FIDO UAF Registry of Predefined Values

FIDO Alliance Review Draft 28 November 2017

This version:

Previous version:

Editor:
Dr. Rolf Lindemann, Nok Nok Labs, Inc.

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

Copyright © 2013-2017 FIDO Alliance All Rights Reserved.

Abstract

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

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current FIDO Alliance publications and the latest revision of this technical report can be found in the FIDO Alliance specifications index at https://www.fidoalliance.org/specifications/.

This document was published by the FIDO Alliance as a Review Draft. This document is intended to become a FIDO Alliance Proposed Standard. If you wish to make comments regarding this document, please Contact Us. All comments are welcome.

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

Implementation of certain elements of this Specification may require licenses under third party intellectual property rights, including without limitation, patent rights. The FIDO Alliance, Inc. and its Members and any other contributors to the Specification are not, and shall not be held, responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

THIS FIDO ALLIANCE SPECIFICATION IS PROVIDED “AS IS” AND WITHOUT ANY WARRANTY OF ANY KIND, INCLUDING, WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTY OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents

1. Notation
   1.1 Key Words
2. Overview
3. Authenticator Characteristics
   3.1 Assertion Schemes
4. Predefined Tags
   4.1 Tags used in the protocol
5. Predefined Extensions
   5.1 User Verification Method Extension
   5.2 User ID Extension
   5.3 Android SafetyNet Extension
   5.4 Android Key Attestation
   5.5 User Verification Caching
      • 5.5.1 UVC Request
      • 5.5.2 UVC Response
      • 5.5.3 Privacy Considerations
      • 5.5.4 Security Considerations
6. Other Identifiers specific to FIDO UAF
   6.1 FIDO UAF Application Identifier (AID)
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.
UAF specific terminology used in this document is defined in [FIDOGlossary].
All diagrams, examples, notes in this specification are non-normative.

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

2. Overview
This section is non-normative.
This document defines the registry of UAF-specific constants that are used and referenced in various UAF specifications. It is expected that, over time, new constants will be added to this registry. For example new authentication algorithms and new types of authenticator characteristics will require new constants to be defined for use within the specifications.
FIDO-specific constants that are common to multiple protocol families are defined in [FIDORegistry].

3. Authenticator Characteristics
This section is normative.

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

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

4. Predefined Tags
This section is normative.
The internal structure of UAF authenticator commands is a "Tag-Length-Value" (TLV) sequence. The tag is a 2-byte unique unsigned value describing the type of field the data represents, the length is a 2-byte unsigned value indicating the size of the value in bytes, and the value is the variable-sized series of bytes which contain data for this item in the sequence.
Although 2 bytes are allotted for the tag, only the first 14 bits (values up to 0x3FFF) should be used to accommodate the limitations of some hardware platforms.
A tag that has the 14th bit (0x2000) set indicates that it is critical and a receiver must abort processing the entire message if it cannot process that tag.
A tag that has the 13th bit (0x1000) set indicates a composite tag that can be parsed by recursive descent.

4.1 Tags used in the protocol
The following tags have been allocated for data types in UAF protocol messages:

TAG_UAFV1_REG_ASSERTION 0x3E01
The content of this tag is the authenticator response to a Register command.
TAG_UAFV1_AUTH_ASSERTION 0x3E02
The content of this tag is the authenticator response to a Sign command.
TAG_UAFV1_KRD 0x3E03
Indicates Key Registration Data.
TAG_UAFV1_SIGNED_DATA 0x3E04
Indicates data signed by the authenticator using UAuth.priv key.
TAG_ATTESTATION_CERT 0x2E05
Indicates DER encoded attestation certificate.
TAG_SIGNATURE 0x2E06
Indicates a cryptographic signature.
TAG_ATTESTATION_BASIC_FULL 0x3E07
Indicates full basic attestation as defined in [JAFProtocol].
TAG_ATTESTATION_BASIC_SURROGATE 0x3E08
Indicates surrogate basic attestation as defined in [JAFProtocol].
TAG_ATTESTATION_ECDAA 0x3E09
Indicates use of elliptic curve based direct anonymous attestation as defined in [FIDOEcdaaAlgorithm]. Support for this attestation type is optional at this time. It might be required by FIDO Certification.
TAG_KEYID 0x2E09
Represents a generated KeyID.
TAG_FINAL_CHALLENGE_HASH 0x2E0A
Represents a generated final challenge hash as defined in [JAFProtocol].
TAG_ATTESTATION_INFO 0x2E0B
Represents an Authenticator Attestation ID as defined in [JAFProtocol].
TAG_COUNTERS 0x2E0D
Represents a generated public key.
TAG_EXTENSION 0x2E0F
Represents the use counters for an authenticator.
TAG_ASSERTION_INFO 0x2E0F
Represents authenticator information necessary for message processing.
TAG_TRANSACTION_CONTENT_HASH 0x2E10
Represents a hash of the transaction content sent to the authenticator.
TAG_EXTENSION 0x3E11, 0x3E12
This is a composite tag indicating that the content is an extension.
TAG_EXTENSION_ID 0x2E13
Represents extension ID. Content of this tag is a UINT8[] encoding of a UTF-8 string.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

