FIDO UAF Application API and Transport Binding Specification

FIDO Alliance Review Draft 28 November 2017

This version:
https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-uaf-client-api-transport-v1.2-rd-20171128.html
Previous version:
https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-client-api-transport-v1.1-id-20170202.html
Editor:
Dr. Rolf Lindemann, Nok Nok Labs, Inc.
Contributors:
Brad Hill, PayPal, Inc.
Davit Baghdasaryan, Nok Nok Labs, Inc.
Bill Blanke, Nok Nok Labs, Inc.
Jeff Hodges, PayPal, Inc.
Ka Yang, Nok Nok Labs, Inc.

Abstract

Describes APIs and an interoperability profile for client applications to utilize FIDO UAF. This includes methods of communicating with a FIDO UAF Client for both Web platform and Android applications, transport requirements, and an HTTPS interoperability profile for sending FIDO UAF messages to a compatible server.

Status of This Document

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

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

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

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

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

Table of Contents

1. Notation

Type names, attribute names and element names are written as`code`.

String literals are enclosed in “”, e.g. “UAF-TLV”.

In formulas we use “|” to denote byte wise concatenation operations.

The notation base64url refers to "Base 64 Encoding with URL and Filename Safe Alphabet" [RFC4648] *without padding*.

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

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.

The FIDO UAF technology replaces traditional username and password-based authentication solutions for online services, with a stronger and simpler alternative. The core UAF protocol consists of four conceptual conversations between a FIDO UAF Client and FIDO Server: Registration, Authentication, Transaction Confirmation, and Deregistration. As specified in the core protocol, these messages do not have a defined network transport, or describe how application software that a user interfaces with can use UAF. This document describes the API surface that a client application can use to communicate with FIDO UAF Client software, and transport patterns and security requirements for delivering UAF Protocol messages to a remote server.

The reader should also be familiar with the FIDO Glossary of Terms [FIDOGlossary] and the UAF Protocol specification [UAFProtocol].

2.1 Audience

This document is of interest to client-side application authors that wish to utilize FIDO UAF, as well as implementers of web browsers, browser plugins and FIDO clients, in that it describes the API surface they need to expose to application authors.

2.2 Scope

This document describes:

- The local ECMAScript [ECMA-262] API exposed by a FIDO UAF-enabled web browser to client-side web applications.
- The mechanisms and APIs for Android [ANDROID] applications to discover and utilize a shared FIDO UAF Client service.
- The general security requirements for applications initiating and transporting UAF protocol exchanges.
- An interoperability profile for transporting FIDO UAF messages over HTTPS [RFC2818].

The following are out of scope for this document:

- The format and details of the underlying UAF Protocol messages
- APIs for, and any details of interactions between FIDO Server software and the server-side application stack.

NOTE

The goal of describing standard APIs and an interoperability profile for the transport of FIDO UAF messages here is to provide an example of how to develop a FIDO-enabled application and to promote the ease of integrating interoperable layers from different vendors to build a complete FIDO UAF solution. For any given application instance, these particular patterns may not be ideal and are not mandatory. Applications may use alternate transports, bundle UAF Protocol messages with other network data, or discover and utilize alternative APIs as they see fit.

2.3 Architecture

The overall architecture of 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:

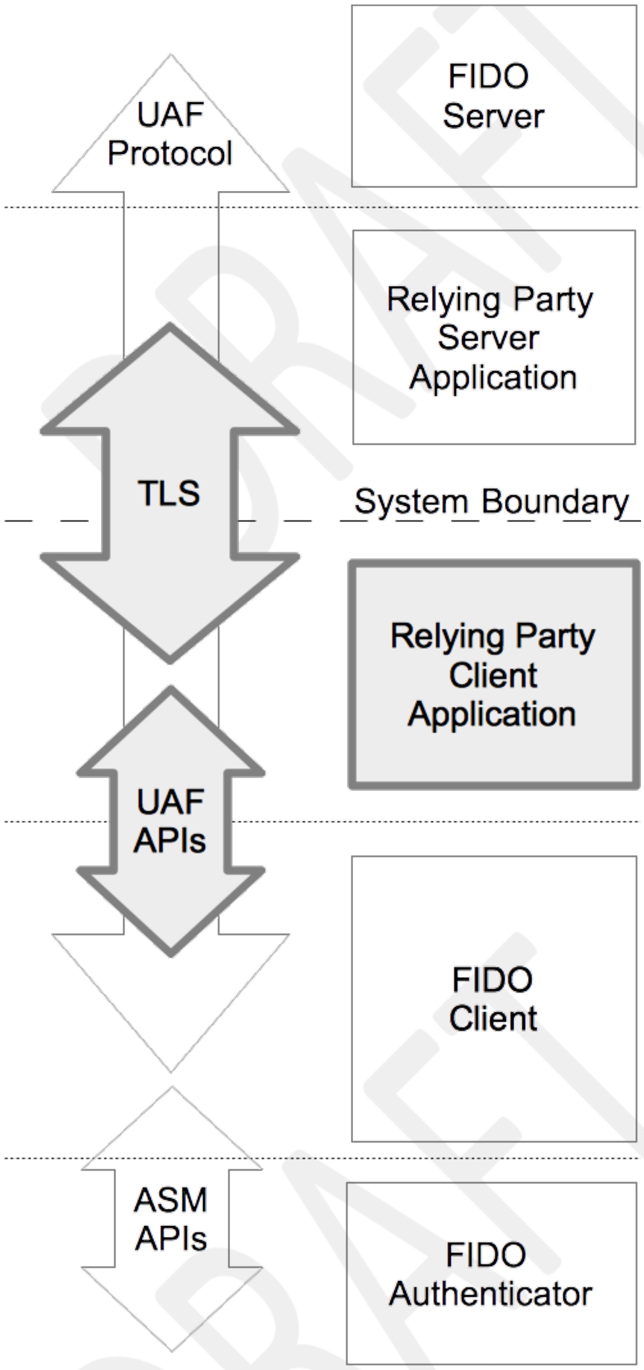

Fig. 1 UAF Application API Architecture and Transport Layers

This document describes the shaded components in Fig 1.

2.3.1 Protocol Conversation

The core UAF protocol consists of five conceptual phases:

- **Discovery** allows the relying party server to determine the availability of FIDO capabilities at the client, including metadata about the available authenticators.
- **Registration** allows the client to generate and associate new key material with an account at the relying party server, subject to policy set by the server and acceptable attestation that the authenticator and registration matches that policy.
- **Authentication** allows a user to provide an account identifier, proof-of-possession of previously registered key material associated with that identifier, and potentially other attested data, to the relying party server.
- **Transaction Confirmation** allows a server to request that a FIDO client and authenticator with the appropriate capabilities display some information to the user, request that the user authenticate locally to their FIDO authenticator to confirm it, and provide proof-of-possession of previously registered key material and an attestation of the confirmation back to the relying party server.
- **Deregistration** allows a relying party server to tell an authenticator to forget selected locally managed key material associated with that relying party in case such keys are no longer considered valid by the relying party.

Discovery does not involve a protocol exchange with the FIDO Server. However, the information available through the discovery APIs might be communicated back to the server in an application-specific manner, such as by obtaining a UAF protocol request message containing an authenticator policy tailored to the specific capabilities of the FIDO user device.

Although the UAF protocol abstractly defines the FIDO server as the initiator of requests, UAF client applications working as described in this document will always transport UAF protocol messages over a client-initiated request/response protocol such as HTTP.

The protocol flow from the point of view of the relying party client application for registration, authentication, and transaction confirmation is as follows:

1. The client application either explicitly contacts the server to obtain a UAF Protocol Request Message, or this message is delivered along with other client application content.

2. The client application invokes the appropriate API to pass the UAF protocol request message asynchronously to the FIDO UAF Client, and receives a set of callbacks.
3. The FIDO UAF Client performs any necessary interactions with the user and authenticator(s) to complete the request and uses a callback to either notify the client application of an error, or to return a UAF response message.
4. The client application delivers the UAF response message to the server over a transport protocol such as HTTP.
5. The server optionally returns an indication of the results of the operation and additional data such as authorization tokens or a redirect.
6. The client application optionally uses the appropriate API to inform the FIDO UAF Client of the results of the operation. This allows the FIDO UAF Client to perform "housekeeping" tasks for a better user experience, e.g. by not attempting to use again later a key that the server refused to register.
7. The client application optionally processes additional data returned to it in an application-specific manner, e.g. processing new authorization tokens, redirecting the user to a new resource or interpreting an error code to determine if and how it should retry a failed operation.

Deregister does not involve a UAF protocol round-trip. If the relying party server instructs the client application to perform a deregistration, the client application simply delivers the UAF protocol Request message to the FIDO UAF Client using the appropriate API. The FIDO UAF Client does not return the results of a deregister operation to the relying party client application or FIDO Server.

UAF protocol Messages are JSON [ECMA-404] structures, but client applications are discouraged from modifying them. These messages may contain embedded cryptographic integrity protections and any modifications might invalidate the messages from the point of view of the FIDO UAF Client or Server.

3. Common Definitions

This section is normative.

These elements are shared by several APIs and layers.

3.1 UAF Status Codes

This table lists UAF protocol status codes.

> NOTE
> These codes indicate the result of the UAF operation at the FIDO Server. They do not represent the HTTP [RFC7230] layer or other transport layers. These codes are intended for consumption by both the client-side web app and FIDO UAF Client to inform application-specific error reporting, retry and housekeeping behavior.

