# FIDO Metadata Statements

## FIDO Alliance Review Draft 28 November 2017

**This version:**
https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-metadata-statement-v1.2-rd-20171128.html
**Previous version:**
https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-metadata-statement-v1.1-id-20170202.html
**Editors:**
Rolf Lindemann, Nok Nok Labs, Inc.
John Kemp, FIDO Alliance
**Contributors:**
Brad Hill, PayPal, Inc.
Davit Baghdasaryan, Nok Nok Labs, Inc.



---

## 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 Review 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.

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 Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words must, must not, required, should, should not, recommended, may, and optional in this specification 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 [FIDOMetadataService].

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 [FIDORegistry].

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 Metadata Service Specification [FIDOMetadataService].

### 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

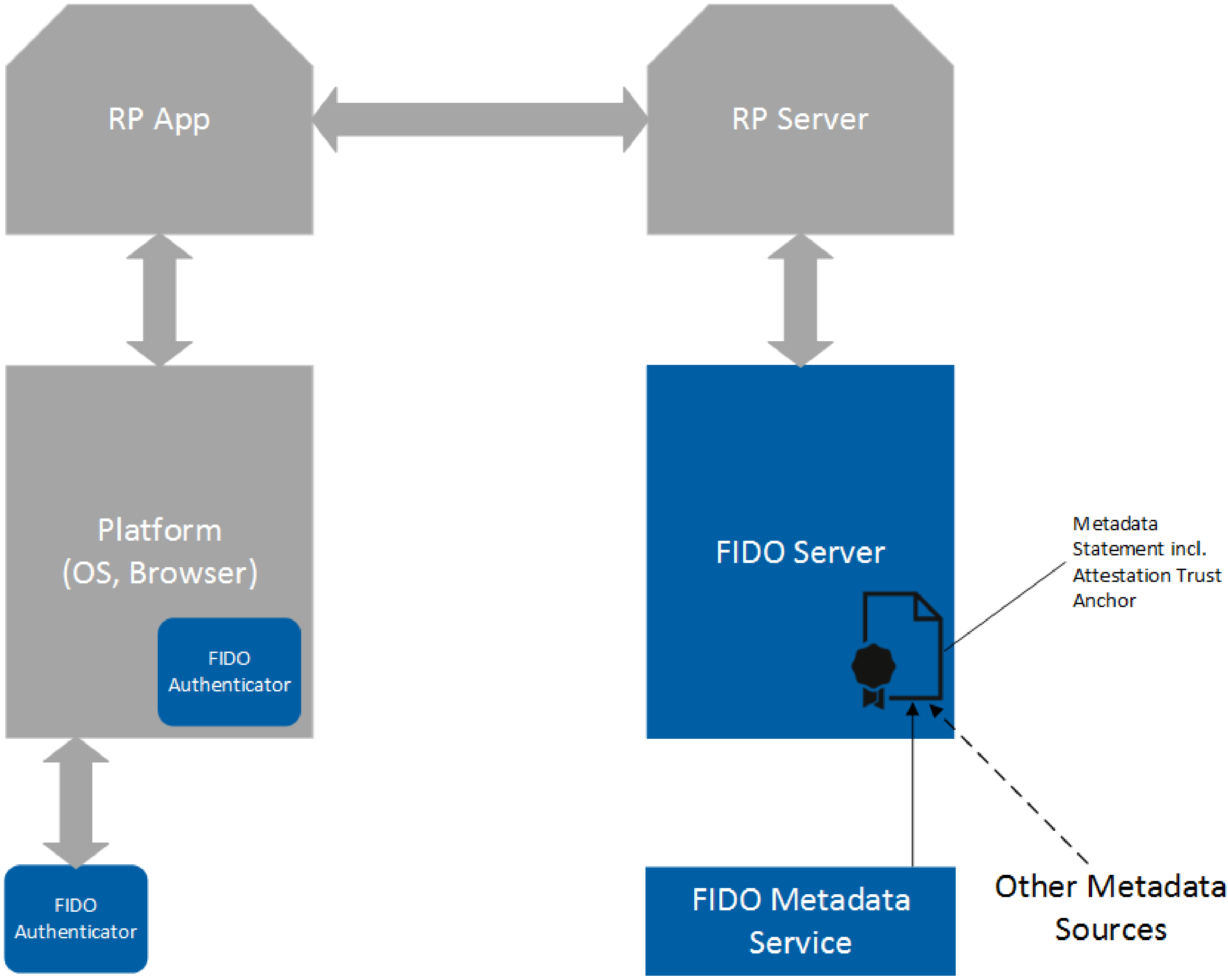


Fig. 1 The FIDO Architecture

*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 [FIDOMetadataService].

