FIDO Metadata Statement

Review Draft, October 16, 2025

This version:
https://fidoalliance.org/specs/mds/fido-metadata-statement-v3.1.1-rd-20251016.html

Previous Versions:
https://fidoalliance.org/specs/mds/fido-metadata-statement-v3.1-ps-20250521.html

Issue Tracking:
GitHub

Editors:
Billy Jack (Microsoft)
Rolf Lindemann (Nok Nok Labs)

Former Editor:
Yuriy Ackermann (FIDO Alliance)

Copyright © 2025 FIDO Alliance. All Rights Reserved.

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://fidoalliance.org/specifications/.

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

This is a Review Draft Specification and is not intended to be a basis for any implementations as the Specification may change. Permission is hereby granted to use the Specification solely for the purpose of reviewing the Specification. No rights are granted to prepare derivative works of this Specification. Entities seeking permission to reproduce portions of this Specification for other uses must contact the FIDO Alliance to determine whether an appropriate license for such use is available.

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

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

Table of Contents

1. Notation§

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 concatenationoperations.

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: 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.

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

2. Overview§

This section is not 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 Metadata statement can be acquired from the Metadata BLOB that is hosted on 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§

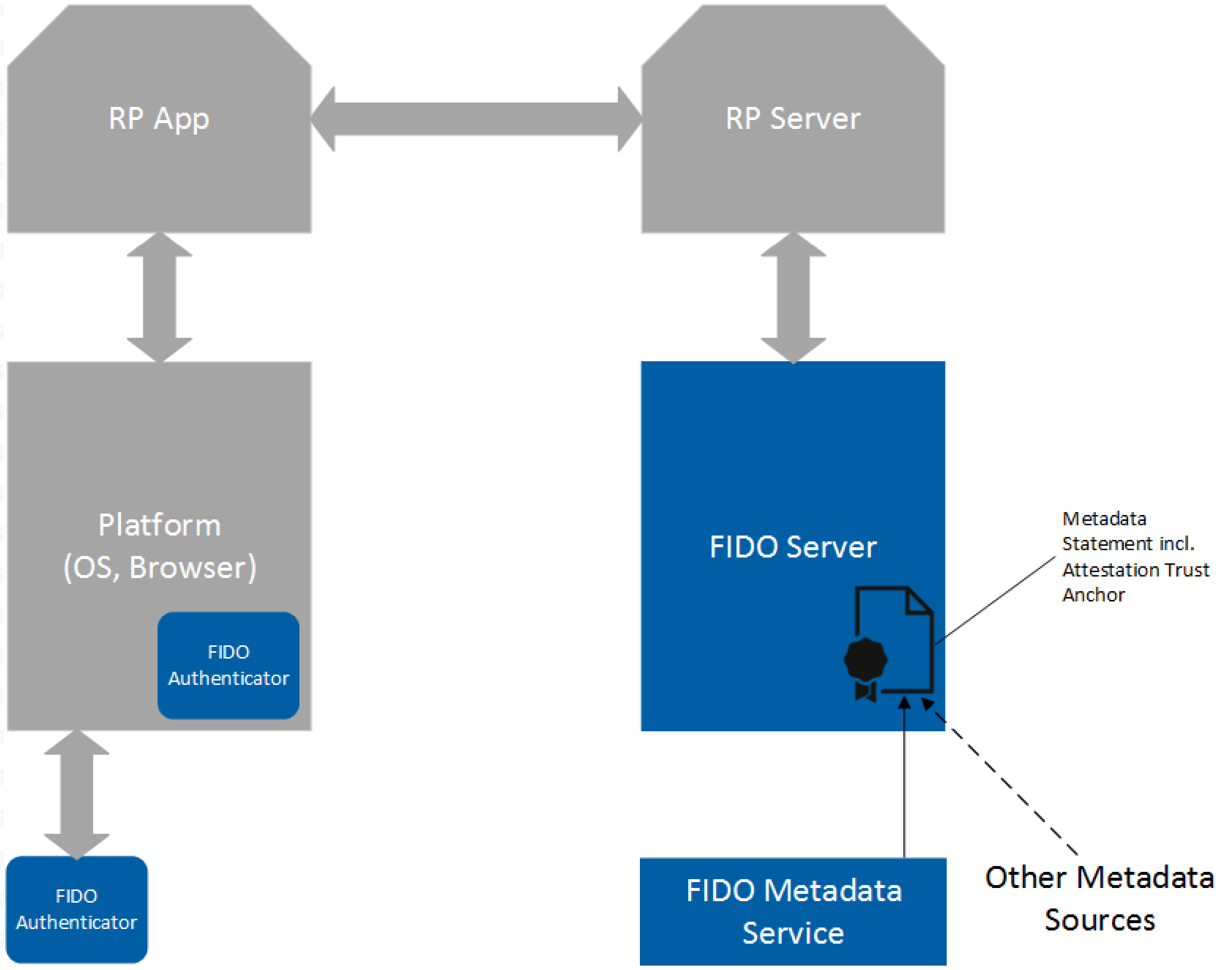

***Figure 1** The FIDO Architecture*

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§

```
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 $1\text{-}2^{-128}$ 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.

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.

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

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

`minLength`, of type 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, see [FIDOBiometricsRequirements].

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 Andoid [AndroidCompatibilityDefinition] and Apple iOS [ApplePlatformSecurity].

