UAF Authenticator-specific Module API

Specification Set: fido-uaf-v1.0-rd-20140209  REVIEW DRAFT

Editors:
Davit Baghdasaryan, NokNok Labs Inc
John Kemp, FIDO Alliance

Contributors:
Roni Sasson, Discretix

Abstract:
Different UAF authenticators may be connected to a user device via various physical interfaces. 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.

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

The document’s intended audience is FIDO Authenticator and FIDO UAF Client vendors.
FIDO UAF Authenticator-specific Module API

19 Status:

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

27 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.

32 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.

36 Copyright © 2014 FIDO Alliance, Inc. All rights reserved.
# Table of Contents

1 Terminology ........................................................................................................5
   1.1 Key Words ...............................................................................................5
   1.2 Overview ..................................................................................................5
   1.1 Notations ................................................................................................6

2 ASM Requests and Responses ...........................................................................7
   2.1 Constants ...............................................................................................7
   2.2 Extension ................................................................................................8
   2.3 Version ....................................................................................................8
   2.4 ASM Request .........................................................................................9
   2.5 ASM Response .....................................................................................10
   2.6 GetInfo Request ..................................................................................10
      2.6.1 GetInfoOut Object ........................................................................11
      2.6.2 AuthenticatorInfo Object ...............................................................11
   2.7 Register Request ..................................................................................14
      2.7.1 RegisterIn Object ..........................................................................14
      2.7.2 RegisterOut Object ......................................................................15
      2.7.3 Detailed Description ......................................................................16
   2.8 Authenticate Request ...........................................................................17
      2.8.1 AuthenticatorIn Object ................................................................18
      2.8.2 Transaction Object .......................................................................18
      2.8.3 AuthenticateOut Object ................................................................19
      2.8.4 Detailed Description ......................................................................19
   2.9 Deregister Request ..............................................................................22
      2.9.1 DeregisterIn Object .....................................................................22
      2.9.2 Detailed Description .....................................................................23
2.10 GetRegistrations Request
2.10.1 GetRegistrationsOut Object
2.10.2 AppRegistration Object
2.10.3 Detailed Description
2.11 Commit Request
2.11.1 CommitIn Object
2.11.2 Detailed Description
2.12 ManageSettings Request
3 Using ASM API
4 ASM API on various platforms
4.1 Android ASM API
4.2 Windows ASM API
5 Security and Privacy Guidelines
5.1 KHAccessToken
5.2 Access Control for ASM APIs
6 Bibliography
1 Terminology

Type names, attribute names and element names are written in *italics*. String literals are enclosed in “”, e.g. “UAF-TLV”. In formulas we use “|” to denote byte wise concatenation operations. UAF specific terminology used in this document is defined in [FIDOGlossary].

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].

1.2 Overview

An Authenticator-specific Module (ASM) is a platform-specific software component offering an API to UAF clients, enabling them to discover and communicate with one or more installed authenticators. A single ASM may report on behalf of multiple authenticators. The intended audience for this document is UAF Authenticator and UAF Client vendors. ASM implementors MUST follow the interface definition of ASM requests and response. The detailed functionality of each request outlined in this specification is not normative however, if implementors do not follow these descriptions. They MUST implement similar or higher security measures than those described in this specification.

The 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:
1.1 Notations

- ASM requests and responses are presented in WebIDL format
- All examples are provided in Javascript language
2 ASM Requests and Responses

The ASM API uses the JSON format [ECMA-404] for exchanging data. In order to send an ASM request, a UAF Client must create an appropriate JSON object, convert it to a string, and send to an ASM. The ASM then parses the string into a JSON object, processes the request and sends the response as a JSON string.

2.1 Constants

```cpp
interface StatusCode {
    const short UAF_STATUS_OK = 0;
    const short UAF_STATUS_ERROR = 1;
    const short UAF_STATUS_ACCESS_DENIED = 2;
    const short UAF_STATUS_USER_CANCELED = 3;
}
```

- **UAF_STATUS_OK**
  - No error condition encountered.
- **UAF_STATUS_ERROR**
  - Unknown error has been encountered during the processing.
- **UAF_STATUS_ACCESS_DENIED**
  - Access to this request is denied.
- **UAF_STATUS_USER_CANCELED**
  - Indicates that user explicitly canceled the request.

```cpp
enum Request {
    "GetInfo",
    "Register",
    "Authenticate",
    "Deregister",
    "GetRegistrations",
    "Commit",
    "ManageSettings"
}
```
2.2 Extension

Dictionary Extension {
    DOMString id;
    DOMString data;
}

id of type DOMString, mandatory
    Extension ID.

data of type DOMString, mandatory
    Extension data.