The workflow around an authenticator metadata statement is as follows:

1. The authenticator vendor produces a metadata statement, that is UTF-8 encoded, describing the characteristics of an authenticator.
2. The metadata statement is submitted to the FIDO Alliance as part of the FIDO certification process. The FIDO Alliance distributes the metadata as described in [FIDOMetadataService].
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. This message can contain such policy statement.
5. Depending on the FIDO protocol being used, either the relying party application or the FIDO UAF Client receives the policy statement as part of the challenge message and processes it. 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 a reference to the authenticator model 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 particular authenticator model. 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 the registration of unexpected authenticator models.
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 authenticator model 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 Authenticator Attestation GUID (AAGUID) typedef

WebIDL

```
typedef DOMString AAGUID;
```

`string[36]`

Some authenticators have an AAGUID, which is a 128-bit identifier that indicates the type (e.g. make and model) of the authenticator. The AAGUID must be chosen by the manufacturer to be identical across all substantially identical authenticators made by that manufacturer, and different (with probability $1-2^{-128}$ or greater) from the AAGUIDs of all other types of authenticators.

The AAGUID is represented as a string (e.g. "7a98c250-6808-11cf-b73b-00aa00b677a7") consisting of 5 hex strings separated by a dash ("-"), see [RFC4122].

### 3.2 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;
};
```

#### 3.2.1 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`.

### 3.3 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.

NOTE

Typical fingerprint sensor characteristics can be found in Google Android 6.0 Compatibility Definition and Apple iOS Security Guide.

WebIDL

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

### 3.3.1 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 `0.002%` would be encoded as `0.00002`.

> 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 `10%` would be encoded as `0.1`.

> 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`.

## 3.4 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;
};
```

#### 3.4.1 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`.

### 3.5 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 [FIDORegistry]

> 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;
};
```

#### 3.5.1 Dictionary `VerificationMethodDescriptor` Members

`userVerification` of type required unsigned long
: a *single* `USER_VERIFY` constant (see [FIDORegistry]), **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`.

### 3.6 verificationMethodANDCombinations typedef

**WebIDL**

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

`VerificationMethodANDCombinations` must be non-empty. It is a list containing the 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.

## 3.7 rgbPaletteEntry dictionary

The `rgbPaletteEntry` is an RGB three-sample tuple palette entry

**WebIDL**

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

### 3.7.1 Dictionary `rgbPaletteEntry` 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

## 3.8 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;
    rgbPaletteEntry[]      plte;
};
```

### 3.8.1 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 *rgbPaletteEntry*
: 1 to 256 palette entries

## 3.9 EcdaaTrustAnchor dictionary

In the case of ECDAA attestation, the ECDAA-Issuer's trust anchormust be specified in this field.

**WebIDL**

```
dictionary EcdaaTrustAnchor {
    required DOMString X;
    required DOMString Y;
    required DOMString c;
    required DOMString sx;
    required DOMString sy;
    required DOMString G1Curve;
};
```

### 3.9.1 Dictionary `EcdaaTrustAnchor` Members

**`X`** of type required DOMString
: base64url encoding of the result of ECPoint2ToB of the ECPoint2 $X = P_2^x$. See [FIDOEcdaaAlgorithm] for the definition of ECPoint2ToB.

**`Y`** of type required DOMString
: base64url encoding of the result of ECPoint2ToB of the ECPoint2 $Y = P_2^y$. See [FIDOEcdaaAlgorithm] for the definition of ECPoint2ToB.

**`c`** of type required DOMString
: base64url encoding of the result of BigNumberToB($c$). See section "Issuer Specific ECDAA Parameters" in [FIDOEcdaaAlgorithm] for an explanation of $c$. See [FIDOEcdaaAlgorithm] for the definition of BigNumberToB.

**`sx`** of type required DOMString
: base64url encoding of the result of BigNumberToB($sx$). See section "Issuer Specific ECDAA Parameters" in [FIDOEcdaaAlgorithm] for an explanation of $sx$. See [FIDOEcdaaAlgorithm] for the definition of BigNumberToB.

**`sy`** of type required DOMString
: base64url encoding of the result of BigNumberToB($sy$). See section "Issuer Specific ECDAA Parameters" in [FIDOEcdaaAlgorithm] for an explanation of $sy$. See [FIDOEcdaaAlgorithm] for the definition of BigNumberToB.

**`G1Curve`** of type required DOMString
: Name of the Barreto-Naehrig elliptic curve for G1. "BN_P256", "BN_P638", "BN_ISOP256", and "BN_ISOP512" are supported. See section "Supported Curves for ECDAA" in [FIDOEcdaaAlgorithm] for details.

