This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of

Web Authentication: An API for accessing Public Key Credentials Level 1

W3C Working Draft, 11 August 2017

This version: https://www.w3.org/TR/2017/WD-webauthn-20170811/

Latest published version: https://www.w3.org/TR/webauthn/

Editor's Draft: https://w3c.github.io/webauthn/

Previous Versions:

https://www.w3.org/TR/2016/WD-webauthn-20160902/
https://www.w3.org/TR/2016/WD-webauthn-20161031/

Issue Tracking: Github

Editors:

Vijay Bharadwaj (Microsoft)
Hubert Le Van Gong (PayPal)
Dirk Bañanz (Google)
Aleksi Czeskis (Google)
Arnar Birgisson (Google)
Jeff Hodges (PayPal)
Michael B. Jones (Microsoft)
Rolf Lindemann (Nok Nok Labs)
J.C. Jones (Mozilla)

Tests:

web-platform-tests/webauthn/ (ongoing work)

Copyright 2017 W3C^ (MIT, ERCIM, Keio, Beihang). W3C liability, trademark and document use rules apply.

Abstract

This specification defines an API enabling the creation and use of strong, attested, scoped, public key-based credentials by web applications, for the purpose of strongly authenticating users. Conceptually, one or more public key credentials, each scoped to a given Relying Party, are created and stored on an authenticator by the user agent in conjunction with the web application. The user agent mediates access to public key credentials in order to preserve user privacy. Authenticators are responsible for ensuring that no operation is performed without user consent. Authenticators provide cryptographic proof of their properties to relying parties via attestation. This specification also describes the functional model for WebAuthn conformant authenticators, including their signature and attestation functionality.

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 W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

This document was published by the Web Authentication Working Group as a Working Draft. This document is intended to become a W3C Recommendation. Feedback and comments on this specification are welcome. Please use Github issues. Discussions may also be found in the public-webauthn@w3.org archives.

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 1 March 2017 W3C Process Document.

Table of Contents

1. 1 Introduction

1. 1.1 Use Cases

1. 1.1.1 Registration

1. 1.2 Authentication

1. 1.2.1 Registration

1. 1.2.2 Authentication

1. 1.3 Other use cases and configurations

2. 2 Conformance

2. 2.1 Dependencies

3. 3 Terminology

4. 4 Web Authentication API

4. 4.1 PublicKeyCredential Interface

4. 4.1.1 CredentialCreationOptions Extension

4. 4.1.2 CredentialRequestOptions Extension

4. 4.1.3 Create a new credential - PublicKeyCredential's [[Create]](options) method

4. 4.1.4 Use an existing credential to make an assertion - PublicKeyCredential's [[Get]](options) method

4. 4.1.5 Platform Authenticator Availability - PublicKeyCredential's isPlatformAuthenticatorAvailable() method

4. 4.2 Authenticator Responses (interface AuthenticatorResponse)

4. 4.2.1 Information about PublicKey Credential (interface AuthenticatorAttestationResponse)

4. 4.2.2 Web Authentication Assertion (interface AuthenticatorAssertionRequest)

3. 3 Parameters for Credential Generation (dictionary PublicKeyCredentialParameters)

4. 4.4 Options for Credential Creation (dictionary PublicKeyCredentialOptions)

4. 4.4.1 Public Key Entity Description (dictionary PublicKeyCredentialEntity)

4. 4.4.2 User Account Parameters for Credential Generation (dictionary PublicKeyCredentialOptions)
4. 9.4 Authenticator Selection Extension (authnSel)
This specification defines an API enabling the creation and use of strongly authenticated, platform-based credentials by web applications, for the purpose of strongly authenticating users. A public key credential is created and stored by an authenticator at the behest of a Relying Party, subject to user consent. Subsequently, the public key credential can only be accessed by origins belonging to that Relying Party. This scopability is enforced jointly by the user, the authenticator, and the authenticator’s environment (e.g., Trust Execution Environment (TEE), a Trusted Platform Module (TPM), or a Secure Element (SE)).

This specification is designed for use in browser applications, for the purpose of strongly authenticating users. Applications using this API can be used by any Relying Party for any purpose without the need to have a dedicated API or service. For the purpose of strongly authenticating users, a public key credential is created and stored by an authenticator at the behest of a Relying Party, subject to user consent. Subsequently, the public key credential can only be accessed by origins belonging to that Relying Party. This scopability is enforced jointly by the user, the authenticator, and the authenticator’s environment (e.g., Trust Execution Environment (TEE), a Trusted Platform Module (TPM), or a Secure Element (SE)).

Relying Parties employ the Web Authentication API during two distinct, but related, ceremonies involving a user. The first is Registration, where a public key credential is created on an authenticator, and associated by a Relying Party with the user’s credentials (the account may already exist or may be created at this time). The second is Authentication, where the Relying Party is presented with an Authentication Assertion proving the presence and consent of the user who registered the public key credential. Functionally, the Web Authentication API comprises a PublicKeyCredential which extends the Credential Management API (CREDENTIAL-MANAGEMENT-1), and infrastructure which allows those credentials to be used with a UserVerification method. The former is used during Registration, and the latter during Authentication.

An authenticator might consist of a Trusted Execution Environment (TEE) applet, a Trusted Platform Module (TPM), or a Secure Element (SE).

Interoperability with other means for user verification, along with appropriate platform software to mediate access to these components’ functionality, Other authenticators may operate autonomously from the computing device running the user agent, and be accessed over a transport such as Universal Serial Bus (USB), Bluetooth Low Energy (BLE) or Near Field Communications (NFC).
1.1. Use Cases

The below use case scenarios illustrate use of two very different types of authenticators, as well as outline further scenarios. Additional scenarios, including sample code, are given later in 11 Sample scenarios.

* On a phone:
  + User navigates to example.com in a browser and signs in to an existing account using whatever method they have been using (possibly a legacy method such as a password), or creates a new account.
  + The phone prompts, "Do you want to register this device with example.com?"
  + User agrees.
  + The phone prompts the user for a previously configured authorization gesture (PIN, biometric, etc.), the user provides this.
  + Website shows message, "Registration complete."

* On a laptop or desktop:
  + User navigates to example.com in a browser, sees an option to "Sign in with your phone."
  + User chooses this option and gets a message from the browser, "Please complete this action on your phone."
  + Next, on their phone:
    + User sees a discrete prompt or notification, "Sign in to example.com."
    + User agrees.
    + User is shown a list of their example.com identities, e.g., "Sign in as Alice / Sign in as Bob."
    + User picks an identity, is prompted for an authorization gesture (PIN, biometric, etc.) and provides this.
    + Website shows message, "Registration complete."

1.1. Registration

A variety of additional use cases and configurations are also possible, including (but not limited to):

* A user navigates to example.com on their laptop, is guided through a flow to create and register a credential on the phone.

* A Rellying Party prompts the user for their authorization gesture in order to authorize a single transaction, such as a payment or financial transaction.

2. Conformance

This specification defines criteria for a Conforming User Agent: A User Agent MUST behave as described in this specification in any way desired. A conforming implementation of the IDL fragments of this specification, as described in the "Web IDL" specification, [WebIDL-1] is considered conformant. Conforming User Agents MAY implement algorithms given in this specification in any way desired, so long as the end result is indistinguishable from the result that would be obtained by the specification's algorithms. A conforming User Agent MUST also be a conforming implementation of the IDL fragments of this specification, as described in the "Web IDL" specification, [WebIDL-1].

This specification also defines a model of a conformant authenticator (see 5 WebAuthn Authenticator model). This is a set of functional and security requirements for an authenticator to be usable by a Conforming User Agent.
User Agent. As described in 1.1 Use Cases, an authenticator may be implemented in the operating system underlying the User Agent, or in external hardware, or a combination of both.

### 2.1. Dependencies

This specification relies on several other underlying specifications, listed below and in Terms defined by reference.

- **Base64url encoding**
  - The term Base64url Encoding refers to the base64 encoding using the URL- and filename-safe character set defined in Section 5 of [RFC4648], with all trailing '=' characters omitted (as permitted by Section 3.2) and without the inclusion of any line breaks, whitespace, or other additional characters.

- **CBOR**
  - A number of structures in this specification, including attestation statements and extensions, are encoded using the Compact Binary Object Representation (CBOR) ([RFC7049]).

- **CDDL**
  - This specification describes the syntax of all CBOR-encoded data using the CBOR Data Definition Language (CDDL) ([CDDL]).

- **COSE**
  - CBOR Object Signing and Encryption (COSE) ([RFC8152]). The IANA COSE Algorithms registry established by this specification is also used.

- **Credential Management**
  - The description of this document is an extension of the Credential concept defined in [CREDENTIAL-MANAGEMENT-1].

- **DOM**
  - DOMException and the DOMException values used in this specification are defined in [DOM].

- **ECMAScript**
  - %ArrayBuffer% is defined in [ECMAScript].

- **HTML**
  - The concepts of relevant settings object, origin, opaque origin, and a registrable domain suffix of or is equal to are defined

---

A Web User Agent MUST behave as described by 5 Web Authentication API in order to be considered conformant. Conforming User Agents MAY implement algorithms given in this specification in any way desired, so long as the end result is indistinguishable from the result that would be obtained by the specification’s algorithms.

A conforming User Agent MUST also be a conforming implementation of the IDL fragments of this specification, as described in the "Web IDL" specification. [WebIDL-1]

### 2.2. Authenticators

An authenticator MUST provide the operations defined by 6 WebAuthn Authenticator model, and those operations MUST behave as described there. This is a set of functional and security requirements for an authenticator to be usable by a Conforming User Agent.

As described in 1.1 Use Cases, an authenticator may be implemented in the operating system underlying the User Agent, or in external hardware, or a combination of both.

### 2.3. Relying Parties

A Relying Party MUST behave as described in 7 Relying Party Operations to get the security benefits offered by this specification.

### 3. Dependencies

This specification relies on several other underlying specifications, listed below and in Terms defined by reference.

- **Base64url encoding**
  - The term Base64url Encoding refers to the base64 encoding using the URL- and filename-safe character set defined in Section 5 of [RFC4648], with all trailing '=' characters omitted (as permitted by Section 3.2) and without the inclusion of any line breaks, whitespace, or other additional characters.

- **CBOR**
  - A number of structures in this specification, including attestation statements and extensions, are encoded using the Compact Binary Object Representation (CBOR) ([RFC7049]).

- **CDDL**
  - This specification describes the syntax of all CBOR-encoded data using the CBOR Data Definition Language (CDDL) ([CDDL]).

- **COSE**
  - CBOR Object Signing and Encryption (COSE) ([RFC8152]). The IANA COSE Algorithms registry established by this specification is also used.

- **Credential Management**
  - The API described in this document is an extension of the Credential concept defined in [CREDENTIAL-MANAGEMENT-1].

- **DOM**
  - DOMException and the DOMException values used in this specification are defined in [DOM].

- **ECMAScript**
  - %ArrayBuffer% is defined in [ECMAScript].

- **HTML**
  - The concepts of relevant settings object, origin, opaque origin, and a registrable domain suffix of or is equal to are defined
Web IDL

Many of the interface definitions and all of the IDL in this specification depend on [WebIDL-1]. This updated version of the Web IDL standard adds support for Promises, which are now the preferred mechanism for asynchronous interaction in all new web APIs.

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].

3. Terminology

3.1.2. Authentication

3.1.2.1. Authentication Assertion

See Authentication Assertion.

3.1.2.2. Attestation

Generally, attestation is a statement serving to bear witness, confirm, or authenticate. In the WebAuthn context, attestation is employed to attest to the provenance of an authenticator and the data it emits; including, for example: credential IDs, credential key pairs, signature counters, etc. An attestation statement is conveyed in an attestation object during registration. See also 5.3 Attestation and Figure 3.

3.1.2.3. Attestation Certificate

A X.509 Certificate for the attestation key pair used by an authenticator to attest to its manufacture and capabilities. At the time of registration, the authenticator uses the attestation private key to sign the Relying Party-specific credential public key (and additional data) that it generates and returns via the authenticatorMakeCredential operation. Relying Parties use the attestation public key conveyed in the attestation certificate to verify the attestation signature. Note that in the case of self attestation, the authenticator has no distinct attestation key pair nor attestation certificate, see self attestation for details.

3.1.2.4. Authentication

The ceremony where a user, and the user’s computing device(s) (containing at least one authenticator) work in concert to cryptographically prove to an Relying Party that the user controls the credential private key associated with a previously-registered public key credential (see Registration). Note that this typically includes employing a test of user presence or user verification.

3.1.2.5. Authentication Assertion

The cryptographically signed AuthenticatorAssertionResponse object returned by an authenticator as the result of a authenticatorGetAssertion operation.

4. Terminology

4.1. Attestation

Generally, attestation is a statement serving to bear witness, confirm, or authenticate. In the WebAuthn context, attestation is employed to attest to the provenance of an authenticator and the data it emits; including, for example: credential IDs, credential key pairs, signature counters, etc. An attestation statement is conveyed in an attestation object during registration. See also 5.3 Attestation and Figure 3. Whether or how the client platform conveys the attestation statement and AAGUID portions of the attestation object to the Relying Party is described by attestation conveyance.

Attestation Certificate

A X.509 Certificate for the attestation key pair used by an authenticator to attest to its manufacture and capabilities. At registration time, the authenticator uses the attestation private key to sign the Relying Party-specific credential public key (and additional data) that it generates and returns via the authenticatorMakeCredential operation. Relying Parties use the attestation public key conveyed in the attestation certificate to verify the attestation signature. Note that in the case of self attestation, the authenticator has no distinct attestation key pair nor attestation certificate, see self attestation for details.

Authentication

The ceremony where a user, and the user’s computing device(s) (containing at least one authenticator) work in concert to cryptographically prove to an Relying Party that the user controls the credential private key associated with a previously-registered public key credential (see Registration). Note that this includes a test of user presence or user verification.

Authentication Assertion

The cryptographically signed AuthenticatorAssertionResponse object returned by an authenticator as the result of a authenticatorGetAssertion operation.

This corresponds to the [CREDENTIAL-MANAGEMENT-1] specification’s single-use credentials.

Authenticator

A cryptographic device used by a WebAuthn Client to (i) generate a public key credential and register it with a Relying Party, and (ii) subsequently used to cryptographically sign and return other data presented by a Relying Party (in concert with the WebAuthn Client) in order to effect authentication.

Authorization Gesture

An authorization gesture is a physical interaction performed by
a user with an authenticator as part of a ceremony, such as registration or authentication. By making such an authorization gesture, a user provides consent for (i.e., authorizes) a ceremony to proceed. This may involve user verification if the employed authenticator is capable, or it may involve a simple test of user presence.

**Biometric Recognition**

The automated recognition of individuals based on their biological and behavioral characteristics

**Ceremony**

The concept of a ceremony [Ceremony] is an extension of the concept of a network protocol, with human nodes alongside computer nodes and with communication links that include user interface(s), human-to-human communication, and transfers of physical objects that carry data. What is out-of-band to a protocol is in-band to a ceremony. In this specification, Registration and Authentication are ceremonies, and an authorization gesture is often a component of those ceremonies.

**Client**

See Conforming User Agent.

**Client-Side**

This refers in general to the combination of the user's platform device, user agent, authenticators, and everything gluing it all together.

**Client-side-resident Credential Private Key**

A Client-side-resident Credential Private Key is stored either on the client platform, or in some cases on the authenticator itself, e.g., in the case of a discrete first-factor roaming authenticator. Such client-side credential private key storage has the property that the authenticator is able to select the private key material to the client platform, but the client platform is not expected to offload the key storage to remote entities (e.g. RP Server).

**Conforming User Agent**

A user agent implementing, in conjunction with the underlying platform, the Web Authentication API and algorithms given in this specification, and handling communication between authenticators and Relying Parties.

**Credential ID**

A probabilistically-unique byte sequence identifying a public key credential source and its authentication assertions.

**Credential IDs are generated by authenticators in two forms:**

1. At least 16 bytes that include at least 100 bits of entropy, or
2. The public key credential source, without its Credential ID, encrypted so only its managing authenticator can decrypt it. This form allows the authenticator to be nearly stateless, i.e., it need not keep any state information about a user with a particular credential source except as it is needed in order to perform the ceremony to proceed. This may involve user verification if the employed authenticator is capable, or it may involve a simple test of user presence.

**Biometric Recognition**

The automated recognition of individuals based on their biological and behavioral characteristics

**Ceremony**

The concept of a ceremony [Ceremony] is an extension of the concept of a network protocol, with human nodes alongside computer nodes and with communication links that include user interface(s), human-to-human communication, and transfers of physical objects that carry data. What is out-of-band to a protocol is in-band to a ceremony. In this specification, Registration and Authentication are ceremonies, and an authorization gesture is often a component of those ceremonies.

**Client**

See Conforming User Agent.

**Client-Side**

This refers in general to the combination of the user's platform device, user agent, authenticators, and everything gluing it all together.

