FIDO UAF Authenticator-Specific Module API

FIDO Alliance Proposed Standard 08 December 2014

This version: https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-asm-api-v1.0-ps-20141208.html


Editors:
Davit Baghdasaryan, Nok Nok Labs, Inc.
John Kemp, FIDO Alliance

Contributors:
Dr. Rolf Lindemann, Nok Nok Labs, Inc.
Brad Hill, PayPal, Inc.
Roni Sasson, Discretix, Inc.

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

Copyright © 2013-2014 FIDO Alliance All Rights Reserved.

Abstract

UAF authenticators may be connected to a user device via various physical interfaces (SPI, USB, Bluetooth, etc). The UAF Authenticator-Specific Module (ASM) is a software interface on top of UAF authenticators which gives a standardized way for FIDO UAF Clients to detect and access the functionality of UAF authenticators and hides internal communication complexity from FIDO UAF Client.

This document describes the internal functionality of ASMs, defines the UAF ASM API and explains how FIDO UAF Clients should use the API.

This document's intended audience is FIDO authenticator and FIDO FIDO UAF Client
vendors.

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 Alliance 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
   1.1 Key Words

2. Overview
   2.1 Code & Example format

3. ASM Requests and Responses
   3.1 Request enum
   3.2 StatusCode Interface
      3.2.1 Constants
   3.3 ASMRequest Dictionary
      3.3.1 Dictionary ASMRequest Members
   3.4 ASMResponse Dictionary
      3.4.1 Dictionary ASMResponse Members
   3.5 GetInfo Request
      3.5.1 GetInfoOut Dictionary
         3.5.1.1 Dictionary GetInfoOut Members
      3.5.2 AuthenticatorInfo Dictionary
         3.5.2.1 Dictionary AuthenticatorInfo Members
   3.6 Register Request
      3.6.1 RegisterIn Object
         3.6.1.1 Dictionary RegisterIn Members
      3.6.2 RegisterOut Object
3.6.2.1 Dictionary RegisterOut Members

3.6.3 Detailed Description for Processing the Register Request

3.7 Authenticate Request

3.7.1 AuthenticateIn Object

3.7.1.1 Dictionary AuthenticateIn Members

3.7.2 Transaction Object

3.7.2.1 Dictionary Transaction Members

3.7.3 AuthenticateOut Object

3.7.3.1 Dictionary AuthenticateOut Members

3.7.4 Detailed Description for Processing the Authenticate Request

3.8 Deregister Request

3.8.1 DeregisterIn Object

3.8.1.1 Dictionary DeregisterIn Members

3.8.2 Detailed Description for Processing the Deregister Request

3.9 GetRegistrations Request

3.9.1 GetRegistrationsOut Object

3.9.1.1 Dictionary GetRegistrationsOut Members

3.9.2 AppRegistration Object

3.9.2.1 Dictionary AppRegistration Members

3.9.3 Detailed Description for Processing the GetRegistrations Request

3.10 OpenSettings Request

4. Using ASM API

5. Using the ASM API on various platforms

5.1 Android ASM Intent API

5.1.1 Discovering ASMs

5.2 Windows ASM API

6. Security and Privacy Guidelines

6.1 KHAccessToken

6.2 Access Control for ASM APIs

A. References

A.1 Normative references

A.2 Informative references

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 “i” to denote byte wise concatenation operations.

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

The notation base64url refers to "Base 64 Encoding with URL and Filename Safe Alphabet" [RFC4648] without padding.
Following [WebIDL-ED], dictionary members are optional unless they are explicitly marked as required.

WebIDL dictionary members must not have a value of null.

Unless otherwise specified, if a WebIDL dictionary member is DOMString, it must not be empty.

Unless otherwise specified, if a WebIDL dictionary member is a List, it must not be an empty list.

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

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.

UAF authenticators may be connected to a user device via various physical interfaces (SPI, USB, Bluetooth, etc). The UAF Authenticator-Specific module (ASM) is a software interface on top of UAF authenticators which gives a standardized way for FIDO UAF Clients to detect and access the functionality of UAF authenticators, and hides internal communication complexity from clients.

The ASM is a platform-specific software component offering an API to FIDO UAF Clients, enabling them to discover and communicate with one or more available authenticators.

A single ASM may report on behalf of multiple authenticators.

The intended audience for this document is FIDO UAF authenticator and FIDO UAF Client vendors.

NOTE

Platform vendors might choose to not expose the ASM API defined in this document to applications. They might instead choose to expose ASM functionality through some other API (such as, for example, the Android KeyStore API, or iOS KeyChain API). In these cases it's important to make sure that the underlying ASM communicates with the FIDO UAF authenticator in a
The FIDO UAF protocol and its various operations is described in the FIDO UAF Protocol Specification [FIDO UAF Protocol]. The following simplified architecture diagram illustrates the interactions and actors this document is concerned with:

![UAF ASM API Architecture Diagram](image)

2.1 Code & Example format

ASM requests and responses are presented in WebIDL format.

3. ASM Requests and Responses

*This section is normative.*

The ASM API is defined in terms of JSON-formatted [ECMA-404] request and reply messages. In order to send a request to an ASM, a FIDO UAF Client creates an appropriate object (e.g., in ECMAscript), "stringifies" it (also known as serialization) into a JSON-formatted string, and sends it to the ASM. The ASM de-serializes the JSON-formatted string, processes the request, constructs a response, stringifies it, returning it as a JSON-formatted string.
Authenticator implementers may create custom authenticator command interfaces other than the one defined in [UAFAuthnrCommands]. Such implementations are not required to implement the exact message-specific processing steps described in this section. However,

1. the command interfaces must present the ASM with external behavior equivalent to that described below in order for the ASM to properly respond to the client request messages (e.g. returning appropriate UAF status codes for specific conditions).
2. all authenticator implementations must support an assertion scheme as defined [UAFRegistry] and must return the related objects, i.e. TAG_UAFV1_REG_ASSERTION and TAG_UAFV1_AUTH_ASSERTION.