> NOTE
>
> Whenever a party uses this trust anchor for the first time, it must first verify that it was correctly generated by verifying $s$, $sx$, $sy$. See [FIDOEcdaaAlgorithm] for details.

## 3.10 ExtensionDescriptor dictionary

This descriptor contains an extension supported by the authenticator.

WebIDL

```
dictionary ExtensionDescriptor {
    required DOMString id;
    unsigned short     tag;
    DOMString          data;
    required boolean   fail_if_unknown;
};
```

### 3.10.1 Dictionary `ExtensionDescriptor` Members

**`id`** of type required DOMString
: Identifies the extension.

**`tag`** of type unsigned short
: The TAG of the extension if this was assigned. TAGs are assigned to extensions if they could appear in an assertion.

**`data`** of type DOMString
: Contains arbitrary data further describing the extension and/or data needed to correctly process the extension.

  This field may be missing or it may be empty.

**`fail_if_unknown`** of type required boolean
: Indicates whether unknown extensions must be ignored (`false`) or must lead to an error (`true`) when the extension is to be processed by the FIDO Server, FIDO Client, ASM, or FIDO Authenticator.

  - A value of `false` indicates that unknown extensions must be ignored
  - A value of `true` indicates that unknown extensions must result in an error.

### 3.11 AlternativeDescriptions dictionary

This descriptor contains description in alternative languages.

WebIDL

```
dictionary AlternativeDescriptions {
    DOMString *IETFLanguageCodes-members...;
};
```

#### 3.11.1 Dictionary `AlternativeDescriptions` Members

`*IETFLanguageCodes-members...` of type DOMString

IETF language codes ([RFC5646]), defined by a primary language subtag, followed by a region subtag based on a two-letter country code from [ISO3166] alpha-2 (usually written in upper case), e.g: Austrian-German - "de-AT". In case of absence of the specific territorial language definition, vendor should fallback to the more general language option, e.g: If "de" is given, but "de-AT" is missing, the use "de" entry instead.

Description values can contain any UTF-8 characters.

For example: `{ "ru-RU": "Пример U2F аутентификатора от FIDO Alliance", "fr-FR": "Exemple U2F authenticator de FIDO Alliance" }`

Each description shall not exceed a maximum length of 200 characters.

## 4. Metadata Keys

*This section is normative.*

WebIDL

```
dictionary MetadataStatement {
    DOMString                                      legalHeader;
    AAID                                           aaid;
    AAGUID                                         aaguid;
    DOMString[]                                    attestationCertificateKeyIdentifiers;
    required DOMString                             description;
    AlternativeDescriptions                        alternativeDescriptions;
    required unsigned short                        authenticatorVersion;
    DOMString                                      protocolFamily;
    required Version[]                             upv;
    required DOMString                             assertionScheme;
    required unsigned short                        authenticationAlgorithm;
    unsigned short[]                               authenticationAlgorithms;
    required unsigned short                        publicKeyAlgAndEncoding;
    unsigned short[]                               publicKeyAlgAndEncodings;
    required unsigned short[]                      attestationTypes;
    required VerificationMethodANDCombinations[]   userVerificationDetails;
    required unsigned short                        keyProtection;
    boolean                                        isKeyRestricted;
    boolean                                        isFreshUserVerificationRequired;
    required unsigned short                        matcherProtection;
    unsigned short                                 cryptoStrength;
    DOMString                                      operatingEnv;
    required unsigned long                         attachmentHint;
    required boolean                               isSecondFactorOnly;
    required unsigned short                        tcDisplay;
    DOMString                                      tcDisplayContentType;
    DisplayPNGCharacteristicsDescriptor[]          tcDisplayPNGCharacteristics;
    required DOMString[]                           attestationRootCertificates;
    EcdaaTrustAnchor[]                             ecdaaTrustAnchors;
    DOMString                                      icon;
    ExtensionDescriptor                            supportedExtensions[];
};
```

### 4.1 Dictionary `MetadataStatement` Members

`legalHeader` of type DOMString
The legalHeader, if present, contains a legal guide for accessing and using metadata, which itself may contain URL(s) pointing to further information, such as a full Terms and Conditions statement.

`aaid` of type AAID
The Authenticator Attestation ID. 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.

All attestationCertificateKeyIdentifier values should be unique within the scope of the Metadata Service.

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

**description** of type required DOMString
: A human-readable, short description of the authenticator, in English.

> NOTE
>
> This description should help an administrator configuring authenticator policies. This description might deviate from the description returned by the ASM for that authenticator.
>
> This description should contain the public authenticator trade name and the publicly known vendor name.

This description must be in English, and only contain ASCII [ECMA-262] characters.

This description shall not exceed a maximum length of 200 characters.