**Client-side-resident Credential Private Key**

A Client-side-resident Credential Private Key is stored either on the client platform, or in some cases on the authenticator itself, e.g., in the case of a discrete first-factor roaming authenticator. Such client-side credential private key storage has the property that the authenticator is able to select the private key material to the client platform, but the client platform is not expected to offload the key storage to remote entities (e.g. RP Server).

**Conforming User Agent**

A user agent implementing, in conjunction with the underlying platform, the Web Authentication API and algorithms given in this specification, and handling communication between authenticators and Relying Parties.
self attestation, the credential key pair is also used as the self attestation key pair, see self attestation for details.

The public key portion of an Relying Party-specific credential is known as the credential public key. Note that in the case of self attestation, the credential key pair is also used as the self attestation key pair, see self attestation for details.

Note: This is a willful violation of [RFC4949]. In English, a "credential" is both a) the thing presented to prove a statement and b) intended to be used multiple times. It's impossible to achieve both criteria securely with a single piece of data in a public key system. [RFC4949] chooses to define a credential as the thing that can be used multiple times (the public key), while this specification gives "credential" the English term's flexibility. This specification uses more specific terms to identify the data related to an [RFC4949] credential:

- A credential source ([CREDENTIAL-MANAGEMENT]1) used by an authenticator to generate authentication assertions. A public key credential source has:
  - A Credential ID.
  - A credential private key.
  - The Relying Party Identifier for the Relying Party that created this credential source.
  - An optional user handle for the person who created this credential source.
  - Optional other information used by the authenticator to inform its UI. For example, this might include the user's display Name.

The authenticatorMakeCredential operation creates a public key credential source bound to a managing authenticator and returns the credential public key associated with its credential private key. The Relying Party can use this public key credential to verify the authentication assertions created by this public key credential source.

Public Key Credential

Generically, a credential is data one entity presents to another in order to authenticate the former to the latter ([RFC4949]). The term public key credential refers to one of: a public key credential source, the possibly-attested credential public key corresponding to a public key credential source, or an authentication assertion. Which one is generally determined by context.

Note: This is a willful violation of [RFC4949]. In English, a "credential" is both a) the thing presented to prove a statement and b) intended to be used multiple times. It's impossible to achieve both criteria securely with a single piece of data in a public key system. [RFC4949] chooses to define a credential as the thing that can be used multiple times (the public key), while this specification gives "credential" the English term's flexibility. This specification uses more specific terms to identify the data related to an [RFC4949] credential:

- A credential source ([CREDENTIAL-MANAGEMENT]1) used by an authenticator to generate authentication assertions. A public key credential source has:
  - A Credential ID.
  - A credential private key.
  - The Relying Party Identifier for the Relying Party that created this credential source.
  - An optional user handle for the person who created this credential source.
  - Optional other information used by the authenticator to inform its UI. For example, this might include the user's display Name.

The authenticatorMakeCredential operation creates a public key credential source bound to a managing authenticator and returns the credential public key associated with its credential private key. The Relying Party can use this public key credential to verify the authentication assertions created by this public key credential source.

Public Key Credential

Generically, a credential is data one entity presents to another in order to authenticate the former to the latter ([RFC4949]). The term public key credential refers to one of: a public key credential source, the possibly-attested credential public key corresponding to a public key credential source, or an authentication assertion. Which one is generally determined by context.

Note: This is a willful violation of [RFC4949]. In English, a "credential" is both a) the thing presented to prove a statement and b) intended to be used multiple times. It's impossible to achieve both criteria securely with a single piece of data in a public key system. [RFC4949] chooses to define a credential as the thing that can be used multiple times (the public key), while this specification gives "credential" the English term's flexibility. This specification uses more specific terms to identify the data related to an [RFC4949] credential:
The process (also known as throttling) by which an authenticator implements controls against brute force attacks by limiting the number of consecutive failed authentication attempts within a given period of time. If the limit is reached, the authenticator should impose a delay that increases exponentially with each successive attempt, or disable the current authentication modality and offer a different authentication factor if available. Rate limiting is often implemented as an aspect of user verification.

The ceremony where a user, a Relying Party, and the user's computing device(s) (containing at least one authenticator) work in concert to create a public key credential and associate it with the user's Relying Party account. Note that this typically includes employing a test of user presence or user verification.

The entity whose web application utilizes the Web Authentication API to register and authenticate users. See Registration and Authentication, respectively.

Note: While the term Relying Party is used in other contexts (e.g., X.509 and OAuth), an entity acting as a Relying Party in one context is not necessarily a Relying Party in other contexts.

The entity whose web application utilizes the Web Authentication API to register and authenticate users. See Registration and Authentication, respectively.

Note: While the term Relying Party is used in other contexts (e.g., X.509 and OAuth), an entity acting as a Relying Party in one context is not necessarily a Relying Party in other contexts.

A valid domain string that identifies the Relying Party on whose behalf a given registration or authentication ceremony is being performed. A public key credential can only be used for authentication with the same entity (as identified by the RP ID) it was registered with. By default, the RP ID for a WebAuthn operation is set to the caller's origin's effective domain. This default may be overridden by the caller, as long as the caller-specified RP ID value is a registrable domain suffix of or is equal to the caller's origin's effective domain. See also 6.1, Create a new credential - PublicKeyCredential's [[Create](options)](method) and 4.4 Use an existing credential - PublicKeyCredential's [[DiscoverFromExternalSource](options)](method).

Note: A Public key credential's scope is for a Relying Party's origin, with the following restrictions and relaxations:

- + The scheme is always https (i.e., a restriction), and, + the host may be equal to the Relying Party's origin's effective domain, or it may be equal to a registrable domain suffix of the Relying Party's origin's effective domain (i.e., an available relaxation), and, + all (TCP) ports on that host (i.e., a relaxation).

This is done in order to match the behavior of pervasively deployed ambient credentials (e.g., cookies, [RFC2618]). Please note that this is a greater relaxation of "same-origin" restrictions than what document.domain's setter provides.

Public Key Credential

Generally, a credential is data one entity presents to another in order to authenticate the former to the latter (RFC4949). A WebAuthn public key credential is a (identifier, type) pair identifying authentication information established by the authenticator and the Relying Party, together, at registration time. The authentication information consists of an asymmetric key pair, where the public key portion is returned to the Relying Party, who then stores it in conjunction with the present user's account. The authenticator maps the private key portion to the Relying Party's RP ID and stores it.

The process (also known as throttling) by which an authenticator implements controls against brute force attacks by limiting the number of consecutive failed authentication attempts within a given period of time. If the limit is reached, the authenticator should impose a delay that increases exponentially with each successive attempt, or disable the current authentication modality and offer a different authentication factor if available. Rate limiting is often implemented as an aspect of user verification.

The ceremony where a user, a Relying Party, and the user's computing device(s) (containing at least one authenticator) work in concert to create a public key credential and associate it with the user's Relying Party account. Note that this typically includes employing a test of user presence or user verification.

The entity whose web application utilizes the Web Authentication API to register and authenticate users. See Registration and Authentication, respectively.

Note: While the term Relying Party is used in other contexts (e.g., X.509 and OAuth), an entity acting as a Relying Party in one context is not necessarily a Relying Party in other contexts.

The entity whose web application utilizes the Web Authentication API to register and authenticate users. See Registration and Authentication, respectively.

Note: While the term Relying Party is used in other contexts (e.g., X.509 and OAuth), an entity acting as a Relying Party in one context is not necessarily a Relying Party in other contexts.

A valid domain string that identifies the Relying Party on whose behalf a given registration or authentication ceremony is being performed. A public key credential can only be used for authentication with the same entity (as identified by the RP ID) it was registered with. By default, the RP ID for a WebAuthn operation is set to the caller's origin's effective domain. This default may be overridden by the caller, as long as the caller-specified RP ID value is a registrable domain suffix of or is equal to the caller's origin's effective domain. See also 5.1, Create a new credential - PublicKeyCredential's [[Create](options)](method) and 4.4 Use an existing credential - PublicKeyCredential's [[Get](options)](method).

Note: A Public key credential's scope is for a Relying Party's origin, with the following restrictions and relaxations:

- + The scheme is always https (i.e., a restriction), and, + the host may be equal to the Relying Party's origin's effective domain, or it may be equal to a registrable domain suffix of the Relying Party's origin's effective domain (i.e., an available relaxation), and, + all (TCP) ports on that host (i.e., a relaxation).

This is done in order to match the behavior of pervasively deployed ambient credentials (e.g., cookies, [RFC2618]). Please note that this is a greater relaxation of "same-origin" restrictions than what document.domain's setter provides.
Test of User Presence

A test of user presence is a simple form of authorization that is a ceremony component often employed to indicate user consent.

User Consent

User consent means the user agrees with what they are being asked, i.e., it encompasses reading and understanding prompts. An authorization gesture is a ceremony component often employed to indicate user consent.

User Verification

The technical process by which an authenticator locally authorizes the invocation of the authenticatorMakeCredential and authenticatorGetAssertion operations. User verification may be instigated through various authorization gesture modalities; for example, through a touch plus pin code, password entry, or biometric recognition (e.g., presenting a fingerprint). The intent is to be able to distinguish individual users. Note that invocation of the authenticatorMakeCredential and authenticatorGetAssertion operations implies use of key material managed by the authenticator. Note that for security, user verification and use of credential private keys must occur within a single logical security boundary defining the authenticator.

User Present

Upon successful completion of a user presence test, the user is said to be "present".

User Verified

Upon successful completion of a user verification process, the user is said to be "verified".

WebAuthn Client

Also referred to herein as simply a client. See also Conforming User Agent.

4. Web Authentication API

This section normatively specifies the API for creating and using public key credentials. The basic idea is that the credentials belong to the user and are managed by an authenticator, with which the Relying Party interacts through the client (consisting of the browser and underlying OS platform). Scripts can (with the user's consent) request the browser to create a new credential for future use by the Relying Party to create a new credential for future use by the Relying Party, as identified by its RP ID, is able to employ the public key credential in authentication ceremonies, via the get() method. The Relying Party uses its stored copy of the credential public key to verify the resultant authentication assertion.
4.1. PublicKeyCredential Interface

The PublicKeyCredential interface inherits from Credential [CREDENTIAL-MANAGEMENT-1], and contains the attributes that are returned to the caller when a new credential is created, or a new assertion is requested. It also contains an interface, PublicKeyCredential : Credential {

**SecureContext**

interface PublicKeyCredential : Credential {

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>SameObject readonly attribute</td>
</tr>
<tr>
<td>rawId</td>
<td>ArrayBuffer</td>
</tr>
</tbody>
</table>

/Users/jehodges/Documents/work/standards/W3C/webauthn/index-master-tr-5e63e57-WD-07.txt, Top line: 741

Part. Scripts can also request the user’s permission to perform authentication operations with an existing credential. All such operations are performed in the authenticator and are mediated by the browser and/or platform on the user’s behalf. At no point does the script get access to the credentials themselves; it only gets information about the credentials in the form of objects.

In addition to the above script interface, the authenticator may implement (or come with client software implementing) a user interface for management. Such an interface may be used, for example, to inform the authenticator to a clean state or to inspect the current state of the authenticator. In other words, such an interface is similar to the user interfaces provided by browsers for managing user state such as, history, saved passwords and cookies. Authenticator management actions such as credential deletion are considered to be the responsibility of such a user interface and are deliberately omitted from the API exposed to scripts.

The security properties of this API are provided by the client and the authenticator working together. The authenticator, which holds and manages credentials, ensures that all operations are scoped to a particular origin, and cannot be replayed against a different origin, by incorporating the origin in its responses. Specifically, as defined in 5.2 Authenticator operations, the full origin of the requester is included, and signed over, in the attestation object produced when a new credential is created as well as in all assertions produced by WebAuthn credentials.

Additionally, to maintain user privacy and prevent malicious Relying Parties from probing for the presence of public key credentials belonging to other Relying Parties, each credential is also associated with a Relying Party identifier, or RP ID. This RP ID is provided to the client to the authenticator for all operations, and the authenticator ensures that credentials created by a Relying Party can only be used in operations requested by the same RP ID. Separating the origin from the RP ID in this way allows the API to be used in cases where a single Relying Party maintains multiple origins.

The client facilitates these security measures by providing the Relying Party’s origin and RP ID to the authenticator for each operation. Since this is an integral part of the WebAuthn security model, user agents must expose this API to callers in secure contexts.

The Web Authentication API is defined by the union of the Web IDL fragments presented in the following sections. A combined IDL listing is given in the IDL index.

5.1. PublicKeyCredential Interface

The PublicKeyCredential interface inherits from Credential [CREDENTIAL-MANAGEMENT-1], and contains the attributes that are returned to the caller when a new credential is created, or a new assertion is requested. It also contains an interface, PublicKeyCredential : Credential {

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>SameObject readonly attribute</td>
</tr>
<tr>
<td>rawId</td>
<td>ArrayBuffer</td>
</tr>
</tbody>
</table>

/Users/jehodges/Documents/work/standards/W3C/webauthn/index-master-tr-598ac41-WD-06.txt, Top line: 623

This attribute is inherited from Credential, though

PublicKeyCredential overrides Credential’s getter, instead returning the base64url encoding of the data contained in the object’s [[identifier]] internal slot.

This attribute returns the ArrayBuffer contained in the [[identifier]] internal slot.
response, of type AuthenticatorResponse, readonly
This attribute contains the authenticator’s response to the
client’s request to either create a public key credential, or
generate an authentication assertion. If the PublicKeyCredential
is created in response to create(), this attribute’s value will
be an AuthenticatorAttestationResponse, otherwise, the
PublicKeyCredential was created in response to get(), and this
attribute’s value will be an AuthenticatorAssertionResponse.

4.1.2. CredentialRequestOptions Extension
To support obtaining assertions via navigator.credentials.get(), this
document extends the CredentialRequestOptions dictionary as follows:
partial dictionary CredentialRequestOptions {
  PublicKeyCredentialRequestOptions publicKey;
}

4.1.1. CredentialCreationOptions Extension
To support registration via navigator.credentials.create(), this
document extends the CredentialCreationOptions dictionary as follows:
partial dictionary CredentialCreationOptions {
  MakePublicKeyCredentialOptions publicKey;
}

1.4.1. CredentialRequestOptions Extension
To support obtaining assertions via navigator.credentials.get(), this
document extends the CredentialRequestOptions dictionary as follows:
partial dictionary CredentialRequestOptions {
  PublicKeyCredentialRequestOptions publicKey;
}

1.3.1. CredentialCreationOptions Extension
To support registration via navigator.credentials.create(), this
document extends the CredentialCreationOptions dictionary as follows:
partial dictionary CredentialCreationOptions {
  MakePublicKeyCredentialOptions publicKey;
}

5.1.1. CredentialCreationOptions Extension
To support registration via navigator.credentials.create(), this
document extends the CredentialCreationOptions dictionary as follows:
partial dictionary CredentialCreationOptions {
  MakePublicKeyCredentialOptions publicKey;
}

5.1.2. CredentialRequestOptions Extension
To support obtaining assertions via navigator.credentials.get(), this
document extends the CredentialRequestOptions dictionary as follows:
partial dictionary CredentialRequestOptions {
  PublicKeyCredentialRequestOptions publicKey;
}
4.1.3. Create a new credential - PublicKeyCredential's \([\text{Create}]\)(options) method

NavigatorCredential's interface object's implementation of the \([\text{Create}]\)(options) method allows scripts to call navigator.credentials.create() to request the creation of a new credential key pair and PublicKeyCredential, managed by an authenticator. The user agent will prompt the user for consent. On success, the returned promise will be resolved with a PublicKeyCredential containing an AuthenticatorAttestationResponse object.

Note: This algorithm is synchronous; the Promise resolution/rejection is handled by navigator.credentials.create().

This method accepts a single argument:

```
options
```

This argument is a CredentialCreationOptions object whose publicKey member contains a MakePublicKeyCredentialOptions object specifying the desired attributes of the to-be-created public key credential.

When this method is invoked, the user agent MUST execute the following algorithm:

1. Assert: options.publicKey is present.
2. If options is a CredentialCreationOptions object, make a new PublicKeyCredential and set its options to options.publicKey.
3. If any of the name member of options.rp, the name member of options.user, the displayName member of options.user, or the id member of options.user are not present, return a TypeError simple exception.
4. If the timeout member of options is present, check if its value lies within a reasonable range as defined by the platform and if not, correct it to the closest value lying within that range. Set adjustedTimeout to this adjusted value. If the timeout member of options is not present, then set adjustedTimeout to a platform-specific default.
5. Let global be the PublicKeyCredential's interface object's environment settings object's global object.
6. Let callerOrigin be the origin specified by this PublicKeyCredential interface object's relevant settings object. If callerOrigin is an opaque origin, return a DOMException whose name is "NotAllowedError", and terminate this algorithm.

This argument is the relevant settings object's origin, as determined by the calling create() implementation.

5.1.3. Create a new credential - PublicKeyCredential's \([\text{Create}]\)(origin, options, sameOriginWithAncestors) method

PublicKeyCredential's interface object's implementation of the \([\text{Create}]\)(origin, options, sameOriginWithAncestors) method allows Relying Party script calls to navigator.credentials.create() to request the creation of a new public key credential source, bound to an authenticator. This navigator.credentials.create() operation can be aborted by leveraging the AbortController: see DOM 3.3 Using AbortController and AbortSignal objects in APIs for detailed instructions.