2.3 Version

Dictionary Version {
    short major;
    short minor;
}

major of type int, mandatory
    Major version.

minor of type int, mandatory
    Minor version.
2.4 ASM Request

All ASM requests are represented as ASMRequest objects.

Dictionary ASMRequest {
    Request requestType;
    Version asmVersion;
    Version aiVersion;
    long authenticatorIndex;
    object args;
    Extension[] exts;
}

*requestType* of type Request, mandatory

Request type.

*asmVersion* of type Version, optional

ASM message version to be used with this request.

*aiVersion* of type Version, optional

Authenticator Interface Version to be used with this request.

*authenticatorIndex* of type short, optional

Authenticator Index. Refer to “GetInfo Request” for more details. This must not be set in GetInfo request.

*args* of type Object, optional

Request specific arguments. If set - this attribute may have one of the following types:

GetInfoIn, RegisterIn, AuthenticateIn, GetRegistrationsIn

*exts* of type Extension[], optional

List of UAF extensions.
2.5 ASM Response

All ASM responses are represented as ASMResponse objects.

Dictionary ASMResponse {
    Request responseType;
    short statusCode;
    object responseData;
    Extension[] exts;
}

responseType of type Request, mandatory
Response type. Must be the same as ASMRequest.requestType.

statusCode of type short, mandatory
Response status. May have one of the values in StatusCode interface.

responseData of type Object, optional
Request specific response data. This attribute may have one of the following types:
GetInfoOut, RegisterOut, AuthenticateOut, GetRegistrationsOut

exts of type Extension[], optional
List of UAF extensions.

2.6 GetInfo Request

Return information about available authenticators.

Unless otherwise specified, all steps are NORMATIVE for ASM implementation.

1. Enumerate all the authenticators this ASM supports
2. Collect information about all of them
3. Assign indices to them (authenticatorIndex)
4. Return the information to UAF Client
ASMRequest must have the following data

- ASMRequest.requestType must be set to "GetInfo"

ASMResponse must have the following data

- ASMResponse.statusCode can have one of the following values
  - UAF_STATUS_OK
  - UAF_STATUS_ERROR
- ASMResponse.responseData must be an object of type GetInfoOutput

### 2.6.1 GetInfoOut Object

```
Dictionary GetInfoOut {
  Version[] asmVersion;
  DOMString vendor;
  AuthenticatorInfo[] authenticators;
}
```

- asmVersions of type Version[], mandatory, non-empty
  - Supported ASM interface versions.
  - Indicates the version of ASM request and response messages that current ASM supports.
- vendor of type DOMString, optional
  - ASM vendor information.
- authenticators of type AuthenticatorInfo[], mandatory
  - List of authenticators reported by the current ASM.

### 2.6.2 AuthenticatorInfo Object
FIDO UAF Authenticator-specific Module API

Dictionary AuthenticatorInfo {
    long authenticatorIndex;
    Version[] aiVersion;
    DOMString aid;
    long userVerification;
    long keyProtection;
    long attachmentHint;
    long secureDisplay;
    long authenticationAlg;
    DOMString scheme;
    long additionalInfo;
    boolean isSecondFactor;
    DOMString[] supportedExtensionIDs;
    DOMString title;
    DOMString description;
    DOMString icon;
    boolean hasSettings;
}

authenticatorIndex of type long, mandatory
Authenticator index. Unique, within the scope of all authenticators reported by the ASM, and non-zero index referring to an authenticator. This index is used by the UAF Client to refer to the appropriate authenticator.

aiVersions of type Version[], mandatory, non-empty
Supported Authenticator Interface Versions.

AAID of type DOMString, readonly, mandatory
The Authenticator Attestation ID, which identifies the type and batch of the authenticator.

userVerification of type long, mandatory
A set of bit flags indicating the user verification method(s) supported by the authenticator. The values are defined by the USER_VERIFY_ constants.
A set of bit flags indicating the key protections used by the authenticator. The values are defined by the KEY_PROTECTION constants in [FIDORegistry].

attachmentHint of type long, mandatory

A set of bit flags indicating how the authenticator is currently connected to the system hosting the UAF Client software. The values are defined by the ATTACHMENT_HINT constants defined in [FIDORegistry].

