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 Implementation 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 Implementation Draft Specification has been prepared by FIDO Alliance, Inc. Permission is hereby granted to use the Specification solely for the purpose of implementing 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
   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
   3.2.2 Mapping Authenticator Status Codes to ASM Status Codes
   3.3 ASMRequest Dictionary
   3.3.1 Dictionary Get Info Request Members
   3.4 ASMRresponse 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 Authenticateln Object
      3.7.1.1 Dictionary Authenticateln 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
   5.1.2 Alternate Android AIDL Service ASM Implementation
   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 `|` 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 [FIDOGLOSSARY].

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

**NOTE**

Note: Certain dictionary members need to be present in order to comply with FIDO requirements. Such members are marked in the WebIDL definitions found in this document, as `required`. The keyword `required` has been introduced by [WebIDL-ED], which is a work-in-progress. If you are using a WebIDL parser which implements [WebIDL], then you may remove the keyword `required` from your WebIDL and use other means to ensure those fields are present.

1.1 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 manner defined in this document.

The FIDO UAF protocol and its various operations is described in the FIDO UAF Protocol Specification [UAFProtocol]. The following simplified architecture diagram illustrates the interactions and actors this document is concerned with:

Fig. 1 UAF ASM API Architecture

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-formated string, and sends it to the ASM. The ASM de-serializes the JSON-formated string, processes the request, constructs a response, stringifies it, returning it as a JSON-formated string.

NOTE
The ASM request processing rules in this document explicitly assume that the underlying authenticator implements the "UAFV1TLV" assertion scheme (e.g. references to TLVs and tags) as described in [UAFProtocol]. If an authenticator supports a different assertion scheme then the corresponding processing rules must be replaced with appropriate assertion scheme-specific rules.

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 as defined in [UAFAuthnrCommands].

3.1 Request enum

WebIDL
enum Request {  
    "GetInfo",  
    "Register",  
    "Authenticate",  
    "Deregister",  
    "GetRegistrations",  
    "OpenSettings"  
};
### 3.2 StatusCode Interface

If the ASM needs to return an error received from the authenticator, it **shall** map the status code received from the authenticator to the appropriate ASM status code as specified here.

If the ASM doesn't understand the authenticator's status code, it **shall** treat it as `UAF_ASM_STATUS_ERROR` and map it to `UAF_ASM_STATUS_ERROR` if it cannot be handled otherwise.

If the caller of the ASM interface (i.e. the FIDO Client) doesn't understand a status code returned by the ASM, it **shall** treat it as `UAF_ASM_STATUS_ERROR`. This might occur when new error codes are introduced.

```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;
  const short UAF_ASM_STATUS_KEY_DISAPPEARED_PERMANENTLY = 0x04;
  const short UAF_ASM_STATUS_CANNOT_RENDER_TRANSACTION_CONTENT = 0x05;
  const short UAF_ASM_STATUS_USER_NOT_RESPONSIVE = 0x06;
  const short UAF_ASM_STATUS_INSUFFICIENT_AUTHENTICATOR_RESOURCES = 0x07;
  const short UAF_ASM_STATUS_USER_LOCKOUT = 0x08;
  const short UAF_ASM_STATUS_USER_NOT_ENROLLED = 0x09;
};
```

#### 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.

`UAF_ASM_STATUS_CANNOT_RENDER_TRANSACTION_CONTENT` of type `short`
- Transaction content cannot be rendered, e.g. format doesn't fit authenticator's need.

`UAF_ASM_STATUS_KEY_DISAPPEARED_PERMANENTLY` of type `short`
- Indicates that the UAuth key disappeared from the authenticator and cannot be restored.

`UAF_ASM_STATUS_AUTHENTICATOR_DISCONNECTED` of type `short`
- Indicates that the authenticator is no longer connected to the ASM.

`UAF_ASM_STATUS_USER_NOT_RESPONSIVE` of type `short`
- The user took too long to follow an instruction, e.g. didn't swipe the finger within the accepted time.

`UAF_ASM_STATUS_INSUFFICIENT_AUTHENTICATOR_RESOURCES` of type `short`
- Insufficient resources in the authenticator to perform the requested task.