This internal method accepts three arguments:

```
origin
options
sameOriginWithAncestors
```

This argument is an opaque origin, as specified in the credentialManagement-1, while allowing Relying Party script access to Web Authentication functionality, e.g., when running in a secure context. The global object, same-origin with its ancestors. However, in the future, this specification (in conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying Parties with more fine-grained control—e.g., ranging from allowing only top-level access to Web Authentication functionality, to allowing cross-origin embedded cases—by leveraging [Feature-Policy] once the latter specification becomes stably implemented in user agents.

When this method is invoked, the user agent MUST execute the following algorithm:

1. Assert: options.publicKey is present.
2. If options is a CredentialCreationOptions object, make a new PublicKeyCredential and set its options to options.publicKey.
3. If the timeout member of options is present, check if its value lies within a reasonable range as defined by the platform and if not, correct it to the closest value lying within that range. Set a timer lifetimeTimer to this adjusted value. If the timeout member of options is not present, then set lifetimeTimer to a platform-specific default.
4. If callerOrigin is an opaque origin, return a DOMException whose name is "NotAllowedError", and terminate this algorithm.
5. Let effectiveDomain be the callerOrigin's effective domain.

Note: This "sameOriginWithAncestors" restriction aims to address allowing cross-origin embedded cases—by leveraging [Feature-Policy] once the latter specification becomes stably implemented in user agents.
7. Let effectiveDomain be the callerOrigin's effective domain. If effective domain is not a valid domain, then return a DOMException whose name is "SecurityError" and terminate this algorithm.

8. Note: An effective domain may resolve to a host, which can be represented in various manners, such as domain, ipv4 address, ipv6 address, opaque host, or empty host. Only the domain format of host is allowed here.

9. Let rpId be effectiveDomain.

10. If options.rp.id is present:
   1. If options.rp.id is not a registrable domain suffix of and is not equal to effectiveDomain, return a DOMException whose name is "SecurityError", and terminate this algorithm.
   2. Set rpid to options.rp.id.

11. Note: rpId represents the caller's RP ID. The RP ID defaults to being the caller's origin's effective domain unless the caller explicitly set options.rp.id when calling create().

12. If credTypesAndPubKeyAlgs is empty and options.pubKeyCredParams is not empty, cancel the timer started in step 2, return a DOMException whose name is "NotSupportedError", and terminate this algorithm.

13. Let clientExtensions be a new map and let authenticatorExtensions be a new map.

14. If the extensions member of options is present, then for each extensionId -> clientExtensionInput of options.extensions:
   1. If extensionId is not supported by this client platform or is not a registration extension, then continue.
   2. Set extClientExtensions[extensionId] to clientExtensionInput.
   3. If extensionId is not an authenticator extension, then continue.
   4. Let authenticatorExtensionInput be the (CBOR) result of running extensionId's client extension processing algorithm on clientExtensionInput. If the algorithm returned an error, continue.
   5. Set authenticatorExtensions[extensionId] to the base64url encoding of authenticatorExtensionInput.

15. Let collectedClientData be a new CollectedClientData instance whose fields are:
   challenge: The base64url encoding of options.challenge.
   origin: The serialization of callerOrigin.
   hashAlgorithm: The recognized algorithm name of the hash algorithm selected by the client for generating the hash of the serialized client data.
   tokenBindingId: The Token Binding ID associated with callerOrigin, if one is available.
   clientExtensions

effective domain is not a valid domain, then return a DOMException whose name is "SecurityError" and terminate this algorithm.

Note: An effective domain may resolve to a host, which can be represented in various manners, such as domain, ipv4 address, ipv6 address, opaque host, or empty host. Only the domain format of host is allowed here.

7. If options.rp.id is present:
   1. If options.rp.id is not a registrable domain suffix of and is not equal to effectiveDomain, return a DOMException whose name is "SecurityError", and terminate this algorithm.

8. Set options.rp.id to effectiveDomain.

Note: options.rp.id represents the caller's RP ID. The RP ID defaults to being the caller's origin's effective domain unless the caller has explicitly set options.rp.id when calling create().

8. Let credTypesAndPubKeyAlgs be a new list whose items are pairs of PublicKeyCredentialType and a COSEAlgorithmIdentifier supported by this implementation, then continue.

9. For each current of options.pubKeyCredParams:
   1. If current.type does not contain a PublicKeyCredentialType supported by this implementation, then continue.
   2. Set alg be current.alg.
   3. Append the pair of current.type and alg to credTypesAndPubKeyAlgs.

10. Let clientExtensions be a new map and let authenticatorExtensions be a new map.

11. If the extensions member of options is present, then for each extensionId -> clientExtensionInput of options.extensions:
   1. If extensionId is not supported by this client platform or is not a registration extension, then continue.
   2. Set extClientExtensions[extensionId] to clientExtensionInput.
   3. If extensionId is not an authenticator extension, then continue.
   4. Let authenticatorExtensionInput be the (CBOR) result of running extensionId's client extension processing algorithm on clientExtensionInput. If the algorithm returned an error, continue.
   5. Set authenticatorExtensions[extensionId] to the base64url encoding of authenticatorExtensionInput.

12. Let collectedClientData be a new CollectedClientData instance whose fields are:
   type: The string "webauthn.create".
   challenge: The base64url encoding of options.challenge.
   origin: The serialization of callerOrigin.
   hashAlgorithm: The recognized algorithm name of the hash algorithm selected by the client for generating the hash of the serialized client data.
   tokenBindingId: The Token Binding ID associated with callerOrigin, if one is available.
   clientExtensions

13. Let collectedClientData be a new CollectedClientData instance whose fields are:
Let initiatedRequest be an ordered set consisting of all authenticators currently available on this platform.

Let issueRequests be a new ordered set.

Let now be the time according to the client's internal clock.

For each authenticator that becomes available on this platform during the lifetime of lifetimeTimer, do the following:

1. Let excludedCredentialDescriptorList be a new list.
2. For each credential descriptor C in options.excludedCredentials:
   a. If C.transports is not empty, and authenticator is not connected over a transport not mentioned in C.transports, the client MAY continue.
   b. Otherwise, Append C to excludedCredentialDescriptorList.
3. In parallel, invoke the authenticatorMakeCredential operation on the authenticator with rp, clientDataJSON, options, rp, options, authExampleExtensions, clientExtensions, and authenticatorExtensions as parameters.
4. Append authenticator to issueRequests.
5. Start a timer for adjustedTimeout milliseconds. Then execute the following steps in parallel. The task source for these tasks is the dom manipulation task source.
   a. If initiatedRequests is not empty, perform the following actions depending upon lifetimeTimer and responses from the authenticators:
      1. Let excludedCredentialDescriptorList be a new list.
      2. For each credential descriptor C in options.excludedCredentials:
         a. If C.transports is not empty, and authenticator is not connected over a transport not mentioned in C.transports, the client MAY continue.
         b. Otherwise, Append C to excludedCredentialDescriptorList.
      3. In parallel, invoke the authenticatorMakeCredential operation on the authenticator with rp, clientDataJSON, options, rp, options, authExampleExtensions, clientExtensions, and authenticatorExtensions as parameters.
      4. Append authenticator to issueRequests.
      5. Start a timer for adjustedTimeout milliseconds. Then execute the following actions:
         a. If initiatedRequests is not empty, perform the following actions depending upon lifetimeTimer and responses from the authenticators:
            1. Let excludedCredentialDescriptorList be a new list.
            2. For each credential descriptor C in options.excludedCredentials:
               a. If C.transports is not empty, and authenticator is not connected over a transport not mentioned in C.transports, the client MAY continue.
               b. Otherwise, Append C to excludedCredentialDescriptorList.
            3. In parallel, invoke the authenticatorMakeCredential operation on the authenticator with rp, clientDataJSON, options, rp, options, authExampleExtensions, clientExtensions, and authenticatorExtensions as parameters.
            4. Append authenticator to issueRequests.
            5. Start a timer for adjustedTimeout milliseconds. Then execute the following actions:
               a. If initiatedRequests is not empty, perform the following actions depending upon lifetimeTimer and responses from the authenticators:
                  1. Let excludedCredentialDescriptorList be a new list.
                  2. For each credential descriptor C in options.excludedCredentials:
                     a. If C.transports is not empty, and authenticator is not connected over a transport not mentioned in C.transports, the client MAY continue.
                     b. Otherwise, Append C to excludedCredentialDescriptorList.
                  3. In parallel, invoke the authenticatorMakeCredential operation on the authenticator with rp, clientDataJSON, options, rp, options, authExampleExtensions, clientExtensions, and authenticatorExtensions as parameters.
                  4. Append authenticator to issueRequests.
                  5. Start a timer for adjustedTimeout milliseconds. Then execute the following actions:
                     a. If initiatedRequests is not empty, perform the following actions depending upon lifetimeTimer and responses from the authenticators:
                        1. Let excludedCredentialDescriptorList be a new list.
                        2. For each credential descriptor C in options.excludedCredentials:
                           a. If C.transports is not empty, and authenticator is not connected over a transport not mentioned in C.transports, the client MAY continue.
                           b. Otherwise, Append C to excludedCredentialDescriptorList.
                        3. In parallel, invoke the authenticatorMakeCredential operation on the authenticator with rp, clientDataJSON, options, rp, options, authExampleExtensions, clientExtensions, and authenticatorExtensions as parameters.
                        4. Append authenticator to issueRequests.
                        5. Start a timer for adjustedTimeout milliseconds. Then execute the following actions:
                           a. If initiatedRequests is not empty, perform the following actions depending upon lifetimeTimer and responses from the authenticators:
                              1. Let excludedCredentialDescriptorList be a new list.
                              2. For each credential descriptor C in options.excludedCredentials:
                                 a. If C.transports is not empty, and authenticator is not connected over a transport not mentioned in C.transports, the client MAY continue.
                                 b. Otherwise, Append C to excludedCredentialDescriptorList.
                              3. In parallel, invoke the authenticatorMakeCredential operation on the authenticator with rp, clientDataJSON, options, rp, options, authExampleExtensions, clientExtensions, and authenticatorExtensions as parameters.
                              4. Append authenticator to issueRequests.
                              5. Start a timer for adjustedTimeout milliseconds. Then execute the following actions:
                                 a. If initiatedRequests is not empty, perform the following actions depending upon lifetimeTimer and responses from the authenticators:
                                    1. Let excludedCredentialDescriptorList be a new list.
                                    2. For each credential descriptor C in options.excludedCredentials:
                                       a. If C.transports is not empty, and authenticator is not connected over a transport not mentioned in C.transports, the client MAY continue.
                                       b. Otherwise, Append C to excludedCredentialDescriptorList.
                                    3. In parallel, invoke the authenticatorMakeCredential operation on the authenticator with rp, clientDataJSON, options, rp, options, authExampleExtensions, clientExtensions, and authenticatorExtensions as parameters.
                                    4. Append authenticator to issueRequests.
                                    5. Start a timer for adjustedTimeout milliseconds. Then execute the following actions:
                                       a. If initiatedRequests is not empty, perform the following actions depending upon lifetimeTimer and responses from the authenticators:
                                          1. Let excludedCredentialDescriptorList be a new list.
                                          2. For each credential descriptor C in options.excludedCredentials:
                                             a. If C.transports is not empty, and authenticator is not connected over a transport not mentioned in C.transports, the client MAY continue.
                                             b. Otherwise, Append C to excludedCredentialDescriptorList.
                                          3. In parallel, invoke the authenticatorMakeCredential operation on the authenticator with rp, clientDataJSON, options, rp, options, authExampleExtensions, clientExtensions, and authenticatorExtensions as parameters.
                                          4. Append authenticator to issueRequests.
                                          5. Start a timer for adjustedTimeout milliseconds. Then execute the following actions:
                                             a. If initiatedRequests is not empty, perform the following actions depending upon lifetimeTimer and responses from the authenticators:
                                                1. Let excludedCredentialDescriptorList be a new list.
                                                2. For each credential descriptor C in options.excludedCredentials:
                                                   a. If C.transports is not empty, and authenticator is not connected over a transport not mentioned in C.transports, the client MAY continue.
                                                   b. Otherwise, Append C to excludedCredentialDescriptorList.
                                                3. In parallel, invoke the authenticatorMakeCredential operation on the authenticator with rp, clientDataJSON, options, rp, options, authExampleExtensions, clientExtensions, and authenticatorExtensions as parameters.
                                                4. Append authenticator to issueRequests.
                                                5. Start a timer for adjustedTimeout milliseconds. Then execute the following actions:
                                                   a. If initiatedRequests is not empty, perform the following actions depending upon lifetimeTimer and responses from the authenticators:
4. Let value be a new PublicKeyCredential object associated with global whose fields are:

3. Let id be attestationObject.authData.attestation operation (which is attObj, as defined in 5.3.4 Generating an Attestation Object).

2. Let attestationObject be a new ArrayBuffer, created using 2. Let attestationObject be a new ArrayBuffer, created using 2. Let attestationObject be a new ArrayBuffer, created using 2. Let attestationObject be a new ArrayBuffer, created using 2. Let attestationObject be a new ArrayBuffer, created using

1. Remove authenticator from issuedRequests.

If any authenticator returns a status indicating the user cancelled the operation,

If any authenticator indicates success,

If any authenticator returns an error status, Remove authenticator from issuedRequests.

If the options.signal is present and its aborted flag is set to true, return a DOMException whose name is "AbortError" and terminate this algorithm.

If any authenticator returns a status indicating that the user cancelled the operation, 1. Remove authenticator from issuedRequests.

For each remaining authenticator in issuedRequests invoke the authenticatorCancel operation on authenticator and remove it from issuedRequests.

If any authenticator returns an error status, Remove authenticator from issuedRequests.

If any authenticator indicates success, 1. Remove authenticator from issuedRequests.

2. Let attestationObject be a new ArrayBuffer, created using global's %ArrayBuffer%, containing the bytes of the value returned from the successful authenticatorMakeCredential operation (which is attObj, as defined in 5.3.4 Generating an Attestation Object).

3. Let id be attestationObject.authData.attestation data credentialID (see 5.3.1 Attestation data and 5.1 Authenticator data).

4. Let value be a new PublicKeyCredential object associated with global whose fields are:

Note: this value is attObj, as defined in 6.3.4 Generating an Attestation Object.

If the adjustedTimeout timer expires, For each authenticator in issuedRequests invoke the authenticatorCancel operation on authenticator and remove authenticator from issuedRequests.

If any authenticator returns a status indicating that the user cancelled the operation, 1. Remove authenticator from issuedRequests.

2. For each remaining authenticator in issuedRequests invoke the authenticatorCancel operation on authenticator and remove it from issuedRequests.

If any authenticator returns an error status, Remove authenticator from issuedRequests.

If any authenticator indicates success, 1. Remove authenticator from issuedRequests.

2. Let credentialCreationData be a struct whose items are:

- attestationObjectResult whose value is the bytes returned from the successful authenticatorMakeCredential operation.

- clientDataJSONResult whose value is the bytes of clientData.JSON.

- attestationConveyancePreferenceOption whose value is the value of options.attestation.

- clientExtensionResults whose value is an AuthenticationExtensions object containing extension identifier -> client extension output entries. The entries are created by running each extension's client extension processing algorithm to create the client extension outputs, for each client extension in clientDataJSON.clientExtensions.

3. Let constructCredentialAlg be an algorithm that takes a global object global, and whose steps are:

1. Let attestationObject be a new ArrayBuffer, created using global's %ArrayBuffer%, containing the bytes of credentialCreationData.attestationObjectResult's value.

2. If credentialCreationData.attestationConveyancePreferenceOption's value is "none" Replace potentially uniquely identifying information (such as AAGUID and attestation certificates) in the
5. For each remaining authenticator in issuedRequests invoke the authenticatorCancel operation on authenticator and remove it from issuedRequests.

6. Return value and terminate this algorithm.
3. If the timeout member of options is present, check if its value

1. Assert: options.publicKey is present.

When this method is invoked, the user agent MUST execute the following

authenticatorGetAssertion operation.

The [[DiscoverFromExternalSource](options)] method is used to discover

select a credential source.

Relying Parties call navigator.credentials.get({publicKey:..., ...}) to
discover and use an existing public key credential, with the user's consent. The
script optionally specifies some criteria to indicate what credentials
are acceptable to it. The user agent and/or platform locates credential sources matching the specified
criteria, and guides the user to pick one that the script will be
allowed to use. The user may choose to decline the entire interaction
even if a credential source is present, for example to maintain privacy. If the user picks a credential source, the user agent then
uses 6.2.2 The authenticatorGetAssertion operation to sign a Relying
Party-provided challenge and other collected data into an assertion,
which is used as a credential.

This argument is a CredentialRequestOptions object whose
attributes of the public key credential to discover.

This internal method accepts three arguments:

options

This argument is a CredentialRequestOptions object whose

This argument is a CredentialRequestOptions object whose
options, sameOriginWithAncestors) method

This argument is a boolean which is true if and only if the
caller's environment settings object is same-origin with its
ancestors.