TAG_USER_VERIFICATION_CACHING 0x0108
This extension allows an app to specify such user verification caching time, i.e. the time for which the user verification status can be "cached" by the authenticator.

The value of this extension is defined as follows:

<table>
<thead>
<tr>
<th>TLV Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 UINT16 Tag</td>
<td>TAG_USER_VERIFICATION_CACHING</td>
</tr>
<tr>
<td>1.1 UINT16 Length</td>
<td>Length of UVC structure in bytes</td>
</tr>
<tr>
<td>1.2 UINT16 maxUVC in seconds</td>
<td></td>
</tr>
<tr>
<td>1.3 UINT8 (optional) verifyIfExceeded. If 0(=false): return error if maxUVC exceeded. If non-zero (=true): verify user if maxUVC exceeded.</td>
<td></td>
</tr>
</tbody>
</table>

Example of the TLV encoded UVC extension (contained in an authentication request)

```
08 01                       -- TAG_USER_VERIFICATION_CACHING (0x0108)
05                          -- length of UVC
2c 01 00 00 -- the UVC value itself: maxUVC = 0x012c (300 secs),
01                          -- followed by verifyIfExceeded = 1 (true)
```

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

5. Predefined Extensions
This section is normative.

5.1 User Verification Method Extension
This extension can be added:

- by FIDO Servers to the UAF Request object (request extension) in the `OperationHeader` in order to ask the authenticator for using a specific user verification method and confirm that in the related response extension.
- by FIDO Clients to the ASM Request object (request extension) in order to ask the authenticator for using a specific user verification method and confirm that in the related response extension.
- by ASMs to the authenticator command (request extension) in order to ask the authenticator for using a specific user verification method and confirm that in the related response extension.
- by Authenticators to the assertion generated in response to a request in order to confirm a specific user verification method that was used for the action.

**Extension identifier**

`fido.uaf.uvm`

**When present in a request (request extension)**

Same as described in Authenticator argument.

**FIDO Client processing**

The client `should` pass the (request) extension through to the Authenticator.

**Authenticator argument**

The payload of this extension is an array of:

```
UINT32 userVerificationMethod
```

The array can have multiple entries. Each entry `shall` have a single bit flag set. In this case the authenticator `shall` verify the user using all (multiple) methods as indicated.

The semantics of the fields are as follows:

- `userVerificationMethod`:
  The authentication method used by the authenticator to verify the user. Available values are defined in [FIDORegistry], “User Verification Methods” section.

**Authenticator processing**

The authenticator supporting this extension

1. `should` limit the user verification methods selectable by the user to the user verification method(s) specified in the request extension.
2. `shall` truthfully report the selected user verification method(s) back in the related response extension added to the assertion.

**Authenticator data**

The payload of this extension is an array of the following structure:

```
UINT32 userVerificationMethod
UINT16 keyProtection
UINT16 matcherProtection
```

The array can have multiple entries describing all user verification methods used.

The semantics of the fields are as follows:

- `userVerificationMethod`:
  The authentication method used by the authenticator to verify the user. Available values are defined in [FIDORegistry], “User Verification Methods” section.
- `keyProtection`:
  The method used by the authenticator to protect the FIDO registration private key material. Available values are defined in [FIDORegistry], “Key Protection Types” section. This value has no meaning in the request extension.
- `matcherProtection`:
  The method used by the authenticator to protect the matcher that performs user verification. Available values are defined in [FIDORegistry], “Matcher Protection Types” section.