**alternativeDescriptions** of type *AlternativeDescriptions*
: A list of human-readable short descriptions of the authenticator in different languages.

**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 [FIDOMetadataService] 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.

**protocolFamily** of type DOMString
: The FIDO protocol family. The values "uaf", "u2f", and "fido2" are supported. If this field is missing, the assumed protocol family is "uaf". Metadata Statements for U2F authenticators must set the value of protocolFamily to "u2f" and FIDO 2.0/WebAuthentication Authenticator implementations must set the value of protocolFamily to "fido2".

**upv** of type array of required Version
: The FIDO unified protocol version(s) (related to the specific protocol family) 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], or to "U2FV1BIN" in the case of the U2F raw message format, or to "FIDOV2" in the case of the FIDO 2/WebAuthentication assertion scheme.

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

**authenticationAlgorithms** of type array of unsigned short
: The list of authentication algorithms supported by the authenticator. Must be set to the *complete list* of the supported `ALG_` constants defined in the FIDO Registry of Predefined Values [FIDORegistry] if the authenticator supports multiple algorithms. Each value must be non-zero.

> NOTE
>
> FIDO UAF Authenticators
> : For verification purposes, the field `SignatureAlgAndEncoding` in the FIDO UAF authentication assertion [UAFAuthnrCommands] should be used to determine the actual signature algorithm and encoding.
>
> FIDO U2F Authenticators
> : FIDO U2F only supports one signature algorithm and encoding: `ALG_SIGN_SECP256R1_ECDSA_SHA256_RAW` [FIDORegistry].

**`publicKeyAlgAndEncoding`** of type required unsigned short
: The preferred public key format used by the authenticator during registration operations. Must be set to one of the `ALG_KEY` constants defined in the FIDO Registry of Predefined Values [FIDORegistry]. 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.

**`publicKeyAlgAndEncodings`** of type array of unsigned short
: The list of public key formats supported by the authenticator during registration operations. Must be set to the *complete list* of the supported `ALG_KEY` constants defined in the FIDO Registry of Predefined Values [FIDORegistry] if the authenticator model supports multiple encodings. 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. Each value must be non-zero.

> NOTE
>
> FIDO UAF Authenticators
> : For verification purposes, the field `PublicKeyAlgAndEncoding` in the FIDO UAF registration assertion [UAFAuthnrCommands] should be used to determine the actual encoding of the public key.
>
> FIDO U2F Authenticators
> : FIDO U2F only supports one public key encoding: `ALG_KEY_ECC_X962_RAW` [FIDORegistry].

**`attestationTypes`** of type array of required unsigned short
: The supported attestation type(s). (e.g. `TAG_ATTESTATION_BASIC_FULL(0x3E07)`, `TAG_ATTESTATION_BASIC_SURROGATE(0x3E08)`).
See section 4.1 of FIDO UAF Registry [UAFRegistry], section 5.2.1 of FIDO UAF Authenticator Commands specification [UAFAuthnrCommands], and section 4.1.2 of FIDO UAF Protocol specification [UAFProtocol] for details.

> NOTE
>
> Even though these tags are defined in FIDO UAF protocol specifications, the attestation types apply to authenticators of all protocol families (e.g. UAF, U2F, ...).

**`userVerificationDetails`** of type array of required VerificationMethodANDCombinations
: A list of *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 [FIDORegistry].

This value must be non-zero.

> NOTE
>
> The keyProtection specified here denotes the effective security of the attestation key and Uauth private key and the effective trustworthiness of the attested attributes in the “sign assertion”. Effective security means that key extraction or injecting malicious attested attributes is only possible if the specified protection method is compromised. For example, if keyProtection=TEE is stated, it shall be impossible to extract the attestation key or the Uauth private key or to inject any malicious attested attributes *without breaking the TEE*.

`isKeyRestricted` of type boolean

This entry is set to `true`, if the Uauth private key is restricted by the *authenticator* to only sign valid FIDO signature assertions.

This entry is set to `false`, if the authenticator doesn't restrict the Uauth key to only sign valid FIDO signature assertions. In this case, the calling application could potentially get any hash value signed by the authenticator.

If this field is missing, the assumed value is isKeyRestricted=`true`

.

NOTE

Note that only in the case of isKeyRestricted=`true`, the FIDO server can trust a signature counter or transaction text to have been correctly processed/controlled by the authenticator.

`isFreshUserVerificationRequired` of type boolean

This entry is set to `true`, if Uauth key usage *always* requires a fresh user verification.

If this field is missing, the assumed value is isFreshUserVerificationRequired=`true`.

This entry is set to `false`, if the Uauth key can be used without requiring a fresh user verification, e.g. without any additional user interaction, if the user was verified a (potentially configurable) caching time ago.