`UAF_ASM_STATUS_USER_LOCKOUT` of type `short`
- The operation failed because the user is locked out and the authenticator cannot automatically trigger an action to change that. Typically the user would have to enter an alternative password (formally: undergo some other alternative user verification method) to re-enable the use of the main user verification method.

- **NOTE** Any method the user can use to (re-) enable the main user verification method is considered an alternative user verification method and must be properly declared as such. For example, if the user can enter an alternative password to re-enable the use of fingerprints or to add additional fingers, the authenticator obviously supports fingerprint or password based user verification.

`UAF_ASM_STATUS_USER_NOT_ENROLLED` of type `short`
- The operation failed because the user is not enrolled to the authenticator and the authenticator cannot automatically trigger user enrollment.

#### 3.2.2 Mapping Authenticator Status Codes to ASM Status Codes

Authenticators are returning a status code in their responses to the ASM. The ASM needs to act on those responses and also map the status code returned by the authenticator to an ASM status code.

The mapping of authenticator status codes to ASM status codes is specified here:

<table>
<thead>
<tr>
<th>Authenticator Status Code</th>
<th>ASM Status Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>UAF_CMD_STATUS_OK</code></td>
<td><code>UAF_ASM_STATUS_OK</code></td>
<td>Pass-through success status.</td>
</tr>
<tr>
<td><code>UAF_CMD_STATUS_ERR_UNKNOWN</code></td>
<td><code>UAF_ASM_STATUS_ERROR</code></td>
<td>Pass-through unspecific error status.</td>
</tr>
<tr>
<td><code>UAF_CMD_STATUS_ACCESS_DENIED</code></td>
<td><code>UAF_ASM_STATUS_ACCESS_DENIED</code></td>
<td>According to [UAFAuthnrCommands], 1</td>
</tr>
</tbody>
</table>
### Authenticator Status Code | ASM Status Code | Comment
--- | --- | ---
UAF_CMD_STATUS_USER_NOT_ENROLLED | UAF_ASM_STATUS_USER_NOT_ENROLLED (or UAF_ASM_STATUS_ACCESS_DENIED in some situations) | Might occur at the `Sign` command or at the `Register` command if the authenticator cannot automatically trigger user enrollment. The mapping depends on the command as follows. In the case of "Register" command, the mapped to UAF_ASM_STATUS_USER_NOT_ENROLLED in order to tell the calling FIDO Client if there is an authenticator present but the user enrollment needs to be triggered outside authenticator. In the case of the "Sign" command, the key needs to be protected by one of the authenticator’s user verification methods at all times. So if this error occurs it is consic internal error and hence mapped to UAF_ASM_STATUS_ACCESS_DENIED.

UAF_CMD_STATUS_CANNOT_RENDER_TRANSACTION_CONTENT | UAF_ASM_STATUS_CANNOT_RENDER_TRANSACTION_CONTENT | Pass-through status code as it indicate problem to be resolved by the entity on the transaction text.

UAF_CMD_STATUS_USER_CANCELLED | UAF_ASM_STATUS_USER_CANCELLED | Map to UAF_ASM_STATUS_USER_Cancelled.

UAF_CMD_STATUS_CMD_NOT_SUPPORTED | UAF_ASM_STATUS_OK or UAF_ASM_STATUS_ERROR | If the ASM is able to handle that command on behalf of the authenticator (e.g. removing key handle in the case of Dereg command for a bound authenticator), the UAF_ASM_STATUS_OK must be returned. Map the status code UAF_ASM_STATUS_ERROR otherwise.

UAF_CMD_STATUS_ATTESTATION_NOT_SUPPORTED | UAF_ASM_STATUS_ERROR | Indicates an ASM issue as the ASM has obviously not requested one of the supported attestation types indicated in the authenticator’s response to the GetInfo command.

UAF_CMD_STATUS_PARAMS_INVALID | UAF_ASM_STATUS_ERROR | Indicates an ASM issue as the ASM has obviously not provided the correct parameters to the authenticator when sending the command.

