FIDO UAF Registry of Predefined Values

FIDO Alliance Proposed Standard 02 February 2017

This version:
https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-reg-v1.1-ps-20170202.html
Previous version:
https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-reg-v1.1-id-20170202.html

Editor:
Dr. Rolf Lindemann, Nok Nok Labs, Inc.
Contributors:
Davit Baghdasaryan, Nok Nok Labs, Inc.
Brad Hill, PayPal

The English version of this specification is the only normative version. Non-normative translations may also be available.

Abstract

This document defines all the strings and constants reserved by UAF protocols. The values defined in this document are referenced by various UAF specifications.

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 Proposed Standard. If you wish to make comments regarding this document, please Contact Us. All comments are welcome.

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.

This document has been reviewed by FIDO Aliance Members and is endorsed as a Proposed Standard. It is a stable document and may be used as reference material or cited from another document. FIDO Alliance's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment.

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.

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

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

1.1 Key Words

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

2. Overview

This section is non-normative.

This document defines the registry of UAF-specific constants that are used and referenced in various UAF specifications. It is expected that, over time, new constants will be added to this registry. For example new authentication algorithms and new types of authenticator characteristics will require new constants to be defined for use within the specifications.

FIDO-specific constants that are common to multiple protocol families are defined in [FIDORegistry].

3. Authenticator Characteristics

This section is normative.

3.1 Assertion Schemes

Names of assertion schemes are strings with a length of 8 characters.

UAF TLV based assertion scheme “UAFV1TLV”
: This assertion scheme allows the authenticator and the FIDO Server to exchange an asymmetric authentication key generated by the authenticator. The authenticator must generate a key pair (UAuth.pub/UAuth.priv) to be used with algorithm suites listed in [FIDORegistry] section "Authentication Algorithms" (with prefix `ALG_`). This assertion scheme is using a compact Tag Length Value (TLV) encoding for the KRD and SignData messages generated by the authenticators. This is the default assertion scheme for the UAF protocol.

4. Predefined Tags

This section is normative.

The internal structure of UAF authenticator commands is a “Tag-Length-Value” (TLV) sequence. The tag is a 2-byte unique unsigned value describing the type of field the data represents, the length is a 2-byte unsigned value indicating the size of the value in bytes, and the value is the variable-sized series of bytes which contain data for this item in the sequence.

Although 2 bytes are allotted for the tag, only the first 14 bits (values up to 0x3FFF) should be used to accommodate the limitations of some hardware platforms.

A tag that has the 14th bit (0x2000) set indicates that it is critical and a receiver must abort processing the entire message if it cannot process that tag.

A tag that has the 13th bit (0x1000) set indicates a composite tag that can be parsed by recursive descent.

4.1 Tags used in the protocol

The following tags have been allocated for data types in UAF protocol messages:

TAG_UAFV1_REG_ASSERTION 0x3E01
: The content of this tag is the authenticator response to a Register command.

TAG_UAFV1_AUTH_ASSERTION 0x3E02

The content of this tag is the authenticator response to a Sign command.

TAG_UAFV1_KRD 0x3E03
: Indicates Key Registration Data.

TAG_UAFV1_SIGNED_DATA 0x3E04
: Indicates data signed by the authenticator using UAuth.priv key.

TAG_ATTESTATION_CERT 0x2E05
: Indicates DER encoded attestation certificate.

TAG_SIGNATURE 0x2E06
: Indicates a cryptographic signature.

TAG_ATTESTATION_BASIC_FULL 0x3E07
: Indicates full basic attestation as defined in [UAFProtocol].

TAG_ATTESTATION_BASIC_SURROGATE 0x3E08
: Indicates surrogate basic attestation as defined in [UAFProtocol].

TAG_ATTESTATION_ECDAA 0x3E09
: Indicates use of elliptic curve based direct anonymous attestation as defined in [FIDOEcdaaAlgorithm]. Support for this attestation type is optional at this time. It might be required by FIDO Certification.

TAG_KEYID 0x2E09
: Represents a generated KeyID.

TAG_FINAL_CHALLENGE_HASH 0x2E0A
: Represents a generated final challenge hash as defined in [UAFProtocol].

TAG_AAID 0x2E0B
: Represents an Authenticator Attestation ID as defined in [UAFProtocol].