**Server processing**

If the FIDO Server requested the UVM extension,

1. `it should` verify that a proper response is provided (if client side support can be assumed), and
2. `it should` verify that the UVM response extension specifies one or more acceptable user verification method(s).

### 5.2 User ID Extension

This extension can be added:

- by FIDO Servers to the UAF Request object (request extension) in the `OperationHeader`. 
- by FIDO Clients to the ASM Request object (request extension).
- by ASMs to the `TAG_UAFV1_REGISTER_CMD` object using `TAG_EXTENSION` (request extension).
- by Authenticators to the registration or authentication assertion using `TAG_EXTENSION` (response extension).

The main purpose of this extension is to allow relying parties finding the related user record by an existing index (i.e. the user ID). This user ID is not intended to be displayed.

Authenticators `should` truthfully indicate support for this extension in their Metadata Statement.

**Extension identifier**

`fido.uaf.userid`

**Extension fail-if-unknown flag**

False, i.e. this (request and response) extension can safely be ignored by all entities.

**Extension data value**

Content of this tag is the `UINT8[]` encoding of the user ID as UTF-8 string.

### 5.3 Android SafetyNet Extension

This extension can be added
by FIDO Servers to the UAF Request object (request extension) in the OperationHeader in order to trigger generation of the related response extension.

by FIDO Clients to the ASM Request object (request extension) in order to trigger generation of the related response extension.

by the ASM to the respective exts array in the ASMRespone object (response extension).

by the FIDO Client to the respective exts array in either the OperationHeader, or the AuthenticatorRegistrationAssertion, or the AuthenticatorSignAssertion of the UAF Response object (response extension).

**Extension identifier**
`fido.uaf.safetynet`

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

**Extension data value**

When present in a request (request extension)

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

```
EXAMPLE 1: SafetyNet Request Extension

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

When present in a response (response extension)

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

```
EXAMPLE 2: SafetyNet Response Extension - not supported by platform

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

```
EXAMPLE 3: SafetyNet Response Extension - temporarily unavailable

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

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

**FIDO Client processing**

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

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

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

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

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

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

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

Where

- `hashAlg` identifies the hash algorithm according to [FIDOSignatureFormat], section IANA Considerations.
- `fcHash` is the base64url encoded hash value of FinalChallenge (see section 3.6.3 and 3.7.4 in [UAFASM] for details on how to compute finalChallengeHash).

We use this method to bind this SafetyNet extension to the respective FIDO UAF message.

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

**Authenticator argument**

N/A

**Authenticator processing**

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

**Authenticator data**

N/A

**Server processing**

If the FIDO Server requested the SafetyNet extension,
5.4 Android Key Attestation

This extension can be added:

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

**Extension identifier**

`fido.uaf.android.key_attestation`

**Extension fail-if-unknown flag**

Empty string, i.e. this (request and response) extension can safely be ignored by all entities.

**Example 4: Android KeyAttestation Request Extension**

```
"exts": ["fido.uaf.android.key_attestation", "\"\", "false\"]
```

**When present in a request (request extension)**

- Empty string, i.e. the FIDO Server might add this extension to the UAF Request with an empty `data` value in order to trigger the generation of this extension for the UAF Response.

**Example 5: Retrieve KeyAttestation and add it as extension**