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.

secureDisplay of type long, mandatory

A set of bit flags indicating the availability and type of secure display. The values are defined by the SECURE_DISPLAY_ constants in [FIDORegistry].

authenticationAlg of type long, mandatory

Indicates the authentication algorithm the authenticator uses. Authentication algorithm identifiers are defined in the UAF Protocol specification. [UAFProtocol]

scheme of type DOMString, mandatory

The encoding scheme the authenticator uses for attested data and signatures. Scheme identifiers are defined in the UAF Protocol specification. [UAFProtocol]

additionalInfo of type long, readonly, optional

RESERVED FOR FUTURE USE

isSecondFactor of type boolean, mandatory

Indicates whether the authenticator can be used only as a second factor.

supportedExtensionIDs of type DOMString[], optional

List of supported UAF extension IDs.

title of type DOMString, mandatory

Human readable short title of Authenticator. It must be localized for current locale.

description of type DOMString, mandatory

Human readable longer description of what the Authenticator represents. It must be localized for current locale.
FIDO UAF Authenticator-specific Module API

icon of type DOMString, optional
    base64url encoded PNG file representing authenticator's icon.

hasSettings of type boolean, mandatory
    A boolean value indicating whether the authenticator has its own settings. If so,
    then a FIDO Client can launch these settings by sending a “ManageSettings” re-
    quest.

2.7 Register Request

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

ASMRequest must have the following attributes

- ASMRequest.requestType must be set to “Register”
- ASMRequest.asmVersion and ASMRequest.aiVersion must be set to desired
  versions
- ASMRequest.authenticatorIndex must be set to corresponding ID
- ASMRequest.args must be set to an object of type RegisterIn

ASMResponse must have the following attributes

- ASMResponse.statusCode can have one of the following values
  - UAF_STATUS_OK
  - UAF_STATUS_ERROR
  - UAF_STATUS_ACCESS_DENIED
- ASMResponse.responseData must be an object of type RegisterOut

2.7.1 RegisterIn Object

Dictionary RegisterIn {
    DOMString appID;
}
DOMString username;
DOMString finalChallenge;
}

appID of type DOMString, mandatory, non-empty
FIDO Server Application Identity.
username of type DOMString, mandatory, non-empty
Human readable user account name.
finalChallenge of type DOMString, mandatory, non-empty, base64url
Challenge data.

2.7.2 RegisterOut Object

Dictionary RegisterOut {
    DOMString assertion;
    DOMString certificateChain;
    DOMString keyID;
}

assertion of type DOMString, mandatory, base64url
Authenticator UAF registration assertion.
certificateChain of typeDOMString, optional, base64url
Authenticator's attestation certificate chain.
keyID of type DOMString, mandatory
KeyID of the newly created record. Among other potential uses KeyID will be
passed to “Commit“ function to notify the ASM to mark the record as “READY".
2.7.3 Detailed Description

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

Unless otherwise specified all steps are NORMATIVE for ASM implementation.

1. Locate authenticator using authenticatorIndex

2. If a user is already enrolled with this authenticator (such as biometric enrollment, PIN setup, etc.) - ASM MUST request the authenticator to verify the user and MAY obtain a User Verification Token (if authenticator is designed to return such token).
   a. If verification fails - return UAF_STATUS_ACCESS_DENIED

3. If User is not enrolled with Authenticator – take the User through enrollment process.
   a. If enrollment fails return UAF_STATUS_ACCESS_DENIED

4. Construct KHAccessToken (see section KHAccessToken for more details)

5. Hash the provided RegisterIn.finalChallenge using authenticator specific hash function (FinalChallengeHash)
   a. Authenticator’s preferred hash function information can be obtained from AuthenticatorInfo.authenticationAlg bitflag.

6. Create TAG_UAFV1_REG_CMD structure and pass it to authenticator
   a. Copy FinalChallengeHash, KHAccessToken, RegisterIn.Username, UserVerificationToken
   b. If this authenticator has a Secure Display – provide TAG_FULL_APPID for display
   c. Invoke the command and receive the response
   d. Parse TAG_UAFV1_REG_CMD_RESP
i. Obtain TAG_UAFV1_REG_RESPONSE

ii. If it's a Bound Authenticator

1. Store CallerID, AppID, TAG_KEYHANDLE and TAG_KEYID in ASM’s database
2. Mark the record as “NOT_READY”. The record must become “READY” only after ASM receives confirmation from FIDO Server (see Commit Request)