TAG_PUB_KEY 0x2E0C
: Represents a generated public key.

TAG_COUNTERS 0x2E0D
: Represents the use counters for an authenticator.

TAG_ASSERTION_INFO 0x2E0E
: Represents authenticator information necessary for message processing.

TAG_AUTHENTICATOR_NONCE 0x2E0F
: Represents a nonce value generated by the authenticator.

TAG_TRANSACTION_CONTENT_HASH 0x2E10
: Represents a hash of the transaction content sent to the authenticator.

TAG_EXTENSION 0x3E11, 0x3E12
: This is a composite tag indicating that the content is an extension.

TAG_EXTENSION_ID 0x2E13
: Represents extension ID. Content of this tag is a UINT8[] encoding of a UTF-8 string.

TAG_EXTENSION_DATA 0x2E14
: Represents extension data. Content of this tag is a UINT8[] byte array.

TAG_RAW_USER_VERIFICATION_INDEX 0x0103
: This is the raw UVI as it might be used internally by authenticators. This TAG shall not appear in assertions leaving the authenticator boundary as it could be used as global correlation handle.

TAG_USER_VERIFICATION_INDEX 0x0104
: The user verification index (UVI) is a value uniquely identifying a user verification data record.

Each UVI value must be specific to the related key (in order to provide unlinkability). It also must contain sufficient entropy that makes guessing impractical. UVI values must not be reused by the Authenticator (for other biometric data or users).

The UVI data can be used by FIDO Servers to understand whether an authentication was authorized by the exact same biometric data as the initial key generation. This allows the detection and prevention of "friendly fraud".

As an example, the UVI could be computed as SHA256(KeyID | SHA256(rawUVI)), where the rawUVI reflects (a) the biometric reference data, (b) the related OS level user ID and (c) an identifier which changes whenever a factory reset is performed for the device, e.g. rawUVI = biometricReferenceData | OSLevelUserID | FactoryResetCounter.

FIDO Servers supporting UVI extensions must support a length of up to 32 bytes for the UVI value.

Example of the TLV encoded UVI extension (contained in an assertion, i.e. TAG_UAFV1_REG_ASSERTION or TAG_UAFV1_AUTH_ASSERTION)

```
...
04 01                     -- TAG_USER_VERIFICATION_INDEX (0x0104)
20                        -- length of UVI
00 43 B8 E3 BE 27 95 8C   -- the UVI value itself
28 D5 74 BF 46 8A 85 CF
46 9A 14 F0 E5 16 69 31
DA 4B CF FF C1 BB 11 32
82
...
```

TAG_RAW_USER_VERIFICATION_STATE 0x0105
: This is the raw UVS as it might be used internally by authenticators. This TAG shall not appear in assertions leaving the authenticator boundary as it could be used as global correlation handle.

TAG_USER_VERIFICATION_STATE 0x0106
: The user verification state (UVS) is a value uniquely identifying the set of active user verification data records.

Each UVS value must be specific to the related key (in order to provide unlinkability). It also must contain sufficient entropy that makes guessing impractical. UVS values must not be reused by the Authenticator (for other biometric data sets or users).

The UVS data can be used by FIDO Servers to understand whether an authentication was authorized by one of the biometric data records already known at the initial key generation.

As an example, the UVS could be computed as SHA256(KeyID I SHA256(rawUVS)), where the rawUVS reflects (a) the biometric reference data sets, (b) the related OS level user ID and (c) an identifier which changes whenever a factory reset is performed for the device, e.g. rawUVS = biometricReferenceDataSet I OSLevelUserID I FactoryResetCounter.

FIDO Servers supporting UVS extensions must support a length of up to 32 bytes for the UVS value.

Example of the TLV encoded UVS extension (contained in an assertion, i.e. TAG_UAFV1_REG_ASSERTION or TAG_UAFV1_AUTH_ASSERTION)

```
...
06 01                         -- TAG_USER_VERIFICATION_STATE (0x0106)
20                            -- length of UVS
00 18 C3 47 81 73 2B 65       -- the UVS value itself
83 E7 43 31 46 8A 85 CF
93 6C 36 F0 AF 16 69 14
DA 4B 1D 43 FE C7 43 24
45
...
```

TAG_RESERVED_5 0x0201
Reserved for future use. Name of the tag will change, value is fixed.