Note: This algorithm is synchronous: the Promise resolution/rejection
is handled by navigator.credentials.get().
lies within a reasonable range as defined by the platform and if
not, correct it to the closest value lying within that range. Set
adjustedTimeout to this adjusted value. If the timeout member of
options is not present, then set adjustedTimeout to a
platform-specific default.

4. Let global be the PublicKeyCredential's interface object's
environment settings object's global object.

5. Let callerOrigin be the origin specified by this
PublicKeyCredential interface object's relevant settings object. If
callerOrigin is an opaque origin, return a DOMException whose name
is "NotAllowedError", and terminate this algorithm.

6. Let effectiveDomain be the callerOrigin's effective domain. If
effective domain is not a valid domain, then return a DOMException
whose name is "SecurityError" and terminate this algorithm.

Note: An effective domain may resolve to a host, which can be
represented in various manners, such as domain, ipv4 address, ipv6
address, opaque host, or empty host. Only the domain format of host
is allowed here.

7. If options.rpid is not present, then set rpid to effectiveDomain.
Otherwise:
   1. If options.rpid is not a registrable domain suffix of and is
      not equal to effectiveDomain, return a DOMException whose name
      is "SecurityError", and terminate this algorithm.
   2. Set rpId to options.rpid.
Note: rpid represents the caller's RP ID. The RP ID defaults
to being the caller's origin's effective domain unless the
caller has explicitly set options.rpid when calling get().

8. Let clientExtensions be a new map and let authenticatorExtensions
be a new map.

9. If the extensions member of options is present, then for each
extensionId -> clientExtensionInput of options.extensions:
   1. if extensionId is not supported by this client platform or is
      not an authentication extension, then continue.
   2. Set clientExtensions[extensionId] to clientExtensionInput.
   3. if extensionId is not an authenticator extension, then
      continue.
   4. Let authenticatorExtensionInput be the (CBOR) result of
      running extensionId's client extension processing algorithm on
      clientExtensionInput. If the algorithm returned an error, then
      continue.
   5. Set authenticatorExtensions[extensionId] to the base64url
      encoding of authenticatorExtensionInput.

10. Let collectedClientData be a new CollectedClientData instance whose
    fields are:

    | field      | type                                      |
    |------------|-------------------------------------------|
    | challenge  | The base64url encoding of options.challenge|
    | origin     | The serialization of callerOrigin.        |
    | hashAlgorithm | The recognized algorithm name of the hash algorithm selected by the client for generating the hash of the serialized client data|
    | tokenBindingId | The Token Binding ID associated with callerOrigin, if one is available. |
2. For each credential descriptor C in
   1. Let distinctTransports be a new ordered set.
   12. Let clientDataJSON be the JSON-serialized client data constructed from collectedClientData.
   13. Let issuedRequests be a new ordered set.
   14. If there are no authenticators currently available on this platform, return a DOMException whose name is "NotFoundError", and terminate this algorithm.
   15. Let authenticator be a platform-specific handle whose value is not empty.
   16. Let allowCredentialDescriptorList be a new list.
   17. For each authenticator currently available on this platform,
       perform the following steps:
       1. Let distinctTransports be a new ordered set.
       2. If allowCredentialDescriptorList has exactly one credential descriptor, let savedCredentialId be a new PublicKeyCredentialDescriptor.id and set its value to allowCredentialDescriptorList[0].id's value (see resolving Issue #613).
       3. If the options.signal is present and its aborted flag is set to true, return a DOMException whose name is "AbortError" and terminate this algorithm.
       4. Let allowCredentialDescriptorList be a new list.
       5. If options.allowCredentials is not empty, execute a platform-specific procedure to determine which, if any, public key credentials described by options.allowCredentials are bound to this authenticator, by matching with rpId, options.allowCredentials.id, and options.allowCredentials.type. Set allowCredentialDescriptorList to this filtered list.
       6. If allowCredentialDescriptorList is not empty
          1. Let distinctTransports be a new ordered set.
          2. For each credential descriptor C in allowCredentialDescriptorList
             1. If options.userVerification is set to required and the authenticator is not capable of performing user verification, continue.
             2. Let userVerification be the effective user verification requirement for assertion, a Boolean value, as follows. If options.userVerification is set to required, let userVerification be true. If the authenticator is capable of user verification, let userVerification be true. If the user is not capable of user verification, let userVerification be false. If userVerification is set to discouraged, let userVerification be false.
             3. Let userPresence be a Boolean value set to the inverse of userVerification.
             4. Let allowCredentialDescriptorList be a new list.
             5. If options.allowCredentials is not empty, execute a platform-specific procedure to determine which, if any, public key credentials described by options.allowCredentials are bound to this authenticator, by matching with rpId, options.allowCredentials.id, and options.allowCredentials.type. Set allowCredentialDescriptorList to this filtered list.
             6. If allowCredentialDescriptorList is not empty
                1. Let distinctTransports be a new ordered set.
                2. If allowCredentialDescriptorList has exactly one value, let savedCredentialId be a new PublicKeyCredentialDescriptor.id and set its value to allowCredentialDescriptorList[0].id's value (see here in 6.2.2 The authenticatorGetAssertion operation for more information).
allowCredentialDescriptorList, append each value, if any, of C.transports to distinctTransports.

Note: This will aggregate only distinct values of transports (for this authenticator) in distinctTransports due to the properties of ordered sets.

3. If distinctTransports is not empty

The client selects one transport value from distinctTransports, possibly incorporating local configuration, knowledge of the appropriate transport to use with authenticator in making its selection.

Then, using transport, invoke in parallel the authenticatorGetAssertion operation, with rpId, clientDataHash, allowCredentialDescriptorList, and authenticatorExtensions as parameters.

is empty

Using local configuration knowledge of the appropriate transport to use with authenticator, invoke in parallel the authenticatorGetAssertion operation on authenticator with rpId, clientDataHash, allowCredentialDescriptorList, and clientExtensions as parameters.

Note: In this case, the Relying Party did not supply a list of acceptable credential descriptors. Thus, the authenticator is being asked to exercise any credential it may possess that is bound to the Relying Party, as identified by rpId.

4. Append authenticator to issuedRequests.

17. Start a timer for adjustedTimeout milliseconds. Then execute the following steps in parallel. The task source for these tasks is the dom manipulation task source.

18. While issuedRequests is not empty, perform the following actions depending upon the adjustedTimeout timer and responses from the authenticators:

If the adjustedTimeout timer expires,

For each authenticator in issuedRequests invoke the authenticatorCancel operation on authenticator and remove authenticator from issuedRequests.

1. For each credential descriptor C in allowCredentialDescriptorList, append each value, if any, of C.transports to distinctTransports.

Note: This will aggregate only distinct values of transports (for this authenticator) in distinctTransports due to the properties of ordered sets.

2. If distinctTransports is not empty

The client selects one transport value from distinctTransports, possibly incorporating local configuration, knowledge of the appropriate transport to use with authenticator in making its selection.

Then, using transport, invoke the authenticatorGetAssertion operation on authenticator, with rpId, clientDataHash, allowCredentialDescriptorList, userPresence, userVerification, and authenticatorExtensions as parameters.

is empty

Using local configuration knowledge of the appropriate transport to use with authenticator, invoke the authenticatorGetAssertion operation on authenticator with rpId, clientDataHash, allowCredentialDescriptorList, userPresence, userVerification, and clientExtensions as parameters.

Note: In this case, the Relying Party did not supply a list of acceptable credential descriptors. Thus, the authenticator is being asked to exercise any credential it may possess that is bound to the Relying Party, as identified by rpId.

7. Append authenticator to issuedRequests.

18. While issuedRequests is not empty, perform the following actions depending upon lifetimeTimer and responses from the authenticators:

If lifetimeTimer expires,

For each authenticator in issuedRequests invoke the authenticatorCancel operation on authenticator and remove authenticator from issuedRequests.

If the signal member is present and the aborted flag is set to
If any authenticator returns a status indicating that the user cancelled the operation,
1. Remove authenticator from issuedRequests.
If any authenticator indicates success,
1. Remove authenticator from issuedRequests.
2. Let value be a new PublicKeyCredential associated with
   global whose fields are:
   
   [[identifier]]
   A new ArrayBuffer, created using global's
   %ArrayBuffer%, containing the bytes of the
   credential ID returned from the successful
   authenticatorGetAssertion operation, as
   defined in 6.2.2 The
   authenticatorGetAssertion operation.
   
   response
   A new AuthenticatorAssertionResponse object

associated with global whose fields are:

clientDataJSON
A new ArrayBuffer, created using
global's %ArrayBuffer%, containing the
Bytes of clientDataJSON

true,
For each authenticator in issuedRequests invoke the
authenticatorCancel operation on authenticator and remove
authenticator from issuedRequests. Then return a
DOMException whose name is "AbortError" and terminate this
algorithm.

If any authenticator returns a status indicating that the user cancelled the operation,
1. Remove authenticator from issuedRequests.
2. For each remaining authenticator in issuedRequests invoke
   the authenticatorCancel operation on authenticator and remove
   it from issuedRequests.
If any authenticator returns an error status,
Remove authenticator from issuedRequests.
If any authenticator indicates success,
1. Remove authenticator from issuedRequests.
2. Let assertionCreationData be a struct whose items are:
   
   credentialIdResult
   if savedCredentialId exists, set the value of credentialIdResult to be the bytes of the
   savedCredentialId. Otherwise, set the value of credentialIdResult to be the bytes of the
   credential ID returned from the successful authenticatorGetAssertion operation, as
   defined in 6.2.2 The
   authenticatorGetAssertion operation.

   clientDataJSONResult
   whose value is the bytes of clientDataJSON.

   authenticatorDataResult
   whose value is the bytes of the authenticator data returned by the authenticator.

   signatureResult
   whose value is the bytes of the signature value returned by the authenticator.

   userHandleResult
   whose value is the bytes of the user handle returned by the authenticator.

   clientExtensionResults
   whose value is an AuthenticationExtensions object containing extension identifier ...
   client extension output entries. The entries are created by running each extension's client
   extension processing algorithm to create the client extension outputs, for each client
   extension in clientDataJSON.clientExtensions.

   3. Let constructAssertionAlg be an algorithm that takes a
global object global, and whose steps are:
   
   1. Let pubKeyCred be a new PublicKeyCredential object
      associated with global whose fields are:
      
      [[identifier]]
      A new ArrayBuffer, created using
      global's %ArrayBuffer%, containing the
      Bytes of assertionCreationData.credentialIdResult

23/109
4.1.5. Platform Authenticator Availability - PublicKeyCredential's

authenticator with which to complete the operation.

During the above process, the user agent SHOULD show some UI to the
user to guide them in the process of selecting and authorizing an
authenticator with which to complete the operation.

19. Return a DOMException whose name is "NotAllowedError".

During the above process, the user agent SHOULD show some UI to the
user to guide them in the process of selecting and authorizing an
authenticator with which to complete the operation.

19. Return a DOMException whose name is "NotAllowedError".

2. Return publicKeyCredential.

5. Return constructAssertionAlg and terminate this algorithm.

19. Return a DOMException whose name is "NotAllowedError".

During the above process, the user agent SHOULD show some UI to the
user to guide them in the process of selecting and authorizing an
authenticator with which to complete the operation.

5.1.5. Store an existing credential - PublicKeyCredential's

[[Store]](credential, sameOriginWithAncestors) method

The [[Store]](credential, sameOriginWithAncestors) method is not
supported for Web Authentication's PublicKeyCredential type, so it
always returns an error.

Note: This algorithm is synchronous; the Promise resolution/rejection
is handled by navigator.credentials.store().

This internal method accepts two arguments:

- credential
- This argument is a PublicKeyCredential object.

3. For each remaining authenticator in issuedRequests invoke

the authenticatorCancel operation on authenticator and
remove it from issuedRequests.

4. Return value and terminate this algorithm.

19. Return a DOMException whose name is "NotAllowedError".

5. Return constructAssertionAlg and terminate this algorithm.

A new AuthenticatorAssertionResponse object associated with global whose
fields are:

- data
- A new ArrayBuffer, created using
global’s %ArrayBuffer%, containing
the bytes of
assertionCreationData.clientDataJSON
ONResult.

- authenticatorData
- A new ArrayBuffer, created using
global’s %ArrayBuffer%, containing the
bytes of the returned authenticatorData

- signature
- A new ArrayBuffer, created using
global’s %ArrayBuffer%, containing the
bytes of the returned signature

- clientExtensionResults
- A new AuthenticationExtensions object
containing the extension identifier -> client
extension output entries created by running
each extension’s client extension processing
algorithm to create the client extension
outputs, for each client extension in
clientDataJSON.clientExtensions.

2. Return publicKeyCredential.

4. For each remaining authenticator in issuedRequests invoke
the authenticatorCancel operation on authenticator and
remove it from issuedRequests.

5. Return constructAssertionAlg and terminate this algorithm.

A new ArrayBuffer, created using
global’s %ArrayBuffer%, containing the
bytes of
assertionCreationData.userHandleResult.

- userHandle
- A new ArrayBuffer, created using
global’s %ArrayBuffer%, containing the
bytes of
assertionCreationData.signatureResults.

- assertion
- A new ArrayBuffer, created using
global’s %ArrayBuffer%, containing the
bytes of
assertionCreationData.authenticatorData

- assertionCreationData
- Client extension output entries created by running
the bytes of
extensionProcessing algorithm to create the
client extension
outputs, for each client extension in
clientDataJSON.clientExtensions.

- extensionProcessing
- The algorithm to create the client extension
outputs, for each client extension in
clientDataJSON.clientExtensions.

- authenticatorData
- A new ArrayBuffer, created using
global’s %ArrayBuffer%, containing the
bytes of the returned authenticatorData

- signature
- A new ArrayBuffer, created using
global’s %ArrayBuffer%, containing the
bytes of the returned signature

- clientExtensionResults
- A new AuthenticationExtensions object
containing the extension identifier -> client
extension output entries created by running
each extension’s client extension processing
algorithm to create the client extension
outputs, for each client extension in
clientDataJSON.clientExtensions.

3. For each remaining authenticator in issuedRequests invoke

the authenticatorCancel operation on authenticator and
remove it from issuedRequests.

4. Return value and terminate this algorithm.

19. Return a DOMException whose name is "NotAllowedError".
The AuthenticatorAttestationResponse interface represents the

4.2.1. Information about Public Key Credential (interface AuthenticatorResponse)

[Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();

5.1.6. Availability of User-Verifying Platform Authenticator -

PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable) method

Relying Parties use this method to determine whether they can create a
new credential using a user-verifying platform authenticator. Upon
invocation, the client employs a platform-specific procedure to
discover available user-verifying platform authenticators. If
successful, the client then assesses whether the user is willing to
create a credential using one of the available user-verifying platform
authenticators. If this assessment is affirmative, the promise is
resolved with the value of True. Otherwise, the promise is resolved with the
value of False.

If the promise will return False, the client SHOULD wait a fixed period
of time from the invocation of the method before returning False. This
is done so that callers can not distinguish between the case where the
user was unwilling to create a credential using one of the available
platform authenticators and the case where no platform authenticator
exists. Trying to make these cases indistinguishable is done in an
attempt to not provide additional information that could be used for
fingerprinting. A timeout value on the order of 10 minutes is
recommended; this is enough time for successful user interactions to be
done so that callers can not distinguish between the case where the
dangling promise will still be unresolved in a reasonably timely fashion.

[SecureContext]

partial interface PublicKeyCredential {
    [Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();
};

5.2. Authentication Responses (interface AuthenticatorResponse)

Authenticators respond to Relying Party requests by returning an object
derived from the AuthenticatorResponse interface:

[SecureContext]

interface AuthenticatorResponse {
    [SameObject] readonly attribute ArrayBuffer clientDataJSON;
};

clientDataJSON, of type ArrayBuffer, readonly

This attribute contains a JSON serialization of the client data
passed to the authenticator by the client in its call to either
create() or get() function.

4.2.1. Information about Public Key Credential (interface

AuthenticatorAttestationResponse)

The AuthenticatorAttestationResponse interface represents the

Relying Parties use this method to determine whether they can create a
new credential using a platform authenticator. Upon
invocation, the client employs a platform-specific procedure to
discover available platform authenticators. If
successful, the client then assesses whether the user is willing to
create a credential using one of the available platform authenticators. This assessment may include various factors, such as:

* Whether the user is running in private or incognito mode.
* Whether the user has previously expressed an unwillingness to
  create a new credential for this Relying Party, as per their
  configuration or by declining a user interface prompt.
* The user's explicitly stated intentions, determined through user
  interaction.

If this assessment is affirmative, the promise is resolved with the
value of True. Otherwise, the promise is resolved with the value of
False. Based on the result, the Relying Party can take further actions
to guide the user to create a credential.

This method has no arguments and returns a boolean value.