```
dictionary BiometricAccuracyDescriptor {
    double          selfAttestedFRR;
    double          selfAttestedFAR;
    double          iAPARThreshold;
    unsigned short maxTemplates;
    unsigned short maxRetries;
    unsigned short blockSlowdown;
};
```

`selfAttestedFRR`, of type double

The false rejection rate [ISOIEC-19795-1] for a single template, i.e. the percentage of verification transactions with truthful claims of identity that are incorrectly denied. For example a FRR of `10%` would be encoded as `0.1`.

This value is self attested and, if the authenticator passed biometric certification, the data is an independently verified FRR as measured when meeting the FRR target specified in the biometric certification requirements [FIDOBiometricsRequirements] for the indicated biometric certification level (see certLevel in related biometricStatusReport as specified in [FIDOMetadataService]).

> Note: The false rejection rate is relevant for user convenience. Lower false rejection rates mean better convenience.

`selfAttestedFAR`, of type double

The false acceptance rate [ISOIEC-19795-1] for a single template, i.e. the percentage of verification transactions with wrongful claims of identity that are incorrectly confirmed. For example a FAR of `0.002%` would be encoded as `0.00002`.

This value is self attested and, if the authenticator passed biometric certification, the data is an independently verified FAR specified in the biometric certification requirements [FIDOBiometricsRequirements] for the indicated biomeric certification level (see certLevel in related biometricStatusReport as specified in [FIDOMetadataService]).

Note: The resulting FAR when all templates are used is approx. `maxTemplates * 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.

`iAPARThreshold`, of type double

Impostor Attack Presentation Accept Rate (IAPAR) is the proportion of impostor attack presentations using the same presentation attack instrument (PAI) species that result in accept [isoiec-30107-3]. For biometric certification requirements [FIDOBiometricsRequirements], certification can be achieved for an IAPAR threshold of less than 7% OR less than 15% for each of the PAI species tested.

If the authenticator did not pass biometric certification, vendor can submit any number, but this number has not been validated for biometric performance requirements.

`maxTemplates`, of type unsigned short

Maximum number of alternative templates from different fingers allowed (for other modalities, multiple parts of the body that can be used interchangeably), e.g. 3 if the user is allowed to enroll up to 3 different fingers to a fingerprint based authenticator.

If the authenticator passed biometric certification this value defaults to 1. For maxTemplates greater than one, it SHALL be independently verified to ensure FAR meets biometric performance requirements of certLevel (of the related biometricStatusReport as specified in [FIDOMetadataService]).

If the authenticator did not pass biometric certification, vendor can submit any number, but this number has not been validated for biometric performance requirements.

`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.

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

`minComplexity`, of type 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.

```
dictionary VerificationMethodDescriptor {
    DOMString                  userVerificationMethod;
    CodeAccuracyDescriptor     caDesc;
    BiometricAccuracyDescriptor baDesc;
    PatternAccuracyDescriptor  paDesc;
};
```

`userVerificationMethod`, of type DOMString

a *single* `USER_VERIFY` constant case-sensitive string name. See section "User Verification Methods" in [FIDO Registry] (e.g. "presence_internal"). This value MUST NOT be empty.

The constant `USER_VERIFY_ALL` MUST NOT be used here.

`caDesc`, of type CodeAccuracyDescriptor

May optionally be used in the case of method `USER_VERIFY_PASSCODE_INTERNAL` or `USER_VERIFY_PASSCODE_EXTERNAL`.

`baDesc`, of type BiometricAccuracyDescriptor

May optionally be used in the case of method `USER_VERIFY_FINGERPRINT_INTERNAL`, `USER_VERIFY_VOICEPRINT_INTERNAL`, `USER_VERIFY_FACEPRINT_INTERNAL`, `USER_VERIFY_EYEPRINT_INTERNAL`, or `USER_VERIFY_HANDPRINT_INTERNAL`.

`paDesc`, of type PatternAccuracyDescriptor

May optionally be used in case of method `USER_VERIFY_PATTERN_INTERNAL` or `USER_VERIFY_PATTERN_EXTERNAL`

3.6. VerificationMethodANDCombinations typedef§

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

`VerificationMethodANDCombinations` list describes a combination of the user verification methods that MUST be passed by the user, in order to achieve successful user verification.

The list MUST NOT be empty.

Each entry in the `VerificationMethodANDCombinations` describes an individual user verification method, that must be passed by the user, as well as some security properties of the user verification method such as pin requirements, biometric properties, etc

3.7. rgbPaletteEntry dictionary§

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

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

r, of type unsigned short
Red channel sample value

g, of type unsigned short
Green channel sample value

b, of type 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)

```
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;
};
```

`width`, of type unsigned long
image width

`height`, of type unsigned long
image height

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

`colorType`, of type octet
Color type defines the PNG image type.

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

`filter`, of type octet
Filter method is the preprocessing method applied to the image data before compression.

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

`plte`, of type rgbPaletteEntry[]
1 to 256 palette entries

3.9. EcdaaTrustAnchor dictionary§

In the case of ECDAA attestation, the ECDAA-Issuer’s trust anchor MUST be specified in this field.

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

X, of type 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 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 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 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 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 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.

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

id, of type 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.

> Examples are `TAG_USER_VERIFICATION_STATE` and `TAG_USER_VERIFICATION_INDEX` as defined in [UAFRegistry].

`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 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. FriendlyNames dictionary§

This descriptor contains friendly names (e.g., public trade name) of the authenticator in multiple languages.

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

***IETFLanguageCodes-members...**
: 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:

```
{
  "en-US": "FIDO Sample Security Key"
}
```

Each entry SHOULD NOT exceed a maximum length of 63 characters to ensure proper display.

3.12. AlternativeDescriptions dictionary§

This descriptor contains description in alternative languages.

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

***IETFLanguageCodes-members...**
: 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.

3.13. AuthenticatorGetInfo dictionary§

This dictionary describes supported versions, extensions, AAGUID of the device and its capabilities.