5. Predefined (untagged) Extensions

This section is normative.

5.1 Android SafetyNet Extension

This extension can be added

- by FIDO Servers to the UAF Request object (request extension) in the `OperationHeader` in order to trigger generation of the related response extension.
- by FIDO Clients to the ASM Request object (request extension) in order to trigger generation of the related response extension.
- by the ASM to the respective `exts` array in the `ASMResponse` object (response extension).
- by the FIDO Client to the respective `exts` array in either the `OperationHeader`, or the `AuthenticatorRegistrationAssertion`, or the `AuthenticatorSignAssertion` of the UAF Response object (response extension).

Extension identifier
fido.uaf.safetynet
Extension fail-if-unknown flag
`false`, i.e. this (request and response) extension can safely be ignored by all entities.
Extension `data` value

When present in a request (request extension)
empty string, i.e. the FIDO Server might add this extension to the UAF Request with an empty `data` value in order to trigger the generation of this extension for the UAF Response.

EXAMPLE 1: SafetyNet Request Extension

```
"exts": [{"id": "fido.uaf.safetynet", "data": "", "fail_if_unknown": false}]
```

When present in a response (response extension)

- If the request extension was successfully processed, the `data` value is set to the JSON Web Signature attestation result as returned by the call to `com.google.android.gms.safetynet.SafetyNetApi.AttestationResult`.
- If the FIDO Client or the ASM support this extension, but the underlying Android platform does not support it (e.g. Google Play Services is not installed), the `data` value is set to the string "p" (i.e. platform issue).

EXAMPLE 2: SafetyNet Response Extension - not supported by platform

```
"exts": [{"id": "fido.uaf.safetynet", "data": "p", "fail_if_unknown": false}]
```

- If the FIDO Client or the ASM support this extension and the underlying Android platform supports it, but the functionality is temporarily unavailable (e.g. Google servers are unreachable), the `data` value is set to the string "a" (i.e. availability issue).

EXAMPLE 3: SafetyNet Response Extension - temporarily unavailable

```
"exts": [{"id": "fido.uaf.safetynet", "data": "a", "fail_if_unknown": false}]
```

> NOTE
>
> If neither the FIDO Client nor the ASM support this extension, it won't be present in the response object.

FIDO Client processing

FIDO Clients running on Android should support processing of this extension.

If the FIDO Client finds this (request) extension with empty `data` value in the UAF Request and it supports processing this extension, then the FIDO Client

1. must call the Android API `SafetyNet.SafetyNetApi.attest(mGoogleApiClient, nonce)` (see SafetyNet online documentation) and add the response (or an error code as described above) as extension to the response object.
2. must not copy the (request) extension to the ASM Request object (deviating from the general rule in [UAFProtocol], section 3.4.6.2 and 3.5.7.2).

If the FIDO Client does not support this extension it must copy this extension from the UAF Request to the ASM Request object (according to the general rule in [UAFProtocol], section 3.4.6.2 and 3.5.7.2).

If the ASM supports this extension it must call the SafetyNet API (see above) and add the response as extension to the ASM Response object. The FIDO Client must copy the extension in the ASM Response to the UAF Response object (according to sections 3.4.6.4. and 3.5.7.4 step 4 in [UAFProtocol]).

When calling the Android API, the nonce parameter must be set to the serialized JSON object with the following structure:

```
{
  "hashAlg": "S256", // the hash algorithm
  "fcHash": "..."    // the finalChallengeHash
}
```

Where

- `hashAlg` identifies the hash algorithm according to [FIDOSignatureFormat], section IANA Considerations.
- `fcHash` is the base64url encoded hash value of FinalChallenge (see section 3.6.3 and 3.7.4 in [UAFASM] for details on how to compute `finalChallengeHash`).
  We use this method to bind this SafetyNet extension to the respective FIDO UAF message.

  Only hash algorithms belonging to the Authentication Algorithms mentioned in [FIDORegistry] shall be used (e.g. SHA256 because it belongs to `ALG_SIGN_SECP256R1_ECDSA_SHA256_RAW`).

Authenticator argument
N/A

Authenticator processing
N/A. This extension is related to the Android platform in general and not to the authenticator in particular. As a consequence there is no need for an authenticator to receive the (request) extension nor to process it.