3.1 Request enum

```webidl
enum Request {
    "GetInfo",
    "Register",
    "Authenticate",
    "Deregister",
    "GetRegistrations",
    "OpenSettings"
};
```

### Enumeration description

<table>
<thead>
<tr>
<th>Request</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetInfo</td>
<td>GetInfo</td>
</tr>
<tr>
<td>Register</td>
<td>Register</td>
</tr>
<tr>
<td>Authenticate</td>
<td>Authenticate</td>
</tr>
<tr>
<td>Deregister</td>
<td>Deregister</td>
</tr>
<tr>
<td>GetRegistrations</td>
<td>GetRegistrations</td>
</tr>
<tr>
<td>OpenSettings</td>
<td>OpenSettings</td>
</tr>
</tbody>
</table>

3.2 StatusCode Interface

```webidl
interface StatusCode {
    const short UAF_ASM_STATUS_OK = 0x00;
    const short UAF_ASM_STATUS_ERROR = 0x01;
    const short UAF_ASM_STATUS_ACCESS_DENIED = 0x02;
    const short UAF_ASM_STATUS_USER_CANCELLED = 0x03;
};
```

3.2.1 Constants
**UAF_ASM_STATUS_OK** of type **short**
No error condition encountered.

**UAF_ASM_STATUS_ERROR** of type **short**
An unknown error has been encountered during the processing.

**UAF_ASM_STATUS_ACCESS_DENIED** of type **short**
Access to this request is denied.

**UAF_ASM_STATUS_USER_CANCELLED** of type **short**
Indicates that user explicitly canceled the request.

### 3.3 ASMRequest Dictionary

All ASM requests are represented as **ASMRequest** objects.

#### 3.3.1 Dictionary **ASMRequest** Members

- **requestType** of type **required Request**
  Request type

- **asmVersion** of type **Version**
  ASM message version to be used with this request. For the definition of the **Version** dictionary see [UAFProtocol]. The ASM version must be 1.0 (i.e. major version is 1 and minor version 0).

- **authenticatorIndex** of type **unsigned short**
  Refer to the **GetInfo** request for more details. Field **authenticatorIndex** must not be set for **GetInfo** request.

- **args** of type **object**
  Request-specific arguments. If set, this attribute may take one of the following types:
  - **RegisterIn**
  - **AuthenticateIn**
  - **DeregisterIn**

- **exts** of type array of **Extension**
  List of UAF extensions. For the definition of the **Extension** dictionary see [UAFProtocol].

### 3.4 ASMResponse Dictionary

All ASM responses are represented as **ASMResponse** objects.
3.4.1 Dictionary `ASMResponse` Members

- **statusCode** of type `required short` must contain one of the values defined in the `StatusCode` interface.

- **responseData** of type `object` Request-specific response data. This attribute must have one of the following types:
  - GetInfoOut
  - RegisterOut
  - AuthenticateOut
  - GetRegistrationOut

- **exts** of type array of `Extension` List of UAF extensions. For the definition of the `Extension` dictionary see [UAFProtocol].

3.5 GetInfo Request

Return information about available authenticators.

1. Enumerate all of the authenticators this ASM supports
2. Collect information about all of them
3. Assign indices to them (`authenticatorIndex`)
4. Return the information to the caller

**NOTE**

Where possible, an `authenticatorIndex` should be a persistent identifier that uniquely identifies an `Authenticator` over time, even if it is repeatedly disconnected and reconnected. This avoids possible confusion if the set of available authenticators changes between a `GetInfo` request and subsequent ASM requests, and allows a FIDO client to perform caching of information about removable authenticators for a better user experience.

For a GetInfo request, the following `ASMRequest` member(s) must have the following value(s). The remaining `ASMRequest` members should be omitted:

- `ASMRequest.requestType` must be set to `GetInfo`

For a GetInfo response, the following `ASMResponse` member(s) must have the following value(s). The remaining `ASMResponse` members should be omitted:

- `ASMResponse.statusCode` must have one of the following values
  - UAF_ASM_STATUS_OK
  - UAF_ASM_STATUS_ERROR
ASMResponse.responseData must be an object of type.getInfoOut

3.5.1 GetInfoOut Dictionary

```webidl
dictionary GetInfoOut {
    required AuthenticatorInfo[] Authenticators;
};
```

3.5.1.1 Dictionary GetInfoOut Members

**Authenticators** of type array of required AuthenticatorInfo
List of authenticators reported by the current ASM. may be empty an empty list.

3.5.2 AuthenticatorInfo Dictionary

```webidl
dictionary AuthenticatorInfo {
    required unsigned short authenticatorIndex;
    required Version[] asmVersions;
    required boolean isUserEnrolled;
    required boolean hasSettings;
    required AAID aaid;
    required DOMString assertionScheme;
    required unsigned short authenticationAlgorithm;
    required attestationTypes;
    required unsigned long userVerification;
    required keyProtection;
    required matcherProtection;
    required attachmentHint;
    required boolean isSecondFactorOnly;
    required boolean isRoamingAuthenticator;
    required DOMString[] supportedExtensionIDs;
    required tcDisplay;
    required tcDisplayPNGCharacteristicsDescriptor[] tcDisplayPNGCharacteristics;
    required DOMString title;
    required DOMString description;
    required DOMString icon;
};
```

3.5.2.1 Dictionary AuthenticatorInfo Members

**authenticatorIndex** of type required unsigned short
Authenticator index. Unique, within the scope of all authenticators reported by the ASM, index referring to an authenticator. This index is used by the UAF Client to refer to the appropriate authenticator in further requests.