```
dictionary AuthenticatorGetInfo {
    DOMString members...;
};
```

members...
The members are the fields of the structure reported by an authenticator when invoking the 'authenticatorGetInfo' method, see [FIDOCTAP]. All binary values are base64 encoded.

4. Metadata Keys§

This section is normative.

```
dictionary MetadataStatement {
    DOMString                                legalHeader;
    AAID                                     aaid;
    AAGUID                                   aaguid;
    DOMString[]                              attestationCertificateKeyIdentifiers;
    FriendlyNames                            friendlyNames;
    required DOMString                       description;
    AlternativeDescriptions                  alternativeDescriptions;
    required unsigned long                   authenticatorVersion;
    required DOMString                       protocolFamily;
    required unsigned short                  schema;
    required Version[]                       upv;
    required DOMString[]                     authenticationAlgorithms;
    required DOMString[]                     publicKeyAlgAndEncodings;
    required DOMString[]                     attestationTypes;
    VerificationMethodANDCombinations[]      userVerificationDetails;
    required DOMString[]                     keyProtection;
    boolean                                  isKeyRestricted;
    boolean                                  isFreshUserVerificationRequired;
    required DOMString[]                     matcherProtection;
    unsigned short                           cryptoStrength;
    DOMString[]                              attachmentHint;
    required DOMString[]                     tcDisplay;
    DOMString                                tcDisplayContentType;
    DisplayPNGCharacteristicsDescriptor[]    tcDisplayPNGCharacteristics;
    required DOMString[]                     attestationRootCertificates;
    EcdaaTrustAnchor[]                       ecdaaTrustAnchors;
    DOMString                                icon;
    DOMString                                iconDark;
    DOMString                                providerLogoLight;
    DOMString                                providerLogoDark;
    ExtensionDescriptor[]                    supportedExtensions;
    DOMString                                multiDeviceCredentialSupport;
    AuthenticatorGetInfo                     authenticatorGetInfo;
    DOMString                                cxConfigURL;
};
```

`legalHeader`, of type DOMString

The legalHeader, which must be in each Metadata Statement, is an indication of the acceptance of the relevant legal agreement for using the MDS.

The example of a Metadata Statement legal header is:

"legalHeader": "https://fidoalliance.org/metadata/metadata-statement-legal-header/".

`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.
> It is always expected that the UAF Authenticator (or at least the UAF ASM) knows and provides the correct AAID.

`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 FIDO2.

> Note: FIDO2 Authenticators support AAGUID, but they don’t support AAID.

> For credentials that are backup eligible, restoring them to a different authenticator model/passkey provider might be supported as well. The AAGUID might change over time as a result. Such credentials either have the "Backup Eligibility" flag set or the authenticator metadata statement has multiDeviceCredentialSupport set to "implicit".

`attestationCertificateKeyIdentifiers`, of type 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.

`friendlyNames`, of type FriendlyNames

A human-readable friendly name of the authenticator / passkey provider in multiple languages. The name is intended to be shown to end users. A name in English language ("en-US") is mandatory, localized names for other languages are optional.

`description`, of type 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 unsigned long

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 `BLOB` 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.

The specified version should equal the value of the 'firmwareVersion' member of the `authenticatorGetInfo` response. If present, see [FIDOCTAP].

> The firmware version of the authenticator providing the attestation can be found in the attestation certificate in extension id-fido-gen-ce-fw-version (OID 1.3.6.1.4.1.45724.1.1.5).

`protocolFamily`, of type DOMString

The FIDO protocol family. The values "uaf", "u2f", and "fido2" are supported.

Metadata Statements for U2F authenticators MUST set the value of `protocolFamily` to "u2f". Metadata statement for UAF authenticator MUST set the value of `protocolFamily` to "uaf", and FIDO2/WebAuthentication Authenticator implementations MUST set the value of `protocolFamily` to "fido2".

`schema`, of type unsigned short

The Metadata Schema version

Metadata schema version defines what schema of the metadata statement is currently present. The schema version of this version of the specification is 3.

`upv`, of type Version[]

The FIDO unified protocol version(s) (related to the specific protocol family) supported by this authenticator. See [UAFProtocol] for the formal definition of the `Version` structure (containing major and minor version numbers).

The unified protocol version is determined as follows:

- in the case of FIDO UAF, use the upv value as specified in the respective "OperationHeader" field, see [UAFProtocol].
- in the case of U2F, use
  - major version 1, minor version 0 for U2F v1.0
  - major version 1, minor version 1 for U2F v1.1
  - major version 1, minor version 2 for U2F v1.2 also known as CTAP1
- in the case of FIDO2/CTAP2, use
  - major version 1, minor version 0 for CTAP 2.0
  - major version 1, minor version 1 for CTAP 2.1

`authenticationAlgorithms`, of type DOMString[]

The list of authentication algorithms supported by the authenticator.

Must be set to the *complete list* of the supported `ALG_` constant case-sensitive string names defined in the FIDO Registry of Predefined Values [FIDORegistry] (section "Authentication Algorithms") if the authenticator supports multiple algorithms. E.g. "secp256r1_ecdsa_sha256_raw", "secp256r1_ecdsa_sha256_der".

The list MUST NOT be empty.

> **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].

`publicKeyAlgAndEncodings`, of type DOMString[]
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` constant case-sensitive string names defined in the FIDO Registry of Predefined Values [FIDORegistry] if the authenticator model supports multiple encodings. See section "Public Key Representation Formats", e.g. "ecc_x962_raw", "ecc_x962_der".

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. The list MUST NOT be empty. If there are multiple values they MUST be ordered by preference.

> **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 DOMString[]
Must be set to the *complete list* of the supported `ATTESTATION_` constant case-sensitive string names. See section "Authenticator Attestation Types" of FIDO Registry [FIDORegistry] for all available attestation formats, e.g. "basic_full".

`userVerificationDetails`, of type VerificationMethodANDCombinations[]
A list of *alternative* `VerificationMethodANDCombinations`.

`userVerificationDetails` is a two dimensional array, that informs RP what `VerificationMethodANDCombinations` user may be required to perform in order to pass user verification, e.g User need to pass fingerprint, or faceprint, or password and palm print, etc.

If this entry is missing, any user verification method could be implemented - including "none", "presence_internal" and other methods - including methods not formally defined in the FIDO Registry [FIDO Registry]. Depending on the "protocolFamily", the authentication assertion might contain further details.

Consider this `userVerificationDetails` example:

```
[
  [
    { "userVerificationMethod": "fingerprint_internal" }
  ],
  // OR
  [
    { "userVerificationMethod": "passcode_internal" }
  ],
  // OR
  [
    { "userVerificationMethod": "faceprint_internal"},
    // AND
    { "userVerificationMethod": "voiceprint_internal"}
  ]
]
```

In this example we have user verification details that describe these potential scenarios: User has an authenticator model that requires

1. Fingerprint, or
2. Passcode, or
3. Faceprint and Voiceprint - where Voiceprint and Faceprint must be provided in order to pass user verification.

The RP verifying attestation or assertion, by checking UV flag in the response knows that one of the user verification combinations been passed.

> Note: FIDO2 "Security Keys" will typically support "none", or "presence_internal", or "passcode_external" [FIDOCTAP], i.e.
>
> ```
> [
>   [
>      { "userVerificationMethod": "none" }
>   ],
>   [
>     { "userVerificationMethod": "presence_internal" }
>   ],
>   [
>     { "userVerificationMethod": "passcode_external" }
>   ],
>   [
>     { "userVerificationMethod": "passcode_external" },
>     { "userVerificationMethod": "presence_internal" }
>   ]
> ]
> ```
>
> The FIDO Client will typically prevent "none" (silent authentication) and "passcode_external" (without "presence_internal") from being used in practice, see [WebAuthn].

`keyProtection`, of type DOMString[]

The list of key protection types supported by the authenticator. Must be set to the*complete list* of the supported `KEY_PROTECTION_` constant case-sensitive string names defined in the FIDO Registry of Predefined Values [FIDORegistry] in section "Key Protection Types" e.g. "secure_element". Each value MUST NOT be empty.

> 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: Only in the case of isKeyRestricted=`true`, the FIDO server can trust a signature counter, transaction text, or any other extension in the signature assertion 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 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 DOMString[]

The list of matcher protections supported by the authenticator. Must be set to the *complete list* of the supported `MATCHER_PROTECTION` constant case-sensitive string names defined in the FIDO Registry of Predefined Values [FIDORegistry]. See section "Matcher Protection Types", e.g. "on_chip". This value MUST NOT be empty.

> If multiple user verification methods are supported, either (a) one entry per method or the minimum security level over all methods could be specified.
> If multiple alternative implementations exist, then each entry must reflect the *weakest* implementation of all alternative implementations.
>
> If a user verification method implementation is split across multiple components, then this value must reflect the *weakest* implementation of all those components.
>
> 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). 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.

> See [FIDOAuthenticatorSecurityRequirements], requirement 2.1.4, "Overall Claimed Cryptographic Strength"

`attachmentHint`, of type DOMString[]

The list of supported attachment hints describing the method(s) by which the authenticator communicates with the FIDO user device. Must be set to the *complete list* of the supported `ATTACHMENT_HINT` constant case-sensitive string names defined in the FIDO Registry of Predefined Values [FIDORegistry]. See section "Authenticator Attachment Hints", e.g. "nfc".

This value MUST be present for all authenticators supporting CTAP 2.2 or newer. This value SHOULD be present for all authenticators supporting CTAP 2.1. If the value is provided, the array MUST NOT be empty.

> 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`.
>
> For FIDO2 the values of attachmentHint MUST correspond to the `authenticatorGetInfo.transports` if present.
>
> See the field `authenticatorGetInfo` for FIDO2 authenticators; which expose similar information in the 'transports' member when invoking the 'authenticatorGetInfo' method. See [FIDOCTAP]

`tcDisplay`, of type DOMString[]

The list of supported transaction confirmation display capabilities. Must be set to include a valid combination, as specified in FIDO Registry of Predefined Values [FIDORegistry] section "Transaction Confirmation Display Types", of the supported `TRANSACTION_CONFIRMATION_DISPLAY` constant case-sensitive string names e.g. "any", "hardware".

This value MUST be empty, if transaction confirmation is not supported by the authenticator.

> 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 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 DOMString[]

List of attestation trust anchors for the batch chain in the authenticator attestation. 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 (1) be the actual certificate presented by the authenticator, or it might (2) be an issuing authority certificate from the vendor that the attestation certificate chains to. In the case of (1), a binary comparison is sufficient to determine if the attestation trust anchor is the attestation certificate itself.