Code	Meaning
1200	OK. Operation completed
1202	Accepted. Message accepted, but not completed at this time. The RP may need time to process the attestation, run risk scoring, etc. The server should not send an authenticationToken with a 1202 response
1400	Bad Request. The server did not understand the message
1401	Unauthorized. The userid must be authenticated to perform this operation, or this KeyID is not associated with this UserID.
1403	Forbidden. The userid is not allowed to perform this operation. Client should not retry
1404	Not Found.
1408	Request Timeout.
1480	Unknown AAID. The server was unable to locate authoritative metadata for the AAID.
1481	Unknown KeyID. The server was unable to locate a registration for the given UserID and KeyID combination. This error indicates that there is an invalid registration on the user's device. It is recommended that FIDO UAF Client deletes the key from local device when this error is received.
1490	Channel Binding Refused. The server refused to service the request due to a missing or mismatched channel binding(s).
1491	Request Invalid. The server refused to service the request because the request message nonce was unknown, expired or the server has previously serviced a message with the same nonce and user ID.
1492	Unacceptable Authenticator. The authenticator is not acceptable according to the server's policy, for example because the capability registry used by the server reported different capabilities than client-side discovery.
1493	Revoked Authenticator. The authenticator is considered revoked by the server.
1494	Unacceptable Key. The key used is unacceptable. Perhaps it is on a list of known weak keys or uses insecure parameter choices.
1495	Unacceptable Algorithm. The server believes the authenticator to be capable of using a stronger mutually-agreeable algorithm than was presented in the request.
1496	Unacceptable Attestation. The attestation(s) provided were not accepted by the server.
1497	Unacceptable Client Capabilities. The server was unable or unwilling to use required capabilities provided supplementally to the authenticator by the client software.
1498	Unacceptable Content. There was a problem with the contents of the message and the server was unwilling or unable to process it.
1500	Internal Server Error

4. Shared Definitions

This section is normative.

> NOTE
>
> This section defines a number of JSON structures, specified with WebIDL [WebIDL-ED]. These structures are shared among APIs for multiple target platforms.

4.1 UAFMessage Dictionary

The UAFMessage dictionary is a wrapper object that contains the raw UAF protocol Message and additional JSON data that may be used to carry application-specific data for use by either the client application or FIDO UAF Client.

WebIDL

```
dictionary UAFMessage {
    required DOMString uafProtocolMessage;
    Object             additionalData;
};
```

4.1.1 Dictionary UAFMessage Members

uafProtocolMessage of type required DOMString
This key contains the UAF protocol Message that will be processed by the FIDO UAF Client or Server. Modification by the client application may invalidate the message. A client application may examine the contents of a message, for example, to determine if a message is still fresh. Details of the structure of the message can be found in the UAF protocol Specification [UAFProtocol].

additionalData of type Object
This key allows the FIDO Server or client application to attach additional data for use by the FIDO UAF Client as a JSON object, or the FIDO UAF Client or client application to attach additional data for use by the client application.

4.2 Version interface

Describes a version of the UAF protocol or FIDO UAF Client for compatibility checking.

WebIDL

```
interface Version {
    readonly    attribute unsigned short major;
    readonly    attribute unsigned short minor;
};
```

4.2.1 Attributes

major of type unsigned short, readonly
Major version number.

minor of type unsigned short, readonly
Minor version number.

4.3 Authenticator interface

Used by several phases of UAF, the Authenticator interface exposes a subset of both verified metadata [FIDOMetadataStatement] and transient information about the state of an available authenticator.

WebIDL

```
interface Authenticator {
    readonly    attribute DOMString                             title;
    readonly    attribute AAID                                  aaid;
    readonly    attribute DOMString                             description;
    readonly    attribute Version[]                             supportedUAFVersions;
    readonly    attribute DOMString                             assertionScheme;
    readonly    attribute unsigned short                        authenticationAlgorithm;
    readonly    attribute unsigned short[]                      attestationTypes;
    readonly    attribute unsigned long                         userVerification;
    readonly    attribute unsigned short                        keyProtection;
    readonly    attribute unsigned short                        matcherProtection;
    readonly    attribute unsigned long                         attachmentHint;
    readonly    attribute boolean                               isSecondFactorOnly;
    readonly    attribute unsigned short                        tcDisplay;
    readonly    attribute DOMString                             tcDisplayContentType;
    readonly    attribute DisplayPNGCharacteristicsDescriptor[] tcDisplayPNGCharacteristics;
    readonly    attribute DOMString                             icon;
    readonly    attribute DOMString[]                           supportedExtensionIDs;
};
```

4.3.1 Attributes

title of type DOMString, readonly
A short, user-friendly name for the authenticator.

> NOTE
>
> This text must be localized for current locale.
>
> If the ASM doesn't return a title in the AuthenticatorInfo object [UAFASM], the FIDO UAF Client must generate a title based on the other fields in AuthenticatorInfo, because title must not be empty (see section 1. Notation).

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

description of type DOMString, readonly
A user-friendly description string for the authenticator.

> NOTE
>
> This text must be localized for current locale.
>
> It is intended to be displayed to the user. It might deviate from the description specified in the authenticator's metadata statement [FIDOMetadataStatement].
>
> If the ASM doesn't return a description in the AuthenticatorInfo object [UAFASM], the FIDO UAF Client must generate a meaningful description to the calling App based on the other fields in AuthenticatorInfo, because description must not be empty (see section 1. Notation).

supportedUAFVersions of type array of *Version*, readonly
Indicates the UAF protocol Versions supported by the authenticator.

assertionScheme of type DOMString, readonly

The assertion scheme the authenticator uses for attested data and signatures.

Assertion scheme identifiers are defined in the UAF Registry of Predefined Values. [UAFRegistry]

authenticationAlgorithm of type unsigned short, readonly
Supported Authentication Algorithm. The value must be related to constants with prefix `ALG_SIGN`.

attestationTypes of type array ofunsigned short, readonly
A list of supported attestation types. The values are defined in [UAFRegistry] by the constants with the prefix `TAG_ATTESTATION`.

userVerification of type unsigned long, readonly
A set of bit flags indicating the user verification methods supported by the authenticator. The values are defined by the constants with the prefix `USER_VERIFY`.

keyProtection of type unsigned short, readonly
A set of bit flags indicating the key protection used by the authenticator. The values are defined by the constants with the prefix `KEY_PROTECTION`.

matcherProtection of type unsigned short, readonly
A set of bit flags indicating the matcher protection used by the authenticator. The values are defined by the constants with the prefix `MATCHER_PROTECTION`.

attachmentHint of type unsigned long, readonly
A set of bit flags indicating how the authenticator is *currently* connected to the FIDO User Device. The values are defined by the constants with the prefix `ATTACHMENT_HINT`.

> NOTE
>
> Because the connection state and topology of an authenticator may be transient, these values are only hints that can be used in applying server-supplied policy to guide the user experience. This can be used to, for example, 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 upon 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 boolean, readonly
Indicates whether the authenticator can only be used as a second-factor.

tcDisplay of type unsigned short, readonly
A set of bit flags indicating the availability and type of transaction confirmation display. The values are defined by the constants with the prefix `TRANSACTION_CONFIRMATION_DISPLAY`.

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

tcDisplayContentType of type DOMString, readonly
The MIME content-type [RFC2045] supported by the transaction confirmation display, such as `text/plain` or `image/png`.

This value must be non-empty if transaction confirmation is supported (`tcDisplay` is non-zero).

tcDisplayPNGCharacteristics of type array ofDisplayPNGCharacteristicsDescriptor, readonly
The set of PNG characteristics *currently* supported by the transaction confirmation display (if any).

> NOTE
>
> See [FIDOMetadataStatement] for additional information on the format of this field and the definition of the `DisplayPNGCharacteristicsDescriptor` structure.

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

icon of type DOMString, readonly
A PNG [PNG] icon for the authenticator, encoded as a`data:` url [RFC2397].

> NOTE
>
> If the ASM doesn't return an icon in the`AuthenticatorInfo` object [UAFASM], the FIDO UAF Client must set a default icon, because `icon` must not be empty (see section 1. Notation).

supportedExtensionIDs of type array ofDOMString, readonly
A list of supported UAF protocol extension identifiers. These may be vendor-specific.

4.3.2 Authenticator Interface Constants

A number of constants are defined for use with the bit flag fields `userVerification`, `keyProtection`, `attachmentHint`, and `tcDisplay`. To avoid duplication and inconsistencies, these are defined in the FIDO Registry of Predefined Values [FIDORegistry].

4.4 DiscoveryData dictionary

WebIDL

```
dictionary DiscoveryData {
    required Version[]       supportedUAFVersions;
    required DOMString       clientVendor;
    required Version         clientVersion;
    required Authenticator[] availableAuthenticators;
};
```

4.4.1 Dictionary DiscoveryData Members

supportedUAFVersions of type array of required Version
: A list of the FIDO UAF protocol versions supported by the client, most-preferred first.

clientVendor of type required DOMString
: The vendor of the FIDO UAF Client.

clientVersion of type required Version
: The version of the FIDO UAF Client. This is a vendor-specific version for the client software, not a UAF version.

availableAuthenticators of type array of required Authenticator
: An array containing Authenticator dictionaries describing the available UAF authenticators. The order is not significant. The list may be empty.

4.5 ErrorCode interface

WebIDL

```
interface ErrorCode {
    const short NO_ERROR = 0x0;
    const short WAIT_USER_ACTION = 0x01;
    const short INSECURE_TRANSPORT = 0x02;
    const short USER_CANCELLED = 0x03;
    const short UNSUPPORTED_VERSION = 0x04;
    const short NO_SUITABLE_AUTHENTICATOR = 0x05;
    const short PROTOCOL_ERROR = 0x06;
    const short UNTRUSTED_FACET_ID = 0x07;
    const short KEY_DISAPPEARED_PERMANENTLY = 0x09;
    const short AUTHENTICATOR_ACCESS_DENIED = 0x0c;
    const short INVALID_TRANSACTION_CONTENT = 0x0d;
    const short USER_NOT_RESPONSIVE = 0x0e;
    const short INSUFFICIENT_AUTHENTICATOR_RESOURCES = 0x0f;
    const short USER_LOCKOUT = 0x10;
    const short USER_NOT_ENROLLED = 0x11;
    const short UNKNOWN = 0xFF;
};
```

4.5.1 Constants

NO_ERROR of type short
: The operation completed with no error condition encountered. Upon receipt of this code, an application should no longer expect an associated `UAFResponseCallback` to fire.

WAIT_USER_ACTION of type short
: Waiting on user action to proceed. For example, selecting an authenticator in the FIDO client user interface, performing user verification, or completing an enrollment step with an authenticator.

INSECURE_TRANSPORT of type short
: `window.location.protocol` is not "https" or the DOM contains insecure mixed content.