7. Create RegisterOut object
   a. Set RegisterOut.scheme as AuthenticatorInfo.scheme
   b. Encode TAG_UAFV1_REG_RESPONSE in base64url format and set as RegisterOut.assertion
   c. Set RegisterOut.certificateChain if applicable to this Authenticator
   d. Set RegisterOut.recordID to KeyID
   e. Return RegisterOut object

2.8 Authenticate Request

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

ASMRequest must have the following attributes

- ASMRequest.requestType must be set to “Authenticate”
- ASMRequest.asmVersion and ASMRequest.aiVersion must be set to desired versions
- ASMRequest.authenticatorIndex must be set to corresponding ID
- ASMRequest.args must be set to an object of type AuthenticateIn

ASMResponse must have the following attributes
ASMResponse.statusCode can have one of the following values
- UAF_STATUS_OK
- UAF_STATUS_ERROR
- UAF_STATUS_ACCESS_DENIED

ASMResponse.responseData must be an object of type AuthenticateOut

2.8.1 AuthenticatorIn Object

Dictionary AuthenticateIn {
  DOMString appID;
  DOMString[] keyIDList;
  DOMString finalChallenge;
  Transaction transaction;
}

appID of type DOMString, mandatory, non-empty
FIDO Server Application Identity.
keyIDList of type DOMString[], optional, non-empty
List of base64url encoded keyIDs.
finalChallenge of type DOMString, mandatory, non-empty, base64url
Opaque challenge data.
transaction of type Transaction, optional
Transaction data to be confirmed by user.

2.8.2 Transaction Object

Dictionary Transaction {
  DOMString contentType;
  DOMString content;
}

Copyright © 2014 FIDO Alliance: REVIEW DRAFT
**FIDO UAF Authenticator-specific Module API**

366  contentType of type DOMString, mandatory
367  Transaction content type. For valid values refer to FIDO Registry of Predefined Values document [FIDORegistry].
369  content of type DOMString, mandatory, non-empty, base64url
370  Transaction content to be shown to user.

371  **2.8.3 AuthenticateOut Object**

372  Dictionary AuthenticateOut {
373      DOMString assertion;
374      DOMString keyID;
375  }

376  assertion of type DOMString, mandatory, base64url
377  Authenticator UAF authentication assertion.
378  keyID of type DOMString, mandatory, base64url
379  KeyID of the used user authentication key.

380  **2.8.4 Detailed Description**

381  Refer to [UAFAuthnrCommands] document for more information about TAGs and structure mentioned in this paragraph.
382  Unless otherwise specified all steps are NORMATIVE for ASM implementation.

384  1. Locate authenticator using authenticatorIndex
385  2. If no user is enrolled with this authenticator (such as biometric enrollment, PIN setup, etc.) – return UAF_STATUS_ACCESS_DENIED
FIDO UAF Authenticator-specific Module API

3. ASM MUST request the authenticator to verify the user and MAY obtain a UserVerificationToken (if authenticator is designed to return such token).
   a. If verification fails - return UAF_STATUS_ACCESS_DENIED

4. Construct KHAccessToken

5. Hash the provided RegisterIn.finalChallenge using authenticator specific hash function (FinalChallengeHash)

6. If this is a Second Factor Authenticator and AuthenticateIn.keyIDList is empty – return UAF_STATUS_ACCESS_DENIED

7. If AuthenticateIn.keyIDList is not empty
   a. If this is a Bound Authenticator - look up ASM’s database with AuthenticateIn.appID and AuthenticateIn.keyIDList and obtain KeyHandles associated with them. Omit records that are marked as “NOT_READY”
      i. Return UAF_STATUS_ACCESS_DENIED if no entry has been found
   b. If this is a Roaming Authenticator – treat AuthenticateIn.keyIDList as KeyHandles

8. Create TAG_UAFV1_SIGN_CMD structure and pass it to authenticator
   a. Copy FinalChallengeHash, KHAccessToken, UserVerificationToken, KeyHandles
   b. If this authenticator has a Secure Display
      i. If AuthenticateIn.Transaction is not empty – copy AuthenticateIn.-Transaction.Content into TAG_UAFV1_SIGN_CMD.Transaction-
      Content
      ii. Copy TAG_FULL_APPID
   c. Invoke the command and receive the response
d. Parse TAG_UAFV1_SIGN_CMD_RESP

i. If it's a First Factor Authenticator and the response includes TAG_UNAME_LIST

1. Show all usernames from TAG_UNAME_LIST to user
2. Ask the user to choose a single Username
3. Set KeyHandles to the single KeyHandle associated with selected username
4. Go to step #8 and send a new TAG_UAFV1_SIGN_CMD command