In the case of isFreshUserVerificationRequired=`false`, the FIDO server must verify the registration response and/or authentication response and verify that the (maximum) caching time (sometimes also called "authTimeout") is acceptable.

This entry solely refers to the user verification. In the case of transaction confirmation, the authenticator must always ask the user to authorize the specific transaction.

NOTE

Note that in the case of isFreshUserVerificationRequired=`false`, the calling App could trigger use of the key without user involvement. In this case it is the responsibility of the App to ask for user consent.

`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 [FIDORegistry].

This value must be non-zero.

NOTE

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

The matcherProtection specified here denotes the effective security of the FIDO authenticator's user verification. This means that a false positive user verification implies breach of the stated method. For example, if matcherProtection=TEE is stated, it shall be impossible to trigger use of the Uauth private key when bypassing the user verification *without breaking the TEE*.

`cryptoStrength` of type unsigned short

The authenticator's **overall claimed cryptographic strength** in bits (sometimes also called security strength or security level). This is the minimum of the cryptographic strength of all involved cryptographic methods (e.g. RNG, underlying hash, key wrapping algorithm, signing algorithm, attestation algorithm), e.g. see [FIPS180-4], [FIPS186-4], [FIPS198-1], [SP800-38B], [SP800-38C], [SP800-38D], [SP800-38F], [SP800-90C], [SP800-90ar1], [FIPS140-2] etc.

If this value is absent, the cryptographic strength is unknown. If the cryptographic strength of one of the involved cryptographic methods is unknown the overall claimed cryptographic strength is also unknown.

`operatingEnv` of type DOMString

Description of the particular operating environment that is used for the Authenticator. These are specified in [FIDORestrictedOperatingEnv].

`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 [FIDORegistry].

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 a combination of the bit flags defined by the `TRANSACTION_CONFIRMATION_DISPLAY` constants in the FIDO Registry of Predefined Values [FIDORegistry].

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

> NOTE
>
> The tcDisplay specified here denotes the effective security of the authenticator's transaction confirmation display. This means that only a breach of the stated method allows an attacker to inject transaction text to be included in the signature assertion which hasn't been displayed and confirmed by the user.

**`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 PNG-image based transaction confirmation is supported, i.e. `tcDisplay` is non-zero and `tcDisplayContentType` is `image/png`.

**`attestationRootCertificates`** of type array ofrequired DOMString
: Each element of this array represents a PKIX [RFC5280] X.509 certificate that is a valid trust anchor for this authenticator model. Multiple certificates might be used for different batches of the same model. The array does not represent a certificate chain, but only the trust anchor of that chain. A trust anchor can be a root certificate, an intermediate CA certificate or even the attestation certificate itself.

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 anchor. 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.
>
> In the case of "uaf" protocol family, the attestation certificate itself and the ordered certificate chain are included in the registration assertion (see [UAFAuthnrCommands]).

Either

1. the manufacturer attestation trust anchor

   or

2. the trust anchor dedicated to a specific authenticator model

must be specified.

In the case (1), the trust anchor certificate might cover multiple authenticator models. In this case, it must be possible to uniquely derive the authenticator model from the Attestation Certificate. When using AAID or AAGUID, this can be achieved by either specifying the AAID or AAGUID in the attestation certificate using the extension id-fido-gen-ce-aaid { 1 3 6 1 4 1 45724 1 1 1 } or id-fido-gen-ce-aaguid { 1 3 6 1 4 1 45724 1 1 4 } or - when neither AAID nor AAGUID are defined - by using the `attestationCertificateKeyIdentifier` method.

In the case (2) this is not required as the trust anchor only covers a single authenticator model.

When supporting surrogate basic attestation only (see [UAFProtocol], section "Surrogate Basic Attestation"), no attestation trust anchor is required/used. So this array must be empty in that case.

**`ecdaaTrustAnchors`** of type array of *EcdaaTrustAnchor*
: A list of trust anchors used for ECDAA attestation. This entry must be present if and only if attestationType

includes TAG_ATTESTATION_ECDAA. The entries in `attestationRootCertificates` have no relevance for ECDAA attestation. Each ecdaaTrustAnchor must be dedicated to a single authenticator model (e.g as identified by its AAID/AAGUID).

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

**`supportedExtensions[]`** of type *ExtensionDescriptor*
: List of extensions supported by 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.

### 5.1 UAF Example

Example of the metadata statement for an UAF authenticator with:

- authenticatorVersion 2.
- Fingerprint based user verification allowing up to 5 registered fingers, with false acceptance rate of 0.002% and rate limiting attempts for 30 seconds after 5 false trials.
- Authenticator is embedded with the FIDO User device.
- The authentication keys are protected by TEE and are restricted to sign valid FIDO sign assertions only.
- 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 `ALG_SIGN_SECP256R1_ECDSA_SHA256_RAW` authentication algorithm.
- It uses the `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 (upv) 1.0 and 1.1.