USER_CANCELLED of type short
: The user declined any necessary part of the interaction to complete the registration.

UNSUPPORTED_VERSION of type short
: The `UAFMessage` does not specify a protocol version supported by this FIDO UAF Client.

NO_SUITABLE_AUTHENTICATOR of type short
: No authenticator matching the authenticator policy specified in the `UAFMessage` is available to service the request, or the user declined to consent to the use of a suitable authenticator.

PROTOCOL_ERROR of type short
: A violation of the UAF protocol occurred. The interaction may have timed out; the origin associated with the message may not match the origin of the calling DOM context, or the protocol message may be malformed or tampered with.

UNTRUSTED_FACET_ID of type short
: The client declined to process the operation because the caller's calculated facet identifier was not found in the trusted list for the application identifier specified in the request message.

KEY_DISAPPEARED_PERMANENTLY of type short
: The UAuth key disappeared from the authenticator and cannot be restored.

> NOTE
>
> The RP App might want to re-register the authenticator in this case.

AUTHENTICATOR_ACCESS_DENIED of type short
: The authenticator denied access to the resulting request.

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

> NOTE
>
> The transaction content format requirements are specified in the authenticator's metadata statement.

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.

INSUFFICIENT_AUTHENTICATOR_RESOURCES of type short
: Insufficient resources in the authenticator to perform the requested task.

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. For example, an authenticator could allow the user to enter an alternative password to re-enable the use of fingerprints after too many failed finger verification attempts. This error will be reported if such method either doesn't exist or the ASM / authenticator cannot automatically trigger it.

`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.

`UNKNOWN` of type short
An error condition not described by the above-listed codes.

5. DOM API

This section is normative.

This section describes the API details exposed by a web browser or browser plugin to a client-side web application executing in a `Document` [DOM] context.

5.1 Feature Detection

FIDO's UAF DOM APIs are rooted in a new `fido` object, a property of `window.navigator` code; the existence and properties of which may be used for feature detection.

EXAMPLE 1

```
<script>

if(!!window.navigator.fido.uaf) { var useUAF = true; }

</script>
```

5.2 uaf Interface

The `window.navigator.fido.uaf` interface is the primary means of interacting with the FIDO UAF Client. All operations are asynchronous.

WebIDL

```
interface uaf {
    void discover (DiscoveryCallback completionCallback, ErrorCallback errorCallback);
    void checkPolicy (UAFMessage message, ErrorCallback cb);
    void processUAFOperation (UAFMessage message, UAFResponseCallback completionCallback, ErrorCallback errorCallback);
    void notifyUAFResult (int responseCode, UAFMessage uafResponse);
};
```

5.2.1 Methods

`discover`

Discover if the user's client software and devices support UAF and if authenticator capabilities are available that it may be willing to accept for authentication.

Parameter	Type	Nullable	Optional	Description
completionCallback	**`DiscoveryCallback`**	✘	✘	The callback that receives **`DiscoveryData`** from the FIDO UAF Client.
errorCallback	**`ErrorCallback`**	✘	✘	A callback function to receive error and progress events.

Return type: `void`

`checkPolicy`

Ask the browser or browser plugin if it would be able to process the supplied request message without prompting the user.

Unlike other operations using an **`ErrorCallback`**, this operation must always trigger the callback and return `NO_ERROR` if it believes that the message can be processed and a suitable authenticator matching the embedded policy is available, or the appropriate **`ErrorCode`** value otherwise.

NOTE

Because this call should not prompt the user, it should not incur a potentially disrupting context-switch even if the FIDO UAF Client is implemented out-of-process.

Parameter	Type	Nullable	Optional	Description
message	**`UAFMessage`**	✘	✘	A **`UAFMessage`** containing the policy and operation to be tested.
cb	**`ErrorCallback`**	✘	✘	The callback function which receives the status of the operation.

Return type: `void`

`processUAFOperation`
Invokes the FIDO UAF Client, transferring control to prompt the user as necessary to complete the operation, and returns to the callback a message in one of the supported protocol versions indicated by the UAFMessage.

Parameter	Type	Nullable	Optional	Description
message	**`UAFMessage`**	✘	✘	The **`UAFMessage`** to be used by the FIDO client software.
completionCallback	**`UAFResponseCallback`**	✘	✘	The callback that receives the client response **`UAFMessage`** from the FIDO UAF Client, to be delivered to the relying party server.
errorCallback	**`ErrorCallback`**	✘	✘	A callback function to receive error and progress events from the FIDO UAF Client.

Return type: `void`

`notifyUAFResult`

Used to indicate the status code resulting from a FIDO UAF message delivered to the remote server. Applications must make this call when they receive a UAF status code from a server. This allows the FIDO UAF Client to perform housekeeping for a better user experience, for example not attempting to use keys that a server refused to register.

NOTE

If, and how, a status code is delivered by the server, is application and transport specific. A non-normative example can be found below in the HTTPS Transport Interoperability Profile.

Parameter	Type	Nullable	Optional	Description
responseCode	int	✘	✘	The uafResult field of a ServerResponse.
uafResponse	UAFMessage	✘	✘	The UAFMessage to which this responseCode applies.

Return type: void

5.3 UAFResponseCallback

A UAFResponseCallback is used upon successful completion of an asynchronous operation by the FIDO UAF Client to return the protocol response message to the client application for transport to the server.

NOTE

This callback is also called in the case of deregistration completion, even though the response object is empty then.

WebIDL

```
callback UAFResponseCallback = void (UAFMessage uafResponse);
```

5.3.1 Callback UAFResponseCallback Parameters

uafResponse of type UAFMessage
The message and any additional data representing the FIDO UAF Client's response to the server's request message.

5.4 DiscoveryCallback

A DiscoveryCallback is used upon successful completion of an asynchronous discover operation by the FIDO UAF Client to return the DiscoveryData to the client application.

WebIDL

```
callback DiscoveryCallback = void (DiscoveryData data);
```

5.4.1 Callback DiscoveryCallback Parameters

data of type DiscoveryData
Describes the current state of FIDO UAF client software and authenticators available to the application.

5.5 ErrorCallback

An ErrorCallback is used to return progress and error codes from asynchronous operations performed by the FIDO UAF Client.

WebIDL

```
callback ErrorCallback = void (ErrorCode code);
```

5.5.1 Callback ErrorCallback Parameters

code of type ErrorCode
A value from the ErrorCode interface indicating the result of the operation.

For certain operations, an ErrorCallback may be called multiple times, for example with the WAIT_USER_ACTION code.

5.6 Privacy Considerations for the DOM API

This section is non-normative.

Differences in the FIDO capabilities on a user device may (among many other characteristics) allow a server to "fingerprint" a remote client and attempt to persistently identify it, even in the absence of any explicit session state maintenance mechanism. Although it may contribute some amount of signal to servers attempting to fingerprint clients, the attributes exposed by the Discovery API are designed to have a large anonymity set size and should present little or no qualitatively new privacy risk. Nonetheless, an unusual configuration of FIDO Authenticators may be sufficient to uniquely identify a user.

It is recommended that user agents expose the Discovery API to all applications without requiring explicit user consent by default, but user agents or FIDO Client implementers should provide users with the means to opt-out of discovery if they wish to do so for privacy reasons.

5.7 Security Considerations for the DOM API

This section is non-normative.

5.7.1 Insecure Mixed Content

When FIDO UAF APIs are called and operations are performed in a Document context in a web user agent, such a contextmust not contain insecure mixed content. The exact definition insecure mixed content is specific to each user agent, but generally includes any script, plugins and other "active" content, forming part of or with access to the DOM, that was not itself loaded over HTTPS.

The UAF APIs must immediately trigger the ErrorCallback with the INSECURE_TRANSPORT code and cease any further processing if any APIs defined in this document are invoked by a Document context that was not loaded over a secure transport and/or which contains insecure mixed content.

5.7.2 The Same Origin Policy, HTTP Redirects and Cross-Origin Content

When retrieving or transporting UAF protocol messages over HTTP, it is important to maintain consistency among the web origin of the document context and the origin embedded in the UAF protocol message. Mismatches may cause the protocol to fail or enable attacks against the protocol. Therefore:

FIDO UAF messages should not be transported using methods that opt-out of the Same Origin Policy [SOP], for example, using `<script src="url">` to non-same-origin URLs or by setting the `Access-Control-Allow-Origin` header at the server.

When transporting FIDO UAF messages using XMLHttpRequest [XHR] the client should not follow redirects that are to URLs with a different origin than the requesting document.

FIDO UAF messages should not be exposed in HTTP responses where the entire response body parses as valid ECMAScript. Resources exposed in this manner may be subject to unauthorized interactions by hostile applications hosted at untrusted origins through cross-origin embedding using `<script src="url">`.

Web applications should not share FIDO UAF messages across origins through channels such as `postMessage()` [webmessaging].

5.8 Implementation Notes for Browser/Plugin Authors

This section is non-normative.

Web applications utilizing UAF depend on services from the web browser as a trusted platform. The APIs for web applications do not provide a means to assert an origin as an application identity for the purposes of FIDO operations as this will be provided to the FIDO UAF Client by the browser based on its privileged understanding of the actual origin context.

The browser must enforce that the web origin communicated to the FIDO UAF Client as the application identity is accurate

The browser must also enforce that resource instances containing insecure mixed-content cannot utilize the UAF DOM APIs.

6. Android Intent API

This section is normative.

This section describes how an Android [ANDROID] client application can locate and communicate with a conforming FIDO Client installation operating on the host device.

> NOTE
>
> As with web applications, a variety of integration patterns are possible on the Android platform. The API described here allows an app to communicate with a shared FIDO UAF Client on the user device in a loosely-coupled fashion using Android *Intents*.

6.1 Android-specific Definitions

6.1.1 org.fidoalliance.uaf.permissions.FIDO_CLIENT

FIDO UAF Clients running on Android versions prior to Android 5must declare the `org.fidoalliance.uaf.permissions.FIDO_CLIENT` permission and they also must declare the related "uses-permission". See the below example of this permission expressed in an Android app manifest file `<permission/>` and `<uses-permission/>` element [AndroidAppManifest].