9. Create AuthenticateOut object

a. Set AuthenticateOut.scheme as AuthenticatorInfo.scheme
b. Encode TAG_UAFV1_SIGN_RESPONSE in base64url format and set as AuthenticateOut.assertion
c. Set AuthenticateOut.keyID
d. Return AuthenticateOut object

**Normative Note:**

Some Authenticators might support Secure Display functionality not inside the Authenticator but within the boundaries of the ASM. Typically these are software based Secure 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 Authenticator includes it in the final assertion. Such Authenticator’s Metadata file MUST clearly indicate the type of Secure Display. Typically the flag of Secure Display will be SECURE_DISPLAY_ANY or SECURE_DISPLAY_PRIVILEGED_SOFTWARE. See [UAF Registry] for flags describing Secure Display type.
2.9 Deregister Request

Delete registered UAF data.

ASMRequest must have the following attributes

- ASMRequest.requestType must be set to “Deregister”
- ASMRequest.asmVersion and ASMRequest.aiVersion must be set to desired versions
- ASMRequest.authenticatorIndex must be set to corresponding index
- ASMRequest.args must be set to an object of type DeregisterIn

ASMResponse must have the following attributes

- ASMResponse.statusCode can have one of the following values
  - UAF_STATUS_OK
  - UAF_STATUS_ERROR
  - UAF_STATUS_ACCESS_DENIED

2.9.1 DeregisterIn Object

Dictionary DeregisterIn {
  DOMString appID;
  DOMString keyID;
}

appID of type DOMString, mandatory, non-empty
FIDO Server Application Identity.

keyID of type DOMString, mandatory, non-empty, base64url
keyID of the authenticator to be deregistered.
2.9.2 Detailed Description

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

Unless otherwise specified all steps are NORMATIVE for ASM implementation.

1. Locate authenticator using authenticatorIndex
2. Construct KHAccessToken
3. If this is a Bound Authenticator
   a. Lookup in ASM database and delete the record associated with DeregisterIn.appID and DeregisterIn.keyID
4. Create TAG_UAFV1_DEREG_CMD structure and pass it to authenticator
   a. Copy KHAccessToken, DeregisterIn.keyID
   b. Invoke the command and receive the response
5. Return

2.10 GetRegistrations Request

Return all registrations made for the calling UAF Client.

ASMRequest must have the following attributes

- ASMRequest.requestType must be set to "GetRegistrations"
- ASMRequest.asmVersion and ASMRequest.aiVersion must be set to desired versions
ASMRequest.authenticatorIndex must be set to corresponding ID

ASMResponse must have the following attributes

- ASMResponse.statusCode can have one of the following values
  - UAF_STATUS_OK
  - UAF_STATUS_ERROR
- ASMResponse.responseData must be an object of type GetRegistrationsOut

2.10.1 GetRegistrationsOut Object

Dictionary GetRegistrationsOut {
    AppRegistration[] appRegs;
}

appRegs of type AppRegistration, mandatory, non-empty
List of appID associated registrations.

2.10.2 AppRegistration Object

Dictionary AppRegistration {
    DOMString appID;
    DOMString[] keyIDs;
}

appID of type DOMString, mandatory, non-empty
FIDO Server Application Identity.
keyIDs of type DOMString[], mandatory, non-empty
List of KeyIDs associated with appID
2.10.3 Detailed Description

1. Locate authenticator using authenticatorIndex
2. If this is a Bound Authenticator
   a. Lookup in ASM database and construct a list of AppRegistration objects
3. Create GetRegistrationsOut object and return
   a. Set GetRegistrationsOut.appRegs

2.11 Commit Request

Locate the corresponding record in credential database and mark it as “READY”.

This function must be called by UAF Client after getting registration response from FIDO Server. There can be scenarios in which the Authenticator successfully registers a key, but the FIDO Server rejects the registration (or FIDO Server never receives it due to application or network error). These cases may result in a bad user experience where system reports that user can use the authenticator to successfully login but then the server rejects it. In order to avoid these scenarios ASM SHOULD mark its records “READY” only after it is sure that registration was successful on server side.