UAF_CMD_STATUS_KEY_DISAPPEARED_PERMANENTLY | UAF_ASM_STATUS_KEY_DISAPPEARED_PERMANENTLY | Pass-through status code. It indicates that Usauth key disappeared permanently or the RP App might want to trigger re-registration the authenticator.

UAF_STATUS_CMD_TIMEOUT | UAF_ASM_STATUS_ERROR | Retry operation and map to UAF_ASM_STATUS_ERROR if the problem persists.

UAF_CMD_STATUS_USER_NOT_RESPONSIVE | UAF_ASM_STATUS_USER_NOT_RESPONSIVE | Pass-through status code.

UAF_CMD_STATUS_INSUFFICIENT_RESOURCES | UAF_ASM_STATUS_INSUFFICIENT_AUTHENTICATOR_RESOURCES | Pass-through status code.

Any other status code | UAF_ASM_STATUS_ERROR | Map any unknown error code to UAF_ASM_STATUS_ERROR. This might happen when an ASM communicates with an author implementing a newer UAF specificatic the ASM.

### 3.3 ASMRequest Dictionary

All ASM requests are represented as `ASMRequest` objects.

```webidl
dictionary ASMRequest {
  required Request requestType;
  Version asmVersion;
  unsigned short authenticatorIndex;
  object args;
  Extension[] exts;
}
```

#### 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 `asmVersion` must be 1.1 (i.e. major version is 1 and minor version is 1) for this version of the specification.
- `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`
3.4 ASMResponse Dictionary

All ASM responses are represented as ASMResponse objects.

```
WebIDL
dictionary ASMResponse {
  required short statusCode;
  object responseData;
  Extension[] exts;
};
```

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.

**NOTE**

It is up to the ASM to decide whether authenticators which are disconnected temporarily will be reported or not. However, if disconnected authenticators are reported, the FIDO Client might trigger an operation via the ASM on those. The ASM will have to notify the user to connect the authenticator and report an appropriate error if the authenticator isn’t connected in time.

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. In the case of an error the values of the fields might be empty (e.g. array with no members).

See section 3.2.2 Mapping Authenticator Status Codes to ASM Status Codes for details on the mapping of authenticator status codes to ASM status codes.

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 or an empty list.

3.5.2 AuthenticatorInfo Dictionary

```
WebIDL
dictionary AuthenticatorInfo {
  string authenticatorIndex;
  string authenticatorVendor;
  string authenticatorModel;
  string authenticatorSerialNumber;
};
```
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.

- **aavVersions** 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) 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.

- **assertionAlgorithm** of type required DOMString
  Indicates the authentication algorithm that the user authentication uses. Authentication algorithm identifiers are defined in are defined in [FIDORegistry] with aad prefix.

- **assertionTagTypes** 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 [FIDORegistry].

- **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 [FIDORegistry].

- **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 [FIDORegistry].

- **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 [FIDORegistry].

**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 [FIDORegistry].

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

- **tcDisplayContentType** of type DOMString
  Supported transaction content type [FIDOMetadataStatement].
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 FIDOMetadataStatement. For the definition of the DisplayPNGCharacteristicsDescriptor structure see [FIDOMetadataStatement].

This list must be present if PNG-image based transaction confirmation is supported, i.e. tcDisplay is non-zero and tcDisplayContentType is image/png.

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 [FIDOMetadataStatement].

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].

description of type DOMString
Portable Network Graphic (PNG) format image file representing the icon encoded as a data: url [RFC2397].

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].

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
  - UAF_ASM_STATUS_AUTHENTICATOR_DISCONNECTED
  - UAF_ASM_STATUS_USER_NOT_RESPONSIVE
  - UAF_ASM_STATUS_INSUFFICIENT_AUTHENTICATOR_RESOURCES
  - UAF_ASM_STATUS_USER_LOCKOUT
  - UAF_ASM_STATUS_USER_NOT_ENROLLED
- ASMResponse.responseData must be an object of type RegisterOut. In the case of an error the values of the fields might be empty (e.g. empty strings).