**asmVersions** of type array of required Version
A list of ASM Versions that this authenticator can be used with. For the definition of the Version dictionary see [UAFProtocol].

**isUserEnrolled** of type required boolean
Indicates whether a user is enrolled with this authenticator. Authenticators which don't have user verification technology must always return true. Bound
authenticators which support different profiles per operating system (OS). The user must report enrollment status for the current OS user.

**hasSettings** of type **required boolean**
A boolean value indicating whether the authenticator has its own settings. If so, then a FIDO UAF Client can launch these settings by sending a `OpenSettings` request.

**aaid** of type **required AAID**
The "Authenticator Attestation ID" (AAID), which identifies the type and batch of the authenticator. See [UAFProtocol] for the definition of the AAID structure.

**assertionScheme** of type **required DOMString**
The assertion scheme the authenticator uses for attested data and signatures.
 AssertionScheme identifiers are defined in the UAF Protocol specification [UAFProtocol].

**authenticationAlgorithm** of type **required unsigned short**
Indicates the authentication algorithm that the authenticator uses. Authentication algorithm identifiers are defined in are defined in [UAFRegistry] with `UAF_ALG` prefix.

**attestationTypes** of type **array of required unsigned short**
Indicates attestation types supported by the authenticator. Attestation type TAGs are defined in [UAFRegistry] with `TAG_ATTESTATION` prefix.

**userVerification** of type **required unsigned long**
A set of bit flags indicating the user verification method(s) supported by the authenticator. The values are defined by the `USER_VERIFY` constants in [UAFRegistry].

**keyProtection** of type **required unsigned short**
A set of bit flags indicating the key protections used by the authenticator. The values are defined by the `KEY_PROTECTION` constants in [UAFRegistry].

**matcherProtection** of type **required unsigned short**
A set of bit flags indicating the matcher protections used by the authenticator. The values are defined by the `MATCHER_PROTECTION` constants in [UAFRegistry].

**attachmentHint** of type **required unsigned long**
A set of bit flags indicating how the authenticator is currently connected to the system hosting the FIDO UAF Client software. The values are defined by the `ATTACHMENT_HINT` constants defined in [UAFRegistry].

**NOTE**
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. These values are not reflected in authenticator metadata and cannot be relied on by the relying party, although some models of authenticator may provide attested measurements with similar semantics as part of UAF protocol messages.
**isSecondFactorOnly** of type `required boolean`
Indicates whether the authenticator can be used only as a second factor.

**isRoamingAuthenticator** of type `required boolean`
Indicates whether this is a roaming authenticator or not.

**supportedExtensionIDs** of type array of `required DOMString`
List of supported UAF extension IDs. may be an empty list.

**tcDisplay** of type `required unsigned short`
A set of bit flags indicating the availability and type of the authenticator’s transaction confirmation display. The values are defined by the `TRANSACTION_CONFIRMATION_DISPLAY` constants in [UAFRegistry].

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

**tcDisplayContentType** of type `DOMString`
Supported transaction content type [UAFAuthnrMetadata].

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

**tcDisplayPNGCharacteristics** of type array of `DisplayPNGCharacteristicsDescriptor`
Supported transaction Portable Network Graphic (PNG) type [UAFAuthnrMetadata]. For the definition of the `DisplayPNGCharacteristicsDescriptor` structure see [UAFAuthnrMetadata].

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

**title** of type `DOMString`
A human-readable short title for the authenticator. It should be localized for the current locale.

NOTE

If the ASM doesn't return a title, the FIDO UAF Client must provide a title to the calling App. See section "Authenticator interface" in [UAFAppAPIAndTransport].

**description** of type `DOMString`
Human-readable longer description of what the authenticator represents.

NOTE

This text should be localized for current locale.

The text is intended to be displayed to the user. It might deviate from the description specified in the metadata statement for the authenticator [UAFAuthnrMetadata].

If the ASM doesn't return a description, the FIDO UAF Client will provide a description to the calling application. See section "Authenticator interface" in [UAFAppAPIAndTransport].
3.6 Register Request

Verify the user and return an authenticator-generated UAF registration assertion.

For a Register request, the following `ASMRequest` member(s) must have the following value(s). The remaining `ASMRequest` members should be omitted:

- `ASMRequest.requestType` must be set to `Register`
- `ASMRequest.asmVersion` must be set to the desired version
- `ASMRequest.authenticatorIndex` must be set to the target authenticator index
- `ASMRequest.args` must be set to an object of type `RegisterIn`

For a Register response, the following `ASMResponse` member(s) must have the following value(s). The remaining `ASMResponse` members should be omitted:

- `ASMResponse.statusCode` must have one of the following values:
  - `UAF_ASM_STATUS_OK`
  - `UAF_ASM_STATUS_ERROR`
  - `UAF_ASM_STATUS_ACCESS_DENIED`
  - `UAF_ASM_STATUS_USER_CANCELLED`
- `ASMResponse.responseData` must be an object of type `RegisterOut`

3.6.1 RegisterIn Object

```webidl
dictionary RegisterIn {
    required DOMString appID;
    required DOMString username;
    required DOMString finalChallenge;
    required unsigned short attestationType;
};
```

3.6.1.1 Dictionary `RegisterIn` Members

- `appID` of type `required DOMString`
  The FIDO server Application Identity.