```
EXAMPLE 5: Retrieve KeyAttestation and add it as extension

Calendar notBefore = Calendar.getInstance();
Calendar notAfter = Calendar.getInstance();
...
for (Certificate cert : certarray) {
    String certArray[] = new String[certarray.length];
    certArray[i] = new String(Base64.encode(buf, Base64.DEFAULT))
    byte[] buf = cert.getEncoded();
    int i=0;
    for (Certificate cert : certarray) {
        byte[] buf = cert.getEncoded();
        certArray[i] = new String(Base64.encode(buf, Base64.DEFAULT));
        i++;
    }
    JSONArray jarray = new JSONArray(certArray);
    JSONObject obj = jarray.getJSONObject(i);
    String key_attestation_data = jarray.toString();
    JSONArray jarray = new JSONArray(certArray);
    for (Certificate cert : certarray) {
        String certArray[] = new String[certarray.length];
        certArray[i] = new String(Base64.encode(buf, Base64.DEFAULT));
        byte[] buf = cert.getEncoded();
        int i=0;
        for (Certificate cert : certarray) {
            byte[] buf = cert.getEncoded();
            certArray[i] = new String(Base64.encode(buf, Base64.DEFAULT));
            i++;
        }
        JSONArray jarray = new JSONArray(certArray);
        JSONObject obj = jarray.getJSONObject(i);
        String key_attestation_data = jarray.toString();
        kpGenerator.generateKeyPair(); // generate Uauth key pair
        Certificate[] certarray = myKeyStore.getCertificateChain(keyUUID);
        String certArray[] = new String[certarray.length];
        certArray[i] = new String(Base64.encode(buf, Base64.DEFAULT));
        byte[] buf = cert.getEncoded();
        int i=0;
        for (Certificate cert : certarray) {
            byte[] buf = cert.getEncoded();
            certArray[i] = new String(Base64.encode(buf, Base64.DEFAULT));
            i++;
        }
        JSONArray jarray = new JSONArray(certArray);
        JSONObject obj = jarray.getJSONObject(i);
        String key_attestation_data = jarray.toString();
        kpGenerator.generateKeyPair(); // generate Uauth key pair
        Certificate[] certarray = myKeyStore.getCertificateChain(keyUUID);
        String certArray[] = new String[certarray.length];
        certArray[i] = new String(Base64.encode(buf, Base64.DEFAULT));
        byte[] buf = cert.getEncoded();
        int i=0;
        for (Certificate cert : certarray) {
            byte[] buf = cert.getEncoded();
            certArray[i] = new String(Base64.encode(buf, Base64.DEFAULT));
            i++;
        }
        JSONArray jarray = new JSONArray(certArray);
        JSONObject obj = jarray.getJSONObject(i);
        String key_attestation_data = jarray.toString();
        kpGenerator.generateKeyPair(); // generate Uauth key pair
        Certificate[] certarray = myKeyStore.getCertificateChain(keyUUID);
        String certArray[] = new String[certarray.length];
        certArray[i] = new String(Base64.encode(buf, Base64.DEFAULT));
        byte[] buf = cert.getEncoded();
        int i=0;
        for (Certificate cert : certarray) {
            byte[] buf = cert.getEncoded();
            certArray[i] = new String(Base64.encode(buf, Base64.DEFAULT));
            i++;
        }
        JSONArray jarray = new JSONArray(certArray);
        JSONObject obj = jarray.getJSONObject(i);
        String key_attestation_data = jarray.toString();
```

**When present in a response (response extension)**

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

**Example 6: Example of successful key attestation extension response**

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

**NOTE**

The package name in AttestationResponse might relate to either the FIDO Client or the ASM.
NOTE
Line-breaks have been added for readability.

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

EXAMPLE 7: KeyAttestation Response Extension - not supported by platform

data = "p", fail_if_unknown = false

EXAMPLE 8: KeyAttestation Response Extension - temporarily unavailable

data = "a", fail_if_unknown = false

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

FIDO Client processing

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

If the ASM supports this extension it must call the KeyStore API (see above) and add the response as extension to the ASM Response object. Copy the extension in the ASM Response to the UFAP Response, then process the UFAP Response according to section 3.4.6.4 step 4 in [UAPProtcol].

More details on Android key attestation can be found at:

- https://developer.android.com/training/articles/keystorage-rsa
- https://developer.android.com/preview/api-overview.html#key_attestation