FIDO UAF Clients running on Android version 5 or latermust not declare this permission and they also must not declare the related "uses-permission".

EXAMPLE 2

```
<permission
    android:name="org.fidoalliance.uaf.permissions.FIDO_CLIENT"
    android:label="Act as a FIDO Client."
    android:description="This application acts as a FIDO Client. It may
       access authentication devices available on the system, create and
       delete FIDO registrations on behalf of other applications."
    android:protectionLevel="dangerous"
/>
<uses-permission android:name="org.fidoalliance.uaf.permissions.FIDO_CLIENT"/>
```

> NOTE
>
> - Since FIDO Clients perform security relevant tasks (e.g. verifying the AppID/FacetID relation and asking for user consent), users should carefully select the FIDO Clients they use. Requiring apps acting as FIDO Clients to declare and use this permission allows them to be identified as such to users.
> - There are not any FIDO Client resources needing "protection" based upon the FIDO_CLIENT permission. The reason for having FIDO Client declare the FIDO_CLIENT permission is solely that users should be able to carefully decide which FIDO Clients to install.
> - Android version 5 changed the way it handles the case where multiple apps declare the same permission [Android5Changes]; it blocks the installation of all subsequent apps declaring that permission.
> - *The best way to flag the fact that an app may act as a FIDO Client needs to be determined for Android version 5.*

6.1.2 org.fidoalliance.uaf.permissions.ACT_AS_WEB_BROWSER

Android applications requesting services from the FIDO UAF Client can do so under their own identity, or they can act as the user's agent by explicitly declaring an RFC6454 [RFC6454] serialization of the remote server's origin when invoking the FIDO UAF Client.

An application that is operating on behalf of a single entity must not set an explicit origin. Omitting an explicit origin will cause the FIDO UAF Client to determine the caller's identity as `android:apk-key-hash:<hash-of-public-key>`. The FIDO UAF Client will then compare this with the list of authorized application facets for the target AppID and proceed if it is listed as trusted.

> NOTE
>
> See the UAF Protocol Specification [UAFProtocol] for more information on application and facet identifiers.

If the application is explicitly intended to operate as the user's agent in the context of an arbitrary number of remote applications (as when implementing a full web browser) it may set its origin to the RFC6454 [RFC6454] Unicode serialization of the remote application's Origin. The application must satisfy the necessary conditions described in Transport Security Requirements for authenticating the remote server before setting the origin.

Use of the origin parameter requires the application to declare the `org.fidoalliance.uaf.permissions.ACT_AS_WEB_BROWSER` permission, and the FIDO UAF Client must verify that the calling application has this permission before processing the operation.

EXAMPLE 3

```
<permission
   android:name="org.fidoalliance.uaf.permissions.ACT_AS_WEB_BROWSER"
   android:label="Act as a browser for FIDO registrations."
   android:description="This application may act as a web browser,
       creating new and accessing existing FIDO registrations for any domain."
   android:protectionLevel="dangerous"
/>
```

6.1.3 channelBindings

This section is non-normative.

In the DOM API, the browser or browser plugin is responsible for supplying any available channel binding information to the FIDO Client, but an Android application, as the direct owner of the transport channel, must provide this information itself.

The `channelBindings` data structure is:

`Map<String,String>`

with the keys as defined for the `ChannelBinding` structure in the UAF Protocol Specification. [UAFProtocol]

The use of channel bindings for TLS helps assure the server that the channel over which UAF protocol messages are transported is the same channel the legitimate client is using and that messages have not been forwarded through a malicious party.

UAF defines support for the `tls-unique` and `tls-server-end-point` bindings from [RFC5929], as well as server certificate and ChannelID [ChannelID] bindings. The client should supply all channel binding information available to it.

Missing or invalid channel binding information may cause a relying party server to reject a transaction.

6.1.4 UAFIntentType enumeration

This enumeration describes the type of operation for the intent implementing the Android API.

> NOTE
>
> UAF uses only a single intent to simplify behavior in the situation even where multiple FIDO clients may be installed. In such a case, the user will be prompted which of the installed FIDO UAF clients should be used to handle an implicit intent.

If the user selected to make different FIDO UAF Clients the default for different intents representing different phases, it could produce inconsistent results or fail to function at all.

If the application workflow requires multiple calls to the client (and it usually does) the application should read the `componentName` from the intent extras it receives from `startActivityForResult()` and pass it to `setComponent()` for subsequent intents to be sure they are explicitly resolved to the same FIDO UAF Client.

WebIDL

```
enum UAFIntentType {
    "DISCOVER",
    "DISCOVER_RESULT",
    "CHECK_POLICY",
    "CHECK_POLICY_RESULT",
    "UAF_OPERATION",
    "UAF_OPERATION_RESULT",
    "UAF_OPERATION_COMPLETION_STATUS"
};
```

Enumeration description	
`DISCOVER`	Discovery
`DISCOVER_RESULT`	Discovery results
`CHECK_POLICY`	Perform a no-op check if a message could be processed.
`CHECK_POLICY_RESULT`	Check Policy results.
`UAF_OPERATION`	Process a Registration, Authentication, Transaction Confirmation or Deregistration message.
`UAF_OPERATION_RESULT`	UAF Operation results.
`UAF_OPERATION_COMPLETION_STATUS`	Inform the FIDO UAF Client of the completion status of a Registration, Authentication, Transaction Confirmation or Deregistration message.

6.2 org.fidoalliance.intent.FIDO_OPERATION Intent

All interactions between a FIDO UAF Client and an application on Android takes place via a single Android intent:

`org.fidoalliance.intent.FIDO_OPERATION`

The specifics of the operation are carried by the MIME media type and various extra data included with the intent.

The operations described in this document are of MIME media type `application/fido.uaf_client+json` and this must be set as the `type` attribute of any intent.

> NOTE
>
> Client applications can discover if a FIDO UAF Client (or several) is available on the system by using `PackageManager.queryIntentActivities(Intent intent, int flags)` with this intent to see if any activities are available.

Extra	Type	Description
`UAFIntentType`	String	One of the **`UAFIntentType`** enumeration values describing the intent.
`discoveryData`	String	**`DiscoveryData`** JSON dictionary.
`componentName`	String	The component name of the responding FIDO UAF Client. It must be serialized using ComponentName.flattenString()

Extra	Type	Description
errorCode	short	ErrorCode value for operation
message	String	UAFMessage request to test or process, depending on UAFIntentType.
origin	String	An RFC6454 Web Origin [RFC6454] string for the request, if the caller has the org.fidoalliance.permissions.ACT_AS_WEB_BROWSER permission.
channelBindings	String	The JSON dictionary of channel bindings for the operation.
responseCode	short	The uafResult field of a ServerResponse.

The following table shows what intent extras are expected, depending on the value of the UAFIntentType extra:

UAFIntentType value	discoveryData	componentName	errorCode	message	origin	channelBindings	responseCode
"DISCOVER"							
"DISCOVER_RESULT"	optional	required	required				
"CHECK_POLICY"				required	optional		
"CHECK_POLICY_RESULT"		required	required				
"UAF_OPERATION"				required	optional	required	
"UAF_OPERATION_RESULT"		required	required	optional			
"UAF_OPERATION_COMPLETION_STATUS"				required			required

6.2.1 UAFIntentType.DISCOVER

This Android intent invokes the FIDO UAF Client to discover the available authenticators and capabilities. The FIDO UAF Client generally will not show a UI associated with the handling of this intent, but immediately return the JSON structure. The calling application cannot depend on this however, as the FIDO UAF Client may show a UI for privacy purposes, allowing the user to choose whether and which authenticators to disclose to the calling application.

This intent must be invoked with `startActivityForResult()`.

6.2.2 UAFIntentType.DISCOVER_RESULT

An intent with this type is returned by the FIDO UAF Client as an argument to `onActivityResult()` in response to receiving an intent of type `DISCOVER`.

If the `resultCode` passed to `onActivityResult()` is `RESULT_OK`, and the intent extra `errorCode` is NO_ERROR, this intent has an extra, `discoveryData`, containing a `String` representation of a **`DiscoveryData`** JSON dictionary with the available authenticators and capabilities.

6.2.3 UAFIntentType.CHECK_POLICY

This intent invokes the FIDO UAF Client to discover if it would be able to process the supplied message without prompting the user. The action handling this intent should not show a UI to the user.

This intent requires the following extras:

- `message`, containing a `String` representation of a **`UAFMessage`** representing the request message to test.
- `origin`, an optional extra that allows a caller with the `org.fidoalliance.uaf.permissions.ACT_AS_WEB_BROWSER` permission to supply an RFC6454 Origin [RFC6454] string to be used instead of the application's own identity.

This intent must be invoked with `startActivityForResult()`.

6.2.4 UAFIntentType.CHECK_POLICY_RESULT

This Android intent is returned by the FIDO UAF Client as an argument to `onActivityResult()` in response to receiving a `CHECK_POLICY` intent.

In addition to the `resultCode` passed to `onActivityResult()`, this intent has an extra, `errorCode`, containing an **`ErrorCode`** value indicating the specific error condition or NO_ERROR if the FIDO UAF Client could process the message.

6.2.5 UAFIntentType.UAF_OPERATION

This Android intent invokes the FIDO UAF Client to process the supplied request message and return a response message ready for delivery to the FIDO UAF Server.

The sender should assume that the FIDO UAF Client will display a user interface allowing the user to handle this intent, for example, prompting the user to complete their verification ceremony.

This intent requires the following extras:

- `message`, containing a `String` representation of a **`UAFMessage`** representing the request message to process.
- `channelBindings`, containing a `String` representation of a JSON dictionary as defined by the `ChannelBinding` structure in the FIDO UAF Protocol Specification [UAFProtocol].
- `origin`, an optional parameter that allows a caller with the `org.fidoalliance.uaf.permissions.ACT_AS_WEB_BROWSER` permission to supply an RFC6454 Origin [RFC6454] string to be used instead of the application's own identity.

This intent must be invoked with `startActivityForResult()`.

6.2.6 UAFIntentType.UAF_OPERATION_RESULT

This intent is returned by the FIDO UAF Client as an argument to `onActivityResult()`, in response to receiving a `UAF_OPERATION` intent.