- `username` of type `required DOMString`
  Human-readable user account name

NOTE

If the ASM doesn't return an icon, the FIDO UAF Client will provide a default icon to the calling application. See section "Authenticator interface" in [UAFAppAPIAndTransport].
**finalChallenge** of type **required DOMString**
base64url-encoded challenge data [RFC4648]

**attestationType** of type **required unsigned short**
Single requested attestation type

### 3.6.2 RegisterOut Object

```webidl
dictionary RegisterOut {
  required DOMString assertion;
  required DOMString assertionScheme;
};
```

#### 3.6.2.1 Dictionary RegisterOut Members

- **assertion** of type **required DOMString**
  FIDO UAF authenticator registration assertion, base64url-encoded

- **assertionScheme** of type **required DOMString**
  Assertion scheme.
  
  AssertionScheme identifiers are defined in the UAF Protocol [UAFProtocol].

### 3.6.3 Detailed Description for Processing the Register Request

Refer to [UAFAuthnrCommands] document for more information about the TAGs and structure mentioned in this paragraph.

1. Locate authenticator using `authenticatorIndex`. If the authenticator cannot be located, then fail with `UAF_ASM_STATUS_ERROR`.
2. If a user is already enrolled with this authenticator (such as biometric enrollment, PIN setup, etc. for example) then the ASM must request that the authenticator verifies the user.

   **NOTE**
   If the authenticator supports `UserVerificationToken` (see [UAFAuthnrCommands]), then the ASM must obtain this token in order to later include it with the `Register` command.

   - If verification fails, return `UAF_ASM_STATUS_ACCESS_DENIED`
3. If the user is not enrolled with the authenticator then take the user through the enrollment process.
   - If enrollment fails, return `UAF_ASM_STATUS_ACCESS_DENIED`
4. Construct `KHAccessToken` (see section [KHAccessTok}}{{\text{en}}} for more details)
5. Hash the provided `RegisterIn.finalChallenge` using the authenticator-specific hash function (`FinalChallengeHash`)

An authenticator's preferred hash function information must meet the algorithm defined in the `AuthenticatorInfo.authenticationAlgorithm` field.
6. Create a `TAG_UAFV1_REGISTER_CMD` structure and pass it to the authenticator.
   1. Copy `FinalChallengeHash`, `KHAccessToken`, `RegisterIn.Username`, `UserVerificationToken`, `RegisterIn.AppID`, `RegisterIn.AttestationType`
   1. Depending on `AuthenticatorType` some arguments may be optional. Refer to [UAFAuthnrCommands] for more information on authenticator types and their required arguments.

7. Invoke the command and receive the response.

8. Parse `TAG_UAFV1_REGISTER_CMD_RESP`
   1. Parse the content of `TAG_AUTHENTICATOR_ASSERTION` (e.g. `TAG_UAFV1_REG_ASSERTION`) and extract `TAG_KEYID`

9. If the authenticator is a bound authenticator

### NOTE

What data an ASM will store at this stage depends on underlying authenticator's architecture. For example some authenticators might store `AppID`, `KeyHandle`, `KeyID` inside their own secure storage. In this case ASM doesn't have to store these data in its database.

10. Create a `RegisterOut` object
    1. Set `RegisterOut.assertionScheme` according to `AuthenticatorInfo.assertionScheme`
    2. Encode the content of `TAG_AUTHENTICATOR_ASSERTION` (e.g. `TAG_UAFV1_REG_ASSERTION`) in base64url format and set as `RegisterOut.assertion`
    3. Return `RegisterOut` object

### 3.7 Authenticate Request

Verify the user and return authenticator-generated UAF authentication assertion.

For an Authenticate request, the following `ASMRequest` member(s) must have the following value(s). The remaining `ASMRequest` members should be omitted:

- `ASMRequest.requestType` must be set to `Authenticate`.
- `ASMRequest.asmVersion` must be set to the desired version.
- `ASMRequest.authenticatorIndex` must be set to the target authenticator index.
- `ASMRequest.args` must be set to an object of type `AuthenticateIn`

For an Authenticate response, the following `ASMResponse` member(s) must have the following value(s). The remaining `ASMResponse` members should be omitted:

- `ASMResponse.statusCode` must have one of the following values:
  - `UAF_ASM_STATUS_OK`
  - `UAF_ASM_STATUS_ERROR`
  - `UAF_ASM_STATUS_ACCESS_DENIED`
  - `UAF_ASM_STATUS_USER_CANCELLED`
3.7.1 Authenticateln Object

**WebIDL**

```webidl
dictionary 
AuthenticateIn { 
  required DOMString appID; 
  DOMString[] keyIDs; 
  required DOMString finalChallenge; 
  Transaction[] transaction; 
};
```

3.7.1.1 Dictionary 
**AuthenticateIn Members**

- **appID** of type required DOMString
  appID string

- **keyIDs** of type array of DOMString
  base64url [RFC4648] encoded keyIDs

- **finalChallenge** of type required DOMString
  base64url [RFC4648] encoded final challenge

- **transaction** of type array of Transaction
  An array of transaction data to be confirmed by user. If multiple transactions are provided, then the ASM must select the one that best matches the current display characteristics.

**NOTE**

This may, for example, depend on whether user’s device is positioned horizontally or vertically at the moment of transaction.