Authenticator data
N/A

Server processing
If the FIDO Server requested the SafetyNet extension,

1. it should verify that a proper response is provided (if client side support can be assumed), and
2. it should verify the SafetyNet AttestationResult (see SafetyNet online documentation).

> NOTE
>
> The package name in AttestationResult might relate to either the FIDO Client or the ASM.

> NOTE
>
> The response extension is not part of the signed assertion generated by the authenticator. If an MITM or MITB attacker would remove the response extension, the FIDO server might not be able to distinguish this from the "SafetyNet extension not supported by FIDO Client/ASM" case.

5.2 Android Key Attestation

This extension can be added

- by FIDO Servers to the UAF Request object (request extension) in the `OperationHeader` in order to trigger generation of the related response extension.

- by FIDO Clients to the ASM Request object (request extension) in order to trigger generation of the related response extension.
- by the ASM to the respective `exts` array in the `ASMResponse` object (response extension).
- by the FIDO Client to the respective `exts` array in either the `OperationHeader`, or the `AuthenticatorRegistrationAssertion`, or the `AuthenticatorSignAssertion` of the UAF Response object (response extension).

Extension identifier
fido.uaf.android.key_attestation

Extension fail-if-unknown flag
`false`, i.e. this (request and response) extension can safely be ignored by all entities.

Extension `data` value

When present in a request (request extension)
empty string, i.e. the FIDO Server might add this extension to the UAF Request with an empty `data` value in order to trigger the generation of this extension for the UAF Response.

EXAMPLE 4: Android KeyAttestation Request Extension

```
"exts": [{"id": "fido.uaf.android.key_attestation", "data": "", "fail_if_unknown": false}]
```

When present in a response (response extension)

- If the request extension was successfully processed, the `data` value is set to a JSON array containing the base64 encoded entries of the array returned by the call to the KeyStore API function getCertificateChain.

EXAMPLE 5: Retrieve KeyAttestation and add it as extension

```
KeyPairGenerator kpGenerator = KeyPairGenerator.getInstance(
  KeyProperties.KEY_ALGORITHM_EC, "AndroidKeyStore");
kpGenerator.initialize(
  new KeyGenParameterSpec.Builder(keyUUID, KeyProperties.PURPOSE_SIGN)
    .setDigests(KeyProperties.DIGEST_SHA256)
    .setAlgorithmParameterSpec(new ECGenParameterSpec("prime256v1"))
    .setCertificateSubject(
      new X500Principal(String.format("CN=%s, OU=%s",
        keyUUID, aContext.getPackageName())))
    .setCertificateSerialNumber(BigInteger.ONE)
    .setCertificateNotBefore(notBefore.getTime())
    .setCertificateNotAfter(notAfter.getTime())
    .setUserAuthenticationRequired(true)
    .setAttestationChallenge(fcHash) -- bind to Final Challenge
    .build());

kpGenerator.generateKeyPair(); // generate Uauth key pair

Certificate[] certarray=myKeyStore.getCertificateChain(keyUUID);
String certArray[]=new String[certarray.length];
int i=0;
for (Certificate cert : certarray) {
    byte[] buf = cert.getEncoded();
    certArray[i] = new String(Base64.encode(buf, Base64.DEFAULT));
    i++;
}

JSONArray jarray=new JSONArray(certArray);
String key_attestation_data=jarray.toString();
```

- If the FIDO Client or the ASM support this extension, but the underlying Android platform does not support it (e.g. Android version doesn't yet support it), the `data` value is set to the string "p" (i.e. platform issue).

EXAMPLE 6: KeyAttestation Response Extension - not supported by platform

```
"exts": [{"id": "fido.uaf.android.key_attestation", "data": "p", "fail_if_unknown": false}]
```

- If the FIDO Client or the ASM support this extension and the underlying Android platform supports it, but the functionality is temporarily unavailable (e.g. Google servers are unreachable), the `data` value is set to the string "a".

EXAMPLE 7: KeyAttestation Response Extension - temporarily unavailable

```
"exts": [{"id": "fido.uaf.android.key_attestation", "data": "a", "fail_if_unknown": false}]
```

NOTE

> If neither the FIDO Client nor the ASM support this extension, it won't be present in the response object.