> 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 EcdaaTrustAnchor[]
A list of trust anchors used for ECDAA attestation. This entry MUST be present if and only if attestationType includes 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).

> Note: This field only applies to UAF authenticators.

`icon`, of type DOMString
A `data:` url [RFC2397] encoded [PNG] or [SVG11] (light mode) icon for the Authenticator (e.g., depicting the security key). This icon is intended to be shown to users by RPs. Use of [SVG11] format is mandatory if any of the `iconDark`, `providerLogoLight` and/or `providerLogoDark` is used in addition to `icon`. Use of [SVG11] is recommended if only `icon` is used. The icon is more specific than the provider logo and should be shown if present.

`iconDark`, of type DOMString
A `data:` url [RFC2397] encoded [SVG11] dark mode icon for the Authenticator (e.g., depicting the security key). This icon is intended to be shown to users by RPs. The icon is more specific than the provider logo and should be shown if present.

`providerLogoLight`, of type DOMString
A `data:` url [RFC2397] encoded [SVG11] light mode icon for the provider (e.g., logomark of the passkey provider). The SVG MUST meet all of the requirements defined in § 4.1 SVG requirements. This icon is intended to be shown to users by RPs.

`providerLogoDark`, of type DOMString
A `data:` url [RFC2397] encoded [SVG11] dark mode icon for the provider (e.g., logomark of the passkey provider). The SVG MUST meet all of the requirements defined in § 4.1 SVG requirements. This icon is intended to be shown to users by RPs.

`supportedExtensions`, of type ExtensionDescriptor[]
List of extensions supported by the FIDO UAF authenticator. This field only applies to UAF authenticators. For FIDO2 authenticators see `authenticatorGetInfo`

`multiDeviceCredentialSupport`, of type DOMString

When set to "unsupported" it means that all private keys relating to thePublic Key Credential Source [WebAuthn] are designed to stay within the authenticator boundary. Consequently, the security characteristics of this Metadata Statement apply to *all* keys.

When set to "explicit" it means that the authenticator explicitly marks keys as either multi-device keys or single-device keys via the "Backup Eligibility" flag [WebAuthn].

When set to "implicit" it means that all private keys relating to Public Key Credential Source [WebAuthn] may be backed up.

The field `authenticatorGetInfo` / `supportedExtensions` will include the "supplementalPubKeys" extension if the authenticator supports it in general.

If this `multiDeviceCredentialSupport` field is missing the implicit value is "unsupported" (to provide backwards compatibility).

`authenticatorGetInfo`, of type AuthenticatorGetInfo

Describes supported versions, extensions, AAGUID of the device and its capabilities.

The information is the same reported by an authenticator when invoking the 'authenticatorGetInfo' method, see [FIDOCTAP].

> Note: This field MUST be present for FIDO2 authenticators that *natively* support FIDO CTAP [FIDOCTAP], i.e., when `protoclFamily` is "fido2".
>
> Authenticators only implementing platform APIs shall not not provide the field `authenticatorGetInfo`.
>
> FIDO UAF and FIDO U2F authenticators do not support`authenticatorGetInfo`.

> The FIDO Alliance does not verify the certification status information in field `certifications` of the authenticatorGetInfo response [FIDOCTAP].
>
> Authoritative information on the authenticator certification status is included in the `StatusReport` field of the respective Metadata entry [FIDOMetadataService].

`cxConfigURL`, of type DOMString

Specifies the URL for retrieving the configuration details for credential exchange (CX).

When exchanging credentials, the passkey provider to export the credential might retrieve the credential exchange configuration details for of the the importing provider in order to provide additional security. More details can be found in Credential Exchange Protocol specification that can be found on the FIDO Credential Exchange Specifications web page.

4.1. SVG requirements§

This section is normative.

All [SVG11] provider icons MUST adhere to the SVG Portable/Secure (SVG-P/S) profile defined in https://datatracker.ietf.org/doc/draft-svg-tiny-ps-abrotman/09/.

Additional requirements:

1. Format: SVG Version: 1.2 with baseProfile as “tiny-ps"
2. Elements: vector-based (cannot contain raster components)
3. Dimensions: square aspect ratio
4. The `<title>` element MUST be populated with the English version of the provider friendly name
5. The SVG MUST not contain comments or extra text

5. Metadata Statement Format§

This section is not 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).
- 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 `ATTESTATION_BASIC_FULL` method (0x3E07=15879 decimal).
- It implements UAF protocol version (upv) 1.0 and 1.1.

EXAMPLE 1 ¶