3.7.2 Transaction Object

**WebIDL**

```webidl
dictionary 
Transaction { 
  required DOMString contentType; 
  required DOMString content; 
  DisplayPNGCharacteristicsDescriptor tcDisplayPNGCharacteristics; 
};
```

3.7.2.1 Dictionary 
**Transaction Members**

- **contentType** of type required DOMString
  Contains the MIME Content-Type supported by the authenticator according to its metadata statement (see [UAFAuthnrMetadata])

- **content** of type required DOMString
  Contains the base64url-encoded [RFC4648] transaction content according to the contentType to be shown to the user.
3.7.3 AuthenticateOut Object

```webidl
dictionary AuthenticateOut {
  required DOMString assertion;
  required DOMString assertionScheme;
};
```

3.7.3.1 Dictionary AuthenticateOut Members

- **assertion** of type `required DOMString`
  Authenticator UAF authentication assertion.
- **assertionScheme** of type `required DOMString`
  Assertion scheme

3.7.4 Detailed Description for Processing the Authenticate Request

Refer to the [UAFAuthnrCommands] document for more information about the TAGs and structure mentioned in this paragraph.

1. Locate the authenticator using `authenticatorIndex`
2. If no user is enrolled with this authenticator (such as biometric enrollment, PIN setup, etc.), return `UAF_ASM_STATUS_ACCESS_DENIED`
3. The ASM **must** request the authenticator to verify the user.
   - If verification fails, return `UAF_ASM_STATUS_ACCESS_DENIED`

NOTE

If the authenticator supports `UserVerificationToken` (see [UAFAuthnrCommands]), the ASM must obtain this token in order to later pass to `Sign` command.

4. Construct `KHAccessToken` (see section `KHAccessToken` for more details)
5. Hash the provided `AuthenticateIn.finalChallenge` using an authenticator-specific hash function (`FinalChallengeHash`).
   - The authenticator’s preferred hash function information **must** meet the algorithm defined in the `AuthenticatorInfo.authenticationAlgorithm` field.

6. If this is a Second Factor authenticator and `AuthenticateIn.keyIDs` is empty, then return `UAF_ASM_STATUS_ACCESS_DENIED`
7. If `AuthenticateIn.keyIDs` is not empty,
   1. If this is a bound authenticator, then look up ASM’s database with `AuthenticateIn.appID` and `AuthenticateIn.keyIDs` and obtain the KeyHandles associated with it.
      - Return `UAF_ASM_STATUS_ACCESS_DENIED` if no entry has been found
2. If this is a roaming authenticator, then treat AuthenticateIn.keyIDs as KeyHandles.

8. Create **TAG_UAFV1_SIGN_CMD** structure and pass it to the authenticator.
   1. Copy AuthenticateIn.AppID, AuthenticateIn.Transaction.content (if not empty), FinalChallengeHash, KHAccessToken, UserVerificationToken, KeyHandles
   
   - Depending on AuthenticatorType some arguments may be optional. Refer to [UAFAuthnrCommands] for more information on authenticator types and their required arguments.
   - If multiple transactions are provided, select the one that best matches the current display characteristics.

```
NOTE
This may, for example, depend on whether user's device is positioned horizontally or vertically at the moment of transaction.
```

- Decode the base64url encoded AuthenticateIn.Transaction.content before passing it to the authenticator.

9. Invoke the command and receive the response.

10. Parse **TAG_UAFV1_SIGN_CMD_RESP**
    1. If it's a first-factor authenticator and the response includes **TAG_USERNAME_AND_KEYHANDLE**, then
       1. Extract usernames from **TAG_USERNAME_AND_KEYHANDLE** fields
       2. If two equal usernames are found, then choose the one which has registered most recently
       3. Show remaining distinct usernames and ask the user to choose a single username
       4. Set **TAG_UAFV1_SIGN_CMD.KeyHandles** to the single KeyHandle associated with the selected username.
       5. Go to step #8 and send a new **TAG_UAFV1_SIGN_CMD** command

11. Create the AuthenticateOut object
    1. Set AuthenticateOut.assertionScheme as AuthenticatorInfo.assertionScheme
    2. Encode the content of **TAG_AUTHENTICATOR_ASSERTION** (e.g. **TAG_UAFV1_AUTH_ASSERTION**) in base64url format and set as AuthenticateOut.assertion
    3. Return the AuthenticateOut object

```
NOTE
Some authenticators might support "Transaction Confirmation Display" functionality not inside the authenticator but within the boundaries of the ASM. Typically these are software based Transaction Confirmation Displays. When processing the sign command with a given transaction such ASM should show transaction content in its own UI and after user confirms it -- pass the content to authenticator so that the authenticator includes it in the final assertion.

See [UAFRegistry] for flags describing Transaction Confirmation Display type.
```
The authenticator metadata statement must truly indicate the type of transaction confirmation display implementation. Typically the "Transaction Confirmation Display" flag will be set to TRANSACTION_CONFIRMATION_DISPLAY_ANY or TRANSACTION_CONFIRMATION_DISPLAY_PRIVILEGED_SOFTWARE.

3.8 Deregister Request

Delete registered UAF record from the authenticator.

