FIDO Registry of Predefined Values

FIDO Alliance Review Draft 02 July 2018

This version:
https://fidoalliance.org/specs/fido-v2.0-rd-20180702/fido-registry-v2.0-rd-20180702.html
Previous version:
https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-registry-v2.0-id-20180227.html

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

Abstract

This document defines all the strings and constants reserved by FIDO protocols. The values defined in this document are referenced by various FIDO 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 Review Draft. This document is intended to become a FIDO Alliance Proposed Standard. If you wish to make comments regarding this document, please Contact Us. All comments are welcome.

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 concatenation operations.

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

Some entries are marked as **"(optional)"** in this spec. The meaning of this is defined in other FIDO specifications referring to this document.

1.1 Conformance

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

The key words MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this specification are to be interpreted as described in [RFC2119].

2. Overview

This section is non-normative.

This document defines the registry of FIDO-specific constants common to multiple FIDO protocol families. 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.

3. Authenticator Characteristics

This section is normative.

3.1 User Verification Methods

The `USER_VERIFY` constants are flags in a bitfield represented as a 32 bit long integer. They describe the methods and capabilities of an UAF authenticator for *locally* verifying a user. The operational details of these methods are opaque to the server. These constants are used in the authoritative metadata for an authenticator, reported and queried through the UAF Discovery APIs, and used to form authenticator policies in UAF protocol messages.

All user verification methods must be performed locally by the authenticator in order to meet FIDO privacy principles.

USER_VERIFY_PRESENCE 0x00000001
: This flag MUST be set if the authenticator is able to confirm user presence in any fashion. If this flag and no other is set for user verification, the guarantee is only that the authenticator cannot be operated without some human intervention, not necessarily that the sensing of "presence" provides any level of user verification (e.g. a device that requires a button press to activate).

USER_VERIFY_FINGERPRINT 0x00000002
: This flag MUST be set if the authenticator uses any type of measurement of a fingerprint for user verification.

USER_VERIFY_PASSCODE 0x00000004
: This flag MUST be set if the authenticator uses a local-only passcode (i.e. a passcode not known by the server) for user verification.

USER_VERIFY_VOICEPRINT 0x00000008
: This flag MUST be set if the authenticator uses a voiceprint (also known as speaker recognition) for user verification.

USER_VERIFY_FACEPRINT 0x00000010
: This flag MUST be set if the authenticator uses any manner of face recognition to verify the user.

USER_VERIFY_LOCATION 0x00000020
: This flag MUST be set if the authenticator uses any form of location sensor or measurement for user verification.

USER_VERIFY_EYEPRINT 0x00000040
: This flag MUST be set if the authenticator uses any form of eye biometrics for user verification.

USER_VERIFY_PATTERN 0x00000080
: This flag MUST be set if the authenticator uses a drawn pattern for user verification.

USER_VERIFY_HANDPRINT 0x00000100
: This flag MUST be set if the authenticator uses any measurement of a full hand (including palm-print, hand geometry or vein geometry) for user verification.

USER_VERIFY_NONE 0x00000200
: This flag MUST be set if the authenticator will respond without any user interaction (e.g. Silent Authenticator).

USER_VERIFY_ALL 0x00000400
: If an authenticator sets multiple flags for user verification types, itMAY also set this flag to indicate that all verification methods will be enforced (e.g. faceprint AND voiceprint). If flags for multiple user verification methods are set and this flag is not set, verification with only one is necessary (e.g. fingerprint OR passcode).

3.2 Key Protection Types

The KEY_PROTECTION constants are flags in a bit field represented as a 16 bit long integer. They describe the method an authenticator uses to protect the private key material for FIDO registrations. Refer to [UAFAuthnrCommands] for more details on the relevance of keys and key protection. These constants are used in the authoritative metadata for an authenticator, reported and queried through the UAF Discovery APIs, and used to form authenticator policies in UAF protocol messages.

When used in metadata describing an authenticator, several of these flags are*exclusive* of others (i.e. can not be combined) - the certified metadata may have at most one of the mutually exclusive bits set to 1. When used in authenticator policy, any bit may be set to 1, e.g. to indicate that a server is willing to accept authenticators using either KEY_PROTECTION_SOFTWARE or KEY_PROTECTION_HARDWARE.