If the promise will return False, the client SHOULD wait a fixed period
of time from the invocation of the method before returning False. This
is done so that callers can not distinguish between the case where the
user was unwilling to create a credential using one of the available
platform authenticators and the case where no platform authenticator
exists. Trying to make these cases indistinguishable is done in an
attempt to not provide additional information that could be used for
fingerprinting. A timeout value on the order of 10 minutes is
recommended; this is enough time for successful user interactions to be
done so that callers can not distinguish between the case where the
dangling promise will still be unresolved in a reasonably timely fashion.

[SecureContext]

partial interface PublicKeyCredential {
    [Unscopable] Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();
};

5.2. Authentication Responses (interface AuthenticatorResponse)

Authenticators respond to Relying Party requests by returning an object
derived from the AuthenticatorResponse interface:

[SecureContext, Exposed=Window]

interface AuthenticatorResponse {
    [SameObject] readonly attribute ArrayBuffer clientDataJSON;
};

clientDataJSON, of type ArrayBuffer, readonly

This attribute contains a JSON serialization of the client data
passed to the authenticator by the client in its call to either
create() or get() function.

4.2.1. Information about Public Key Credential (interface

AuthenticatorAttestationResponse)

The AuthenticatorAttestationResponse interface represents the

Relying Parties use this method to determine whether they can create a
new credential using a platform authenticator. Upon
invocation, the client employs a platform-specific procedure to
discover available platform authenticators. If
successful, the client then assesses whether the user is willing to
create a credential using one of the available platform authenticators. This assessment may include various factors, such as:

* Whether the user is running in private or incognito mode.
* Whether the user has previously expressed an unwillingness to
  create a new credential for this Relying Party, as per their
  configuration or by declining a user interface prompt.
* The user's explicitly stated intentions, determined through user
  interaction.

If this assessment is affirmative, the promise is resolved with the
value of True. Otherwise, the promise is resolved with the value of
False. Based on the result, the Relying Party can take further actions
to guide the user to create a credential.

This method has no arguments and returns a boolean value.

If the promise will return False, the client SHOULD wait a fixed period
of time from the invocation of the method before returning False. This
is done so that callers can not distinguish between the case where the
user was unwilling to create a credential using one of the available
platform authenticators and the case where no platform authenticator
exists. Trying to make these cases indistinguishable is done in an
attempt to not provide additional information that could be used for
fingerprinting. A timeout value on the order of 10 minutes is
recommended; this is enough time for successful user interactions to be
done so that callers can not distinguish between the case where the
dangling promise will still be unresolved in a reasonably timely fashion.

[SecureContext]

partial interface PublicKeyCredential {
    [Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();
};

5.2. Authentication Responses (interface AuthenticatorResponse)

Authenticators respond to Relying Party requests by returning an object
derived from the AuthenticatorResponse interface:

[SecureContext, Exposed=Window]

interface AuthenticatorResponse {
    [SameObject] readonly attribute ArrayBuffer clientDataJSON;
};

clientDataJSON, of type ArrayBuffer, readonly

This attribute contains a JSON serialization of the client data
passed to the authenticator by the client in its call to either
create() or get() function.

4.2.1. Information about Public Key Credential (interface

AuthenticatorAttestationResponse)

The AuthenticatorAttestationResponse interface represents the
4.3. Parameters for Credential Generation

The Authenticator's response to a client's request for the creation of a new public key credential. It contains information about the new credential that can be used to identify it for later use, and metadata that can be used by the Relying Party to assess the characteristics of the credential during registration.

interface AuthenticatorAttestationResponse : AuthenticatorResponse {

    [SameObject] readonly attribute ArrayBuffer attestationObject;
};

clientDataJSON

This attribute, inherited from AuthenticatorResponse, contains the JSON-serialized client data (see 5.3 Attestation) passed to the authenticator by the client in order to generate this credential. The exact JSON serialization must be preserved, as the hash of the serialized client data has been computed over it.

attestationObject, of type ArrayBuffer, readonly

This attribute contains an attestation object, which is opaque to, and cryptographically protected against tampering by, the client. The attestation object contains both authenticator data and an attestation statement. The former contains the AAGUID, a unique credential ID, and the cryptographic public key. The latter contains the attestation statement format used by the authenticator. It also contains any additional information that the Relying Party's server requires to validate the attestation statement, as well as to decode and validate the authenticator data along with the JSON-serialized client data. For more details, see 5.3 Attestation, 5.4 Generating an Attestation Object, and Figure 3.

4.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)

The AuthenticatorAssertionResponse interface represents an authenticator's response to a client's request for the creation of a new authentication assertion given the Relying Party's challenge and optional list of credentials it is aware of. This response contains a cryptographic signature proving possession of the credential private key, and optionally evidence of user consent to a specific transaction.

interface AuthenticatorAssertionResponse : AuthenticatorResponse {

    [SameObject] readonly attribute ArrayBuffer authenticatorData;

    [SameObject] readonly attribute ArrayBuffer signature;
};

clientDataJSON

This attribute, inherited from AuthenticatorResponse, contains the JSON-serialized client data (see 4.7.1 Client data used in WebAuthn signatures (dictionary CollectedClientData)) passed to the authenticator by the client in order to generate this assertion. The exact JSON serialization must be preserved, as the hash of the serialized client data has been computed over it.

authenticatorData, of type ArrayBuffer, readonly

This attribute contains the raw signature returned by the authenticator. See 5.1 Authenticator data.

signature, of type ArrayBuffer, readonly

This attribute contains the raw signature returned from the authenticator. See 5.2.2 The authenticatorGetAssertion operation.

4.3. Parameters for Credential Generation

This attribute, inherited from AuthenticatorResponse, contains the JSON-serialized client data (see 5.3 Attestation) passed to the authenticator by the client in order to generate this credential. The exact JSON serialization must be preserved, as the hash of the serialized client data has been computed over it.

attestationObject, of type ArrayBuffer, readonly

This attribute contains an attestation object, which is opaque to, and cryptographically protected against tampering by, the client. The attestation object contains both authenticator data and an attestation statement. The former contains the AAGUID, a unique credential ID, and the cryptographic public key. The latter contains the attestation statement format used by the authenticator. It also contains any additional information that the Relying Party's server requires to validate the attestation statement, as well as to decode and validate the authenticator data along with the JSON-serialized client data. For more details, see 5.3 Attestation, 5.4 Generating an Attestation Object, and Figure 3.

5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)

The AuthenticatorAssertionResponse interface represents an authenticator's response to a client's request for generation of a new authentication assertion given the Relying Party's challenge and optional list of credentials it is aware of. This response contains a cryptographic signature proving possession of the credential private key, and optionally evidence of user consent to a specific transaction.

interface AuthenticatorAssertionResponse : AuthenticatorResponse {

    [SameObject] readonly attribute ArrayBuffer authenticatorData;

    [SameObject] readonly attribute ArrayBuffer signature;

    [SameObject] readonly attribute ArrayBuffer userHandle;
};

clientDataJSON

This attribute, inherited from AuthenticatorResponse, contains the JSON-serialized client data (see 5.1 Client data used in WebAuthn signatures (dictionary CollectedClientData)) passed to the authenticator by the client in order to generate this assertion. The exact JSON serialization must be preserved, as the hash of the serialized client data has been computed over it.

authenticatorData, of type ArrayBuffer, readonly

This attribute contains the raw signature returned by the authenticator. See 5.1 Authenticator data.

signature, of type ArrayBuffer, readonly

This attribute contains the raw signature returned from the authenticator. See 5.2.2 The authenticatorGetAssertion operation.
Public Key Credential Parameters

dictionary PublicKeyCredentialParameters {
    required PublicKeyCredentialType type;
    required COSEAlgorithmIdentifier alg;
};

This dictionary is used to supply additional parameters when creating a new credential.

The type member specifies the type of credential to be created.

The alg member specifies the cryptographic signature algorithm with which the newly generated credential will be used, and thus also the type of asymmetric key pair to be generated, e.g., RSA or Elliptic Curve.

Note: we use "alg" as the latter member name, rather than spelling-out "algorithm", because it will be serialized into a message to the authenticator, which may be sent over a low-bandwidth link.

4.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions)

dictionary MakePublicKeyCredentialOptions {
    required PublicKeyCredentialEntity rp;
    required PublicKeyCredentialUserEntity user;
    required BufferSource challenge;
    required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
    unsigned long timeout;
    sequence<PublicKeyCredentialDescriptor> excludeCREDENTIALS = [];
    AuthenticatorSelectionCriteria authenticatorSelection;
    AuthenticationExtensions extensions;
};

rp, of type PublicKeyCredentialEntity

This member contains data about the Relying Party responsible for the request.

Its value's name member is required, and contains the friendly name of the Relying Party (e.g. "Acme Corporation", "Widgets, Inc.", or "Awesome Site").

Its value's id member specifies the relying party identifier with which the credential should be associated. If omitted, its value will be the CredentialsContainer object's relevant settings object's origin's effective domain.

user, of type PublicKeyCredentialUserEntity

This member contains data about the user account for which the Relying Party is requesting attestation.

Its value's name member is required, and contains a name for the user account (e.g., "John P. Smith").

Its value's displayName member contains a friendly name for the user account (e.g., "John P. Smith").

Its value's id member is required, and contains an identifier for the account, specified by the Relying Party. This is not meant to be displayed to the user, but is used by the Relying Party to control the number of credentials - an authenticator will never contain more than one credential for a given Relying Party.

5.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions)

dictionary MakePublicKeyCredentialOptions {
    required PublicKeyCredentialRPEntity rp;
    required PublicKeyCredentialUserEntity user;
    required BufferSource challenge;
    required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
    unsigned long timeout;
    sequence<PublicKeyCredentialDescriptor> excludeCREDENTIALS = [];
    AuthenticatorSelectionCriteria authenticatorSelection;
    AuthenticationExtensions extensions;
};

rp, of type PublicKeyCredentialRPEntity

This member contains data about the Relying Party responsible for the request.

Its value's name member contains the friendly name of the Relying Party (e.g. "Acme Corporation", "Widgets, Inc.", or "Awesome Site").

Its value's id member specifies the relying party identifier with which the credential should be associated. If omitted, its value will be the CredentialsContainer object's relevant settings object's origin's effective domain.

user, of type PublicKeyCredentialUserEntity

This member contains data about the user account for which the Relying Party is requesting attestation.

Its value's name member contains the name of the user account (e.g., "john.p.smith@example.com") or "+14255551234".

Its value's displayName member contains a friendly name for the user account (e.g., "John P. Smith").

Its value's id member contains the user handle for the account, specified by the Relying Party.
4.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

The PublicKeyCredentialEntity dictionary describes a user account, or a Relying Party, with which a public key credential is associated.

```javascript
{  id: DOMString,  name: DOMString,  icon: USVString },
```

Id, of type DOMString
A unique identifier for the entity. For a relying party entity, sets the RP ID. For a user account entity, this will be an arbitrary string specified by the relying party.

name, of type DOMString
A human-friendly identifier for the entity. For example, this could be a company name for a Relying Party, or a user's name.

This identifier is intended for display.

4.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity) (continued)

This member contains additional parameters requesting additional processing by the client and authenticator. For example, the caller may request that only authenticators with certain capabilities be used to create the credential, or that particular information be returned in the attestation object. Some extensions are defined in 9 WebAuthn Extensions; consult the IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registries] for an up-to-date list of registered WebAuthn Extensions.

extensions, of type AuthenticationExtensions
This member contains additional parameters requesting additional processing by the client and authenticator. For example, the caller may request that only authenticators with certain capabilities be used to create the credential, or that particular information be returned in the attestation object. Some extensions are defined in 9 WebAuthn Extensions; consult the IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registries] for an up-to-date list of registered WebAuthn Extensions.

This member is intended for use by Relying Parties that wish to select the appropriate authenticators to participate in the create() or get() operation.

excludeCredentials, of type sequence<PublicKeyCredentialDescriptor>, defaulting to None
This member is intended for use by Relying Parties that wish to limit the creation of multiple credentials for the same account on a single authenticator. The platform is requested to return an error if the new credential would be created on an authenticator that also contains one of the credentials enumerated in this parameter.

This member contains additional parameters requesting additional processing by the client and authenticator. For example, the caller may request that only authenticators with certain capabilities be used to create the credential, or that particular information be returned in the attestation object. Some extensions are defined in 9 WebAuthn Extensions; consult the IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registries] for an up-to-date list of registered WebAuthn Extensions.

This member is intended for use by Relying Parties that wish to limit the creation of multiple credentials for the same account on a single authenticator. The platform is requested to return an error if the new credential would be created on an authenticator that also contains one of the credentials enumerated in this parameter.

This member is intended for use by Relying Parties that wish to select the appropriate authenticators to participate in the create() or get() operation.

attestation, of type AttestationConveyancePreference, defaulting to "none"
This member is intended for use by Relying Parties that wish to express their preference for attestation conveyance. The default is none.

extensions, of type AuthenticationExtensions
This member contains additional parameters requesting additional processing by the client and authenticator. For example, the caller may request that only authenticators with certain capabilities be used to create the credential, or that particular information be returned in the attestation object. Some extensions are defined in 9 WebAuthn Extensions; consult the IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registries] for an up-to-date list of registered WebAuthn Extensions.

This member is intended for use by Relying Parties that wish to limit the creation of multiple credentials for the same account on a single authenticator. The platform is requested to return an error if the new credential would be created on an authenticator that also contains one of the credentials enumerated in this parameter.

This member contains additional parameters requesting additional processing by the client and authenticator. For example, the caller may request that only authenticators with certain capabilities be used to create the credential, or that particular information be returned in the attestation object. Some extensions are defined in 9 WebAuthn Extensions; consult the IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registries] for an up-to-date list of registered WebAuthn Extensions.

This member is intended for use by Relying Parties that wish to select the appropriate authenticators to participate in the create() or get() operation.

```javascript
{  id: DOMString,  name: DOMString,  icon: USVString },
```

Id, of type DOMString
A unique identifier for the entity. For a relying party entity, sets the RP ID. For a user account entity, this will be an arbitrary string specified by the relying party.

name, of type DOMString
A human-friendly identifier for the entity. For example, this could be a company name for a Relying Party, or a user's name.

This identifier is intended for display.

Accept and store a 64 byte minimum length for a name member's value. Authenticators MAY truncate a name member's value to a length equal to or greater than 64 bytes.

This member contains a challenge intended to be used for generating the newly created credential's attestation object.

```javascript
{  id: DOMString,  name: DOMString,  icon: USVString },
```

Id, of type DOMString
A unique identifier for the entity. For a relying party entity, sets the RP ID. For a user account entity, this will be an arbitrary string specified by the relying party.

name, of type DOMString
A human-friendly identifier for the entity. For example, this could be a company name for a Relying Party, or a user's name.

This identifier is intended for display. Authenticators MUST accept and store a 64 byte minimum length for a name member's value. Authenticators MAY truncate a name member's value to a length equal to or greater than 64 bytes.

This member contains a challenge intended to be used for generating the newly created credential's attestation object.

```javascript
{  id: DOMString,  name: DOMString,  icon: USVString },
```

Id, of type DOMString
A unique identifier for the entity. For a relying party entity, sets the RP ID. For a user account entity, this will be an arbitrary string specified by the relying party.

name, of type DOMString
A human-friendly identifier for the entity. For example, this could be a company name for a Relying Party, or a user's name.

This identifier is intended for display. Authenticators MUST accept and store a 64 byte minimum length for a name member's value. Authenticators MAY truncate a name member's value to a length equal to or greater than 64 bytes.

This member contains a challenge intended to be used for generating the newly created credential's attestation object.

```javascript
{  id: DOMString,  name: DOMString,  icon: USVString },
```

Id, of type DOMString
A unique identifier for the entity. For a relying party entity, sets the RP ID. For a user account entity, this will be an arbitrary string specified by the relying party.

name, of type DOMString
A human-friendly identifier for the entity. For example, this could be a company name for a Relying Party, or a user's name.

This identifier is intended for display. Authenticators MUST accept and store a 64 byte minimum length for a name member's value. Authenticators MAY truncate a name member's value to a length equal to or greater than 64 bytes.
4.4.2. User Account Parameters for Credential Generation (dictionary)

4.4.3. Authenticator Selection Criteria (dictionary)

4.4.4. RP Parameters for Credential Generation (dictionary)

4.4.5. User Account Parameters for Credential Generation (dictionary)
4.5. Options for Assertion Generation

dictionary

4.5.1. Options for Assertion Generation (dictionary)

A Credential's options must specify whether an assertion - PublicKeyCredential's

MUST perform user verification when performing the create

operation and future 4.1.4 Use an existing credential to make

an assertion - PublicKeyCredential's

[DiscoverFromExternalSource](options) method operations when

it is requested to verify the credential.

Note: These identifiers are intentionally short, rather than

descriptive, because they will be serialized into a message to the

authentication, which may be sent over a low-bandwidth link.

4.4.4. Authenticator Attachment enumeration (enum AuthenticatorAttachment)

enum AuthenticatorAttachment {

 plat,  // Platform attachment

cross-platform,  // Cross-platform attachment

};

Clients may communicate with authenticators using a variety of

mechanisms. For example, a client may use a platform-specific API to

communicate with an authenticator which is physically bound to a

platform. On the other hand, a client may use a variety of standardized

cross-platform transport protocols such as Bluetooth (see 4.7.4

Authenticator Transport enumeration (enum AuthenticatorTransport)) to

discover and communicate with cross-platform attached authenticators.