For a Deregister request, the following ASMRequest member(s) must have the following value(s). The remaining ASMRequest members should be omitted:

- ASMRequest.requestType must be set to Deregister
- ASMRequest.asmVersion must be set to the desired version
- ASMRequest.authenticateIndex must be set to the target authenticator index
- ASMRequest.args must be set to an object of type DeregisterIn

For a Deregister response, the following ASMResponse member(s) must have the following value(s). The remaining ASMResponse members should be omitted:

- ASMResponse.statusCode must have one of the following values:
  - UAF_ASM_STATUS_OK
  - UAF_ASM_STATUS_ERROR
  - UAF_ASM_STATUS_ACCESS_DENIED

3.8.1 DeregisterIn Object

WebIDL

```
dictionary DeregisterIn {
  required DOMString appID;
  required DOMString keyID;
};
```

3.8.1.1 Dictionary DeregisterIn Members

- **appID** of type required DOMString
  
  FIDO Server Application Identity

- **keyID** of type required DOMString
  
  Base64url-encoded [RFC4648] key identifier of the authenticator to be deregistered.

3.8.2 Detailed Description for Processing the Deregister Request

Refer to [UAFAuthnrCommands] for more information about the TAGs and structures mentioned in this paragraph.

1. Locate the authenticator using authenticateIndex
2. Construct KHAccessToken (see section KHAccessToken for more details).
3. If this is a bound authenticator, then
   - Lookup the authenticator related data in the ASM database and delete the
4. Create the `TAG_UAFV1_DEREGISTER_CMD` structure, copy `KHAccessToken`, `DeregisterIn.appID`, and `DeregisterIn.keyID` and pass it to the authenticator.

5. Invoke the command and receive the response

### 3.9 GetRegistrations Request

Return all registrations made for the calling FIDO UAF Client.

For a GetRegistrations request, the following `ASMRequest` member(s) must have the following value(s). The remaining `ASMRequest` members should be omitted:

- `ASMRequest.requestType` must be set to `GetRegistrations`
- `ASMRequest.asmVersion` must be set to the desired version
- `ASMRequest.authenticatorIndex` must be set to corresponding ID

For a GetRegistrations response, the following `ASMResponse` member(s) must have the following value(s). The remaining `ASMResponse` members should be omitted:

- `ASMResponse.statusCode` must have one of the following values:
  - `UAF_ASM_STATUS_OK`
  - `UAF_ASM_STATUS_ERROR`
- The `ASMResponse.responseData` must be an object of type `GetRegistrationsOut`

#### 3.9.1 GetRegistrationsOut Object

```
WebIDL
dictionary GetRegistrationsOut {
  required AppRegistration[] appRegs;
};
```

```
3.9.1.1 Dictionary GetRegistrationsOut Members

**appRegs** of type array of required `AppRegistration`
- List of registrations associated with an `appID` (see `AppRegistration` below). `appRegs` may be an empty list.

#### 3.9.2 AppRegistration Object

```
WebIDL
dictionary AppRegistration {
  required DOMString appID;
  required DOMString[] keyIDs;
};
```

```
3.9.2.1 Dictionary AppRegistration Members

**appID** of type required `DOMString`
- FIDO Server Application Identity.
3.9.3 Detailed Description for Processing the GetRegistrations Request

1. Locate the authenticator using authenticatorIndex
2. If this is bound authenticator, then
   - Lookup the registrations associated with CallerID and AppID in the ASM database and construct a list of AppRegistration objects

   **NOTE**
   Some ASMs might not store this information inside their own database. Instead it might have been stored inside the authenticator's secure storage area. In this case the ASM must send a proprietary command to obtain the necessary data.

3. Create GetRegistrationsOut object and return

3.10 OpenSettings Request

Display the authenticator-specific settings interface. If the authenticator has its own built-in user interface, then the ASM must invoke TAG_UAFV1_OPEN_SETTINGS_CMD to display it.

For an OpenSettings request, the following ASMRequest member(s) must have the following value(s). The remaining ASMRequest members should be omitted:

- ASMRequest.requestType must be set to OpenSettings
- ASMRequest.asmVersion must be set to the desired version
- ASMRequest.authenticatorIndex must be set to the target authenticator index

For an OpenSettings response, the following ASMResponse member(s) must have the following value(s). The remaining ASMResponse members should be omitted:

- ASMResponse.statusCode must have one of the following values:
  - UAF_ASM_STATUS_OK

4. Using ASM API

*This section is non-normative.*

In a typical implementation, the FIDO UAF Client will call GetInfo during initialization and obtain information about the authenticators. Once the information is obtained it will typically be used during FIDO UAF message processing to find a match for given FIDO UAF policy. Once a match is found the FIDO UAF Client will send the appropriate request (Register/Authenticate/Deregister...) to this ASM.

The FIDO UAF Client may use the information obtained from a GetInfo response to display relevant information about an authenticator to the user.

5. Using the ASM API on various platforms
5.1 Android ASM Intent API

On Android systems FIDO UAF ASMs may be implemented as a separate APK-packaged application.

The FIDO UAF Client invokes ASM operations via Android Intents. All interactions between the FIDO UAF Client and an ASM on Android takes place through the following intent identifier:

```
org.fidoalliance.intent.FIDO_OPERATION
```

To carry messages described in this document, an intent must also have its type attribute set to `application/fido.uaf_asm+json`.

ASMs must register that intent in their manifest file and implement a handler for it.

FIDO UAF Clients must append an extra, `message`, containing a String representation of a `ASMRequest`, before invoking the intent.

FIDO UAF Clients must invoke ASMs by calling `startActivityForResult()`.

FIDO UAF Clients should assume that ASMs will display an interface to the user in order to handle this intent, e.g. prompting the user to complete the verification ceremony. However, the ASM should not display any user interface when processing a `GetInfo` request.