> NOTE
>
> These flags must be set according to the*effective* security of the keys, in order to follow the assumptions made in [FIDOSecRef]. For example, if a key is stored in a secure element *but* software running on the FIDO User Device could call a function in the secure element to export the key either in the clear or using an arbitrary wrapping key, then the effective security is KEY_PROTECTION_SOFTWARE and not KEY_PROTECTION_SECURE_ELEMENT.

KEY_PROTECTION_SOFTWARE 0x0001
: This flag MUST be set if the authenticator uses software-based key management. Exclusive in authenticator metadata with KEY_PROTECTION_HARDWARE, KEY_PROTECTION_TEE, KEY_PROTECTION_SECURE_ELEMENT

KEY_PROTECTION_HARDWARE 0x0002
: This flag SHOULD be set if the authenticator uses hardware-based key management. Exclusive in authenticator metadata with KEY_PROTECTION_SOFTWARE

KEY_PROTECTION_TEE 0x0004
: This flag SHOULD be set if the authenticator uses the Trusted Execution Environment [TEE] for key management. In authenticator metadata, this flag should be set in conjunction with KEY_PROTECTION_HARDWARE. Mutually exclusive in authenticator metadata with KEY_PROTECTION_SOFTWARE, KEY_PROTECTION_SECURE_ELEMENT

KEY_PROTECTION_SECURE_ELEMENT 0x0008

This flag SHOULD be set if the authenticator uses a Secure Element [SecureElement] for key management. In authenticator metadata, this flag should be set in conjunction with KEY_PROTECTION_HARDWARE. Mutually exclusive in authenticator metadata with KEY_PROTECTION_TEE, KEY_PROTECTION_SOFTWARE

KEY_PROTECTION_REMOTE_HANDLE 0x0010

This flag MUST be set if the authenticator does not store (wrapped) UAuth keys at the client, but relies on a server-provided key handle. This flag MUST be set in conjunction with one of the otherKEY_PROTECTION flags to indicate how the local key handle wrapping key and operations are protected. Servers MAY unset this flag in authenticator policy if they are not prepared to store and return key handles, for example, if they have a requirement to respond indistinguishably to authentication attempts against userIDs that do and do not exist. Refer to [UAFProtocol] for more details.

3.3 Matcher Protection Types

The MATCHER_PROTECTION constants are flags in a bit field represented as a 16 bit long integer. They describe the method an authenticator uses to protect the matcher that performs user verification. These constants are used in the authoritative metadata for an authenticator, reported and queried through the UAF Discovery APIs, and used to form authenticator policies in UAF protocol messages. Refer to [UAFAuthnrCommands] for more details on the matcher component.

> NOTE
>
> These flags must be set according to the*effective* security of the matcher, in order to follow the assumptions made in [FIDOSecRef]. For example, if a passcode based matcher is implemented in a secure element, but the passcode is expected to be provided as unauthenticated parameter, then the effective security is MATCHER_PROTECTION_SOFTWARE and not MATCHER_PROTECTION_ON_CHIP.

MATCHER_PROTECTION_SOFTWARE 0x0001

This flag MUST be set if the authenticator's matcher is running in software. Exclusive in authenticator metadata with MATCHER_PROTECTION_TEE, MATCHER_PROTECTION_ON_CHIP

MATCHER_PROTECTION_TEE 0x0002

This flag SHOULD be set if the authenticator's matcher is running inside the Trusted Execution Environment [TEE]. Mutually exclusive in authenticator metadata with MATCHER_PROTECTION_SOFTWARE, MATCHER_PROTECTION_ON_CHIP

MATCHER_PROTECTION_ON_CHIP 0x0004

This flag SHOULD be set if the authenticator's matcher is running on the chip. Mutually exclusive in authenticator metadata with MATCHER_PROTECTION_TEE, MATCHER_PROTECTION_SOFTWARE

3.4 Authenticator Attachment Hints

The ATTACHMENT_HINT constants are flags in a bit field represented as a 32 bit long. They describe the method an authenticator uses to communicate with the FIDO User Device. These constants are reported and queried through the UAF Discovery APIs [UAFAppAPIAndTransport], and used to form Authenticator policies in UAF protocol messages. Because the connection state and topology of an authenticator may be transient, these values are only hints that can be used by server-supplied policy to guide the user experience, e.g. to prefer a device that is connected and ready for authenticating or confirming a low-value transaction, rather than one that is more secure but requires more user effort.