FIDO Client processing

FIDO Clients running on Android must pass this (request) extension with empty `data` value to the ASM.

If the ASM supports this extension it must call the KeyStore API (see above) and add the response as extension to the ASM Response object. The FIDO Client must copy the extension in the ASM Response to the UAF Response object (according to sections 3.4.6.4. and 3.5.7.4 step 4 in [UAFProtocol]).

More details on Android key attestation can be found at:

- https://developer.android.com/preview/api-overview.html#key_attestation
- https://source.android.com/security/keystore/
- https://source.android.com/security/keystore/implementer-ref.html

Authenticator argument
N/A

Authenticator processing
The authenticator generates the attestation response. The call keyStore.getCertificateChain is finally processed by the authenticator.

Authenticator data
N/A

Server processing
If the FIDO Server requested the key attestation extension,

1. it must follow the registration response processing rules (see FIDO UAF Protocol, section 3.4.6.5) before processing this extension
2. it must verify the syntax of the key attestation extension and it must perform RFC5280 compliant chain validation of the entries in the array to one attestationRootCertificate specified in the Metadata Statement.
3. it must determine the leaf certificate from that chain, and it must perform the following checks on this leaf certificate
    1. Verify that KeyDescripion.attestationChallenge == FCHash (see FIDO UAF Protocol, section 3.4.6.5 Step 6.)
    2. Verify that the public key included in the leaf certificate is identical to the public key included in the FIDO UAF Surrogate attestation block
    3. If the related Metadata Statement claims keyProtection KEY_PROTECTION_TEE, then refer to KeyDescription.teeEnforced using "authzList". If the related Metadata Statement claims keyProtection KEY_PROTECTION_SOFTWARE, then refer to KeyDescription.softwareEnforced using "authzList".
    4. Verify that
        1. authzList.origin == KM_TAG_GENERATED
        2. authzList.purpose == KM_PURPOSE_SIGN
        3. authzList.keySize is acceptable, i.e. =2048 (bit) RSA or =256 (bit) ECDSA.
        4. authzList.digest == KM_DIGEST_SHA_2_256.
        5. authzList.authType only contains acceptable user verification methods.
        6. authzList.authTimeout == 0 (or *not* present).
        7. authzList.noAuthRequired is *not* present (unless the Metadata Statement marks this authenticator as silent authenticator, i.e. userVerificaton set to USER_VERIFY_NONE).
        8. authzList.allApplications is *not* present, since FIDO Uauth keys must be bound to the generating app (AppID).

> NOTE
>
> The response extension is not part of the signed assertion generated by the authenticator. If an MITM or MITB attacker would remove the response extension, the FIDO server might not be able to distinguish this from the "KeyAttestation extension not supported by ASM/Authenticator" case.

ExtensionDescriptor `data` value (for Metadata Statement)
In the case of extension id="fido.uaf.android.key_attestation", the data field of the ExtensionDescriptor as included in the Metadata Statement will contain a dictionary containing the following data fields

DOMString attestationRootCertificates[]
Each element of this array represents a PKIX [RFC5280] X.509 certificate that is valid 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.

Each array element is a base64-encoded (section 4 of [RFC4648]), DER-encoded [ITU-X690-2008] PKIX certificate value.

> NOTE
>
> A certificate listed here is either a root certificate or an intermediate CA certificate.

> NOTE
>
> The field `data` is specified with type DOMString in [FIDOMetadataStatement] and hence will contain the serialized object as described above.

An example for the `supportedExtensions` field in the Metadata Statement could look as follows (with line breaks to improve readability):