Therefore, we use AuthenticatorAttachment to describe an

authenticator’s attachment modality. We define authenticators that are

part of the client’s platform as having a platform attachment, and

refer to them as platform authenticators. While those that are

reachable via cross-platform transport protocols are defined as having

cross-platform attachment, and refer to them as roaming authenticators.

* cross-platform attachment - the respective authenticator is attached

using platform-specific transports. Usually, authenticators of this

class are non-removable from the platform.

* platform attachment - the respective authenticator is attached using cross-platform transports. Authenticators of this

class are removable from, and can “roam” among, client platforms.

This distinction is important because there are use-cases where only

platform authenticators are acceptable to a Relying Party, and

conversely ones where only roaming authenticators are employed. As a

concrete example of the former, a credential on a platform

authenticator may be used by Relying Parties to quickly and

conveniently reauthenticate the user with a minimum of friction, e.g.,

the user will not have to dig around in their pocket for their key fob

or phone. As a concrete example of the latter, when the user is

accessing the Relying Party for the first time, they may be required to use a roaming authenticator which was

originally registered with the Relying Party using a different client.

4.6. Options for Assertion Generation (dictionary)

5.4.6. Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference)

enum AttestationConveyancePreference {

 "none",

 "direct",

 "indirect",

 "indirect - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"direct - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"indirect - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"direct - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"direct - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"indirect - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"direct - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"direct - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"indirect - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"indirect - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"direct - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"direct - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.

"indirect - indicates that the Relying Party prefers an attestation

process, but this preference is not binding and the Relying Party

may decide to relay, another attestation process, such as

platform-specific attestation or attestation via a trusted attestation

channel.
PublicKeyCredentialRequestOptions)

The PublicKeyCredentialRequestOptions dictionary supplies get( ) with the data it needs to generate an assertion. Its challenge member must be present, while its other members are optional.

dictionary PublicKeyCredentialRequestOptions {
  required BufferSource challenge;
  unsigned long timeout;
  USVString rpId;
  sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
  AuthenticationExtensions extensions;
};

challenge, of type BufferSource
  This member represents a challenge that the selected authenticator signs, along with other data, when producing an authentication assertion.

timeout, of type unsigned long
  This optional member specifies a time, in milliseconds, that the caller is willing to wait for the call to complete. The value is treated as a hint, and may be overridden by the platform.

rpId, of type USVString
  This optional member specifies the relying party identifier claimed by the caller. If omitted, its value will be the origin’s effective domain.

allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,
  defaulting to None
  This optional member contains a list of PublicKeyCredentialDescriptor objects representing public key credentials acceptable to the caller, in descending order of the caller’s preference (the first item in the list is the most preferred credential, and so on down the list).

extensions, of type AuthenticationExtensions
  This optional member contains additional parameters requesting additional processing by the client and authenticator. For example, if transaction confirmation is sought from the user, then the prompt string might be included as an extension.

4.6. Authentication Extensions (typedef AuthenticationExtensions)
typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> AuthenticationExtensions;

dictionary CollectedClientData {
    string-valued keys. Values may be any type that has a valid encoding in Party and the client platform. It is a key-value mapping with 
}

dict CollectedClientData)

typedef record<DOMString, any> Authentication Extensions (typedef AuthenticationExtensions)
The challenge member contains the base64url encoding of the challenge provided by the RP.

The origin member contains the fully qualified origin of the requestor, as provided to the authenticator by the client, in the syntax defined by [RFC6454].

The hashAlgorithm member is a recognized algorithm name that supports the "digest" operation, which specifies the algorithm used to compute the hash of the serialized client data. This algorithm is chosen by the client at its sole discretion.

The tokenBindingId member contains the base64url encoding of the Token Binding ID that this client uses for the Token Binding protocol when communicating with the Relying Party. This can be omitted if no Token Binding has been negotiated between the client and the Relying Party.

The optional clientExtensions and authenticatorExtensions members contain additional parameters generated by processing the extensions passed in by the Relying Party. WebAuthn extensions are detailed in Section 8 WebAuthn Extensions.

This structure is used by the client to compute the following quantities:

- JSON-serialized client data
- This is the UTF-8 encoding of the result of calling the initial value of JSON.stringify on a CollectedClientData dictionary.
- Hash of the serialized client data
- This is the hash (computed using hashAlgorithm) of the JSON-serialized client data, as constructed by the client.

### 4.7.2. Credential Type enumeration (enum PublicKeyCredentialType)

enum PublicKeyCredentialType {
  "public-key"
};

This enumeration defines the valid credential types. It is an extension point; values may be added to it in the future, as more credential types are defined. The values of this enumeration are used for versioning the Authentication Assertion and attestation structures according to the type of the authenticator.

Currently one credential type is defined, namely "public-key".

### 4.7.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)

dictionary PublicKeyCredentialDescriptor {
  required PublicKeyCredentialType type;
  required BufferSource id;
  sequence<AuthenticatorTransport> transports;
};

This dictionary contains the attributes that are specified by a caller when referring to a credential as an input parameter to the create() or get() methods. It mirrors the fields of the PublicKeyCredential object returned by the latter methods.

The type member contains the type of the credential the caller is referring to.

The id member contains the identifier of the credential that the caller is referring to.

### 4.7.4. Authenticator Transport enumeration (enum AuthenticatorTransport)

These enumerations are referring to.

The hashAlgorithm member is a recognized algorithm name that supports the "digest" operation, which specifies the algorithm used to compute the hash of the serialized client data. This algorithm is chosen by the client at its sole discretion.

The tokenBindingId member contains the base64url encoding of the Token Binding ID that this client uses for the Token Binding protocol when communicating with the Relying Party. This can be omitted if no Token Binding has been negotiated between the client and the Relying Party.

The optional clientExtensions and authenticatorExtensions members contain additional parameters generated by processing the extensions passed in by the Relying Party. WebAuthn extensions are detailed in Section 8 WebAuthn Extensions.

This structure is used by the client to compute the following quantities:

- JSON-serialized client data
- This is the UTF-8 encoding of the result of calling the initial value of JSON.stringify on a CollectedClientData dictionary.
- Hash of the serialized client data
- This is the hash (computed using hashAlgorithm) of the JSON-serialized client data, as constructed by the client.

### 5.8.2. Credential Type enumeration (enum PublicKeyCredentialType)

enum PublicKeyCredentialType {
  "public-key"
};

This enumeration defines the valid credential types. It is an extension point; values may be added to it in the future, as more credential types are defined. The values of this enumeration are used for versioning the Authentication Assertion and attestation structures according to the type of the authenticator.

Currently one credential type is defined, namely "public-key".

### 5.8.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)

dictionary PublicKeyCredentialDescriptor {
  required PublicKeyCredentialType type;
  required BufferSource id;
  sequence<AuthenticatorTransport> transports;
};

This dictionary contains the attributes that are specified by a caller when referring to a credential as an input parameter to the create() or get() methods. It mirrors the fields of the PublicKeyCredential object returned by the latter methods.

The type member contains the type of the credential the caller is referring to.

The id member contains the identifier of the credential that the caller is referring to.

### 5.8.4. Authenticator Transport enumeration (enum AuthenticatorTransport)
The API defined in this specification implies a specific abstract functional model for an authenticator. This section describes the authenticator model.

Client platforms may implement and expose this abstract model in any way desired. However, the behavior of the client’s Web Authentication API implementation, when operating on the authenticators supported by that platform, MUST be indistinguishable from the behavior specified in 4 Web Authentication API.

For authenticators, this model defines the logical operations that they must support, and the data formats that they expose to the client and the Relying Party. However, it does not define the details of how
Each authenticator stores some number of public key credentials. Each public key credential has an identifier which is unique (or extremely unlikely to be duplicated) among all public key credentials. Each credential is also associated with a relying party, whose identity is represented by a relying party identifier (RP ID).

Each authenticator has an AAGUID, which is a 128-bit identifier that indicates the type (e.g. make and model) of the authenticator. The AAGUID MUST be chosen by the manufacturer to be identical across all substantially identical authenticators made by that manufacturer, and different (with probability 1-2^-128 or greater) from the AAGUIDs of all other types of authenticators. The RP MAY use the AAGUID to infer certain properties of the authenticator, such as certification level and strength of key protection, using information from other sources.

The primary function of the authenticator is to provide WebAuthn signatures, which are bound to various contextual data. These data are obtained, and added at different levels of the authenticator, for request passes from the server to the authenticator. In verifying a signature, the server checks these bindings against expected values. These contextual bindings are divided into two: Those added by the RP or the client, referred to as client data; and those added by the authenticator, referred to as authenticator data. The authenticator signs over the client data, but is otherwise not interested in its contents. To save bandwidth and processing requirements on the authenticator, the client hashes the client data and sends only the result to the authenticator. The authenticator signs over the combination of the hash of the serialized client data, and its own authenticator data.

The goals of this design can be summarized as follows.

- The scheme for generating signatures should accommodate cases where the link between the client platform and authenticator is very limited, in bandwidth and/or latency. Examples include Bluetooth Low Energy and Near-Field Communication.
- The data processed by the authenticator should be small and easy to interpret in low-level code. In particular, authenticators should not have to parse high-level encodings such as JSON.
- Both the client platform and authenticator should have the flexibility to add contextual bindings as needed.
- The design aims to reuse as much as possible of existing encoding formats in order to aid adoption and implementation.

Authenticators produce cryptographic signatures for two distinct purposes:
1. An attestation signature is produced when a new public key credential is created via an authenticatorMakeCredential operation.
2. An attestation signature provides cryptographic proof of certain properties of the the authenticator and the credential. For
instance, an attestation signature asserts the authenticator type (as denoted by its AAGUID) and the credential public key. The attestat
signature is signed by an attestation private key, which is chosen depending on the type of attestation desired. For more details on attestation, see 5.3 Attestation.

2. An assertion signature is produced when the authenticatorGetAssertion method is invoked. It represents an assertion by the authenticator that the user has consented to a specific transaction, such as logging in, or completing a purchase. Thus, an assertion signature asserts that the authenticator possesses a particular credential private key and has established, to the best of its ability, that the user requesting this transaction is the same user who consented to creating that particular public key credential. It also asserts additional information, termed client data, that may be useful to the caller, such as the means by which user consent was provided, and the prompt shown to the user by the authenticator. The assertion signature format is illustrated in Figure 2, below.

The formats of these signatures, as well as the procedures for generating them, are specified below.

5.1. Authenticator data

The authenticator data structure encodes contextual bindings made by the authenticator. These bindings are controlled by the authenticator itself, and derive their trust from the Relying Party’s assessment of the security properties of the authenticator. In one extreme case, the authenticator may be embedded in the client, and its bindings may be no more trustworthy than the client data. At the other extreme, the authenticator may be a discrete entity with high-security hardware and software, connected to the client over a secure channel. In both cases, the Relying Party receives the authenticator data in the same format, and uses its knowledge of the authenticator to make trust decisions.

The authenticator data has a compact but extensible encoding. This is desirable since authenticators can have different capabilities and low power requirements, with much simpler software stacks than the client platform components.

The authenticator data structure is a byte array of 37 bytes or more, as follows.

Length (in bytes) Description
32 SHA-256 hash of the RP ID associated with the credential.
1 Flags (bit 0 is the least significant bit):
  * Bit 0: User Present (UP) result.
  * Bit 1: Reserved for future use (RFU1).
  * Bit 2: User Verified (UV) result.
  * Bit 3-5: Reserved for future use (RFU2).
  * Bit 6: Attestation data included (AT).
  * Bit 7: Extension data included (ED).

+ Indicates whether the authenticator added attestation data.
+ Indicates whether the authenticator added attestation data.

4 Signature counter (signCount), 32-bit unsigned big-endian integer.
variable (if present) attestation data (if present). See 5.3.1
for details. See 5.3.1 for details. See 5.3.1
for details. See 5.3.1 for details. This is a CBOR (RFC7049) map with extension identifiers as keys, and
authenticator extension outputs as values. See WebAuthn Extensions
for details.

instance, an attestation signature asserts the authenticator type (as denoted by its AAGUID) and the credential public key. The attestat
signature is signed by an attestation private key, which is chosen depending on the type of attestation desired. For more details on attestation, see 6.3 Attestation.

2. An assertion signature is produced when the authenticatorGetAssertion method is invoked. It represents an assertion by the authenticator that the user has consented to a specific transaction, such as logging in, or completing a purchase. Thus, an assertion signature asserts that the authenticator possesses a particular credential private key and has established, to the best of its ability, that the user requesting this transaction is the same user who consented to creating that particular public key credential. It also asserts additional information, termed client data, that may be useful to the caller, such as the means by which user consent was provided, and the prompt shown to the user by the authenticator. The assertion signature format is illustrated in Figure 2, below.

The formats of these signatures, as well as the procedures for generating them, are specified below.

6.1. Authenticator data

The authenticator data structure encodes contextual bindings made by the authenticator. These bindings are controlled by the authenticator itself, and derive their trust from the Relying Party’s assessment of the security properties of the authenticator. In one extreme case, the authenticator may be embedded in the client, and its bindings may be no more trustworthy than the client data. At the other extreme, the authenticator may be a discrete entity with high-security hardware and software, connected to the client over a secure channel. In both cases, the Relying Party receives the authenticator data in the same format, and uses its knowledge of the authenticator to make trust decisions.

The authenticator data has a compact but extensible encoding. This is desirable since authenticators can have different capabilities and low power requirements, with much simpler software stacks than the client platform components.

The authenticator data structure is a byte array of 37 bytes or more, as follows.

Name Length (in bytes) Description
rpIdHash 32 SHA-256 hash of the RP ID associated with the credential.
flags 1 Flags (bit 0 is the least significant bit):
  * Bit 0: User Present (UP) result.
  + 1 means the user is present.
  * Bit 1: Reserved for future use (RFU1).
  * Bit 2: User Verified (UV) result.
  + 1 means the user is verified.
  * Bit 3: Not user is verified.
  * Bit 4: Reserved for future use (RFU2).
  * Bit 5: Attestation data included (AT).
  + Indicates whether the authenticator added attestation data.
  + Indicates whether the authenticator added attestation data.

4 Signature counter (signCount), 32-bit unsigned big-endian integer.
variable (if present) attestation data (if present). See 5.3.1
for details. See 5.3.1 for details. See 5.3.1
for details. See 5.3.1 for details. This is a CBOR (RFC7049) map with extension identifiers as keys, and
authenticator extension outputs as values. See WebAuthn Extensions
for details.

+ Indicates whether the authenticator added attestation data.
+ Indicates whether the authenticator added attestation data.
The RP ID is originally received from the client when the credential is created, and again when an assertion is generated. However, it differs from other client data in some important ways. First, unlike the client data, the RP ID of a credential does not change between operations but instead remains the same for the lifetime of that credential. Secondly, it is validated by the authenticator during the authenticatorGetAssertion operation. By verifying that the RP ID associated with the requested credential exactly matches the RP ID requested by the client, and that the RP ID is a registerable domain suffix of or is equal to the effective domain of the RP's origin's effective domain.

The UP flag SHALL be set if and only if the authenticator detected a user through the authenticator specific gesture. The RFU bits SHALL be set to zero.

For attestation signatures, the authenticator MUST set the AT flag and include the attestation data. For authentication signatures, the AT flag MUST NOT be set and the attestation data MUST NOT be included.

If the authenticator does not include any extension data, it MUST set the ED flag to zero, and to one if extension data is included.

The figure below shows a visual representation of the authenticator data structure.

![Authenticator Data Layout](image1.png)

5.2. Authenticator operations

NOTE: The names in the Name column in the above table are only for reference within this document, and are not present in the actual representation of the authenticator data.
A client must connect to an authenticator in order to invoke any of the operations of that authenticator. This connection defines an authenticator session. An authenticator must maintain isolation between sessions. It may do this by only allowing one session to exist at any particular time, or by providing more complicated session management.

The following operations can be invoked by the client in an authenticator session.

### 5.2.1. The authenticatorMakeCredential operation

This operation must be invoked in an authenticator session which has no other operations in progress. It takes the following input parameters:

- The caller's RP ID, as determined by the user agent and the client.
- The hash of the serialized client data, provided by the client.
- The Relying Party's PublicKeyCredentialEntity.
- The user account's PublicKeyCredentialUserEntity.
- A sequence of pairs of PublicKeyCredentialType and COSEAlgorithmIdentifier requested by the Relying Party. This sequence is ordered from most preferred to least preferred. The platform makes a best-effort to create the most preferred credential that it can.
- An optional list of PublicKeyCredentialDescriptor objects provided by the Relying Party with the intention that, if any of these are known to the authenticator, it should not create a new credential.
- The `rk` member of the options.authenticatorSelection dictionary.
- A sequence of pairs of PublicKeyCredentialType and public key algorithms (COSEAlgorithmIdentifier) requested by the Relying Party. This sequence is ordered from most preferred to least preferred. The platform makes a best-effort to create the most preferred credential that it can.
- Extension data created by the client based on the extensions requested by the Relying Party, if any.