> NOTE
>
> These flags are not a mandatory part of authenticator metadata and, when present, only indicate possible states that may be reported during authenticator discovery.

ATTACHMENT_HINT_INTERNAL 0x0001

This flag MAY be set to indicate that the authenticator is permanently attached to the FIDO User Device.

A device such as a smartphone may have authenticator functionality that is able to be used both locally and remotely. In such a case, the FIDO client MUST filter and exclusively report only the relevant bit during Discovery and when performing policy matching.

This flag cannot be combined with any otherATTACHMENT_HINT flags.

ATTACHMENT_HINT_EXTERNAL 0x0002

This flag MAY be set to indicate, for a hardware-based authenticator, that it is removable or remote from the FIDO User Device.

A device such as a smartphone may have authenticator functionality that is able to be used both locally and remotely. In such a case, the FIDO UAF Client MUST filter and exclusively report only the relevant bit during discovery and when performing policy matching.
This flag MUST be combined with one or more other ATTACHMENT_HINT flag(s).

ATTACHMENT_HINT_WIRED 0x0004

This flag MAY be set to indicate that an external authenticator currently has an exclusive wired connection, e.g. through USB, Firewire or similar, to the FIDO User Device.

ATTACHMENT_HINT_WIRELESS 0x0008

This flag MAY be set to indicate that an external authenticator communicates with the FIDO User Device through a personal area or otherwise non-routed wireless protocol, such as Bluetooth or NFC.

ATTACHMENT_HINT_NFC 0x0010

This flag MAY be set to indicate that an external authenticator is able to communicate by NFC to the FIDO User Device. As part of authenticator metadata, or when reporting characteristics through discovery, if this flag is set, the ATTACHMENT_HINT_WIRELESS flag SHOULD also be set as well.

ATTACHMENT_HINT_BLUETOOTH 0x0020

This flag MAY be set to indicate that an external authenticator is able to communicate using Bluetooth with the FIDO User Device. As part of authenticator metadata, or when reporting characteristics through discovery, if this flag is set, the ATTACHMENT_HINT_WIRELESS flag SHOULD also be set.

ATTACHMENT_HINT_NETWORK 0x0040

This flag MAY be set to indicate that the authenticator is connected to the FIDO User Device over a non-exclusive network (e.g. over a TCP/IP LAN or WAN, as opposed to a PAN or point-to-point connection).

ATTACHMENT_HINT_READY 0x0080

This flag MAY be set to indicate that an external authenticator is in a "ready" state. This flag is set by the ASM at its discretion.

> NOTE
>
> Generally this should indicate that the device is immediately available to perform user verification without additional actions such as connecting the device or creating a new biometric profile enrollment, but the exact meaning may vary for different types of devices. For example, a USB authenticator may only report itself as ready when it is plugged in, or a Bluetooth authenticator when it is paired and connected, but an NFC-based authenticator may always report itself as ready.

ATTACHMENT_HINT_WIFI_DIRECT 0x0100

This flag MAY be set to indicate that an external authenticator is able to communicate using WiFi Direct with the FIDO User Device. As part of authenticator metadata and when reporting characteristics through discovery, if this flag is set, the ATTACHMENT_HINT_WIRELESS flag SHOULD also be set.

3.5 Transaction Confirmation Display Types

The TRANSACTION_CONFIRMATION_DISPLAY constants are flags in a bit field represented as a 16 bit long integer. They describe the availability and implementation of a transaction confirmation display capability required for the transaction confirmation operation. These constants are used in the authoritative metadata for an authenticator, reported and queried through the UAF Discovery APIs, and used to form authenticator policies in UAF protocol messages. Refer to [UAFAuthnrCommands] for more details on the security aspects of TransactionConfirmation Display.