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

finalChallenge of type required DOMString
base64url-encoded challenge data [RFC4648]
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 specification [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_AUTHENTICATOR_DISCONNECTED`.
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 the user is locked out (e.g. too many failed attempts to get verified) and the authenticator cannot automatically trigger unblocking, return `UAF_ASM_STATUS_USER_LOCKOUT`.

   - 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 neither the ASM nor the Authenticator can trigger the enrollment process, return `UAF_ASM_STATUS_USER_NOT_ENROLLED`.
   - If enrollment fails, return `UAF_ASM_STATUS_ACCESS_DENIED`.
4. Construct `KHAccessToken` (see section KHAccessToken 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.authenticationAlgorithms` field.

6. Create a `TAG_UAFV1_REG_ASSERTION` structure and pass it to the authenticator

   1. Copy `FinalChallengeHash`, `KHAccessToken`, `RegisterIn.Username`, `UserVerificationToken`, `RegisterIn.AppID`, `RegisterIn.AssertionScheme`

   2. 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. If the authenticator returns an error, handle that error appropriately. If the connection to the authenticator gets lost and cannot be restored, return `UAF_ASM_STATUS_AUTHENTICATOR_DISCONNECTED`. If the operation finally fails, map the authenticator error code to the appropriate ASM error code (see section 3.2.2 Mapping Authenticator Status Codes to ASM Status Codes for details).

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.responseType` must be set to `AuthenticateResponse`.
- `ASMResponse.authenticatorIndex` must be set to the authenticator index.
- `ASMResponse.errorCode` must be set to the appropriate error code.
• 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
  • UAF_ASM_STATUS_CANT_RENDER_TRANSACTION_CONTENT
  • UAF_ASM_STATUS_KEY_DISAPPEARED_PERMANENTLY
  • UAF_ASM_STATUS_AUTHENTICATOR_DISCONNECTED
  • UAF_ASM_STATUS_USER_NOT_RESPONSIVE
  • UAF_ASM_STATUS_USER_LOCKOUT
  • UAF_ASM_STATUS_USER_NOT_ENROLLED

• ASMResponse.responseData must be an object of type AuthenticateOut. In the case of an error the values of the fields might be empty (e.g. empty strings).