When this operation is invoked, the authenticator must perform the following procedure: 1. Check if all the supplied parameters are syntactically well-formed and of the correct length. If not, return an error code equivalent to "UnknownError" and terminate the operation.
2. If at least one of the specified combinations of PublicKeyCredentialType and cryptographic parameters is supported, allocate the counter, associate it with the RP ID, and return success.

3. Check if any credential bound to this authenticator matches an item of excludeCredentialDescriptorList. A match occurs if a credential matches rpEntity.id and an excludeCredentialDescriptorList item’s excludeCredentialDescriptorListId type. If so, return an error code equivalent to “NotAllowedError” and terminate the operation.

4. If requireResidentKey is true and the authenticator cannot create a Client-side-resident Credential Private Key, return an error code equivalent to “ConstraintError” and terminate the operation.

5. If requireUserVerification is true and the authenticator cannot perform user verification, return an error code equivalent to “ConstraintError” and terminate the operation.

6. Obtain user consent for creating a new credential. The prompt for obtaining this consent is shown by the authenticator if it has its own output capability, or by the user agent otherwise. The prompt should display a userEntity.name, userEntity.displayName, if possible.

7. If requireUserPresence is true, the method of obtaining user consent MUST include user verification. If the user denies consent or if user verification fails, return an error code equivalent to “NotAllowedError” and terminate the operation.

8. Once user consent has been obtained, generate a new credential object:
   - Generate a set of cryptographic keys using the most preferred combination of PublicKeyCredentialType and cryptographic parameters supported by this authenticator.
   - Generate an identifier for this credential, such that this identifier is globally unique with high probability across all credentials with the same type across all authenticators.
   - Associate the credential with the specified RP ID and the user’s account identifier user.id.
   - Delete any older credentials with the same RP ID and user.id that are stored locally by the authenticator.
   - If any error occurred while creating the new credential object, return an error code equivalent to “UnknownError” and terminate the operation.

9. Process all the supported extensions requested by the client, and generate the authenticator data with attestation data as specified in 5.1 Authenticator data. Use this authenticator data and the hash of the serialized client data to create an attestation object for the new credential using the procedure specified in 5.3.4.4 Generating an Attestation Object. For more details on attestation, see 5.3 Attestation.

10. If the authenticator supports:
   - a per-RP ID signature counter, allocate the counter, associate it with the RP ID, and initialize the counter value as zero.
   - a global signature counter
     * Use the global signature counter’s actual value when generating authenticator data.
     * Allocate a per credential signature counter
     * Allocate the counter, associate it with the new credential, and initialize the counter value as zero.

11. Let attestedCredentialData be the attested credential data byte array including the credentialId and publicKey.

12. Let authenticatorData be the byte array specified in 6.1 Authenticator data, including attestedCredentialData as the attestedCredentialData and processedExtensions, if any, as the extensions.

13. Return the attestation object for the new credential created by the authenticator.
Obtain user consent for using this credential. The prompt for

* Prompt the user to select a credential from among the above list.
* If the previous step resulted in an empty list, return an error code equivalent to "NotAllowedError" and terminate the operation.

credentials stored for the caller's RP ID (as determined by an
authenticator. If no list was supplied, create a list with all
removing those credentials that are not present on this
removing those credentials that are not present on this
removing those credentials that are not present on this

* If a list of credentials was supplied by the client, filter it by

To "UnknownError" and terminate the operation.

Note: Before performing this operation, all other operations in
progress in the authenticator session must be aborted by running the
authenticatorCancel operation.

When this method is invoked, the authenticator must perform the
following procedure:

1. Check if all the supplied parameters are syntactically well-formed
and of the correct length. If not, return an error code equivalent
to "NotAllowedError" and terminate the operation.
2. If requireUserVerification is true and the authenticator cannot
perform user verification, return an error code equivalent to
"ConstraintError" and terminate the operation.
3. If allowCredentialDescriptorList was not supplied, set it to a list
of all credentials stored for the caller's RP ID (as determined by an
exact match of the RP ID).
4. Prompt the user to select a credential from among the above list.
Obtain user consent for using this credential. The prompt for

Prompt the user to select selectedCredential from the
is greater than 1
is exactly 1
is greater than 1
is exactly 1
is greater than 1
is equal to 1
is less than 1
is equal to 1
is greater than 1
which does not have an authenticatorMakeCredential or
This operation is ignored if it is invoked in an authenticator session

responses from the authenticator for the canceled operation.

stops prompting for, or accepting, any user input related to
authenticatorMakeCredential or authenticatorGetAssertion operation
session, it has the effect of terminating any

This operation takes no input parameters and returns no result.
When this operation is invoked by the client in an authenticator
session, it has the effect of terminating any
authenticatorMakeCredential or authenticatorGetAssertion operation
currently in progress in that authenticator session. The authenticator
stops prompting for, or accepting, any user input related to
authorizing the canceled operation. The client ignores any further
responses from the authenticator for the canceled operation.

This operation is ignored if it is invoked in an authenticator session
which does not have an authenticatorMakeCredential or
Authenticators must also provide some form of attestation. The basic requirement is that the authenticator must produce, for each credential, an attestation public key, an attestation statement verifiable by the Relying Party. Typically, this attestation statement contains a signature by an attestation private key over the attested credential public key and a challenge, as well as a certificate or similar data providing provenance information for the attestation public key, enabling the Relying Party to make a trust decision. However, if an attestation key pair is not available, then the authenticator MUST perform self attestation of the credential public key with the corresponding credential private key. All this information is returned by authenticators any time a new public key credential is generated, in the overall form of an attestation object. The relationship of the attestation object with authenticator data (containing attestation data) and the attestation statement is illustrated in figure 3, below.

This figure illustrates only the packed attestation statement format. Several additional attestation statement formats are defined in 7 Defined Attestation Statement Formats.

An important component of the attestation object is the attestation statement. This is a specific type of signed data object containing statements about a public key credential and the authenticator that created it. It contains an attestation signature created using the key of the attesting authority (except for the case of self attestation, when it is created using the credential private key). In order to correctly interpret an attestation statement, a Relying Party needs to understand these two aspects of attestation:

1. The attestation statement format is the manner in which the signature is represented and the various contextual bindings are incorporated into the attestation statement by the authenticator. In other words, this defines the syntax of the statement. Various existing devices and platforms (such as TPMs and the Android OS) have previously defined statement formats. This specification supports a variety of such formats in an extensible way, as defined in 3.3 Attestation Object Layout Formats.

2. The attestation type defines the semantics of attestation statements and their underlying trust models. Specifically, it defines how a Relying Party establishes trust in a particular attestation statement, after verifying that it is cryptographically valid. This specification supports a number of attestation types, as described in 6.3.2 Attestation Types.

In general, there is no simple mapping between attestation statement formats and attestation types. For example, the "packed" attestation statement format defined in 7.2 Packed Attestation Statement Format can be used in conjunction with all attestation types, while other formats and types have more limited applicability.

The privacy, security and operational characteristics of attestation depend on:

* The attestation type, which determines the trust model, the attestation statement format, which may constrain the strength of the attestation by limiting what can be expressed in an attestation statement, and the characteristics of the individual authenticator, such as its construction, whether part or all of it runs in a secure operating environment, and so on.

It is expected that most authenticators will support a small number of attestation types and attestation statement formats, while Relying

AuthenticatorsGetAssertion operation currently in progress.
Parties will decide what attestation types are acceptable to them by policy. Relying Parties will also need to understand the characteristics of the authenticators that they trust, based on information they have about these authenticators. For example, the FIDO Metadata Service [FIDOMetadataService] provides one way to access such information.

5.3.1. Attestation data

Attestation data is added to the authenticator data when generating an attestation object for a given credential. It has the following format:

- **Length (in bytes)**: 16 The AAGUID of the authenticator.
- **2 Byte length L of Credential ID**: L Credential ID
- **Variable: The credential public key encoded in COSE. Key format, as defined in Section 7 of [RFC8152]. The encoded credential public key MUST contain the "alg" parameter and MUST NOT contain any other optional parameters. The "alg" parameter MUST contain a COSEAlgorithmIdentifier value.**

5.3.2. Attestation Statement Formats

As described above, an attestation statement format is a data format which represents a cryptographic signature by an authenticator over a set of contextual bindings. Each attestation statement format MUST be defined using the following template:

- **Attestation statement format identifier**: A COSE algorithm identifier.
- **Supported attestation types**: The syntex of an attestation statement produced in this format, defined using [CDDL] for the extension point SatStmtFormat defined in 5.3.4 Generating an Attestation Object.
- **Signing procedure**: The signing procedure for computing an attestation statement in this format given the public key credential to be attested, the authenticator data structure containing the authenticator data for the attestation, and the hash of the serialized client data.
- **Verification procedure**: The procedure for verifying an attestation statement, which takes as inputs the authenticator data structure containing the authenticator data claimed to have been used for the attestation and the hash of the serialized client data, and returns either:
  - An error indicating that the attestation is invalid, or
  - The attestation type, and the trust path of the attestation.

**The initial list of specified attestation statement formats is in 7 Defined Attestation Statement Formats.**

5.3.3. Attestation Types

WebAuthn supports multiple attestation types:

- Basic Attestation
  - In the case of basic attestation [IUAProtocol], the authenticator’s attestation key pair is specific to an authenticator model. Thus, authenticators of the same model often share the same attestation key pair. See 5.3.5.1 Privacy for further information.
Self Attestation

In the case of self attestation, also known as surrogate basic attestation [UAFProtocol], the Authenticator does not have any specific attestation key. Instead it uses the authentication key itself to create the attestation signature. Authenticators without meaningful protection measures for an attestation private key typically use this attestation type.

Privacy CA

In this case, the Authenticator owns an authenticator-specific (endorsement) key. This key is used to securely communicate with a trusted third party, the Privacy CA. The Authenticator can generate multiple attestation key pairs and asks the Privacy CA to issue an attestation certificate for it. Using this approach, the Authenticator can limit the exposure of the endorsement key (which is a global correlation handle) to Privacy CA(s).

Attestation keys can be requested for each public key credential individually.

Note: This concept typically leads to multiple attestation certificates. The attestation certificate requested most recently is called "active".

Elliptic Curve based Direct Anonymous Attestation (ECDAA)

In this case, the Authenticator receives direct anonymous attestation (DAA) credentials from a single DAA-Issuer. These DAA credentials are used along with blinding to sign the attestation data. The concept of blinding avoids the DAA credentials being misused as global correlation handle. WebAuthn supports DAA using elliptic curve cryptography and bilinear pairings, called ECDAA (see [FIDOEcdaaAlgorithm]) in this specification. Consequently, we denote the DAA-Issuer as ECDAA-Issuer (see [FIDOEcdaaAlgorithm]).

5.3.4. Generating an Attestation Object

This section specifies the algorithm for generating an attestation object (see: Figure 3) for any attestation statement format.

In order to construct an attestation object for a given public key credential using a particular attestation statement format, the authenticator MUST first generate the authenticator data. The authenticator MUST then run the signing procedure for the desired attestation statement format with this authenticator data and the hash of the serialized client data as input, and use this to construct an attestation statement in that attestation statement format.

Finally, the authenticator MUST construct the attestation object as a CBOR map with the following syntax:

```
attObj = ({attStmt: bytes, $attStmtType})
attStmtTemplate = ({fmt: text, attStmt: bytes})

; Every attestation statement format must have the above fields
```

Self Attestation

In the case of self attestation, also known as surrogate basic attestation [UAFProtocol], the Authenticator does not have any specific attestation key. Instead it uses the authentication key itself to create the attestation signature. Authenticators without meaningful protection measures for an attestation private key typically use this attestation type.

Privacy CA

In this case, the Authenticator owns an authenticator-specific (endorsement) key. This key is used to securely communicate with a trusted third party, the Privacy CA. The Authenticator can generate multiple attestation key pairs and asks the Privacy CA to issue an attestation certificate for it. Using this approach, the Authenticator can limit the exposure of the endorsement key (which is a global correlation handle) to Privacy CA(s).

Attestation keys can be requested for each public key credential individually.

Note: This concept typically leads to multiple attestation certificates. The attestation certificate requested most recently is called "active".

Elliptic Curve based Direct Anonymous Attestation (ECDAA)

In this case, the Authenticator receives direct anonymous attestation (DAA) credentials from a single DAA-Issuer. These DAA credentials are used along with blinding to sign the attested credential data. The concept of blinding avoids the DAA credentials being misused as global correlation handle. WebAuthn supports DAA using elliptic curve cryptography and bilinear pairings, called ECDAA (see [FIDOEcdaaAlgorithm]) in this specification. Consequently, we denote the DAA-Issuer as ECDAA-Issuer (see [FIDOEcdaaAlgorithm]).

6.3.4. Generating an Attestation Object

To generate an attestation object (see: Figure 3) given:

- `attestationFormat`: An attestation statement format.
- `authData`: A byte array containing authenticator data.
- `hash`: The hash of the serialized client data.
- the authenticator MUST:
  1. Let attStmt be the result of running attestationFormat's signing procedure given authData and hash.
  2. Let fmt be attestationFormat's attestation statement format identifier.
  3. Return the attestation object as a CBOR map with the following syntax, filled in with variables initialized by this algorithm:

```
attObj = {authData: bytes, $attStmtType}
attStmtTemplate = {
  fmt: text,  
  attStmt: bytes
}

; Every attestation statement format must have the above fields
```
attStmtTemplate .within $SatStmtType

5.3.5. Security Considerations

5.3.5.1. Privacy

Attestation keys may be used to track users or link various online identities of the same user together. This may be mitigated in several ways, including:

- A WebAuthn authenticator manufacturer may choose to ship all of their devices with the same (or a fixed number of) attestation keys (called Basic Attestation). This will anonymize the user at the risk of not being able to revoke a particular attestation key should its WebAuthn Authenticator be compromised.

- A WebAuthn Authenticator attestation certificate must be revoked by the issuer CA if its key has been compromised. A WebAuthn Authentication certificate is compromised. A WebAuthn Authenticator manufacturer may need to ship a firmware update and inject new attestation keys and certificates into already manufactured WebAuthn Authenticators, if the exposure was due to a firmware flaw. (The process by which this happens is out of scope for this specification.)

- If the WebAuthn Authenticator manufacturer does not have this capability, then it may not be possible for Relying Parties to trust any further attestation statements from the affected WebAuthn Authenticators.

If attestation certificate validation fails due to a revoked intermediate attestation CA certificate, and the Relying Party's policy requires rejecting the registration/authentication request in these situations, then it is recommended that the Relying Party also un-registers (or marks with a trust level equivalent to “self
attestation”) public key credentials that were registered after the CA compromise date using an attestation certificate chaining up to the same intermediate CA. It is thus recommended that Relying Parties remember intermediate attestation CA certificates during Authenticator registration in order to un-register related public key credentials if the registration was performed after revocation of such certificates.

If an ECDAA attestation key has been compromised, it can be added to the RevokedList (i.e., the list of revoked authenticators) maintained by the related ECDAA-issuer. The Relying Party should verify whether an attestation certificate belongs to the RevokedList when performing ECDAA verify (see section 3.6 in [FIDOecdaaAlgorithm]). For example, the FIDO Metadata Service [FIDOMetadataService] provides one way to access such information.

6.3.5.3. Attestation Certificate Hierarchy

A 3-tier hierarchy for attestation certificates is recommended (i.e., Attestation Root, Attestation Issuing CA, Attestation Certificate). It is also recommended that for each WebAuthn Authenticator device line (i.e., model), a separate issuing CA is used to help facilitate isolating problems with a specific version of a device.

If the attestation root certificate is not dedicated to a single WebAuthn Authenticator device line (i.e., AAGUID), the AAGUID should be specified in the attestation certificate itself, so that it can be verified against the authenticator data.

7. Relying Party Operations

Upon successful execution of create() or get(), the Relying Party’s system perceives a PublicKeyCredential containing an AuthenticatorAttestationResponse or AuthenticatorAssertionResponse structure, respectively, from the client. It must then deliver the contents of this structure to the Relying Party server, using methods outside the scope of this specification. This section describes the operations that the Relying Party must perform upon receipt of these structures.

7.1. Registering a new credential

When registering a new credential, represented by a AuthenticatorAttestationResponse structure, as part of a registration ceremony, a Relying Party MUST proceed as follows:

1. Perform JSON deserialization on the credentialDataJSON field of the AuthenticatorAttestationResponse object to extract the client data C claimed as collected during the credential creation.
2. Verify that the challenge in C matches the challenge that was sent to the authenticator in the create() call.
3. Verify that the origin in C matches the Relying Party’s origin.
4. Verify that the tokenBindingId in C matches the Token Binding ID claimed as collected during the credential creation.
5. Verify that the clientExtensions in C is a proper subset of the extensions requested by the RP and that the authenticatorExtensions in C is also a proper subset of the extensions requested by the RP.
6. Compute the hash of clientDataJSON using the algorithm identified in C hashAlgorithm.
7. Perform CBOR decoding on the attestationObject field of the AuthenticatorAttestationResponse structure to obtain the attestation statement format fmt, the authenticator data authData, and the attestation statement attStmt.
8. Verify that the RP ID hash in authData is indeed the SHA-256 hash of the RP ID expected by the RP.
9. Determine the attestation statement format by performing an USASCII case-sensitive match on fmt against the set of supported WebAuthn Attestation Statement Format Identifiers. The set of up-to-date publicly registered WebAuthn Attestation Statement Format Identifier values is maintained in the IANA registry of the same name [WebAuthn-Registrars].
11. Verify that attStmt is a correct, validly-signed attestation statement, using the attestation statement format fmt's verification procedure given attStmt, authData and the format fmt's signature verification procedure given attStmt, authData and the hash of the serialized client data computed in step 6.