TRANSACTION_CONFIRMATION_DISPLAY_ANY 0x0001

This flag MUST be set to indicate that a transaction confirmation display, of any type, is available on this authenticator. Other TRANSACTION_CONFIRMATION_DISPLAY flags MAY also be set if this flag is set. If the authenticator does not support a transaction confirmation display, then the value of TRANSACTION_CONFIRMATION_DISPLAY MUST be set to 0.

TRANSACTION_CONFIRMATION_DISPLAY_PRIVILEGED_SOFTWARE 0x0002

This flag MUST be set to indicate, that a software-based transaction confirmation display operating in a privileged context is available on this authenticator.

A FIDO client that is capable of providing this capability MAY set this bit (in conjunction with TRANSACTION_CONFIRMATION_DISPLAY_ANY) for all authenticators of type ATTACHMENT_HINT_INTERNAL, even if the authoritative metadata for the authenticator does not indicate this capability.

> NOTE
>
> Software based transaction confirmation displays might be implemented within the boundaries of the ASM rather than by the authenticator itself [UAFASM].

This flag is mutually exclusive with TRANSACTION_CONFIRMATION_DISPLAY_TEE and TRANSACTION_CONFIRMATION_DISPLAY_HARDWARE.

TRANSACTION_CONFIRMATION_DISPLAY_TEE 0x0004

This flag SHOULD be set to indicate that the authenticator implements a transaction confirmation display in a Trusted Execution Environment ([TEE], [TEESecureDisplay]). This flag is mutually exclusive with TRANSACTION_CONFIRMATION_DISPLAY_PRIVILEGED_SOFTWARE and TRANSACTION_CONFIRMATION_DISPLAY_HARDWARE.

TRANSACTION_CONFIRMATION_DISPLAY_HARDWARE 0x0008

This flag SHOULD be set to indicate that a transaction confirmation display based on hardware assisted capabilities is available on this authenticator. This flag is mutually exclusive with TRANSACTION_CONFIRMATION_DISPLAY_PRIVILEGED_SOFTWARE and TRANSACTION_CONFIRMATION_DISPLAY_TEE.

TRANSACTION_CONFIRMATION_DISPLAY_REMOTE 0x0010

This flag SHOULD be set to indicate that the transaction confirmation display is provided on a distinct device from the FIDO User Device. This flag can be combined with any other flag.

3.6 Tags used for crypto algorithms and types

These tags indicate the specific authentication algorithms, public key formats and other crypto relevant data.

3.6.1 Authentication Algorithms

The ALG_SIGN constants are 16 bit long integers indicating the specific signature algorithm and encoding.

> NOTE
>
> FIDO UAF supports RAW and DER signature encodings in order to allow small footprint authenticator implementations.

ALG_SIGN_SECP256R1_ECDSA_SHA256_RAW 0x0001

An ECDSA signature on the NIST secp256r1 curve whichMUST have raw R and S buffers, encoded in big-endian order. This is the signature encoding as specified in [ECDSA-ANSI].

I.e. [R (32 bytes), S (32 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_ECC_X962_RAW
- ALG_KEY_ECC_X962_DER
- ALG_KEY_COSE(kty: 2, alg: -7, crv: 1)

ALG_SIGN_SECP256R1_ECDSA_SHA256_DER 0x0002

DER [ITU-X690-2008] encoded ECDSA signature [RFC5480] on the NIST secp256r1 curve.

I.e. a DER encoded SEQUENCE { r INTEGER, s INTEGER }

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_ECC_X962_RAW
- ALG_KEY_ECC_X962_DER
- ALG_KEY_COSE(kty: 2, alg: -7, crv: 1)

ALG_SIGN_RSASSA_PSS_SHA256_RAW 0x0003

RSASSA-PSS [RFC3447] signature MUST have raw S buffers, encoded in big-endian order [RFC4055] [RFC4056]. The default parameters as specified in [RFC4055] MUST be assumed, i.e.

- Mask Generation Algorithm MGF1 with SHA256
- Salt Length of 32 bytes, i.e. the length of a SHA256 hash value.
- Trailer Field value of 1, which represents the trailer field with hexadecimal value0xBC.