```
{
  "legalHeader": "https://fidoalliance.org/metadata/metadata-statement-legal-header/",
  "friendlyNames": {"en-US": "FIDO UAF Sample"},
  "description": "FIDO Alliance Sample UAF Authenticator",
  "aaid": "1234#5678",
  "alternativeDescriptions": {
      "ru-RU": "Пример UAF аутентификатора от FIDO Alliance",
      "fr-FR": "Exemple UAF authenticator de FIDO Alliance"
  },
  "authenticatorVersion": 2,
  "protocolFamily": "uaf",
  "schema": 3,
  "upv": [
    { "major": 1, "minor": 0 },
    { "major": 1, "minor": 1 }
  ],
  "authenticationAlgorithms": ["secp256r1_ecdsa_sha256_raw"],
  "publicKeyAlgAndEncodings": ["ecc_x962_raw"],
  "attestationTypes": ["basic_full"],
  "userVerificationDetails": [
    [{
      "userVerificationMethod": "fingerprint_internal",
      "baDesc": {
        "selfAttestedFAR": 0.00002,
        "maxRetries": 5,
        "blockSlowdown": 30,
        "maxTemplates": 5
      }
```

```
    }]
  ],
  "keyProtection": ["hardware", "tee"],
  "isKeyRestricted": true,
  "matcherProtection": ["tee"],
  "cryptoStrength": 128,
  "attachmentHint": ["internal"],
  "tcDisplay": ["any", "tee"],
  "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
    884zu9NdlnGTfZJP2n3n0++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/STtf1HDpM3b592y7h3Thx5ozK69HLpYWuAwaqS5cv26q7ceb8efWPUBUfyP3iFU8zj1knSw
    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/cKtVm4EBxa9VOGDzYr3LrUMRjHEKkk7zaEKYOA2hGOU1
```

```
    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 ¶

```
[
  [
    {
      "userVerificationMethod": "fingerprint_internal",
      "baDesc": {
        "selfAttestedFAR": 0.000002,
        "maxTemplates": 5,
        "maxRetries": 5,
        "blockSlowdown": 0
      }
    }
  ],
  [
    {
      "userVerificationMethod": "passcode_internal",
      "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. From the perspective of the authenticator, the presence check is optional for U2F_AUTHENTICATE.
- The Authenticator is a pure second factor authenticator.
- It uses the `ALG_SIGN_SECP256R1_ECDSA_SHA256_RAW` authentication algorithm.
- It uses the `ALG_KEY_ECC_X962_RAW` public key format.

- It only implements the `ATTESTATION_BASIC_FULL` method.
- It implements U2F protocol versions 1.2, 1.1 and 1.0

EXAMPLE 3


```
{
  "legalHeader": "https://fidoalliance.org/metadata/metadata-statement-legal-header/",
  "friendlyNames": {"en-US": "FIDO U2F Sample"},
  "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": "□□FIDO Alliance□□□U2F□□□□□"
  },
  "attestationCertificateKeyIdentifiers": ["7c0903708b87115b0b422def3138c3c864e44573"],
  "protocolFamily": "u2f",
  "schema": 3,
  "authenticatorVersion": 2,
  "upv": [
    { "major": 1, "minor": 0 },
    { "major": 1, "minor": 1 },
    { "major": 1, "minor": 2 }
  ],
  "authenticationAlgorithms": ["secp256r1_ecdsa_sha256_raw"],
  "publicKeyAlgAndEncodings": ["ecc_x962_raw"],
  "attestationTypes": ["basic_full"],
  "userVerificationDetails": [
    [
      {"userVerificationMethod": "none"}
    ],
    [
      {"userVerificationMethod": "presence_internal"}
    ]
  ],
  "keyProtection": ["hardware", "secure_element"],
  "matcherProtection": ["on_chip"],
  "cryptoStrength": 128,
  "attachmentHint": ["external", "wired", "nfc"],
  "tcDisplay": [],
  "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
    dtuWBIuffr6oWpV0FPNLhow1751Nm21LvPH3rVtWjfz66Lfql8tX7FRl9YFSXsmSseb9ceOGbbk6
    MNUcGPg8ZsbMe9rfQUaaV/JMX9sqdzBG64i
```

```
    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=="
}
```


5.3. FIDO2 Example§

Example of the metadata statement for an FIDO2 authenticator with:

- AAGUID is set to 0132d110-bf4e-4208-a403-ab4f5f12efe5.
- authenticatorVersion is set to 2.
- Touch based user presence check, and external pin(ClientPin Protocol) support.
- Authenticator is a USB pluggable hardware token with support for NFC.
- The authentication keys are protected by a secure element.
- The user presence check is implemented in the chip. From the perspective of the authenticator, the presence check is optional for getAssertion.
- It uses the `ALG_SIGN_SECP256R1_ECDSA_SHA256_RAW` and `ALG_SIGN_RSASSA_PKCSV15_SHA256_RAW` authentication algorithms.
- It uses the `ALG_KEY_COSE` public key format.
- It only implements the `ATTESTATION_BASIC_FULL` method.
- It implements FIDO2 protocol version 1.0.

EXAMPLE 4 ¶


```
{
  "legalHeader": "https://fidoalliance.org/metadata/metadata-statement-legal-header/",
  "friendlyNames": {"en-US": "FIDO Sample Security Key"},
  "description": "FIDO Alliance Sample FIDO2 Authenticator",
  "aaguid": "0132d110-bf4e-4208-a403-ab4f5f12efe5",
  "alternativeDescriptions": {
      "ru-RU": "Пример FIDO2 аутентификатора от FIDO Alliance",
      "fr-FR": "Exemple FIDO2 authenticator de FIDO Alliance",
      "zh-CN": "□□FIDO Alliance□□□FIDO2□□□□□"
  },
  "protocolFamily": "fido2",
  "schema": 3,
```