After processing is complete the ASM will return the response intent as an argument to `onActivityResult()`. The response intent will have an extra, `message`, containing a String representation of a `ASMResponse`.

5.1.1 Discovering ASMs

FIDO UAF Clients can discover the ASMs available on the system by using `PackageManager.queryIntentActivities(Intent intent, int flags)` with the FIDO Intent described above to see if any activities are available.

A typical FIDO UAF Client will enumerate all ASM applications using this function and will invoke the `GetInfo` operation for each one discovered.

5.2 Windows ASM API

On Windows, an ASM is implemented in the form of a Dynamic Link Library (DLL). The following is an example `asmplugin.h` header file defining a Windows ASM API.
#define ASM_FUNC extern "C" ASM_API
#define ASM_NULL 0

/*! rief Error codes returned by ASM Plugin API.
* Authenticator specific error codes are returned in JSON form.
* See JSON schemas for more details.
*/

enum asmResult_t
{
    Success = 0, /**< Success */
    Failure /**< Generic failure */
};

/*! rief Generic structure containing JSON string in UTF-8
* format.
* This structure is used throughout functions to pass and receives
* JSON data.
*/

struct asmJSONData_t
{
    int length; /**< JSON data length */
    char pData; /**< JSON data */
};

/*! rief Enumeration event types for authenticators.
These events will be fired when an authenticator becomes
available (plugged) or unavailable (unplugged).
*/

enum asmEnumerationType_t
{
    Plugged = 0, /**< Indicates that authenticator Plugged to system */
    Unplugged /**< Indicates that authenticator Unplugged from system */
};

namespace ASM
{
/*! rief Callback listener.
FIDO UAF Client must pass an object implementing this interface to
Authenticator::Process function. This interface is used to provide
ASM JSON based response data.*/
class ICallback
{
public
    virtual ~ICallback() {} /**< This function is called when ASM's response is ready. */
    virtual void Callback(const asmJSONData_t &response,
                          asmJSONData_t &exchangeData) = 0;
};

/*! rief Authenticator Enumerator.
FIDO UAF Client must provide an object implementing this
interface. It will be invoked when a new authenticator is plugged or
when an authenticator has been unplugged. */
class IEnumerator
{
public
    virtual ~IEnumerator() {} /**< This function is called when an authenticator is plugged or unplugged. */
}
A Windows-based FIDO UAF Client must look for ASM DLLs in the following registry paths:

HKCU\Software\FIDO\UAF\ASM

HKLM\Software\FIDO\UAF\ASM

The FIDO UAF Client iterates over all keys under this path and looks for "path" field:

[HK**\Software\FIDO\UAF\ASM\<exampleASMName>]

"path"="<ABSOLUTE_PATH_TO_ASM>.dll"

path must point to the absolute location of the ASM DLL.

6. Security and Privacy Guidelines

This section is normative.

ASM developers must carefully protect the FIDO UAF data they are working with. ASMs must follow these security guidelines:

- ASMs must implement a mechanism for isolating UAF credentials registered by two different FIDO UAF Clients from one another. One FIDO UAF Client must not have access to FIDO UAF credentials that have been registered via a different FIDO UAF Client. This prevents malware from exercising credentials associated with a legitimate FIDO Client.
**NOTE**

ASMs must properly protect their sensitive data against malware using platform-provided isolation capabilities in order to follow the assumptions made in [FIDOSecRef]. Malware with root access to the system or direct physical attack on the device are out of scope for this requirement.

**NOTE**

The following are examples for achieving this:

- If an ASM is bundled with a FIDO UAF Client, this isolation mechanism is already built-in.
- If the ASM and FIDO UAF Client are implemented by the same vendor, the vendor may implement proprietary mechanisms to bind its ASM exclusively to its own FIDO UAF Client.
- On some platforms ASMs and the FIDO UAF Clients may be assigned with a special privilege or permissions which regular applications don't have. ASMs built for such platforms may avoid supporting isolation of UAF credentials per FIDO UAF Clients since all FIDO UAF Clients will be considered equally trusted.

- An ASM designed specifically for bound authenticators must ensure that FIDO UAF credentials registered with one ASM cannot be accessed by another ASM. This is to prevent an application pretending to be an ASM from exercising legitimate UAF credentials.
  - Using a `KHAccessToken` offers such a mechanism.

- An ASMs must implement platform-provided security best practices for protecting UAF related stored data.

- ASMs must not store any sensitive FIDO UAF data in its local storage, except the following:
  - `CallerID, ASMToken, PersonaID, KeyID, KeyHandle, AppID`

**NOTE**

An ASM, for example, must never store a username provided by a FIDO Server in its local storage in a form other than being decryptable exclusively by the authenticator.

- ASMs should ensure that applications cannot use silent authenticators for tracking purposes. ASMs implementing support for a silent authenticator must show, during every registration, a user interface which explains what a silent authenticator is, asking for the users consent for the registration. Also, it is
recommended that ASMs designed to support roaming silent authenticators either

- Run with a special permission/privilege on the system, or
- Have a built-in binding with the authenticator which ensures that other applications cannot directly communicate with the authenticator by bypassing this ASM.

6.1 KHAccessToken

KHAccessToken is an access control mechanism for protecting an authenticator's FIDO UAF credentials from unauthorized use. It is created by the ASM by mixing various sources of information together. Typically, a KHAccessToken contains the following four data items in it: AppID, PersonaID, ASMToken and CallerID.

AppID is provided by the FIDO Server and is contained in every FIDO UAF message.

PersonaID is obtained by the ASM from the operational environment. Typically a different PersonaID is assigned to every operating system user account.