I.e. [S (256 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_RSA_2048_RAW
- ALG_KEY_RSA_2048_DER
- ALG_KEY_COSE(kty: 3, alg: -37)

ALG_SIGN_RSASSA_PSS_SHA256_DER 0x0004

DER [ITU-X690-2008] encoded OCTET STRING (not BIT STRING!) containing the RSASSA-PSS [RFC3447] signature [RFC4055] [RFC4056]. The default parameters as specified in [RFC4055] MUST be assumed, i.e.

- Mask Generation Algorithm MGF1 with SHA256
- Salt Length of 32 bytes, i.e. the length of a SHA256 hash value.
- Trailer Field value of 1, which represents the trailer field with hexadecimal value0xBC.

I.e. a DER encoded OCTET STRING (including its tag and length bytes).

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_RSA_2048_RAW
- ALG_KEY_RSA_2048_DER
- ALG_KEY_COSE(kty: 3, alg: -37)

ALG_SIGN_SECP256K1_ECDSA_SHA256_RAW 0x0005

An ECDSA signature on the secp256k1 curve which MUST have raw R and S buffers, encoded in big-endian order.

I.e.[R (32 bytes), S (32 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_ECC_X962_RAW
- ALG_KEY_ECC_X962_DER
- ALG_KEY_COSE(kty: 2, alg: 7, crv: 8)

ALG_SIGN_SECP256K1_ECDSA_SHA256_DER 0x0006

DER [ITU-X690-2008] encoded ECDSA signature [RFC5480] on the secp256k1 curve.

I.e. a DER encoded SEQUENCE { r INTEGER, s INTEGER }

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_ECC_X962_RAW
- ALG_KEY_ECC_X962_DER
- ALG_KEY_COSE(kty: 2, alg: 7, crv: 8)

ALG_SIGN_SM2_SM3_RAW 0x0007 (optional)

Chinese SM2 elliptic curve based signature algorithm combined with SM3 hash algorithm [OSCCA-SM2][OSCCA-SM3]. We use the 256bit curve [OSCCA-SM2-curve-param].

This algorithm is suitable for authenticators using the following key representation format: ALG_KEY_ECC_X962_RAW.

ALG_SIGN_RSA_EMSA_PKCS1_SHA256_RAW 0x0008

This is the EMSA-PKCS1-v1_5 signature as defined in [RFC3447]. This means that the encoded message EM will be the input to the cryptographic signing algorithm RSASP1 as defined in [RFC3447]. The result s of RSASP1 is then encoded using function I2OSP to produce the raw signature octets.

- EM = 0x00 | 0x01 | PS | 0x00 | T
- with the padding string PS with length=emLen - tLen - 3 octets having the value 0xff for each octet, e.g. (0x) ff ff ff ff ff ff ff ff
- with the DER [ITU-X690-2008] encoded DigestInfo value T: (0x)30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20 | H, where H denotes the bytes of the SHA256 hash value.

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_RSA_2048_RAW
- ALG_KEY_RSA_2048_DER

> NOTE
>
> Implementers should verify that their implementation of the PKCS#1 V1.5 signature follows the recommendations in [RFC3218] to protect against adaptive chosen-ciphertext attacks such as Bleichenbacher.

ALG_SIGN_RSA_EMSA_PKCS1_SHA256_DER 0x0009

DER [ITU-X690-2008] encoded OCTET STRING (not BIT STRING!) containing the EMSA-PKCS1-v1_5 signature as defined in [RFC3447]. This means that the encoded message EM will be the input to the cryptographic signing algorithm RSASP1 as defined in [RFC3447]. The result s of RSASP1 is then encoded using function I2OSP to produce the raw signature. The raw signature is DER [ITU-X690-2008] encoded as an OCTET STRING to produce the final signature octets.

- EM = 0x00 | 0x01 | PS | 0x00 | T
- with the padding string PS with length=emLen - tLen - 3 octets having the value 0xff for each octet, e.g. (0x) ff ff ff ff ff ff ff ff
- with the DER encoded DigestInfo value T:(0x)30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20 | H, where H denotes the bytes of the SHA256 hash value.

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_RSA_2048_RAW
- ALG_KEY_RSA_2048_DER

> NOTE
>
> Implementers should verify that their implementation of the PKCS#1 V1.5 signature follows the recommendations in [RFC3218] to protect against adaptive chosen-ciphertext attacks such as Bleichenbacher.