EXAMPLE 1: MetadataStatement for UAF Authenticator

```
{
  "aaid": "1234#5678",
  "description": "FIDO Alliance Sample UAF Authenticator",
  "alternativeDescriptions": {
      "ru-RU": "Пример UAF аутентификатора от FIDO Alliance",
      "fr-FR": "Exemple UAF authenticator de FIDO Alliance"
  },
  "authenticatorVersion": 2,
  "upv": [
    { "major": 1, "minor": 0 },
    { "major": 1, "minor": 1 }
  ],
  "assertionScheme": "UAFV1TLV",
  "authenticationAlgorithm": 1,
  "publicKeyAlgAndEncoding": 256,
  "attestationTypes": [15879],
  "userVerificationDetails": [
    [{
      "userVerification": 2,
      "baDesc": {
        "FAR": 0.00002,
        "maxRetries": 5,
        "blockSlowdown": 30,
        "maxReferenceDataSets": 5
      }
    }]
  ],
  "keyProtection": 6,
  "isKeyRestricted": true,
  "matcherProtection": 2,
  "cryptoStrength": 128,
  "operatingEnv": "TEEs based on ARM TrustZone HW",
  "attachmentHint": 1,
  "isSecondFactorOnly": "false",
  "tcDisplay": 5,
  "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+ZcaNmGimE8yXN3RUd3al8nF0fUlovZ+0CTzWpd2Vj+eOm1bEyy6Dx4i5pUMGWveo506q227
        dtuWBIuffr6oWpV0FPNLhow1751Nm21LvPH3rVtWjfz66Lfql8tX7FRl9YFSXsmSseb9ceOGbYk7
        MNUcGPg8ZsbMe9rfQUaaV/JMX9sqdzDCSvp0kZHmTZg9x7bLHcMnThb16eJ+mVfQq8yaUZQNG64i
        XZ+0/kq6uOZFO0QtatdWKfXnRQ99Bj91R5OIFnk54jN0mkUiqlO3XDW+Ml+98mKB6tW7rWpZcPc+
        0zg4tLrYlUc86E6eGDjIMubVpcusearfgIYGRk6brhZVr/JcHzooL7550jedLExopWcApi2ZUqhu
        7JLvrVsQU81zkzOPeemMRYvVuQsX7PbiDQY5JvZonftK+1VY8H9utx530h0ob+jmRYqj6ouaYvEe
        nW/WlYjp8cwbMm682tPwqW1R4tj/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/STtf1HDpM3b592y7h3Thx5ozK69HLpYWuAwaqS5cv26q7ceb8efVYaReP3iFU8zj1knSw
        ZXHMmnCjY0Ogalo7UQfSCM3qQQr2H/XFP7ssXx45Yl91ByeCep4moZoH+1fG3xD4tT7x8kwyj8nw
        b9ev26V0B6d+7H4zKvudAH537FjqyzOHdJnHEuzmXq/WjxObvNMbv7nhywsX2aVsWtC8+48aLeap
        E7p5wKZi0A2AQRV5nvR4E+uJc+b61kApqInxBgmd/4V5QP/mt18HDC7sRHftmeu5lmhV0rn/ALX2
        32bqd4BFnDx7Vi1cWS2uff0IbB47qexxmUj9QutYjupd3tYD6abWBBMrh+apNbOKrNF1+ugCa4ri
        XGfwMPPtViavhU3YMOAAnuUb/R07L0yOSeOadE88ApsXFGff30ynhlJgM51CU6vN9EzgnpvHBFUy
        iVraePiwJ53DF5ZTZnomENg85kNUd2oJi2Wpr4OmmkfN4x4zHfiVFc8Dv8NzuhNqOidilGvA6DGu
        eZwO78AAQn6ciEk6+rw5VcvjvqNDYPOoIUwaKShrxAuXLlkH4aYuGfMYDc10WF5Ta31hPJOfcUhr
        U/JlINi6c6elRYdBpo6++Yfjx61lGNfRm4MD5rJ1j3FoGHnjDSBNarYUgMLyMszKpb7tXpoHfPs8
        h3Wp1LzNfNk54XxC1wDGUmYzXYefh6z/cKtVm4EBxa9VQGDzYr3LrUMRjHEKkk7zaFKYQA2hGQU1
        z+85NFWpXDrkz3vx10GqxQ6BzeNboBk5n8k4nebRh+k1hWfxTF0D1EyWUs5nv+dgQqKaxzuCdE0i
        sHl02NQ8ah0mXr12La3m0f9wik9+wLNTMY/86MPo8yi31OfxmT6PWoqG9+DZukYna56mSZt5WWSy
        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 into the authenticator 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 } } ]
]
```