If the `resultCode` passed to `onActivityResult()` is `RESULT_CANCELLED`, this intent will have an extra, `errorCode` parameter, containing an **`ErrorCode`** value indicating the specific error condition.

If the `resultCode` passed to `onActivityResult()` is `RESULT_OK`, and the `errorCode` is NO_ERROR, this intent has a `message`, containing a `String` representation of a **`UAFMessage`**, being the UAF protocol response message to be delivered to the FIDO Server.

6.2.7 UAFIntentType.UAF_OPERATION_COMPLETION_STATUS

This intent must be delivered to the FIDO UAF Client to indicate the processing status of a FIDO UAF message delivered to the remote server. This is especially important as a new registration may be considered by the client to be in a pending state until it is communicated that the server accepted it.

6.3 Alternate Android AIDL Service UAF Client Implementation

The Android Intent API can also be implemented using Android AIDL services as an alternative transport mechanism to Android Intents. While Android Intents work at the UI layer, Android AIDL services are performed at a lower level. This can ease integration with relying party apps, since UAF requests can be fulfilled without interfering with existing relying party app UI and application lifecycle behavior.

The UAF Android AIDL service needs to be defined in the UAF client manifest. This is done using the `<service>` tag for an Android AIDL service instead of the `<activity>` tag in Android Intents. Just as with Android intents, the manifest definition for the AIDL service uses an intent filter (note `org.fidoalliance.aidl.FIDO_OPERATION` versus `org.fidoalliance.intent.FIDO_OPERATION`) to identify itself as a FIDO UAF client to the relying party app:

EXAMPLE 4

```
<service android:name="foo" >
<intent-filter>
<action android:name="org.fidoalliance.aidl.FIDO_OPERATION" />
<category android:name="android.intent.category.DEFAULT" />
<data android:mimeType="application/fido.uaf_client+json" />
</intent-filter>
</service>
```

Once the relying party app chooses a UAF client from the list discovered by `PackageManager.queryIntentServices()`, the relying party app and the FIDO UAF client share the following AIDL interface to service UAF requests:

EXAMPLE 5

```
package org.fidoalliance.aidl

oneway interface IUAFOperation
{
 void process(in Intent uafRequest, in IUAFResponseListener uafResponseListener);
}
```

NOTE

Android AIDL services use `Binder.getCallingUid()` instead of `Activity.getCallingActivity()` with Android Intents to identify the caller and obtain FacetID information.

For consistency, the Intents for the Android AIDL service are the same as defined in the Android Intent specification in the UAF standard. In `process()`, the `uafRequest` parameter is the Intent that would be passed to `startActivityForResult()`. The `uafResponseListener` parameter is a listener interface that receives the result. The following AIDL defines this interface:

EXAMPLE 6

```
package org.fidoalliance.aidl

interface IUAFResponseListener
{
 void onResult(in Intent uafResponse);
}
```

In the listener, the `uafResponse` parameter is the Intent that would be passed to `onActivityResult`.

6.4 Security Considerations for Android Implementations

This section is non-normative.

Android applications may choose to implement the user-interactive portion of FIDO in at least two ways:

- by authoring an Android Activity using Android-native user interface components, or
- with an HTML-based experience by loading an Android WebView and injecting the UAF DOM APIs with `addJavaScriptInterface()`.

An application that chooses to inject the UAF interface into a WebView must follow all appropriate security considerations that apply to usage of the DOM APIs, *and* those that apply to user agent implementers.

In particular, the content of a WebView into which an API will be injected must be loaded only from trusted local content or over a secure channel as specified in Transport Security Requirements and must not contain insecure mixed-content.

Applications should not declare the `ACT_AS_WEB_BROWSER` permission unless they need to act as the user's agent for an un-predetermined number of third party applications. Where an Android application has an explicit relationship with a relying party application(s), the preferred method of access control is for those applications to list the Android application's identity as a trusted facet. See the UAF Protocol Specification [UAFProtocol] for more information on application and facet identifiers.

To protect against a malicious application registering itself as a FIDO UAF Client, relying party applications can obtain the identity of the responding application, and utilize it in risk management decisions around the authentication or transaction events.

For example, a relying party might maintain a list of application identities known to belong to malware and refuse to accept operations completed with such clients, or a list of application identities of known-good clients that receive preferred risk-scoring.

Relying party applications running on Android versions prior to Android 5must make sure that a FIDO UAF Client has the "uses-permission" for `org.fidoalliance.uaf.permissions.FIDO_CLIENT`. Relying party applications running on Android 5 should not implement this check.

NOTE

Relying party applications should implement the check on Android prior to 5 by using the package manager to verify that the FIDO Client indeed declared the `org.fidoalliance.uaf.permissions.FIDO_CLIENT` permission (see example below). Relying party applications should not use a "uses-permission" for `FIDO_CLIENT`.

EXAMPLE 7

```
boolean checkFIDOClientPermission(String packageName)
                                  throws NameNotFoundException {
  for (String requestedPermission :
         getPackageManager().getPackageInfo(packageName,
             PackageManager.GET_PERMISSIONS).requestedPermissions) {
      if (requestedPermission.matches(
           "org.fidoalliance.uaf.permissions.FIDO_CLIENT"))
        return true;
  }
 return false;
}
```

Relying party applications which use the AIDL service implementation of the UAF Client Intent API must use an explicit intent to bind to the AIDL service. Failing to do so may result in binding to an unexpected and possibly malicious service, because intent filter resolution depends on application installation order and intent filter priority. Android 5.0 and later will throw a `SecurityException` if an implicit intent is used, but earlier versions do not enforce this behavior.

7. iOS Custom URL API

This section is normative.

This section describes how an iOS relying party application can locate and communicate with a conforming FIDO UAF Client installed on the host device.

NOTE

Because of sandboxing and no true multitasking support, the iOS operating system offers very limited ways to do interprocess communication (IPC).

Any IPC solution for a FIDO UAF Client must be able to:

1. Identify the calling app in order to provide FacetID approval.
2. Allow transition to another app without user intervention

Currently the only IPC method on iOS that satisfies both of these requirements is custom URL handlers.

Custom URL handlers use the iOS operating system to handle URL requests from the sender, launch the receiving app, and then pass the request to the receiving app for processing. By enabling custom URL handlers for two different applications, it is possible to achieve bidirectional IPC between them--one custom URL handler to send data from app A to app B and another custom URL handler to send data from app B to app A.

Because iOS has no true multitasking, there must be an app transition to process each request and response. Too many app transitions can negatively affect the user experience, so relying party applications must carefully choose when it is necessary to query the FIDO UAF Client.

7.1 iOS-specific Definitions

7.1.1 X-Callback-URL Transport

When the relying party application communicates with the FIDO UAF Client, it sends a URL with the standard `x-callback-url` format (see x-callback-url.com):

EXAMPLE 8

```
FidoUAFClient1://x-callback-url/[UAFxRequestType]?x-success=[RelyingPartyURL]://x-callback-url/
        [UAFxResponseType]&
        key=[SecretKey]&
        state=[STATE]&
        json=[Base64URLEncodedJSON]
```

- `FidoUAFClient1` is the iOS custom URL scheme used by FIDO UAF Clients. As specified in the `x-callback-url` standard, version information for the transport layer is encoded in the URL scheme itself (in this case, `FidoUAFClient1`). This is so other applications can check for support for the 1.0 version by using the `canOpenURL` call.
- `[UAFxRequestType]` is the type that should be used for request operations, which are described later in this document.
- `[RelyingPartyURL]` is the URL that the relying party app has registered in order to receive the response. According to the `x-callback-url` standard, this is defined using the `x-success` parameter.
- `[UAFxResponseType]` is the type that should be used for response operations, which are described later in this document.
- `[SecretKey]` is a base64url-encoded, without padding, random key generated for each request by the calling application.

  The response from the FIDO UAF Client will be encrypted with this key in order to prevent rogue applications from obtaining information by spoofing the return URL.

- `[STATE]` is data that can be used to match the request with the response.
- Finally `[Base64URLEncodedJSON]` contains the message to be sent to the FIDO UAF Client.

  Items are stored in JSON format and then base64url-encoded without padding.

For FIDO UAF Clients, the custom URL scheme handler entrypoint is the openURL() function:

Objective-C

EXAMPLE 9

```
(BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:(NSString *)sourceApplication annotation:(id)ann
```

SWIFT

EXAMPLE 10

```
func application(_ application: UIApplication, open url: URL, sourceApplication: String?, annotation: Any) -> Bool {
    ...
```

```
}
```

Here, the URL above is received via the url parameter. For security considerations, the sourceApplication parameter contains the iOS bundle ID of the relying party application. This bundle ID must be used to verify the application FacetID.

Conversely, when the FIDO UAF Client responds to the request, it sends the following URL back in standard x-callback-url format:

EXAMPLE 11

```
[RelyingPartyURL]://x-callback-url/
       [UAFxResponseType]?
       state=[STATE]&
       json=[Base64URLEncodedJWE]
```

The parameters in the response are similar to those of the request, except that the [Base64URLEncodedEncryptedJSON] parameter is encrypted with the public key before being base64url-encoded without padding. [STATE] is the same STATE as was sent in the request--it is echoed back to the sender to verify the matched response.

In the relying party application's openURL() handler, the url parameter will be the URL listed above and the sourceApplication parameter will be the iOS bundle ID for the FIDO client application.

7.1.2 Secret Key Generation

A new secret encryption key must be generated by the calling application every time it sends a request to FIDO UAF Client. The FIDO UAF Client must then use this key to encrypt the response message before responding to the caller.

JSON Web Encryption (JWE), JSON Serialization (JWE Section 7.2) format must be used to represent the encrypted response message.

The encryption algorithm is that specified in "A128CBC-HS256" where the JWE "Key Management Mode" employed is "Direct Encryption" and the JWE "Content Encryption Key (CEK)" is the secret key generated by the calling application and passed to the FIDO UAF Client in the key parameter of the request.

