteristics claimed in the metadata) protect `UAuth` keys and other sensitive data from the other "virtual" authenticator - just as a normal authenticator would do.

NOTE

Authentication keys (`UAuth.pub`) registered for one AAID cannot be used by authenticators reporting a different AAID - even when running on the same hardware (see section "Authentication Response Processing Rules for FIDO Server" in [UAFProtocol]).

A. References

A.1 Normative references

[ISO19795-1]
ISO/IEC JTC 1/SC 37, *Information Technology - Biometric peformance testing and reporting - Part 1: Principles and framework*, URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=41447

[ISO30107-1]
ISO/IEC JTC 1/SC 37, *Information Technology - Biometrics - Presentation attack detection - Part 1: Framework*, URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=53227

[RFC2049]
N. Freed, N. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples (RFC 2049)*, IETF, November 1996, URL: http://www.ietf.org/rfc/rfc2049.txt

[RFC2397]
L. Masinter. *The "data" URL scheme*. August 1998. Proposed Standard. URL: https://tools.ietf.org/html/rfc2397

[WebIDL-ED]
Cameron McCormack, *Web IDL*, W3C. Editor's Draft 13 November 2014. URL: http://heycam.github.io/webidl/

A.2 Informative references

[AndroidUnlockPattern]
Android Unlock Pattern Security Analysis. Sinustrom.info web site. URL: http://www.sinustrom.info/2012/05/21/android-unlock-pattern-security-analysis/'

[ECMA-262]
ECMAScript Language Specification, Edition 5.1. June 2011. URL: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[FIDOGlossary]
R. Lindemann, D. Baghdasaryan, B. Hill, J. Kemp *FIDO Technical Glossary v1.0*. FIDO Alliance Review Draft (Work in progress.) URL: http://fidoalliance.org/specs/fido-glossary-v1.0-rd-20140209.pdf

[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-I/en

[MoreTopWorstPasswords]
10000 Top Passwords, Mark Burnett (Accessed July 11, 2014) URL: https://xato.net/passwords/more-top-worst-passwords/

[PNG]
Tom Lane. *Portable Network Graphics (PNG) Specification (Second Edition)*. 10 November 2003. W3C Recommendation. URL: http://www.w3.org/TR/PNG

[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

[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

[UAFAuthnrCommands]
D. Baghdasaryan, J. Kemp *FIDO UAF Authenticator Commands v1.0*. FIDO Alliance Review Draft (Work in progress.) URL: http://fidoalliance.org/specs/fido-authnr-cmds-v1.0-rd-20140209.pdf

[UAFMetadataService]
R. Lindemann *FIDO UAF Metadata Service v1.0*. FIDO Alliance Working Draft (Work in progress.) URL: TODO

[UAFProtocol]
R. Lindemann, D. Baghdasaryan, E. Tiffany *FIDO UAF Protocol Specification v1.0*. FIDO Alliance Review Draft (Work in progress.) URL: http://fidoalliance.org/specs/fido-uaf-protocol-v1.0-rd-20140209.pdf

[UAFRegistry]
R. Lindemann, D. Baghdasaryan, B. Hill, *FIDO UAF Registry of Predefined Values v1.0*. FIDO Alliance Review Draft (Work in progress.) URL: http://fidoalliance.org/specs/fido-uaf-reg-v1.0-rd-20140209.pdf

[WebIDL]
Cameron McCormack. *Web IDL*. 19 April 2012. W3C Candidate Recommendation. URL: http://www.w3.org/TR/WebIDL/

[iPhonePasscodes]
Most Common iPhone Passcodes, Daniel Amitay (Accessed July 11, 2014) URL: http://danielamitay.com/blog/2011/6/13/most-common-iphone-passcodes