## 5.2 U2F Example

Example of the metadata statement for an U2F authenticator with:

- authenticatorVersion 2.
- Touch based user presence check.
- Authenticator is a USB pluggable hardware token.
- The authentication keys are protected by a secure element.
- The user presence check is implemented in the chip.
- The Authentiator is a pure second factor authenticator.
- It supports the "U2FV1BIN" assertion scheme.
- It uses the `ALG_SIGN_SECP256R1_ECDSA_SHA256_RAW` authentication algorithm.
- It uses the `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 U2F protocol version 1.0 only.

EXAMPLE 3: MetadataStatement for U2F Authenticator


```
{
  "description": "FIDO Alliance Sample U2F Authenticator",
  "alternativeDescriptions": {
      "ru-RU": "Пример U2F аутентификатора от FIDO Alliance",
      "fr-FR": "Exemple U2F authenticator de FIDO Alliance",
      "zh-CN": "來自FIDO Alliance的示例U2F身份驗證器"
  },
  "attestationCertificateKeyIdentifiers": ["7c0903708b87115b0b422def3138c3c864e44573"],
  "protocolFamily": "u2f",
  "authenticatorVersion": 2,
  "upv": [
    { "major": 1, "minor": 0 }
  ],
  "assertionScheme": "U2FV1BIN",
  "authenticationAlgorithm": 1,
  "publicKeyAlgAndEncoding": 256,
  "attestationTypes": [15879],
  "userVerificationDetails": [
    [{ "userVerification": 1 }]
  ],
  "keyProtection": 10,
  "matcherProtection": 4,
  "cryptoStrength": 128,
  "operatingEnv": "Secure Element (SE)",
  "attachmentHint": 2,
  "isSecondFactorOnly": "true",
  "tcDisplay": 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/WlYjp8cwbMm682tPwqW1R4tj/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/STtf1HDpM3b592y7h3Thx45YNyj8nw
    b9ev26V0B6d+7H4zKvudAH537FjqyzOHdJnHEuzmXq/WjxObvNMbv7nhywsX2aVsWtC8+48aLeap
    E7p5wKZi0A2AQRV5nvR4E+uJc+b61kApqInxBgmd/4V5QP/mt18HDC7sRHftmeu5lmhV0rn/ALX2
    32bqd4BFnDx7Vi1cWS2uff0IbB47qexxmUj9QutYjupd3tYD6abWBBMrh+apNbOKrNF1+ugCa4ri
    XGfwMPPtViavhU3YMOAAnuUb/R07L0yOSeOadE88ApsXFGff30ynhlJgM51CU6vN9EzgnpvHBFUy
    iVraePiwJ53DF5ZTZnomENg85kNUd2oJi2Wpr4OmmkfN4x4zHfiVFc8Dv8NzuhNqOidilGvA6DGu
    eZwO78AAQn6ciEk6+rw5VcvjvqNDYPOoIUwaKShrxAuXLlkH4aYuGfMYDc10WF5Ta31hPJOfcUhr
    U/JlINi6c6elRYdBpo6++Yfjx61lGNfRm4MD5rJ1j3FoGHnjDSBNarYUgMLyMszKpb7tXpoHfPs8
    h3Wp1LzNfNk54XxC1wDGUmYzXYefh6z/cKtVm4EBxa9VQGDzYr3LrUMRjHEKkk7zaFKYQA2hGQU1
    z+85NFWpXDrkz3vx10GqxQ6BzeNboBk5n8k4nebRh+k1hWfxTF0D1EyWUs5nv+dgQqKaxzuCdE0i
    sHl02NQ8ah0mXr12La3m0f9wik9+wLNTMY/86MPo8yi31OfxmT6PWoqG9+DZukYna56mSZt5WWSy
    5qVA1rwUyJqXAlnzkiai/gHSD7RkTyihogAAAABJRU5ErkJggg=="
}
```


## 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 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


**[ECMA-262]**
*ECMAScript Language Specification*. URL: https://tc39.github.io/ecma262/

**[FIDORestrictedOperatingEnv]**
Laurence Lundblade; Meagan Karlsson. *FIDO Authenticator Allowed Restricted Operating Environments List*. August 2016. Draft. URL: https://fidoalliance.org/specs/fido-security-requirements-v1.1-fd-20171108/fido-authenticator-allowed-restricted-operating-environments-list-v1.1-fd-20171108.html