ASMToken is a randomly generated secret which is maintained and protected by the ASM.

NOTE

In a typical implementation an ASM will randomly generate an ASMToken when it is launched the first time and will maintain this secret until the ASM is uninstalled.

CallerID is the ID the platform has assigned to the calling FIDO UAF Client (e.g. "bundle ID" for iOS). On different platforms the caller ID can be obtained differently.

NOTE

For example on Android platform ASM can use the hash of the caller's apk-signing-cert.

The ASM uses the KHAccessToken to establish a link between the ASM and the key handle that is created by authenticator on behalf of this ASM.

The ASM provides the KHAccessToken to the authenticator with every command which works with key handles.

NOTE

The following example describes how the ASM constructs and uses KHAccessToken.

- During a Register request
  - Append AppID
    - KHAccessToken = AppID
  - If a bound authenticator, append ASMToken, PersonaID and CallerID
Bound authenticators must support a mechanism for binding generated key handles to ASMs. The binding mechanism must have at least the same security characteristics as mechanism for protecting KHAccessToken described above. As a consequence it is recommended to securely derive KHAccessToken from AppID, ASMToken, PersonaID and the CallerID.

6.2 Access Control for ASM APIs

The following table summarizes the access control requirements for each API call.

ASMs must implement the access control requirements defined below. ASM vendors may implement additional security mechanisms.

Terms used in the table:

- **NoAuth** -- no access control
- **CallerID** -- FIDO UAF Client's platform-assigned ID is verified
- **UserVerify** -- user must be explicitly verification
- **KeyIDList** -- must be known to the caller
<table>
<thead>
<tr>
<th>Commands</th>
<th>First-factor bound authenticator</th>
<th>Second-factor bound authenticator</th>
<th>First-factor roaming authenticator</th>
<th>Second-factor roaming authenticator</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetInfo</td>
<td>NoAuth</td>
<td>NoAuth</td>
<td>NoAuth</td>
<td>NoAuth</td>
</tr>
<tr>
<td>OpenSettings</td>
<td>NoAuth</td>
<td>NoAuth</td>
<td>NoAuth</td>
<td>NoAuth</td>
</tr>
<tr>
<td>Register</td>
<td>UserVerify</td>
<td>UserVerify</td>
<td>UserVerify</td>
<td>UserVerify</td>
</tr>
<tr>
<td>Authenticate</td>
<td>UserVerify AppID CallerID PersonaID</td>
<td>UserVerify AppID KeyIDList CallerID PersonaID</td>
<td>UserVerify AppID KeyIDList</td>
<td>UserVerify AppID KeyIDList</td>
</tr>
<tr>
<td>GetRegistrations*</td>
<td>CallerID PersonaID</td>
<td>CallerID PersonaID</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Deregister</td>
<td>AppID KeyID CallerID PersonaID</td>
<td>AppID KeyID CallerID PersonaID</td>
<td>AppID KeyID</td>
<td>AppID KeyID</td>
</tr>
</tbody>
</table>

A. References

A.1 Normative references

[ECMA-262]  

[FIDO Glossary]  
R. Lindemann, D. Baghdasaryan, B. Hill, J. Hodges, FIDO Technical Glossary. FIDO Alliance Proposed Standard. URLs:  
HTML: fido-glossary-v1.0-ps-20141208.html
PDF: fido-glossary-v1.0-ps-20141208.pdf

[RFC2119]  

[RFC4648]  
S. Josefsson, The Base16, Base32, and Base64 Data Encodings (RFC 4648), IETF, October 2006, URL: http://www.ietf.org/rfc/rfc4648.txt

[UAAuthnrCommands]  
D. Baghdasaryan, J. Kemp, R. Lindemann, R. Sasson, B. Hill, FIDO UAF Authenticator Commands v1.0. FIDO Alliance Proposed Standard. URLs:  
HTML: fido-uaf-authnr-cmds-v1.0-ps-20141208.html
PDF: fido-uaf-authnr-cmds-v1.0-ps-20141208.pdf

[UAAuthnrMetadata]  
B. Hill, D. Baghdasaryan, J. Kemp, FIDO UAF Authenticator Metadata Statements v1.0. FIDO Alliance Proposed Standard. URLs:  
HTML: fido-uaf-authnr-metadata-v1.0-ps-20141208.html
PDF: fido-uaf-authnr-metadata-v1.0-ps-20141208.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-v1.0-ps-20141208.html
PDF: fido-uaf-protocol-v1.0-ps-20141208.pdf

[UAFRegistry]  
R. Lindemann, D. Baghdasaryan, B. Hill, FIDO UAF Registry of Predefined.
A.2 Informative references

[ECMA-404]
   The JSON Data Interchange Format. 1 October 2013. Standard. URL:

[FIDOSecRef]
   R. Lindemann, D. Baghdasaryan, B. Hill, FIDO Security Reference. FIDO
   Alliance Proposed Standard. URLs:
   HTML: fido-security-ref-v1.0-ps-20141208.html
   PDF: fido-security-ref-v1.0-ps-20141208.pdf

[RFC2397]
   L. Masinter. The "data" URL scheme. August 1998. Proposed Standard. URL:

[UAFAppAPIAndTransport]
   B. Hill, D. Baghdasaryan, B. Blanke, FIDO UAF Application API and Transport
   Binding Specification. FIDO Alliance Proposed Standard. URLs:
   HTML: fido-uaf-client-api-transport-v1.0-ps-20141208.html
   PDF: fido-uaf-client-api-transport-v1.0-ps-20141208.pdf

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