ALG_SIGN_RSASSA_PSS_SHA384_RAW 0x000A

RSASSA-PSS [RFC3447] signature MUST have raw S buffers, encoded in big-endian order [RFC4055] [RFC4056]. The default parameters as specified in [RFC4055] MUST be assumed, i.e.

- Mask Generation Algorithm MGF1 with SHA384
- Salt Length of 48 bytes, i.e. the length of a SHA384 hash value.
- Trailer Field value of 1, which represents the trailer field with hexadecimal value0xBC.

I.e. [S (256 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_COSE(kty: 3, alg: -38)

ALG_SIGN_RSASSA_PSS_SHA512_RAW 0x000B

RSASSA-PSS [RFC3447] signature MUST have raw S buffers, encoded in big-endian order [RFC4055] [RFC4056]. The default parameters as specified in [RFC4055] MUST be assumed, i.e.

- Mask Generation Algorithm MGF1 with SHA512
- Salt Length of 64 bytes, i.e. the length of a SHA512 hash value.
- Trailer Field value of 1, which represents the trailer field with hexadecimal value0xBC.

I.e. [S (256 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_COSE(kty: 3, alg: -39)

ALG_SIGN_RSASSA_PKCSV15_SHA256_RAW 0x000C

RSASSA-PKCS1-v1_5 [RFC3447] with SHA256(aka RS256) signature MUST have raw S buffers, encoded in big-endian order [RFC8017] [RFC4056]

I.e. [S (256 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_COSE(kty: 3, alg: -257)

ALG_SIGN_RSASSA_PKCSV15_SHA384_RAW 0x000D

RSASSA-PKCS1-v1_5 [RFC3447] with SHA384(aka RS384) signature MUST have raw S buffers, encoded in big-endian order [RFC8017] [RFC4056]

I.e. [S (256 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_COSE(kty: 3, alg: -258)

ALG_SIGN_RSASSA_PKCSV15_SHA512_RAW 0x000E

RSASSA-PKCS1-v1_5 [RFC3447] with SHA512(aka RS512) signature MUST have raw S buffers, encoded in big-endian order [RFC8017] [RFC4056]

I.e. [S (256 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_COSE(kty: 3, alg: -259)

ALG_SIGN_RSASSA_PKCSV15_SHA1_RAW 0x000F

RSASSA-PKCS1-v1_5 [RFC3447] with SHA1(aka RS1) signature MUST have raw S buffers, encoded in big-endian order [RFC8017] [RFC4056]

I.e. [S (256 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_COSE(kty: 3, alg: -65535)

ALG_SIGN_SECP384R1_ECDSA_SHA384_RAW 0x0010

An ECDSA signature on the NIST secp384r1 curve with SHA384(aka: ES384) whichMUST have raw R and S buffers, encoded in big-endian order. This is the signature encoding as specified in [ECDSA-ANSI].

I.e. [R (48 bytes), S (48 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_COSE(kty: 2, alg: -35, crv: 2)

ALG_SIGN_SECP521R1_ECDSA_SHA512_RAW 0x0011

An ECDSA signature on the NIST secp512r1 curve with SHA512(aka: ES512) whichMUST have raw R and S buffers, encoded in big-endian order. This is the signature encoding as specified in [ECDSA-ANSI].

I.e. [R (66 bytes), S (66 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_COSE(kty: 2, alg: -36, crv: 3)

ALG_SIGN_ED25519_EDDSA_SHA256_RAW 0x0012

An EdDSA signature on the curve 25519, which MUST have raw R and S buffers, encoded in big-endian order. This is the signature encoding as specified in [RFC8032].

I.e. [R (32 bytes), S (32 bytes)]

This algorithm is suitable for authenticators using the following key representation formats:

- ALG_KEY_COSE(kty: 1, alg: -8, crv: 6)

3.6.2 Public Key Representation Formats

The ALG_KEY constants are 16 bit long integers indicating the specific Public Key algorithm and encoding.

> NOTE
>
> FIDO UAF supports RAW and DER encodings in order to allow small footprint authenticator implementations. By definition, the authenticator must encode the public key as part of the registration assertion.