EXAMPLE 12

```
{
   "unprotected": {"alg": "dir", "enc": "A128CBC-HS256"},
   "iv": "...",
   "ciphertext": "...",
   "tag": "..."
}
```

7.1.3 Origin

iOS applications requesting services from the FIDO Client can do so under their own identity, or they can act as the user's agent by explicitly declaring an RFC6454 [RFC6454] serialization of the remote server's origin when invoking the FIDO UAF Client.

An application that is operating on behalf of a single entity must not set an explicit origin. Omitting an explicit origin will cause the FIDO UAF Client to determine the caller's identity as "ios:bundle-id". The FIDO UAF Client will then compare this with the list of authorized application facets for the target AppID and proceed if it is listed as trusted.

See the UAF Protocol Specification [UAFProtocol] for more information on application and facet identifiers.

If the application is explicitly intended to operate as the user's agent in the context of an arbitrary number of remote applications (as when implementing a full web browser) it may set origin to the RFC6454 [RFC6454] Unicode serialization of the remote application's Origin. The application must satisfy the necessary conditions described in Transport Security Requirements for authenticating the remote server before setting origin.

7.1.4 channelBindings

This section is non-normative.

In the DOM API, the browser or browser plugin is responsible for supplying any available channel binding information to the FIDO Client, but an iOS application, as the direct owner of the transport channel, must provide this information itself.

The channelBindings data structure is Map<String,String> with the keys as defined for the ChannelBinding structure in the FIDO UAF Protocol Specification. [UAFProtocol]

The use of channel bindings for TLS helps assure the server that the channel over which UAF protocol messages are transported is the same channel the legitimate client is using and that messages have not been forwarded through a malicious party. UAF defines support for the tls-unique and tls-server-end-point bindings from [RFC5929], as well as server certificate and ChannelID [ChannelID] bindings. The client should supply all channel binding information available to it.

Missing or invalid channel binding information may cause a relying party server to reject a transaction.

7.1.5 UAFxType

This value describes the type of operation for the x-callback-url operations implementing the iOS API.

WebIDL

```
enum UAFxType {
    "DISCOVER",
    "DISCOVER_RESULT",
    "CHECK_POLICY",
    "CHECK_POLICY_RESULT",
    "UAF_OPERATION",
    "UAF_OPERATION_RESULT",
    "UAF_OPERATION_COMPLETION_STATUS"
};
```

Enumeration description	
DISCOVER	Discovery
DISCOVER_RESULT	Discovery results
CHECK_POLICY	Perform a no-op check if a message could be processed.
CHECK_POLICY_RESULT	Check Policy results.

UAF_OPERATION	The UAF message operation type (for example Registration).
UAF_OPERATION_RESULT	UAF Operation results.
UAF_OPERATION_COMPLETION_STATUS	Inform the FIDO UAF Client of the completion status of a UAF operation (such as Registration.

7.2 JSON Values

The specifics of the UAFxType operation are carried by various JSON values encoded in the json x-callback-url parameter.

JSON value	Type	Description
discoveryData	String	**DiscoveryData** JSON dictionary.
errorCode	short	**ErrorCode** value for operation
message	String	**UAFMessage** request to test or process, depending on **UAFxType**.
origin	String	An RFC6454 Web Origin [RFC6454] string for the request.
channelBindings	String	The channel bindings JSON dictionary for the operation.
responseCode	short	The uafResult field of a ServerResponse.

The following table shows what JSON values are expected, depending on the value of the UAFxType x-callback-url operation:

UAFxType operation	discoveryData	errorCode	message	origin	channelBindings	responseCode
"DISCOVER"						
"DISCOVER_RESULT"	optional	required				
"CHECK_POLICY"			required	optional		
"CHECK_POLICY_RESULT"		required				
"UAF_OPERATION"			required	optional	required	
"UAF_OPERATION_RESULT"		required	optional			
"UAF_OPERATION_COMPLETION_STATUS"			required			required

7.2.1 DISCOVER

This operation invokes the FIDO UAF Client to discover the available authenticators and capabilities. The FIDO UAF Client generally will not show a user interface associated with the handling of this operation, but will simply return the resulting JSON structure.

The calling application cannot depend on this however, as the client may show a user interface for privacy purposes, allowing the user to choose whether and which authenticators to disclose to the calling application.

> NOTE
>
> iOS custom URL scheme handlers always require an application switch for every request and response, even if no user interface is displayed.

7.2.2 DISCOVER_RESULT

An operation with this type is returned by the FIDO UAF Client in response to receiving an x-callback-url operation of type DISCOVER.

If x-callback-url JSON value errorCode is NO_ERROR, this x-callback-url operation has a JSON value, discoveryData, containing a String representation of a **DiscoveryData** JSON dictionary listing the available authenticators and their capabilities.

7.2.3 CHECK_POLICY

This operation invokes the FIDO UAF Client to discover if the client would be able to process the supplied message, without prompting the user.

The related Action handling this operation should not show an interface to the user.

> NOTE
>
> iOS custom URL scheme handlers always require an application switch for every request and response, even if no UI is displayed.

This x-callback-url operation requires the following JSON values:

- message, containing a String representation of a **UAFMessage** representing the request message to test.
- origin, an optional JSON value that allows a caller to supply an RFC6454 Origin [RFC6454] string to be used instead of the application's own identity.

7.2.4 CHECK_POLICY_RESULT

This operation is returned by the FIDO UAF Client in response to receiving a CHECK_POLICY x-callback-url operation.

The x-callback-url JSON value errorCode containing an **ErrorCode** value indicating the specific error condition or NO_ERROR if the FIDO Client could process the message.

7.2.5 UAF_OPERATION

This operation invokes the FIDO UAF Client to process the supplied request message and return a result message ready for delivery to the FIDO UAF Server. The sender should assume that the FIDO UAF Client will display a UI to the user to handle this x-callback-url operation, e.g. prompting the user to complete their verification ceremony.

This x-callback-url operation requires the following JSON values:

- `message`, containing a `String` representation of a **`UAFMessage`** representing the request message to process.
- `channelBindings`, containing a `String` representation of a JSON dictionary as defined by the `ChannelBinding` structure in the UAF Protocol Specification [UAFProtocol].
- `origin`, an optional JSON value that allows a caller to supply an RFC6454 Origin [RFC6454] string to be used instead of the application's own identity.

7.2.6 UAF_OPERATION_RESULT

This x-callback-url operation is returned by the FIDO UAF Client in response to receiving a `UAF_OPERATION` x-callback-url operation.

The x-callback-url JSON value `errorCode` containing an **`ErrorCode`**value indicating the specific error condition.

If x-callback-url JSON value `errorCode` is NO_ERROR, this x-callback-url operation has a JSON value, `message`, containing a `String` representation of a **`UAFMessage`**, being the UAF protocol response message to be delivered to the FIDO Server.

7.2.7 UAF_OPERATION_COMPLETION_STATUS

This x-callback-url operation must be delivered to the FIDO UAF Client to indicate the completion status of a FIDO UAF message delivered to the remote server. This is especially important as, e.g. a new registration may be considered in a pending status until it is known the server accepted it.

7.3 Implementation Guidelines for iOS Implementations

Each iOS Custom URL based request results in a human-noticeable context switch between the App and FIDO UAF Client and vice versa. This will be most noticeable when invoking DISCOVER and CHECK_POLICY requests since typically these requests will be invoked automatically, without user's involvement. Such a context switch impacts the User Experience and therefore it's recommended to avoid making these two requests and integrate FIDO without using them.

7.4 Security Considerations for iOS Implementations

This section is non-normative.

A security concern with custom URLs under iOS is that any app can register any custom URL. If multiple applications register the same custom URL, the behavior for handling the URL call in iOS is undefined.

On the FIDO UAF Client side, this issue with custom URL scheme handlers is solved by using the `sourceApplication` parameter which provides the bundle ID of the URL originator. This is effective as long as the device has not been jailbroken and as long as Apple has done due diligence vetting submissions to the app store for malware with faked bundle IDs. The `sourceApplication` parameter can be matched with the FacetID list to ensure that the calling app is approved to use the credentials for the relying party.

On the relying party app side, encryption is used to prevent a rogue app from spoofing the relying party app's response URL. The relying party app generates a random encryption key on every request and sends it to the FIDO client. The FIDO client then encrypts the response to this key. In this manner, only the relying party app can decrypt the response. Even in the event that malware is able to spoof the relying party app's URL and intercept the response, it would not be able to decode it.

To protect against potentially malicious applications registering themselves to handle the FIDO UAF Client custom URL scheme, relying party Applications can obtain the bundle-id of the responding app and utilize it in risk management decisions around the authentication or transaction events. For example, a relying party might maintain a list of bundle-ids known to belong to malware and refuse to accept operations completed with such clients, or a list of bundle-ids of known-good clients that receive preferred risk-scoring.

8. Transport Binding Profile

This section is normative.

This section describes general normative security requirements for how a client application transports FIDO UAF protocol messages, gives specific requirements for Transport Layer Security (TLS), and describes an interoperability profile for using HTTP over TLS [RFC2818] with the FIDO UAF protocol.

8.1 Transport Security Requirements

This section is non-normative.

The UAF protocol contains no inherent means of identifying a relying party server, or for end-to-end protection of UAF protocol messages. To perform a secure UAF protocol exchange, the following abstract requirements apply:

1. The client application must securely authenticate the server endpoint as authorized, from that client's viewpoint, to represent the Web origin [RFC6454] (scheme:host:port tuple) reported to the FIDO UAF Client by the client application. Most typically this will be done by using TLS and verifying the server's certificate is valid, asserts the correct DNS name, and chains up to a root trusted by the client platform. Clients may also utilize other means to authenticate a server, such as via a pre-provisioned certificate or key that is distributed with an application, or alternative network authentication protocols such as Kerberos [RFC4120].
2. The transport mechanism for UAF protocol messages must provide confidentiality for the message, to prevent disclosure of their contents to unauthorized third parties. These protections should be cryptographically bound to proof of the server's identity as described above.
3. The transport mechanism for UAF protocol messages must protect the integrity of the message from tampering by unauthorized third parties. These protections should be cryptographically bound to proof of the server's identity in as described above.

8.2 TLS Security Requirements

This section is non-normative.