EXAMPLE 8: Example of a supportedExtensions field in Metadata Statement


```
"supportedExtensions": [{
        "id": "fido.uaf.android.key_attestation",
        "data": "{ \"attestationRootCertificates\": [
\"MIICPTCCAeOgAwIBAgIJAOuexvU3Oy2wMAoGCCqGSM49BAMCMHsxIDAeBgNVBAMM
F1NhbXBsZSBBdHRlc3RhdGlvbiBSb290MRYwFAYDVQQKDA1GSURPIEFsbGlhbmNl
MREwDwYDVQQLDAhVQUYgVFdHLDESMBAGA1UEBwwJUGFsbyBBbHRvMQswCQYDVQQI
DAJDQTELMAkGA1UEBhMCVVMwHhcNMTQwNjE4MTMzMzMyWhcNNDExMTAzMTMzMzMy
WjB7MSAwHgYDVQQDDBdTYW1wbGUgQXR0ZXN0YXRpb24gUm9vdDEWMBQGA1UECgwN
RklETyBBbGxpYW5jZTERMA8GA1UECwwIVUFGIFRXRywxEjAQBgNVBAcMCVBhbG8g
QWx0bzELMAkGA1UECAwCQ0ExCzAJBgNVBAYTAlVTMFkwEwYHKoZIzj0CAQYIKoZI
zj0DAQcDQgAEH8hv2D0HXa59/BmpQ7RZehL/FMGzFd1QBg9vAUpOZ3ajnuQ94PR7
aMzH33nUSBr8fHYDrqOBb58pxGqHJRyX/6NQME4wHQYDVR0OBBYEFPoHA3CLhxFb
C0It7zE4w8hk5EJ/MB8GA1UdIwQYMBaAFPoHA3CLhxFbC0It7zE4w8hk5EJ/MAwG
A1UdEwQFMAMBAf8wCgYIKoZIzj0EAwIDSAAwRQIhAJ06QSXt9ihIbEKYKIjsPkri
VdLIgtfsbDSu7ErJfzr4AiBqoYCZf0+zI55aQeAHjIzA9Xm63rruAxBZ9ps9z2XN
lQ==\"] }",
        "fail_if_unknown": false
                        }]
```


6. Other Identifiers specific to FIDO UAF

6.1 FIDO UAF Application Identifier (AID)

This AID [ISOIEC-7816-5] is used to identify FIDO UAF authenticator applications in a Secure Element.

The FIDO UAF AID consists of the following fields:

Field	RID	AC	AX
Value	0xA000000647	0xAF	0x0001

Table 1: FIDO UAF Applet AID

A. References

A.1 Normative references

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

[FIDOGlossary]
R. Lindemann, D. Baghdasaryan, B. Hill, J. Hodges, *FIDO Technical Glossary*. FIDO Alliance Implementation Draft. URLs:
HTML: fido-glossary-v1.1-id-20170202.pdf

[FIDORegistry]
R. Lindemann, D. Baghdasaryan, B. Hill, *FIDO Registry of Predefined Values*. FIDO Alliance Implementation Draft. URLs:
HTML: fido-registry-v1.1-id-20170202.pdf

[ISOIEC-7816-5]
ISO 7816-5: Identification cards - Integrated circuit cards - Part 5: Registration of application providers

[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

A.2 Informative references

[FIDOMetadataStatement]
B. Hill, D. Baghdasaryan, J. Kemp, *FIDO Metadata Statements v1.0*. FIDO Alliance Implementation Draft. URLs:
HTML: fido-metadata-statements.pdf

[FIDOSignatureFormat]
FIDO 2.0: Signature format. URL: https://fidoalliance.org/specs/fido-v2.0-ps-20150904/fido-signature-format-v2.0-

ps-20150904.html
[ITU-X690-2008]
X.690: Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), (T-REC-X.690-200811). International Telecommunications Union, November 2008 URL: http://www.itu.int/rec/T-REC-X.690-200811-I/en
[RFC4648]
S. Josefsson, *The Base16, Base32, and Base64 Data Encodings (RFC 4648)*, IETF, October 2006, URL: http://www.ietf.org/rfc/rfc4648.txt
[RFC5280]
D. Cooper, S. Santesson, s. Farrell, S.Boeyen, R. Housley, W. Polk;*Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, IETF, May 2008, URL:http://www.ietf.org/rfc/rfc5280.txt
[UAFASM]
D. Baghdasaryan, J. Kemp, R. Lindemann, B. Hill, R. Sasson,*FIDO UAF Authenticator-Specific Module API*. FIDO Alliance Implementation Draft. URLs:
HTML: fido-uaf-asm-api-v1.1-id-20170202.pdf
[UAFProtocol]
R. Lindemann, D. Baghdasaryan, E. Tiffany, D. Balfanz, B. Hill, J. Hodges,*FIDO UAF Protocol Specification v1.0*. FIDO Alliance Proposed Standard. URLs:
HTML: fido-uaf-protocol-v1.1-id-20170202.pdf