ALG_KEY_ECC_X962_RAW 0x0100

Raw ANSI X9.62 formatted Elliptic Curve public key [SEC1].

I.e. [0x04, X (32 bytes), Y (32 bytes)] . Where the byte 0x04 denotes the uncompressed point compression method.

ALG_KEY_ECC_X962_DER 0x0101

DER [ITU-X690-2008] encoded ANSI X.9.62 formatted SubjectPublicKeyInfo [RFC5480] specifying an elliptic curve public key.

I.e. a DER encoded SubjectPublicKeyInfo as defined in [RFC5480].

Authenticator implementations MUST generate namedCurve in the ECParameters object which is included in the AlgorithmIdentifier. A FIDO UAF Server MUST accept namedCurve in the ECParameters object which is included in the AlgorithmIdentifier.

ALG_KEY_RSA_2048_RAW 0x0102

Raw encoded 2048-bit RSA public key [RFC3447].

That is, [n (256 bytes), e (N-256 bytes)] . Where N is the total length of the field.

This total length should be taken from the object containing this key, e.g. the TLV encoded field.

ALG_KEY_RSA_2048_DER 0x0103

ASN.1 DER [ITU-X690-2008] encoded 2048-bit RSA [RFC3447] public key [RFC4055].

That is a DER encoded SEQUENCE { n INTEGER, e INTEGER } .

ALG_KEY_COSE 0x0104

COSE_Key format, as defined in Section 7 of [RFC8152]. This encoding includes its own field for indicating the public key algorithm.

3.6.3 Authenticator Attestation Types

The ATTESTATION constants are 16 bit long integers indicating the specific attestation that authenticator supports.

ATTESTATION_BASIC_FULL 0x3E07

Indicates full basic attestation, based on an attestation private key shared among a class of authenticators (e.g. same model). Authenticators must provide its attestation signature during the registration process for the same reason. The attestation trust anchor is shared with FIDO Servers out of band (as part of the Metadata). This sharing process shouldt be done according to [UAFMetadataService].

ATTESTATION_BASIC_SURROGATE 0x3E08

Just syntactically a Basic Attestation. The attestation object self-signed, i.e. it is signed using the UAuth.priv key, i.e. the key corresponding to the UAuth.pub key included in the attestation object. As a consequence it does not provide a cryptographic proof of the security characteristics. But it is the best thing we can do if the authenticator is not able to have an attestation private key.

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.

ATTESTATION_ATTCA 0x3E0A

Indicates PrivacyCA attestation as defined in [TCG-CMCProfile-AIKCertEnroll]. Support for this attestation type is optional at this time. It might be required by FIDO Certification.

3.7 Authenticator Attestation Types

The ATTESTATION constants are 16 bit long integers indicating the specific attestation that authenticator supports.

ATTESTATION_BASIC_FULL 0x3E07

Indicates full basic attestation, based on an attestation private key shared among a class of authenticators (e.g. same model). Authenticators must provide its attestation signature during the registration process for the same reason. The attestation trust anchor is shared with FIDO Servers out of band (as part of the Metadata). This sharing process shouldt be done according to [UAFMetadataService].

ATTESTATION_BASIC_SURROGATE 0x3E08

Just syntactically a Basic Attestation. The attestation object self-signed, i.e. it is signed using the UAuth.priv key, i.e. the key corresponding to the UAuth.pub key included in the attestation object. As a consequence it does not provide a cryptographic proof of the security characteristics. But it is the best thing we can do if the authenticator is not able to have an attestation private key.

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.

ATTESTATION_ATTCA 0x3E0A

Indicates PrivacyCA attestation as defined in [TCG-CMCProfile-AIKCertEnroll]. Support for this attestation type is optional at this time. It might be required by FIDO Certification.