If using HTTP over TLS ([RFC2246] [RFC4346], [RFC5246] or [TLS13draft02]) to transport an UAF protocol exchange, the following specific requirements apply:

1. If there are any TLS errors, whether "warning" or "fatal" or any other error level with the TLS connection, the HTTP client must terminate the connection without prompting the user. For example, this includes any errors found in certificate validity checking that HTTP clients employ, such as via TLS server identity checking [RFC6125], Certificate Revocation Lists (CRLs) [RFC5280], or via the Online Certificate Status Protocol (OCSP) [RFC2560].
2. Whenever comparisons are made between the presented TLS server identity (as presented during the TLS handshake, typically within the server certificate) and the intended source TLS server identity (e.g., as entered by a user, or embedded in a link), [RFC6125] server identity checking must be employed. The client must terminate the connection without prompting the user upon any error condition.
3. The TLS server certificate must either be provisioned explicitly out-of-band (e.g. packaged with an app as a "pinned certificate") or be trusted by chaining to a root included in the certificate store of the operating system or a major browser by virtue of being currently in compliance with their root store program requirements. The client must terminate the connection without user recourse if there are any error conditions when building the chain of trust.
4. The "anon" and "null" crypto suites are not allowed and insecure cryptographic algorithms in TLS (e.g. MD4, RC4, SHA1) should be avoided (see NIST SP800-131A [SP800-131A]).

5. The client and server should use the latest practicable TLS version.
6. The client should supply, and the server should verify whatever practicable channel binding information is available, including a `ChannelID` [ChannelID] public key, the `tls-unique` and `tls-server-end-point` bindings [RFC5929], and TLS server certificate binding [UAFProtocol]. This information provides protection against certain classes of network attackers and the forwarding of protocol messages, and a server may reject a message that lacks or has channel binding data that does not verify correctly.

8.3 HTTPS Transport Interoperability Profile

This section is normative.

Conforming applications may support this profile.

Complex and highly-optimized applications utilizing UAF will often transport UAF protocol messages in-line with other application protocol messages. The profile defined here for transporting UAF protocol messages over HTTPS is intended to:

- Provide an interoperability profile to enable easier composition of client-side application libraries and server-side implementations for FIDO UAF-enabled products from different vendors.
- Provide detailed illustration of specific necessary security properties for the transport layer and HTTP interfaces, especially as they may interact with a browser-hosted application.
- This profile is also utilized in the examples that constitute the appendices of this document. This profile is optional to implement. RFC 2119 key words are used in this section to indicate necessary security and other properties for implementations that intend to use this profile to interoperate [RFC2119].

> NOTE
>
> Certain FIDO UAF operations, in particular, transaction confirmation, will always require an application-specific implementation. This interoperability profile only provides a skeleton framework suitable for replacing username/password authentication.

8.3.1 Obtaining a UAF Request message

A UAF-enabled web application might typically deliver request messages as part of a response body containing other application content, e.g. in a script block as such:

EXAMPLE 13

```
...

<script type="application/json">
{
"initialRequest": {
// initial request message here
},

"lifetimeMillis": 60000; // hint: this initial request is valid for 60 seconds
}
</script>

...
```

However, request messages have a limited lifetime, and an installed application cannot be delivered with a request, so client applications generally need the ability to retrieve a fresh request.

When sending a request message over HTTPS with XMLHttpRequest [XHR] or another HTTP API:

1. The URI of the server endpoint, and how it is communicated to the client, is application-specific.
2. The client must set the HTTP method to POST. [RFC7231]
3. The client should set the HTTP "Content-Type" header to `"application/fido+uaf; charset=utf-8"`. [RFC7231]
4. The client should include `"application/fido+uaf"` as a media type in the HTTP "Accept" header [RFC7231]. Conforming servers must accept `"application/fido+uaf"` as media type.
5. The client may need to supply additional headers, such as a HTTP Cookie [RFC6265], to demonstrate, in an application-specific manner, their authorization to perform a request.
6. The entire POST body must consist entirely of a JSON [ECMA-404] structure described by the `GetUAFRequest` dictionary.
7. The server's response should set the HTTP "Content-Type" to `"application/fido+uaf; charset=utf-8"`
8. The client should decode the response byte string as UTF-8 with error handling. [HTML5]
9. The decoded body of the response must consist entirely of a JSON structure described by the `ReturnUAFRequest` interface.

8.3.2 Operation enum

Describes the operation type of a FIDO UAF message or request for a message.

WebIDL

```
enum Operation {
    "Reg",
    "Auth",
    "Dereg"
};
```

Enumeration description	
`Reg`	Registration
`Auth`	Authentication or Transaction Confirmation
`Dereg`	Deregistration

8.3.3 GetUAFRequest dictionary

WebIDL

```
dictionary GetUAFRequest {
    Operation op;
```

```
    DOMString previousRequest;
    DOMString context;
};
```

8.3.3.1 Dictionary `GetUAFRequest` Members

`op` of type *Operation*
: The type of the UAF request message desired. Allowable string values are defined by the Operation enum. This field is optional but must be set if the operation is not known to the server through other context, e.g. an operation-specific URL endpoint.

`previousRequest` of type DOMString
: If the application is requesting a new UAF request message because a previous one has expired, this optional key can include the previous one to assist the server in locating any state that should be re-associated with a new request message, should one be issued.

`context` of type DOMString
: Any additional contextual information that may be useful or necessary for the server to generate the correct request message. This key is optional and the format and nature of this data is application-specific.

8.3.4 ReturnUAFRequest dictionary

WebIDL

```
dictionary ReturnUAFRequest {
    required unsigned long statusCode;
    DOMString              uafRequest;
    Operation              op;
    long                   lifetimeMillis;
};
```

8.3.4.1 Dictionary `ReturnUAFRequest` Members

`statusCode` of type required unsigned long
: The UAF Status Code for the operation (see section 3.1 UAF Status Codes).

`uafRequest` of type DOMString
: The new UAF Request Message, optional, if the server decided to issue one.

`op` of type *Operation*
: An optional hint to the client of the operation type of the message, useful if the server might return a different type than was requested. For example, a server might return a deregister message if an authentication request referred to a key it no longer considers valid. Allowable string values are defined by the Operation enum.

`lifetimeMillis` of type long
: If the server returned a `uafRequest`, this is an optional hint informing the client application of the lifetime of the message in milliseconds.

8.3.5 SendUAFResponse dictionary

WebIDL

```
dictionary SendUAFResponse {
    required DOMString uafResponse;
    DOMString          context;
};
```

8.3.5.1 Dictionary `SendUAFResponse` Members

`uafResponse` of type required DOMString
: The UAF Response Message. It must be set to `UAFMessage.uafProtocolMessage` returned by FIDO UAF Client.

`context` of type DOMString
: Any additional contextual information that may be useful or necessary for the server to process the response message. This key is optional and the format and nature of this data is application-specific.

8.3.6 Delivering a UAF Response

Although it is not the only pattern possible, an asynchronous HTTP request is a useful way of delivering a UAF Response to the remote server for either web applications or standalone applications.

When delivering a response message over HTTPS with XMLHttpRequest [XHR] or another API:

1. The URI of the server endpoint and how it is communicated to the client is application-specific.
2. The client must set the HTTP method to POST. [RFC7231]
3. The client must set the HTTP "Content-Type" header to `"application/fido+uaf; charset=utf-8"`. [RFC7231]
4. The client should include `"application/fido+uaf"` as a media type in the HTTP "Accept" header. [RFC7231]
5. The client may need to supply additional headers, such as a HTTP Cookie [RFC6265], to demonstrate, in an application-specific manner, their authorization to perform an operation.
6. The entire POST body must consist entirely of a JSON [ECMA-404] structure described by the `SendUAFResponse`.
7. The server's response should set the "Content-Type" to `"application/fido+uaf; charset=utf-8"` and the body of the response must consist entirely of a JSON structure described by the `ServerResponse` interface.

8.3.7 ServerResponse Interface

The `ServerResponse` interface represents the completion status and additional application-specific additional data that results from successful processing of a Register, Authenticate, or Transaction Confirmation operation. This message is not formally part of the UAF protocol, but the `statusCode` should be posted to the FIDO UAF Client, for housekeeping, using the `notifyUAFResult()` operation.

WebIDL

```
interface ServerResponse {
    readonly    attribute int       statusCode;
    [Optional]
```

```
    readonly    attribute DOMString description;
    [Optional]
    readonly    attribute Token[]  additionalTokens;
    [Optional]
    readonly    attribute DOMString location;
    [Optional]
    readonly    attribute DOMString postData;
    [Optional]
    readonly    attribute DOMString newUAFRequest;
};
```

8.3.7.1 Attributes

`statusCode` of type int, readonly
: The FIDO UAF response status code. Note that this status code describes the result of processing the tunneled UAF operation, not the status code for the outer HTTP transport.

`description` of type DOMString, readonly
: A detailed message describing the status code or providing additional information to the user.

`additionalTokens` of type array of *Token*, readonly
: This key contains new authentication or authorization token(s) for the client that are not natively handled by the HTTP transport. Tokens should be processed prior to processing of `location`.

`location` of type DOMString, readonly
: If present, indicates to the client web application that it should navigate the Document context to the URI contained on this field after processing any tokens.

`postData` of type DOMString, readonly
: If present in combination with `location`, indicates that the client should POST the contents to the specified location after processing any tokens.

`newUAFRequest` of type DOMString, readonly
: The server may use this to return a new UAF protocol message. This might be used to supply a fresh request to retry an operation in response to a transient failure, to request additional confirmation for a transaction, or to send a deregistration message in response to a permanent failure.

8.3.8 Token interface

NOTE

The UAF Server is not responsible for creating additional tokens returned as part of a UAF response. Such tokens exist to provide a means for the relying party application to update the authentication/authorization state of the client in response to a successful UAF operation. For example, these fields could be used to allow UAF to serve as the initial authentication leg of a federation protocol, but the scope and details of any such federation are outside of the scope of UAF.

WebIDL

```
interface Token {
    readonly    attribute TokenType type;
    readonly    attribute DOMString value;
};
```

8.3.8.1 Attributes

`type` of type *TokenType*, readonly
: The type of the additional authentication / authorization token.