3.7.1 AuthenticateIn Object

```
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
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 [FIDOMetadataStatement])
- content of type required DOMString
  Contains the base64url-encoded [RFC4648] transaction content according to the contentType to be shown to the user.
- tcDisplayPNGCharacteristics of type DisplayPNGCharacteristicsDescriptor
  Transaction content PNG characteristics. For the definition of the displayPNGCharacteristicsDescriptor structure See [FIDOMetadataStatement].

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 \texttt{authenticatorIndex}. If the authenticator cannot be located, then fail with \texttt{UAF\_ASM\_STATUS\_AUTHENTICATOR\_DISCONNECTED}.

2. If no user is enrolled with this authenticator (such as biometric enrollment, PIN setup, etc.), return \texttt{UAF\_ASM\_STATUS\_ACCESS\_DENIED}.

3. The ASM must request the authenticator to verify the user.
   - If the user is locked out (e.g. too many failed attempts to get verified) and the authenticator cannot automatically trigger unblocking, return \texttt{UAF\_ASM\_STATUS\_USER\_LOCKOUT}.
   - If verification fails, return \texttt{UAF\_ASM\_STATUS\_ACCESS\_DENIED}

4. Construct \texttt{KHAccessToken} (see section \texttt{KHAccessToken} for more details).

5. Hash the provided \texttt{ASMRequest.finalChallenge} using an authenticator-specific hash function (\texttt{FinalChallengeHash}).

   The authenticator's preferred hash function information must meet the algorithm defined in the \texttt{AuthenticatorInfo.authenticationAlgorithm} field.

6. If this is a Second Factor authenticator and \texttt{AuthenticatorIn.keyIDs} is empty, then return \texttt{UAF\_ASM\_STATUS\_ACCESS\_DENIED}.

7. If \texttt{AuthenticatorIn.keyIDs} is not empty,
   - If there is a bound authenticator, then look up ASM's database with \texttt{AuthenticatorIn.appID} and \texttt{AuthenticatorIn.keyIDs} and obtain the KeyHandles associated with it.
     - Return \texttt{UAF\_ASM\_STATUS\_KEY\_DISAPPEARED\_PERMANENTLY} if the related key disappeared permanently from the authenticator.
     - Return \texttt{UAF\_ASM\_STATUS\_ACCESS\_DENIED} if no entry has been found.
   - If this is a roaming authenticator, then treat \texttt{AuthenticatorIn.keyIDs} as KeyHandles.

8. Create \texttt{TAG\_UAF\_V1\_SIGN\_CMD} structure and pass it to the authenticator.
   - Copy \texttt{AuthenticatorIn.appID}, \texttt{AuthenticatorIn.Transaction.content} (if not empty), \texttt{FinalChallengeHash}, \texttt{KHAccessToken}, \texttt{UserVerificationToken}, and \texttt{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.

9. Invoke the command and receive the response. If the authenticator returns an error, handle that error appropriately. If the connection to the authenticator gets lost and cannot be restored, return \texttt{UAF\_ASM\_STATUS\_AUTHENTICATOR\_DISCONNECTED}. If the operation finally fails, map the authenticator error code to the appropriate ASM error code (see section 3.2.2 Mapping Authenticator Status Codes to ASM Status Codes for details).

10. Parse \texttt{TAG\_UAF\_V1\_SIGN\_CMD\_RESP}
    - If it's a first-factor authenticator and the response includes \texttt{TAG\_USERNAME\_AND\_KEYHANDLE}, then
      1. Extract usernames from \texttt{TAG\_USERNAME\_AND\_KEYHANDLE} fields.
      2. If two or more equal usernames are found, then choose the one which has registered most recently.

11. Create the \texttt{AuthenticateOut} object
    - Set \texttt{AuthenticateOut.assertionScheme} as \texttt{AuthenticatorInfo.assertionScheme}
    - Encode the content of \texttt{TAG\_AUTHENTICATOR\_ASSERTION} (e.g. \texttt{TAG\_UAF\_V1\_AUTH\_ASSERTION}) in base64url format and set as \texttt{AuthenticateOut.assertion}.
    - Return the \texttt{AuthenticateOut} object.

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 \texttt{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 [FIDORegistry] 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 \texttt{TRANSACTION\_CONFIRMATION\_DISPLAY\_ART} (bitwise) or \texttt{TRANSACTION\_CONFIRMATION\_DISPLAY\_PRIVILEGED\_SOFTWARE}.

3.8 Deregister Request

Delete registered UAF record from the authenticator.

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

- \texttt{ASMRequest.requestType} must be set to \texttt{Deregister}
- **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 `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`
  - `UAF_ASM_STATUS_AUTHENTICATOR_DISCONNECTED`

### 3.8.1 DeregisterIn Object

<table>
<thead>
<tr>
<th>WebIDL</th>
</tr>
</thead>
</table>
| ```
  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 de-registered. The `keyID` can be an empty string. In this case all `keyID` related to this `appID` must 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 **authenticatorIndex**
2. Construct **KHAccessToken** (see section [KHAccessToken] for more details).
3. If this is a bound authenticator, then
   - If the value of `DeregisterIn.keyID` is an empty string, then lookup all pairs of this `appID` and any `keyID` mapped to this `authenticatorIndex` and delete them. Go to step 4.
   - Otherwise, lookup the authenticator related data in the ASM database and delete the record associated with `DeregisterIn.appID` and `DeregisterIn.keyID`. Go to step 4.

4. Create the **TAG_UAFV1_DEREGISTER_CMD** structure, copy **KHAccessToken** and `DeregisterIn.keyID` and pass it to the authenticator.

5. Invoke the command and receive the response. If the authenticator returns an error, handle that error appropriately. If the connection to the authenticator gets lost and cannot be restored, return `UAF_ASM_STATUS_AUTHENTICATOR_DISCONNECTED`. If the operation finally fails, map the authenticator error code to the appropriate ASM error code (see section 3.2.2 Mapping Authenticator Status Codes to ASM Status Codes for details). Return proper **ASMResponse**.

### 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`
  - `UAF_ASM_STATUS_AUTHENTICATOR_DISCONNECTED`

  The **ASMResponse.responseData** must be an object of type `GetRegistrationsOut`. In the case of an error the values of the fields might be empty (e.g. empty strings).

### 3.9.1 GetRegistrationsOut Object

<table>
<thead>
<tr>
<th>WebIDL</th>
</tr>
</thead>
</table>
| ```
  dictionary GetRegistrationsOut {
    required AppRegistration[] appRegs;
  };
``` |