ASMRequest must have the following attributes

- ASMRequest.requestType must be set to “Commit”
- ASMRequest.asmVersion and ASMRequest.aiVersion must be set to desired versions
- ASMRequest.authenticatorIndex must be set to corresponding ID

ASMResponse must have the following attributes

- ASMResponse.statusCode can have one of the following values
  - UAF_STATUS_OK
2.11.1 CommitIn Object

Dictionary CommitIn {
  DOMString appID;
  DOMString keyID;
  boolean commit;
}

- `appID` of type `DOMString`, mandatory, non-empty
  FIDO Server Application Identity.
- `keyID` of type `DOMString`, mandatory, non-empty, base64url
  KeyID to be committed.
- `commit` of type `boolean`, mandatory, non-empty
  Indicates if the record must be committed or deleted. `true` means commit and `false` means delete.

2.11.2 Detailed Description

1. Locate authenticator using authenticatorIndex
2. If this is a Bound Authenticator
   a. Locate the record with that record ID in ASM database
   b. If the record is already marked as “READY” - return UAF_STATUS_SUCCESS
   c. If CommitIn.commit== true
      Mark the record as “READY”
   d. Else
      i. Delete the record
3. If this is a Roaming Authenticator
   a. Construct and send “Commit” command to Authenticator. *This command is currently not specified in Authenticator Commands function.*

2.12 ManageSettings Request

Display UI for management of authenticator specific settings.

ASMRequest must have the following attributes

- ASMRequest.requestType must be set to “ManageSettings”
- ASMRequest.authenticatorIndex must be set to corresponding ID

ASMResponse must have the following attributes

- ASMResponse.statusCode can have one of the following values
  - *UAF_STATUS_OK*
In a typical implementation of the FIDO Client, it will call GetInfo during initialization and obtain information about authenticators. Once the information is obtained it will typically be used during UAF message processing to find a match for a given UAF policy. Once a match is found the FIDO Client will call the appropriate function (Register/Authenticate/Deregister) for this authenticator.

After calling Register function, the FIDO Client will wait for FIDO Server response and call Commit function to allow ASM to commit or delete the created record.

Also, FIDO Clients may use the information obtained after GetInfo function to display relevant information about an authenticator to the user.
4 ASM API on various platforms

4.1 Android ASM API

On Android systems, ASM is implemented as a separate APK-packaged application.
In order to be recognized by a FIDO Client, an ASM Plugin application must be designed as an Android Service

```java
// IASMService.aidl
package org.fidoalliance.uaf.asm;

import org.fidoalliance.uaf.asm.IASMResponseListener;

/**
 * UAF ASM interface between UAF Client and UAF ASM.
 */
interface IASMService
{
    /**
     * The asynchronous function processes UAF ASM request and sends the response via provided listener.
     */
    void process(String request, IASMResponseListener listener);
}
```

```java
// IASMResponseListener.aidl
package org.fidoalliance.uaf.asm;

interface IASMResponseListener
{
    /**
     * The callback function is fired by UAF ASM when the response is ready.
     */
    void response(String response);
}
```
The following code demonstrates how to register an ASMService.

```java
public class ASMService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        return mAsmService;
    }

    private IASMService.Stub mAsmService = new IASMService.Stub() {

        @Override
        public void process(String request, IASMResponseListener l) throws RemoteException {
            // ASM code goes here
        }
    };
}
```

Additionally each ASM Plugin APK must include the following entry in its manifest file:

```xml
<service android:name=".exampleASMPluginName">
    <intent-filter>
        <action android:name="org.fidoalliance.uaf.asm.FIDO_INTENT_ENUM_ASM" />
    </intent-filter>
</service>
```

The FIDO Client will find ASM packages installed on the system by looking for packages which have a registered “org.fidoalliance.uaf.asm.FIDO_INTENT_ENUM_ASM” intent.