A. References

A.1 Normative references

[FIDOEcdaaAlgorithm]

R. Lindemann; J. Camenisch; M. Drijvers; A. Edgington; A. Lehmann; R. Urian. *FIDO ECDAA Algorithm*. Review Draft. URL: https://fidoalliance.org/specs/fido-v2.0-rd-20180702/fido-ecdaa-algorithm-v2.0-rd-20180702.html

[FIDOGlossary]

R. Lindemann; D. Baghdasaryan; B. Hill; J. Hodges. *FIDO Technical Glossary*. Implementation Draft. URL: https://fidoalliance.org/specs/fido-v2.0-rd-20180702/fido-glossary-v2.0-rd-20180702.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: http://www.itu.int/rec/T-REC-X.690-200811-I/en

[OSCCA-SM2]

SM2: Public Key Cryptographic Algorithm SM2 Based on Elliptic Curves: Part 1: General December 2010. URL: http://www.oscca.gov.cn/UpFile/2010122214822692.pdf

[OSCCA-SM2-curve-param]

SM2: Elliptic Curve Public-Key Cryptography Algorithm: Recommended Curve Parameters December 2010. URL: http://www.oscca.gov.cn/UpFile/2010122214836668.pdf

[OSCCA-SM3]

SM3 Cryptographic Hash Algorithm. December 2010. URL: http://www.oscca.gov.cn/UpFile/20101222141857786.pdf

[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

[RFC3447]

J. Jonsson; B. Kaliski. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. February 2003. Informational. URL: https://tools.ietf.org/html/rfc3447

[RFC4055]

J. Schaad; B. Kaliski; R. Housley. *Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. June 2005. Proposed Standard. URL: https://tools.ietf.org/html/rfc4055

[RFC4056]
J. Schaad. *Use of the RSASSA-PSS Signature Algorithm in Cryptographic Message Syntax (CMS)* June 2005. Proposed Standard. URL: https://tools.ietf.org/html/rfc4056

[RFC5480]
S. Turner; D. Brown; K. Yiu; R. Housley; T. Polk. *Elliptic Curve Cryptography Subject Public Key Information*. March 2009. Proposed Standard. URL: https://tools.ietf.org/html/rfc5480

[RFC8017]
K. Moriarty, Ed.; B. Kaliski; J. Jonsson; A. Rusch. *PKCS #1: RSA Cryptography Specifications Version 2.2* November 2016. Informational. URL: https://tools.ietf.org/html/rfc8017

[RFC8152]
J. Schaad. *CBOR Object Signing and Encryption (COSE)*. July 2017. Proposed Standard. URL: https://tools.ietf.org/html/rfc8152

[SEC1]
SEC1: Elliptic Curve Cryptography, Version 2.0 September 2000. URL: http://secg.org/download/aid-780/sec1-v2.pdf

A.2 Informative references

[ECDSA-ANSI]
Public Key Cryptography for the Financial Services Industry - Key Agreement and Key Transport Using Elliptic Curve Cryptography ANSI X9.63-2011 (R2017). 2017. URL: https://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+X9.63-2011+(R2017)

[FIDOSecRef]
R. Lindemann; D. Baghdasaryan; B. Hill. *FIDO Security Reference*. Implementation Draft. URL: https://fidoalliance.org/specs/fido-v2.0-rd-20180702/fido-security-ref-v2.0-rd-20180702.html

[RFC3218]
E. Rescorla. *Preventing the Million Message Attack on Cryptographic Message Syntax* January 2002. Informational. URL: https://tools.ietf.org/html/rfc3218

[RFC8032]
S. Josefsson; I. Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. January 2017. Informational. URL: https://tools.ietf.org/html/rfc8032

[SecureElement]
GlobalPlatform Card Specifications. URL: https://www.globalplatform.org/specifications.asp

[TEE]
GlobalPlatform Trusted Execution Environment Specifications. URL: https://www.globalplatform.org/specifications.asp

[TEESecureDisplay]
GlobalPlatform Trusted User Interface API Specifications. URL: https://www.globalplatform.org/specifications.asp

[UAFASM]
D. Baghdasaryan; J. Kemp; R. Lindemann; B. Hill; R. Sasson. *FIDO UAF Authenticator-Specific Module API*. Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-uaf-asm-api-v1.2-rd-20171128.html

[UAFAppAPIAndTransport]
B. Hill; D. Baghdasaryan; B. Blanke. *FIDO UAF Application API and Transport Binding Specification* Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-uaf-client-api-transport-v1.2-rd-20171128.html

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

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