- https://source.android.com/security/keystore
- https://source.android.com/security/keystore/implmenter-ref.html

Authenticator argument

N/A

Authenticator processing

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

Authenticator data

N/A

Server processing

If the FIDO Server requested the key attestation extension, it must execute the following steps:

1. It must follow the registration response processing rules (see FIDO UAF Protocol, section 3.4.6.5) before processing this extension.
2. It must verify the syntax of the key attestation extension and if it performs RFC5280 compliant chain validation of the entries in the array to one attestRootCertificate specified in the Metadata Statement - accepting that the keyCertSign bit in the key usage extension of the certificate issuing the leaf certificate is NOT set (which is a deviation from RFC5280).
3. It must determine the leaf certificate from that chain, and it must perform the following checks on this leaf certificate:
   1. Verify that KeyDescription.attestationChallenge == FCHash (see FIDO UAF Protocol, section 3.4.6.5 Step 6). It is recommended to use HMacSHA256. If the key is a private key, use HMacSHA256 with the private key issued from the FIDO UAF protocol.
   2. Verify that keyProtection.KEY_PROTECTION_TEE then refer to KeyDescription.teeEnforced using "authzList".
   3. Verify that KeyDescription.attestationChallenge == FCHash (see FIDO UAF Protocol, section 3.4.6.5 Step 6). It is recommended to use HMacSHA256. If the key is a private key, use HMacSHA256 with the private key issued from the FIDO UAF protocol.
   4. Verify that keyProtection.KEY_PROTECTION_SOFTWare then refer to KeyDescription.teeEnforced using "authzList".

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

In several cases it is good enough for the relying party to know whether the user was verified by the authenticator "some time" ago. This extension allows an app to specify such user verification caching time, i.e. the time for which the user verification status can be "cached" by the authenticator.

For example: Do not ask the user for a fresh user verification to authorize a payment of 4€ if the user was verified by this authenticator within the past 300 seconds.

This extension also allows the authenticator to bridge the gap between a "silent" authenticator, i.e. an authenticator never verifying the user and a "traditional" authenticator, i.e. an authenticator always asking for fresh user verification.

We formally define one extension for the request and a separate extension for the response as the request extension can be safely ignored, but the response extension cannot.

Authenticator supporting this extension must truthfully specify both, the UVC Request and UVC Response extension in the supportedExtensions list of the related Metadata Statement [FIDOMetadataStatement]. The TAG of the UVC Response extension must be specified in that list.

5.5.1 UVC Request

This extension can be added by FIDO Servers to the UAF Request object (extension request) in the OperationHeader in order to trigger generation of the related response extension.

Extension Identifier

fido.uaf.uvc-req

Extension fail-if-unknown flag

false, i.e. the request extension can safely be ignored by all entities.

UVC Extension data value

A (base64-url-encoded) TLV object as defined in the description of [TAG_USER_VERIFICATION_CACHING]. In the UVC Extension provided through the DOM API [IFAPPAPIAndTransport], the field verifyItExceeded may NOT be present. The FIDO Client may add the field verifyItExceeded in order to improve processing.

FIDO Client processing

- In a registration request: Simple pass-through to the platform preferred authenticator.
- In a sign request: Simple pass-through to an authenticator which would never require fresh user verification and still meets all other authentication selection criteria (if such authenticator exists). If this is not possible, then use the preferred authenticator (as normal) but pass-through this extension.

Authenticator argument

Same TLV object as defined in "Extension data value", but as binary object included in the Registration / Authentication command.

Authenticator processing

In a registration request:

The Authenticator must always freshly verify the user and create a key marked with the maximum user verification caching time as specified (referred to as regMaxUVC), i.e. in signAssertion the acceptable maximum user verification time can never exceed this value.

The field (verifyItExceeded) is not allowed in a registration request.

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