```
  "authenticatorVersion": 5,
  "upv": [
    { "major": 1, "minor": 0 }
  ],
  "authenticationAlgorithms": ["secp256r1_ecdsa_sha256_raw", "rsassa_pkcsv15_sha256_raw"],
  "publicKeyAlgAndEncodings": ["cose"],
  "attestationTypes": ["basic_full"],
  "userVerificationDetails": [
    [
      {"userVerificationMethod": "none"}
    ],
    [
     {"userVerificationMethod": "presence_internal"}
    ],
    [{
      "userVerificationMethod": "passcode_external",
      "caDesc": {
        "base": 10,
        "minLength": 4
      }
    }],
    [{
      "userVerificationMethod": "passcode_external",
      "caDesc": {
        "base": 10,
        "minLength": 4
     }},
     {"userVerificationMethod": "presence_internal"}
    ]
  ],
  "keyProtection": ["hardware", "secure_element"],
  "matcherProtection": ["on_chip"],
  "cryptoStrength": 128,
  "attachmentHint": ["external", "wired", "wireless", "nfc"],
  "tcDisplay": [],
  "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,
    iVBORw0KGgoAAAANSUhEUgAAAE8AAAAvCAYAAACiwJfcAAAAAXNSR0IArs4c6QAAARnQU1BAACx
    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/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==",
  "authenticatorGetInfo": {
      "versions": [ "U2F_V2", "FIDO_2_0" ],
      "extensions": [ "credProtect", "hmac-secret" ],
      "aaguid": "0132d110bf4e4208a403ab4f5f12efe5",
      "options": {
        "plat": false,
        "rk": true,
        "clientPin": true,
        "up": true,
        "uv": true,
        "uvToken": false,
        "config": false
      },
      "maxMsgSize": 1200,
      "pinUvAuthProtocols": [1],
      "maxCredentialCountInList": 16,
      "maxCredentialIdLength": 128,
      "transports": ["usb", "nfc"],
      "algorithms": [{
          "type": "public-key",
          "alg": -7
        },
        {
          "type": "public-key",
          "alg": -257
        }
      ],
      "maxAuthenticatorConfigLength": 1024,
      "defaultCredProtect": 2,
      "firmwareVersion": 5
  }
}
```


6. Additional Considerations§

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 authenticator model identifier (AAID/AAGUID/attestationCertificateKeyIdentifiers).

> 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.

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 authenticator model identifier (AAID/AAGUID/attestationCertificateKeyIdentifiers) to this authenticator.

A single authenticator implementation could report itself as two "virtual" authenticators using different authenticator model identifiers (AAIDs/AAGUIDs/attestationCertificateKeyIdentifiers). 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 authenticator model (e.g. as identified by AAID/AAGUID/attestationCertificateKeyIdentifiers) cannot be used by authenticators reporting a different authenticator model identifier (AAID/AAGUID/attestationCertificateKeyIdentifiers) - even when running on the same hardware (see section "Authentication Response Processing Rules for FIDO Server" in [UAFProtocol]).

> Note: To remain compatible with future versions the FIDO Server SHOULD ignore unrecognized fields when processing any element of an entry. The addition, subtraction or change in interpretation of any fields in an entry of this specification which substantively changes the processing logic of a consumer will only occur alongside an update to the major version number of the specification.

Index§

Terms defined by this specification§

overall claimed cryptographic strength

paDesc

PatternAccuracyDescriptor

plte

protocolFamily

providerLogoDark

providerLogoLight

publicKeyAlgAndEncodings

r

rgbPaletteEntry

schema

selfAttestedFAR

selfAttestedFRR

supportedExtensions

sx

sy

tag

tcDisplay

tcDisplayContentType

tcDisplayPNGCharacteristics

upv

userVerificationDetails

userVerificationMethod

VerificationMethodANDCombinations

VerificationMethodDescriptor

width

X

Y

Terms defined by reference§

[WebIDL] defines the following terms:

- DOMString
- boolean
- double
- octet
- unsigned long
- unsigned short

References§

Normative References§

[ECMA-262]

ECMAScript Language Specification. URL: https://tc39.es/ecma262/multipage/

[FIDOAuthenticatorSecurityRequirements]
Rolf Lindemann; Dr. Joshua E. Hill; Douglas Biggs. *FIDO Authenticator Security Requirements*. November 2020. Final Draft. URL: https://fidoalliance.org/specs/fido-security-requirements/fido-authenticator-security-requirements-v1.4-fd-20201102.html

[FIDOBiometricsRequirements]
Stephanie Schuckers; et al. *FIDO Biometrics Requirements*. October 2020. URL: https://fidoalliance.org/specs/biometric/requirements/Biometrics-Requirements-v2.0-fd-20201006.html

[FIDOCTAP]
C. Brand; et al. *FIDO 2.0: Client To Authenticator Protocol*. 30 January 2019. URL: https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html

[FIDOMetadataService]
B. Jack; R. Lindemann; Y. Ackermann. *FIDO Metadata Service*. 21 May 2025. Proposed Standard. URL: https://fidoalliance.org/specs/mds/fido-metadata-service-v3.1-ps-20250521.html

[FIDORegistry]
R. Lindemann; et al. *FIDO Registry of Predefined Values*. 23 May 2022. Proposed Standard. URL: https://fidoalliance.org/specs/common-specs/fido-registry-v2.2-ps-20220523.html

[ISOIEC-19795-1]
ISO/IEC 19795-1:2021 Information technology — Biometric performance testing and reporting — Part 1: Principles and framework. 2021. URL: https://www.iso.org/standard/73515.html

[ISOIEC-30107-3]
ISO/IEC 30107-3:2017 Information technology — Biometric presentation attack detection — Part 3: Testing and reporting. 2017. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:30107:-3:ed-1:v1:en

[PNG]
Chris Lilley; et al. *Portable Network Graphics (PNG) Specification (Third Edition)*. URL: https://w3c.github.io/png/

[RFC2049]
N. Freed; N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples (RFC 2049)*. November 1996. November 1996. URL: https://tools.ietf.org/html/rfc2049

[RFC2397]
L. Masinter. *The "data" URL scheme*. August 1998. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc2397

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

[SVG11]
Erik Dahlström; et al. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. 16 August 2011. REC. URL: https://www.w3.org/TR/SVG11/

[UAFProtocol]
R. Lindemann; et al. *FIDO UAF Protocol Specification v1.2*. Proposed Standard. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-ps-20201020/fido-uaf-protocol-v1.2-ps-20201020.html

[WebAuthn]
Dirk Balfanz (Google); et al. *Web Authentication: An API for accessing Public Key Credentials Level 2*. 8 April 2021. TR. URL: https://www.w3.org/TR/webauthn-2/

[WebIDL]
Edgar Chen; Timothy Gu. *Web IDL Standard*. Living Standard. URL: https://webidl.spec.whatwg.org/

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

Informative References§

[AndroidCompatibilityDefinition]
Android 14 Compatibility Definition. URL: https://source.android.com/docs/compatibility/14/android-14-cdd

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

[ApplePlatformSecurity]
Apple Platform Security. URL: https://support.apple.com/guide/security/welcome/web

[FIDOEcdaaAlgorithm]
R. Lindemann; et al. *FIDO ECDAA Algorithm*. 23 May 2022. Proposed Standard. URL: https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-ecdaa-algorithm-v2.0-id-20180227.html

[FIDOGlossary]
R. Lindemann; et al. *FIDO Technical Glossary*. 23 May 2022. Proposed Standard. URL: https://fidoalliance.org/specs/common-specs/fido-glossary-v2.1-ps-20220523.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

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

[ISO3166]
Codes for the representation of names of countries and their subdivisions — Part 1: Country code. August 2020. Published. URL: https://www.iso.org/standard/72482.html

[ITU-X690-2008]
X.690: Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), (T-REC-X.690-200811). November 2008. URL: https://www.itu.int/rec/T-REC-X.690-200811-S

[MoreTopWorstPasswords]
Mark Burnett. *10000 Top Passwords*. URL: https://xato.net/passwords/more-top-worst-passwords/

[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; et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. May 2008. URL: https://tools.ietf.org/html/rfc5280

[RFC5646]
A. Phillips, Ed.; M. Davis, Ed.. *Tags for Identifying Languages*. September 2009. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc5646

[UAFAuthnrCommands]
D. Baghdasaryan; et al. *FIDO UAF Authenticator Commands*. Proposed Standard. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-ps-20201020/fido-uaf-authnr-cmds-v1.2-ps-20201020.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-ps-20201020/fido-uaf-reg-v1.2-ps-20201020.html

IDL Index§