The following code demonstrates how UAF Clients can find ASMs and invoke the “process” function.
public class ASMAgent {
    private IASMService mAsmService = null;
    private IASMResponseListener.Stub mResponseListener = new IASMResponseListener.Stub() {
        @Override
        public void response(String response) throws RemoteException {
            // The ASM response handler
        }
    };
    private ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder service) {
            mAsmService = IASMService.Stub.asInterface(service);
            try {
                String request = "{'requestType': 'GetInfo' }";
                mAsmService.process(request, mResponseListener);
            } catch (RemoteException e) {
            }
        }
        public void onServiceDisconnected(ComponentName className) {
        }
    };
    void doBindService() {
        // Find ASM packages
        PackageManager pm = getApplicationContext().getPackageManager();
        List<ResolveInfo> asmlist = pm.queryIntentServices(
            new Intent("org.fidoalliance.uaf.asm.FIDO_INTENT_ENUM_ASM"),
            PackageManager.GET_INTENT_FILTERS);
        for (ResolveInfo info : asmlist) {
            getApplicationContext().bindService(
                new Intent().setClassName(info.serviceInfo.packageName,
                    info.serviceInfo.name),
                mConnection,
                Context.BIND_AUTO_CREATE);
        }
    }
}
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 Windows ASM API:

```c
/*! @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
{
```
FIDO UAF Authenticator-specific Module API

```cpp
Plugged     = 0, /**< Indicates that Authenticator Plugged to system */
Unplugged        /**< Indicates that Authenticator Unplugged from system */
};
namespace ASM
{
  /*!
   * \brief Callback listener.
   * UAF Client must pass an object implementating 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.
     * @param response JSON based event data
     * @param exchangeData must be provided by ASM if it needs some data back right after calling the callback function.
     * The lifecycle of this parameter must be managed by ASM. ASM must allocate enough memory for getting the data back.
     */
    virtual void Callback(const asmJSONData_t &response, asmJSONData_t &exchangeData) = 0;
  }; 

  /*!
   * \brief Authenticator Enumerator.
   * 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.
     * @param eventType event type (plugged/unplugged)
     * @param authenticatorInfo JSON based GetInfoResponse object
     */
    virtual void Notify(const asmEnumerationType_t eventType, const asmJSONData_t &authenticatorInfo) = 0;
  }

  /*!
   * Initializes ASM plugin. This is the first function to be called.
   */
```
FIDO UAF Authenticator-specific Module API

```c
/*
 * @param  pEnumerationListener caller provided Enumerator
 */
ASM_FUNC asmResult_t asmInit(ASM::IEnumerator *pEnumerationListener);

/**
 * Process given JSON request and returns JSON response.
 * If the caller wants to execute a function defined in ASM JSON schema - this is the function that must be called.
 * @param  pInData input JSON data
 * @param  pListener event listener for receiving events from ASM
 */
ASM_FUNC asmResult_t asmProcess(const asmJSONData_t *pInData, ASM::ICallback *pListener);

/**
 * Unitializes ASM plugin.
 */
ASM_FUNC asmResult_t asmUninit();
```

A Windows-based FIDO Client looks for ASM DLLs in the following registry paths:

- `HKCU\Software\FIDO\UAF\ASM`
- `HKLM\Software\FIDO\UAF\ASM`

The FIDO 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 ASM DLL.
588 5 Security and Privacy Guidelines

589 ASM developers must carefully protect UAF data they are working with. The following
590 are security requirements ASMs must follow:

591 • ASM MUST implement a mechanism for isolating UAF credentials registered by
592 two different FIDO Clients from one another. One FIDO Client must not have ac-
593 cess to UAF credentials that have been registered via a different FIDO Client.
594 This prevents an application pretending to be a FIDO Client from exercising or
595 acquiring the credentials associated with a legitimate FIDO Client.
596 ○ ASMs must do their best to protect their sensitive data against malware
597 using platform provided isolation capabilities. Malware with root access to
598 the system or direct physical attack on the device are out of scope.

599 The following are examples for achieving this.

600 ○ If an ASM is bundled with a FIDO Client - this isolation mechanism is al-
601 ready built-in.
602 ○ If the ASM and FIDO Client are implemented by the same vendor - the
603 vendor may implement proprietary mechanisms to bind its ASM only with
604 its own FIDO Client.
605 ○ On some platforms, the ASM and the FIDO Client may be assigned with a
606 special privilege or permissions which regular applications don’t have.
607 ASMs built for such platforms may avoid supporting isolation of UAF cre-
608 dentials per FIDO Clients since all FIDO Clients will be considered equally
609 trusted.

610 • ASM designed specifically for UAF Bound Authenticators must ensure that UAF
611 credentials registered with one ASM cannot be accessed by another ASM. This
612 is to prevent an application pretending to be an ASM from exercising legitimate
613 UAF credentials.

614 ○ The KHAccessKeyId mechanism described in 5.1 is one such mecha-
615 nism.

616 • ASM must implement platform provided security best practices for protecting
617 UAF related stored data.
FIDO UAF Authenticator-specific Module API

- An ASM must not store any UAF sensitive data other than the following data in its local storage:
  - CallerID, ASMToken, PersonaID, KeyID, KeyHandle, AppID

An ASM, for example, must never store FIDO Server provided username in its local storage in a plaintext form. This ensures that a minimum of sensitive information is stored outside of the authenticator’s security boundary.