```json
"supportedExtensions": [ {
    "id": "fido.uaf.android.key_attestation",
    "data": "{ "attestationRootCertificates": [
        { "id": "fido.uaf.android.key_attestation", "value": "true" }
    ] }",
    "isSupported": true
}]
```
In a sign request:
If the authenticator supports specifying user verification caching time in a sign request:
1. compute \( \text{maxUVC} = \min(\text{maxUVC}, \text{regMaxUVC}) \)
2. compute \( \text{elapsedTime} \), i.e. the time since last user verification.
3. If \( (\text{elapsedTime} > \text{maxUVC}) \) AND \( \text{verifyIfExceeded} = \text{false} \) then return error
4. If \( (\text{elapsedTime} > \text{maxUVC}) \) AND \( (\text{verifyIfExceeded} = \text{true}) \) OR \( (\text{verifyIfExceeded} \text{ is NOT PRESENT}) \) then verify user
5. If \( (\text{elapsedTime} < \text{maxUVC}) \) then sign the assertion as normal
6. Add the UVC Response extension to the assertion.

If the authenticator does not support specifying user verification caching time in a sign request, this extension will be ignored by the authenticator. This will be detected by the server since no extension output will be generated by the authenticator.

<table>
<thead>
<tr>
<th>Authenticator data</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server processing</td>
<td>N/A</td>
</tr>
</tbody>
</table>

5.5.2 UVC Response
This extension can be added by the Authenticator to the AuthenticatorRegistrationAssertion, or the AuthenticatorSignAssertion of the UAF Response object (response extension).

**Extension Identifier**
fido.uaf.uvc-resp (TAG_USER_VERIFICATION_CACHING)

**Extension fail-if-unknown flag**
I.e. the response extension (included in the UAF assertion) may NOT be ignored if unknown. If the server is not prepared to process the UVC response extension, it must fail.

**Extension data value**
N/A

<table>
<thead>
<tr>
<th>FIDO Client processing</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authenticator argument</td>
<td>N/A</td>
</tr>
<tr>
<td>Authenticator processing</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Authenticator data**
If the extension is supported and the request extension was received and evaluated during the respective call, then the binary TLV object as described in the description of TAG_USER_VERIFICATION_CACHING will be included in the assertion generated by the Authenticator.

Where the field \( \text{maxUVC} \) contains an upper bound of \( \text{trueUVC} \) and where the field \( \text{verifyIfExceeded} \) will not be present.

The upper bound value is to be computed as follows:
1. Compute the elapsed seconds since last user verification (=trueUVC).
2. Compute some upper bound of trueUVC, must not exceed \( \min(\text{command.maxUVC}, \text{regMaxUVC}) \).

Where \( \text{command.maxUVC} \) refers to the maxUVC value of the related UVC Request.

Where \( \text{regMaxUVC} \) is the maxUVC value specified in the related registration call (see above) or 0 if no such value was provided at registration time:

For example, use \( \min(\text{maxUVC}, \text{createMaxUVC}) \) or \( \min(\text{round trueUVC to 5 seconds}, \text{maxUVC}, \text{createMaxUVC}) \).

**Server processing**
If the FIDO Server requested the UVC extension,
1. Verify that the Metadata Statement related to this Authenticator indicates support for this extension in the field \( \text{supportedExtensions} \).
2. Verify that assertion.maxUVC is less or equal to request.maxUVC, fail if it isn't.
3. Verify that assertion.maxUVC is acceptable, fail if it isn't.

If the FIDO Server did not request the UVC extension (but encounters it in the response) or if the server doesn't understand the UVC response extension, it must fail.

5.5.3 Privacy Considerations
Using the UVC Request extension with \( \text{verifyIfExceeded} \) set to \( \text{true} \) might allow the caller to triage the last time the user was verified without requiring any input from the user and without notifying the user. We do not allow this field to be set through the DOM API (i.e. by web pages). However, native applications can use this field and hence could be able to determine the last time the user was verified. Native applications have substantially more permissions and hence can have more detailed knowledge about the user's behavior than web pages (e.g. track whether the device is used by evaluating accelerometers).

In the UVC Response extension the Authenticator can provide an upper bound of the \( \text{trueUVC} \) value in order to prevent disclosure of exact time of user verification.

5.5.4 Security Considerations
FIDO Servers not expecting user verification being used, might expect a fresh user verification and an explicit user consent being provided. Authenticators supporting this extension shall only use it when they are asked for that (i.e. UVC Request extension is present). Additionally the authenticator must indicate if the user was not freshly verified using the UVC Response extension. This response extension is marked with "fail-if-unknown" set to true, to make sure that servers receiving this extension know that the user might not have been freshly verified.

6. Other Identifiers specific to FIDO UAF

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

The FIDO UAF AID consists of the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>RID</th>
<th>AC</th>
<th>AX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0xA000000647</td>
<td>0xAF</td>
<td>0x0001</td>
</tr>
</tbody>
</table>
A. References

A.1 Normative references

[FIDOEcdaaAlgorithm]

[FIDO Glossary]

[FIDOMetadataStatement]

[FIDORegistry]
R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Registry of Predefined Values. Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-registry-v1.2-rd-20171128.html

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

[RFC2119]

A.2 Informative references

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

[ITU-X690-2008]

[RFC4648]

[RFC5280]

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

[UAFAppAPIAndTransport]

[UAFProtocol]