```
typedef DOMString AAGUID;

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

```
};

dictionary BiometricAccuracyDescriptor {
    double         selfAttestedFRR;
    double         selfAttestedFAR;
    double         iAPARThreshold;
    unsigned short maxTemplates;
    unsigned short maxRetries;
    unsigned short blockSlowdown;
};

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

dictionary VerificationMethodDescriptor {
    DOMString                   userVerificationMethod;
    CodeAccuracyDescriptor      caDesc;
    BiometricAccuracyDescriptor baDesc;
    PatternAccuracyDescriptor   paDesc;
};

typedef VerificationMethodDescriptor[] VerificationMethodANDCombinations;

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

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;
};

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

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

dictionary MetadataStatement {
    DOMString                                    legalHeader;
    AAID                                         aaid;
    AAGUID                                       aaguid;
```

```
    DOMString[]                                    attestationCertificateKeyIdentifiers;
    FriendlyNames                                  friendlyNames;
    required DOMString                             description;
    AlternativeDescriptions                        alternativeDescriptions;
    required unsigned long                         authenticatorVersion;
    required DOMString                             protocolFamily;
    required unsigned short                        schema;
    required Version[]                             upv;
    required DOMString[]                           authenticationAlgorithms;
    required DOMString[]                           publicKeyAlgAndEncodings;
    required DOMString[]                           attestationTypes;
    VerificationMethodANDCombinations[]            userVerificationDetails;
    required DOMString[]                           keyProtection;
    boolean                                        isKeyRestricted;
    boolean                                        isFreshUserVerificationRequired;
    required DOMString[]                           matcherProtection;
    unsigned short                                 cryptoStrength;
    DOMString[]                                    attachmentHint;
    required DOMString[]                           tcDisplay;
    DOMString                                      tcDisplayContentType;
    DisplayPNGCharacteristicsDescriptor[]          tcDisplayPNGCharacteristics;
    required DOMString[]                           attestationRootCertificates;
    EcdaaTrustAnchor[]                             ecdaaTrustAnchors;
    DOMString                                      icon;
    DOMString                                      iconDark;
    DOMString                                      providerLogoLight;
    DOMString                                      providerLogoDark;
    ExtensionDescriptor[]                          supportedExtensions;
    DOMString                                      multiDeviceCredentialSupport;
    AuthenticatorGetInfo                           authenticatorGetInfo;
    DOMString                                      cxConfigURL;
};
```