- 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, an ASM designed to support Roaming Silent Authenticators must 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.

5.1 KHAccessToken

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

- **AppID** is provided by FIDO Server and is contained within every UAF message.
- **PersonaID** is obtained by ASM from operational environment. Typically a different PersonaID is assigned to every user account.
- **ASMToken** is a random secret generated secret which is maintained and protected by ASM. In a typical implementation ASM will randomly generate an ASMToken when it’s launch the first time and will maintain this secret until it’s uninstalled.
- **CallerID** is the the calling FIDO Client’s platform assigned ID (e.g. bundle ID for iOS). On different platforms the caller ID can be obtained differently. For example on Android platform ASM can use the hash of caller’s apk-signing-cert.
The ASM uses KHAccessToken to establish a link between ASM and the KeyHandle that is created by Authenticator on behalf of this ASM.

ASM provides the KHAccessToken to the Authenticator with every command which works with KeyHandles.

The following describes how ASM constructs and uses KHAccessToken.

- During Register request
  - Append AppID
    - KHAccessToken = AppID
  - If it's a Bound Authenticator, append also ASMToken, PersonalID and CallerID
    - KHAccessToken |= ASMToken | PersonalID | CallerID
  - Hash KHAccessToken
    - Hash KHAccessToken using the authenticator's hashing algorithm.
    - The reason of using Authenticator specific hash function is to make sure of interoperability between ASMs. If interoperability is not required – ASM can use any other secure hash function it wants.
    - KHAccessToken=hash(KHAccessToken)
  - Provide KHAccessToken to Authenticator
  - Authenticator puts the KHAccessToken into RawKeyHandle (see [UAFAuthnCommands] for more details)

- During other commands which require KHAccessToken as input argument
  - ASM calculates KHAccessToken the same way as during Register command and provides it to Authenticator along with other arguments
  - Authenticator unwraps provided KeyHandle(s) and proceeds with the command only if RawKeyHandle.KHAccessToken equals to provided KHAccessToken

**Normative Notes:**

Bound Authenticators MUST support a mechanism for binding generated KeyHandles with ASMs. The mechanism MUST have at least the same security characteristics as KHAccessToken described above.
It is RECOMMENDED that for Roaming Authenticators the KHAccessToken contains only AppID since otherwise users won’t be able to use them on different machines (PersonaID, ASMToken and CallerID are platform specific). However if Authenticator vendor decides to do that to address a specific use case - they MAY do it.

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

5.2 Access Control for ASM APIs

ASMs may implement various mechanisms to guard the access to the various APIs. This is API access control.

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

Terms used in the table:

- NoAuth – no access control
- CallerID – FIDO Client's platform assigned ID
- UserVerify – explicit user verification
- KeyIDList – must be known to the caller
### FIDO UAF Authenticator-specific Module API

<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>ManageSettings*</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 (provided by FC) CallerID PersonaID</td>
<td>UserVerify AppID</td>
<td>UserVerify AppID KeyIDList (provided by FC)</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 CallerID</td>
<td>AppID KeyID PersonaID CallerID</td>
<td>AppID KeyID</td>
<td>AppID KeyID</td>
</tr>
<tr>
<td>Commit</td>
<td>AppID KeyID CallerID PersonaID CallerID</td>
<td>AppID KeyID PersonaID CallerID</td>
<td>AppID KeyID</td>
<td>AppID KeyID</td>
</tr>
</tbody>
</table>

Table 1: Access Control for ASM API

* Note that commands marked with asterisk are only exposed to FIDO Client and not to RP Apps

**Normative Note:**

ASMs MUST implement the access control requirements defined above.

ASM vendors MAY implement additional security mechanisms to those defined in this document.
6 Bibliography

FIDO Alliance Documents:


Other References:


[FIPS 186-4] NIST DIGITAL SIGNATURE STANDARD (DSS) (FIPS 186-4), National Institute of Standards and Technology, July 2013

[RFC2119] Key words for use in RFCs to Indicate Requirement Levels (RFC2119), S. Bradner, March 1997


[RFC4648] The Base16, Base32, and Base64 Data Encodings (RFC 4648), S. Josefsson, October 2006

[SmartCard] Global Platform Card Specifications (Specifications)
FIDO UAF Authenticator-specific Module API


[TEE] Global Platform Trusted Execution Environment Specifications (Specifications)

[TEESecureDisplay] Trusted User Interface API Specification (Specifications)

[TPM] TPM Main Specification (TPM Specifications)