`value` of type DOMString, readonly
: The string value of the additional authentication / authorization token.

8.3.9 TokenType enum

WebIDL

```
enum TokenType {
    "HTTP_COOKIE",
    "OAUTH",
    "OAUTH2",
    "SAML1_1",
    "SAML2",
    "JWT",
    "OPENID_CONNECT"
};
```

Enumeration description	
`HTTP_COOKIE`	If the user agent is a standard web browser or other HTTP native client with a cookie store, this TokenType should not be used. Cookies should be set directly with the Set-Cookie HTTP header for processing by the user agent. For non-HTTP or non-browser contexts this indicates a token intended to be set as an HTTP cookie. [RFC6265] For example, a native VPN client that authenticates with UAF might use this TokenType to automatically add a cookie to the browser cookie jar.
`OAUTH`	Indicates that the token is of type OAUTH. [RFC5849].
`OAUTH2`	Indicates that the token is of type OAUTH2. [RFC6749].
`SAML1_1`	Indicates that the token is of type SAML 1.1. [SAML11].
`SAML2`	Indicates that the token is of type SAML 2.0. [SAML2-CORE]
`JWT`	Indicates that the token is of type JSON Web Token (JWT). [JWT]
`OPENID_CONNECT`	Indicates that the token is an OpenID Connect “id_token”. [OpenIDConnect]

8.3.10 Security Considerations

This section is non-normative.

It is important that the client set, and the server require, the method be POST and the “Content-Type” HTTP header be the correct values. Because the response body is valid ECMAScript, to protect against unauthorized cross-origin access, a server must not respond to the type of request that can be generated by a script tag, e.g. `<script src="https://example.com/fido/uaf/getRequest">`. The request a user agent generates with this kind of embedding cannot set custom headers.

Likewise, by requiring a custom “Content-Type” header, cross-origin requests cannot be made with an XMLHttpRequest [XHR] without triggering a CORS preflight access check. [CORS]

As FIDO UAF messages are only valid when used same-origin, servers should not supply an “Access-Control-Allow-Origin” [CORS] header with responses that would allow them to be read by non-same-origin content.

To protect from some classes of cross-origin, browser-based, distributed denial-of-service attacks, request endpoints should ignore, without performing additional processing, all requests with an “Access-Control-Request-Method” [CORS] HTTP header or an incorrect “Content-Type” HTTP header.

If a server chooses to respond to requests made with the GET method and without the custom “Content-Type” header, it should apply a prefix string such as “`while(1);`” or “`&&&BEGIN_UAF_RESPONSE&&&`” to the body of all replies and so prevent their being read through cross-origin <script> tag embedding. Legitimate same-origin callers will need to (and alone be able to) strip this prefix string before parsing the JSON content.

A. References

A.1 Normative references

[AndroidAppManifest]
Android App Manifest. Work in Progress. URL:http://developer.android.com/guide/topics/manifest/manifest-intro.html
[ChannelID]
D. Balfanz. *Transport Layer Security (TLS) Channel IDs*. Work In Progress. URL:http://tools.ietf.org/html/draft-balfanz-tls-channelid
[DOM]
Anne van Kesteren. *DOM Standard*. Living Standard. URL:https://dom.spec.whatwg.org/
[ECMA-262]
ECMAScript Language Specification. URL: https://tc39.github.io/ecma262/
[ECMA-404]
The JSON Data Interchange Format. 1 October 2013. Standard. URL:https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf
[FIDOGlossary]
R. Lindemann; D. Baghdasaryan; B. Hill; J. Hodges. *FIDO Technical Glossary*. Implementation Draft. URL:https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-glossary-v1.2-rd-20171128.html
[FIDOMetadataStatement]
B. Hill; D. Baghdasaryan; J. Kemp. *FIDO Metadata Statements v1.0*. Implementation Draft. URL:https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-metadata-statement-v1.2-rd-20171128.html
[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
[HTML5]
I. Hickson; R.Berjon; S. Faulkner; T. Leithead; E. D. Navara; E. O'Connor; S. Pfeiffer.*HTML5: A vocabulary and associated APIs for HTML and XHTML*. 28 October 2014. W3C Recommendation. URL:http://www.w3.org/TR/html5/
[JWT]
M. Jones; J. Bradley; N. Sakimura. *JSON Web Token (JWT)*. May 2015. RFC. URL:https://tools.ietf.org/html/rfc7519
[OpenIDConnect]
OpenID Connect. Work in Progress. URL:http://openid.net/connect/
[PNG]
Tom Lane. *Portable Network Graphics (PNG) Specification (Second Edition)*. 10 November 2003. W3C Recommendation. URL: https://www.w3.org/TR/PNG/
[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL:https://tools.ietf.org/html/rfc2119
[RFC2397]
L. Masinter. *The "data" URL scheme*. August 1998. Proposed Standard. URL:https://tools.ietf.org/html/rfc2397
[RFC2818]
E. Rescorla. *HTTP Over TLS*. May 2000. Informational. URL:https://tools.ietf.org/html/rfc2818
[RFC4648]
S. Josefsson. *The Base16, Base32, and Base64 Data Encodings (RFC 4648)*. October 2006. URL:http://www.ietf.org/rfc/rfc4648.txt
[RFC5849]
E. Hammer-Lahav. *The OAuth 1.0 Protocol (RFC 5849)*. April 2010. URL:http://www.ietf.org/rfc/rfc5849.txt
[RFC5929]
J. Altman; N. Williams; L. Zhu.*Channel Bindings for TLS (RFC 5929)*. July 2010. URL:http://www.ietf.org/rfc/rfc5929.txt
[RFC6125]
P. Saint-Andre; J. Hodges. *Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS) (RFC 6125)*. March 2011. URL:http://www.ietf.org/rfc/rfc6125.txt
[RFC6265]
A. Barth. *HTTP State Management Mechanism*. April 2011. Proposed Standard. URL:https://tools.ietf.org/html/rfc6265
[RFC6454]
A. Barth. *The Web Origin Concept (RFC 6454)*. June 2011. URL:http://www.ietf.org/rfc/rfc6454.txt
[RFC6749]
D. Hardt, Ed.. *The OAuth 2.0 Authorization Framework (RFC 6749)*. October 2012. URL:http://www.ietf.org/rfc/rfc6749.txt
[RFC7230]
R. Fielding, Ed.; J. Reschke, Ed..*Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7230
[RFC7231]
R. Fielding, Ed.; J. Reschke, Ed..*Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7231
[SAML11]
E. Maler; P. Mishra; R. Philpott.*The Security Assertion Markup Language (SAML) v1.1*. October 2003. URL:https://www.oasis-open.org/standards#samlv1.1
[SAML2-CORE]
Scott Cantor; John Kemp; Rob Philpott; Eve Maler.*Assertions and Protocols for SAML V2.0*. 15 March 2005. URL:http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
[UAFProtocol]
R. Lindemann; D. Baghdasaryan; E. Tiffany; D. Balfanz; B. Hill; J. Hodges.*FIDO UAF Protocol Specification v1.0*. Proposed Standard. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-uaf-protocol-v1.2-rd-20171128.html
[UAFRegistry]
R. Lindemann; D. Baghdasaryan; B. Hill. *FIDO UAF Registry of Predefined Values*. Proposed Standard. URL:https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-uaf-reg-v1.2-rd-20171128.html
[WebIDL-ED]
Cameron McCormack. *Web IDL*. 13 November 2014. Editor's Draft. URL:http://heycam.github.io/webidl/

A.2 Informative references

[ANDROID]
The Android™ Operating System. URL: http://developer.android.com/
[Android5Changes]
Android 5.0 Behavior Changes. Work in progress. URL:http://developer.android.com/about/versions/android-5.0-changes.html

[CORS]
Anne van Kesteren. *Cross-Origin Resource Sharing*. 16 January 2014. W3C Recommendation. URL: https://www.w3.org/TR/cors/
[RFC2045]
N. Freed; N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies* November 1996. Draft Standard. URL: https://tools.ietf.org/html/rfc2045
[RFC2246]
T. Dierks; E. Rescorla. *The TLS Protocol Version 1.0*. January 1999. URL: http://www.ietf.org/rfc/rfc2246.txt
[RFC2560]
M. Myers; R. Ankney; A. Malpani; S. Galperin; C. Adams. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. June 1999. Proposed Standard. URL: https://tools.ietf.org/html/rfc2560
[RFC4120]
C. Neuman; T. Yu; S. Hartman; K. Raeburn. *The Kerberos Network Authentication Protocol (V5) (RFC 4120)*. July 2005. URL: http://www.ietf.org/rfc/rfc4120.txt
[RFC4346]
T. Dierks; E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.1*. April 2006. URL: http://www.ietf.org/rfc/rfc4346.txt
[RFC5246]
T. Dierks; E. Rescorla. *The Transport Layer Security (TLS) Protocol*. August 2008. URL: http://www.ietf.org/rfc/rfc5246.txt
[RFC5280]
D. Cooper; S. Santesson; S. Farrell; S.Boeyen; R. Housley; W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. May 2008. URL: http://www.ietf.org/rfc/rfc5280.txt
[SOP]
Same Origin Policy for JavaScript. January 2014. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Same_origin_policy_for_JavaScript
[SP800-131A]
E. Barker; A. Roginsky. *NIST Special Publication 800-131A: Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*. January 2011. URL: http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf
[TLS13draft02]
T. Dierks; E. Rescorla. *The Transport Layer Security (TLD) Protocol Version 1.3 (draft 02)* July 2014. URL: https://tools.ietf.org/html/draft-ietf-tls-tls13-02
[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
[WebIDL]
Cameron McCormack; Boris Zbarsky; Tobie Langel. *Web IDL*. 15 December 2016. W3C Editor's Draft. URL: https://heycam.github.io/webidl/
[XHR]
Anne van Kesteren. *XMLHttpRequest Standard*. Living Standard. URL: https://xhr.spec.whatwg.org/
[webmessaging]
Ian Hickson. *HTML5 Web Messaging*. 19 May 2015. W3C Recommendation. URL: https://www.w3.org/TR/webmessaging/