**[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)*. November 1996. URL: http://www.ietf.org/rfc/rfc2049.txt

**[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

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

**[RFC4122]**
P. Leach. *A Universally Unique IDentifier (UUID) URN Namespace*. July 2005. URL: https://tools.ietf.org/html/rfc4122

**[UAFProtocol]**
R. Lindemann; D. Baghdasaryan; E. Tiffany; D. Balfanz; B. Hill; J. Hodges. *FIDO UAF Protocol Specification v1.0*. Proposed Standard. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-uaf-protocol-v1.2-rd-20171128.html

**[UAFRegistry]**
R. Lindemann; D. Baghdasaryan; B. Hill. *FIDO UAF Registry of Predefined Values*. Proposed Standard. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-uaf-reg-v1.2-rd-20171128.html

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


## A.2 Informative references


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

**[FIDOEcdaaAlgorithm]**
R. Lindemann; J. Camenisch; M. Drijvers; A. Edgington; A. Lehmann; R. Urian. *FIDO ECDAA Algorithm*. Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-ecdaa-algorithm-v1.2-rd-20171128.html

**[FIDOGlossary]**
R. Lindemann; D. Baghdasaryan; B. Hill; J. Hodges. *FIDO Technical Glossary*. Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-glossary-v1.2-rd-20171128.html

**[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

**[FIDOMetadataService]**
R. Lindemann; B. Hill; D. Baghdasaryan. *FIDO Metadata Service v1.0*. Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-metadata-service-v1.2-rd-20171128.html

**[FIDORegistry]**
R. Lindemann; D. Baghdasaryan; B. Hill. *FIDO Registry of Predefined Values*. Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-registry-v1.2-rd-20171128.html

**[FIPS140-2]**
*FIPS PUB 140-2: Security Requirements for Cryptographic Modules*. May 2001. URL: http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

**[FIPS180-4]**
*FIPS PUB 180-4: Secure Hash Standard (SHS)*. March 2012. URL: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

**[FIPS186-4]**
*FIPS PUB 186-4: Digital Signature Standard (DSS)*. July 2013. URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

**[FIPS198-1]**
*FIPS PUB 198-1: The Keyed-Hash Message Authentication Code (HMAC)*. July 2008. URL: http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

**[ISO3166]**
*ISO 3166: Codes for the representation of names of countries and their subdivisions*. 2013. ISO 3166-1:2013. URL: http://www.iso.org/iso/catalogue_detail?csnumber=63545

**[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)*. November 2008. URL: http://www.itu.int/rec/T-REC-X.690-200811-I/en

**[MoreTopWorstPasswords]**
Mark Burnett. *10000 Top Passwords*. 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: https://www.w3.org/TR/PNG/

**[RFC4648]**
S. Josefsson. *The Base16, Base32, and Base64 Data Encodings (RFC 4648)*. 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*. May 2008. URL: http://www.ietf.org/rfc/rfc5280.txt

**[RFC5646]**
A. Phillips, Ed.; M. Davis, Ed.. *Tags for Identifying Languages*. September 2009. Best Current Practice. URL: https://tools.ietf.org/html/rfc5646

**[SP800-38B]**
M. Dworkin. *NIST Special Publication 800-38B: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*. May 2005. URL: http://dx.doi.org/10.6028/NIST.SP.800-38B

**[SP800-38C]**
M. Dworkin. *NIST Special Publication 800-38C: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*. July 2007. URL: http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf

**[SP800-38D]**
M. Dworkin. *NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. November 2007 URL: http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf

**[SP800-38F]**
M. Dworkin. *NIST Special Publication 800-38F: Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*. December 2012. URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf

**[SP800-90C]**
Elaine Barker; John Kelsey. *NIST Special Publication 800-90C: Recommendation for Random Bit Generator (RBG) Constructions*. August 2012. URL: http://csrc.nist.gov/publications/drafts/800-90/sp800_90c_second_draft.pdf

**[SP800-90ar1]**
Elaine Barker; John Kelsey. *NIST Special Publication 800-90a: Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. August 2012. URL: http://dx.doi.org/10.6028/NIST.SP.800-90Ar1

**[UAFAuthnrCommands]**
D. Baghdasaryan; J. Kemp; R. Lindemann; R. Sasson; B. Hill. *FIDO UAF Authenticator Commands v1.0*. Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-uaf-authnr-cmds-v1.2-rd-20171128.html

**[WebIDL]**
Cameron McCormack; Boris Zbarsky; Tobie Langel. *Web IDL*. 15 December 2016. W3C Editor's Draft. URL: https://heycam.github.io/webidl/

**[iPhonePasscodes]**
Daniel Amitay. *Most Common iPhone Passcodes*. URL: http://danielamitay.com/blog/2011/6/13/most-common-iphone-passcodes