### 3.9.1.1 Dictionary GetRegistrationsOut Members

- **NOTE**
  - In the case of roaming authenticators, the `keyID` passed to the authenticator might be an empty string. The authenticator is supposed to deregister all keys related to this `appID` in this case.
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.

- `keyIDs` of type `array of required DOMString`
  
  List of key identifiers associated with the `appID`.

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

This section is normative.

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, 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.

### 5.1.2 Alternate Android AIDL Service ASM Implementation

The Android Intent API can also be implemented using Android AIDL services as an alternative transport mechanism to Android Intents. Please see Android Intent API section [UAFAppAPIAndTransport] for differences between the Android AIDL service and Android Intent implementation.

### 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:

```c
EXAMPLE 1

/*! @file asm.h
 */

#ifndef __ASM_H
#define __ASM_H

#ifdef _WIN32
#define ASM_API __declspec(dllexport)
#endif

#ifdef _WIN32
#pragma warning ( disable : 4251 )
#endif

#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,
 const 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 IEnumerate
{
 public
 virtual ~IEnumerate() {}
 /**
 This function is called when an authenticator is plugged or
 unplugged.
 */
 virtual void Notify(const asmEnumerationType_t eventType,
 const asmJSONData_t &AuthenticatorInfo) = 0;
};

/* Initializes ASM plugin. This is the first function to be called.
 */
ASM_FUNC asmResult_t asmInit(ASM::IEnumerater)
```
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.

- 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 associated with a legitimate FIDO Client.

- 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

- 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 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.

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
  - Set KHAccessToken to a secret value only known to the ASM. This value will always be the same for this ASM.
  - Append AppID
    - KHAccessToken = AppID
  - If a bound authenticator, append ASMToken, PersonaID and CallerID
    - KHAccessToken |= ASMToken | PersonaID | CallerID
  - Hash KHAccessToken
    - KHAccessToken = hash(KHAccessToken)
  - Provide KHAccessToken to the authenticator
  - The authenticator puts the KHAccessToken into RawKeyHandle (see [UAFAuthnrCommands] for more details)
  - During other commands which require KHAccessToken as input argument
    - The ASM computes KHAccessToken the same way as during the Register request and provides it to the authenticator along with other arguments.
    - The authenticator unwraps the provided key handle(s) and proceeds with the command only if RawKeyHandle.KHAccessToken is equal to the provided KHAccessToken.

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 as described above. As a consequence it is recommended to securely derive KHAccessToken from AppID, ASMToken, PersonaID and the CallerID.

NOTE
It is recommended for roaming authenticators that the KHAccessToken contains only the AppID since otherwise users won’t be able to use them on different machines (PersonaID, ASMToken and CallerID are platform specific). If the authenticator vendor decides to do that in order to address a specific use case, however, it is allowed.

Including PersonaID in the KHAccessToken is optional for all types of authenticators. However an authenticator designed for multi-user systems will likely have to support it.

If an ASM for roaming authenticators doesn’t use a KHAccessToken which is different for each AppID, the ASM must include the AppID in the command for a deregister request containing an empty KeyID.

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 verified
- 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>Function</td>
<td>Parameters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authenticate</td>
<td>UserVerify, AppID, CallerID, PersonaID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GetRegistrations</td>
<td>CallerID, PersonaID, KeyIDList, CallerID, PersonaID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deregister</td>
<td>AppID, KeyID, PersonaID, CallerID</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### A. References

#### A.1 Normative references


#### A.2 Informative references


[WebIDL] Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15 December 2016. W3C Working Draft. URL: https://www.w3.org/TR/WebIDL-1/