11. If validation is successful, obtain a list of acceptable trust anchors (attestation root certificates or ECDAA-Issuer public keys) for the attestation type, and attestation format fmt. If attStmt was returned by a trusted method of determining acceptable trust anchors (attestation root certificates or ECDAA-Issuer public keys) from a trusted source or from policy. For example, the FIDO Metadata Service [FIDOMetadataService] provides one way to obtain such information, using the aaguid in the attestedCredentialData in authData.

12. Assess the attestation trustworthiness using the outputs of the verification procedure in step 10, as follows:

12. If self attestation was used, check if self attestation is acceptable under Relying Party policy.
12. Otherwise, use the X.509 certificate returned by the verification procedure to verify that the attestation public key correctly chains up to an acceptable root certificate.
12. If the attestation statement attStmt successfully verified and is found to be trustworthy, then register the credential with the account that was denoted in the options.user passed to create(), by associating it with the credential ID and credential public key contained in authData's data as appropriate for the Relying Party's system.
12. If the attestation statement attStmt successfully verified but is not trustworthy per step 12 above, the Relying Party SHOULD fail the registration ceremony.

13. Assess the attestation trustworthiness using the outputs of the verification procedure in step 10, as follows:

13. If self attestation was used, check if self attestation is acceptable under Relying Party policy.
13. Otherwise, use the X.509 certificate returned by the verification procedure to verify that the attestation public key correctly chains up to an acceptable root certificate.
13. If the attestation statement attStmt successfully verified and is found to be trustworthy, then register the credential with the account that was denoted in the options.user passed to create(), by associating it with the credential ID and credential public key contained in authData's data as appropriate for the Relying Party's system.

Verification of attestation objects requires that the Relying Party has a trusted method of determining acceptable trust anchors in step 11 above. Also, if certificates are being used, the Relying Party must have access to certificate status information for the intermediate CA certificates. The Relying Party must also be able to build the attestation certificate chain if the client did not provide this chain in the attestation information.

To avoid ambiguity during authentication, the Relying Party SHOULD check that each credential is registered to no more than one user. If registration is requested for a credential that is already registered to a different user, the Relying Party SHOULD fail this ceremony, or it MAY decide to accept the registration, e.g. while deleting the older registration.

6.2. Verifying an authentication assertion

When verifying a given PublicKeyCredential structure (credential) as part of an authentication ceremony, the Relying Party MUST perform the following steps:
1. 1. Check that credential's id attribute (or the corresponding rawId, if base64url encoding is inappropriate for your use case), look up the corresponding credential public key.
2. 2. Let cData, aData and sig denote the value of credential's response's clientDataJSON, authenticatorData, and signature respectively.

Verification of attestation objects requires that the Relying Party has a trusted method of determining acceptable trust anchors in step 11 above. Also, if certificates are being used, the Relying Party must have access to certificate status information for the intermediate CA certificates. The Relying Party must also be able to build the attestation certificate chain if the client did not provide this chain in the attestation information.

To avoid ambiguity during authentication, the Relying Party SHOULD check that each credential is registered to no more than one user. If registration is requested for a credential that is already registered to a different user, the Relying Party SHOULD fail this ceremony, or it MAY decide to accept the registration, e.g. while deleting the older registration.

7.2. Verifying an authentication assertion

When verifying a given PublicKeyCredential structure (credential) as part of an authentication ceremony, the Relying Party MUST perform the following steps:
1. 1. Using credential's id attribute (or the corresponding rawId, if base64url encoding is inappropriate for your use case), look up the corresponding credential public key.
2. 2. Let cData, aData and sig denote the value of credential's response's clientDataJSON, authenticatorData, and signature respectively.

7.2. Verifying an authentication assertion

When verifying a given PublicKeyCredential structure (credential) as part of an authentication ceremony, the Relying Party MUST perform the following steps:
1. 1. Using credential's id attribute (or the corresponding rawId, if base64url encoding is inappropriate for your use case), look up the corresponding credential public key.
2. 2. Let cData, aData and sig denote the value of credential's response's clientDataJSON, authenticatorData, and signature respectively.
3. Perform JSON deserialization on cData to extract the client data C used for the signature.

4. Verify that the challenge member of C matches the challenge that was sent to the authenticator in the PublicKeyCredentialRequestOptions passed to the get() call.

5. Verify that the origin member of C matches the Relying Party’s origin.

6. Verify that the tokenBindingId member of C (if present) matches the Token Binding ID for the TLS connection over which the signature was obtained.

7. Verify that the clientExtensions member of C is a proper subset of the extensions requested by the Relying Party and that the authenticatorExtensions in C is also a proper subset of the extensions requested by the Relying Party.

8. Verify that the RP ID hash in aData is the SHA-256 hash of the RP ID expected by the Relying Party.

9. Let hash be the result of computing a hash over the cData using the algorithm represented by the hashAlgorithm member of C.

10. Using the credential public key looked up in step 1, verify that sig is a valid signature over the binary concatenation of aData and hash.

11. If all the above steps are successful, continue with the authentication ceremony as appropriate. Otherwise, fail the authentication ceremony.

7. Defined Attestation Statement Formats

WebAuthn supports pluggable attestation statement formats. This section defines an initial set of such formats.

7.1. Attestation Statement Format Identifiers

Attestation statement formats are identified by a string, called the attestation statement format identifier, chosen by the author of the attestation statement format.

Attestation statement format identifiers SHOULD be registered per [WebAuthn-Registries] “Registrries for Web Authentication (WebAuthn).”

All registered attestation statement format identifiers are unique amongst themselves as a matter of course.

Unregistered attestation statement format identifiers SHOULD use lowercase reverse domain-name naming, using a domain name registered by the developer, in order to assure uniqueness of the identifier. All attestation statement format identifiers MUST be a maximum of 32 octets in length and MUST consist only of printable USASCII characters.

8. Defined Attestation Statement Formats

WebAuthn supports pluggable attestation statement formats. This section defines an initial set of such formats.

8.1. Attestation Statement Format Identifiers

Attestation statement formats are identified by a string, called a attestation statement format identifier, chosen by the author of the attestation statement format.

Attestation statement format identifiers SHOULD be registered per [WebAuthn-Registries] “Registrries for Web Authentication (WebAuthn).”

All registered attestation statement format identifiers are unique amongst themselves as a matter of course.

Unregistered attestation statement format identifiers SHOULD use lowercase reverse domain-name naming, using a domain name registered by the developer, in order to assure uniqueness of the identifier. All attestation statement format identifiers MUST be a maximum of 32 octets in length and MUST consist only of printable USASCII characters.
excluding backslash and doublequote, i.e., VCHAR as defined in [RFC2234] but without %x22 and %x5C.

Note: This means attestation statement format identifiers based on domain names MUST incorporate only LDH Labels [RFC8234].

Implementations MUST match WebAuthn attestation statement format identifiers in a case-sensitive fashion.

Attestation statement formats that may exist in multiple versions SHOULD include a version in their identifier. In effect, different versions are thus treated as different formats, e.g., packed2 as a new version of the packed attestation statement format.

The following sections present a set of currently-defined and registered attestation statement formats and their identifiers. The up-to-date list of registered WebAuthn Extensions is maintained in the IANA "WebAuthn Attestation Statement Format Identifier" registry established by [WebAuthn-Registries].

### 7.2. Packed Attestation Statement Format

This is a WebAuthn optimized attestation statement format. It uses a very compact but still extensible encoding method. It is implementable by authenticators with limited resources (e.g., secure elements).

Attestation statement format identifier: packed

Attestation types supported:

All

Syntax:

The syntax of a Packed Attestation statement is defined by the following CDDL:

```cddl
$sattStmtType // {  
    fmt: "packed",  
    attStmt: packedStmtFormat
}
```

```cddl
packedStmtFormat = {  
    alg: rsaAlgName / eccAlgName,  
    sig: bytes,  
    x5C: [ attestCert: bytes, "(caCert: bytes) " ]  
} //  
```

```cddl
alg: "ED256" / "ED512",  
```

```cddl
sig: bytes,  
ecdaaKeyId: bytes  
```

The semantics of the fields are as follows:

```cddl
alg  
A text string containing the name of the algorithm used to generate the attestation signature. The types rsaAlgName and eccAlgName are as defined in 5.3.1 Attestation data. "ED256" and "ED512" refer to algorithms defined in [FIDOEcdaaAlgorithm].
```

```cddl
sig  
A byte string containing the attestation signature.
```

```cddl
x5C  
The elements of this array contain the attestation certificate and its certificate chain, each encoded in X.509 format. The attestation certificate must be the first element in the array.
```
ecdaaKeyId
The identifier of the ECDAA-Issuer public key. This is the
BigNumberToUB encoding of the component "c" of the
ECDAA-Issuer public key as defined section 3.3, step 3.5
in [FIDOEcdaaAlgorithm].

Sign procedure
The signing procedure for this attestation statement format is
similar to the procedure for generating assertion signatures.
Let authenticatorData denote the authenticator data for the
attestation, and let clientDataHash denote the hash of the
serialized client data.

1. Verify that attStmt is valid CBOR conforming to the syntax
defined above, and perform CBOR decoding on it to extract the
contained fields.
2. If x5c is present, this indicates that the attestation type is
cisco.
   a. Verify that sig is a valid signature over the concatenation of
      authenticatorData and clientDataHash using the ECDAA-Issuer public key in x5c with
      the algorithm specified in alg.
   b. Verify that x5c meets the requirements in 7.2.1 Packed
      attestation statement certificate requirements.
   c. If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4
      (id-fido-gen-ce-aaguid) verify that the value of this
      extension matches the AAGUID in authenticatorData.
   d. If successful, return attestation type Basic and trust path
      x5c.
3. If ecdaaKeyId is present, then the attestation type is ECDAA. In
   this case:
   a. Verify that sig is a valid signature over the concatenation of
      authenticatorData and clientDataHash using ECDAA-Verify with
      the ECDAA-Issuer public key identified by ecdaaKeyId (see
      [FIDOEcdaaAlgorithm]).
4. If self attestation is in use, the authenticator produces sig by
   concatenating authenticatorData and clientDataHash, and signing the result using the
credential private key. It sets alg to the algorithm of the
credential private key, and omits the other
fields.
Verification procedure
Given the verification procedure inputs attStmt,
authenticatorData and clientDataHash, the verification procedure
is as follows:
1. Verify that attStmt is valid CBOR conforming to the syntax
defined above, and perform CBOR decoding on it to extract the
contained fields.
2. If x5c is present, this indicates that the attestation type is
cisco.
   a. Verify that sig is a valid signature over the
      concatenation of authenticatorData and clientDataHash using the
      ECDAA-Issuer public key in x5c with the
      algorithm specified in alg.
   b. Verify that x5c meets the requirements in 8.2.1 Packed
      attestation statement certificate requirements.
   c. If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4
      (id-fido-gen-ce-aaguid) verify that the value of this
      extension matches the AAGUID in authenticatorData.
   d. If successful, return attestation type Basic and trust path
      x5c.
3. If ecdaaKeyId is present, then the attestation type is ECDAA. In
   this case:
   a. Verify that sig is a valid signature over the
      concatenation of authenticatorData and clientDataHash using ECDAA-Verify with
      the ECDAA-Issuer public key identified by ecdaaKeyId (see
      [FIDOEcdaaAlgorithm]).
   b. If successful, return attestation type ECDAA and
      trust path x5c.
3. If ECDAA is in use, the authenticator produces sig by
   concatenating authenticatorData and clientDataHash, and signing the result using the ECDAA-Issuer public key related to the ECDAA signature private key through an
authenticated-specific mechanism (see section 3.5 of
[FIDOEcdaaAlgorithm]). It sets alg to the algorithm of the selected ECDAA-Issuer public key and ecdaaKeyId to the identifier of the
ECDAA-Issuer public key (see above).
4. If self attestation is in use, the authenticator produces sig by
   concatenating authenticatorData and clientDataHash, and signing the result using the credential private key. It sets
alg to the algorithm of the credential private key, and omits the other
fields.
Verification procedure
Given the verification procedure inputs attStmt,
authenticatorData and clientDataHash, the verification procedure
is as follows:
1. Verify that attStmt is valid CBOR conforming to the syntax
defined above, and perform CBOR decoding on it to extract the
contained fields.
2. If x5c is present, this indicates that the attestation type is
cisco.
   a. Verify that sig is a valid signature over the
      concatenation of authenticatorData and clientDataHash using the
      ECDAA-Issuer public key in x5c with the
      algorithm specified in alg.
   b. Verify that x5c meets the requirements in 8.2.1 Packed
      attestation statement certificate requirements.
   c. If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4
      (id-fido-gen-ce-aaguid) verify that the value of this
      extension matches the AAGUID in authenticatorData.
   d. If successful, return attestation type Basic and trust path
      x5c.
3. If ecdaaKeyId is present, then the attestation type is ECDAA. In
   this case:
   a. Verify that sig is a valid signature over the
      concatenation of authenticatorData and clientDataHash using ECDAA-Verify with
      the ECDAA-Issuer public key identified by ecdaaKeyId (see
      [FIDOEcdaaAlgorithm]).
   b. If successful, return attestation type ECDAA and
      trust path x5c.
The syntax of a TPM Attestation statement is as follows:

```plaintext
* Version must be set to 3.
* Subject field MUST be set to:
  Subject-C
    Country where the Authenticator vendor is incorporated
  Subject-O
    Legal name of the Authenticator vendor
  Subject-CN
    Authenticator Attestation
  Subject-OU
    Country where the Authenticator vendor is incorporated
* If the related attestation root certificate is used for multiple authenticator models, the Extension OID 1.3.6.1.4.1.45724.1.1.4 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as value.
* The Basic Constraints extension MUST have the CA component set to false.
  An Authority Information Access (AIA) extension with entry id-ocsp and a CRL Distribution Point extension [RFC5280] are both optional as the status of many attestation certificates is available through authenticator metadata services. See, for example, the FIDO Metadata Service [FIDOMetadataService].
```

7.3. TPM Attestation Statement Format

This attestation statement format is generally used by authenticators that use a Trusted Platform Module as their cryptographic engine.

Attestation statement format identifier
tpm
Attestation types supported
Privacy CA, ECDAA
Syntax
The syntax of a TPM Attestation statement is as follows:

```plaintext
SattStmtType // =
  attStmt: tpmStmtFormat

tpmStmtFormat =
  alg: rsaAlgName / eccAlgName
  x5c: [ aikCert: bytes, * (caCert: bytes) ]
```

8.2.1. Packed attestation statement certificate requirements

The attestation certificate MUST have the following fields/extensions:

* Subject field MUST be set to:
  Subject-C
    Country where the Authenticator vendor is incorporated
  Subject-O
    Legal name of the Authenticator vendor
  Subject-CN
    Authenticator Attestation
  Subject-OU
    Country where the Authenticator vendor is incorporated
* Version must be set to 3.
* If the related attestation root certificate is used for multiple authenticator models, the Extension OID 1.3.6.1.4.1.45724.1.1.4 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as value.
* The Basic Constraints extension MUST have the CA component set to false.
  An Authority Information Access (AIA) extension with entry id-ocsp and a CRL Distribution Point extension [RFC5280] are both optional as the status of many attestation certificates is available through authenticator metadata services. See, for example, the FIDO Metadata Service [FIDOMetadataService].
```

8.3. TPM Attestation Statement Format

This attestation statement format is generally used by authenticators that use a Trusted Platform Module as their cryptographic engine.

Attestation statement format identifier
tpm
Attestation types supported
Privacy CA, ECDAA
Syntax
The syntax of a TPM Attestation statement is as follows:

```plaintext
SattStmtType // =
  fmt: "tpm",
  attStmt: tpmStmtFormat

tpmStmtFormat =
  alg: [ rsaAlgName / eccAlgName ]
  x5c: [ aikCert: bytes, * (caCert: bytes) ]
```

If neither x5c nor ecdaaKeyId is present, self attestation is in use.

- Validate that alg matches the algorithm of the credential private key in authenticatorData.
- Verify that sig is a valid signature over the concatenation of authenticatorData and clientDataHash using the credential public key with alg.
- If successful, return attestation type Self and empty trust path.
Verify that the given attestation statement is valid CBOR.

The semantics of the above fields are as follows:

- **alg**: The name of the algorithm used to generate the attestation signature. The types `ed256` and `ed512` refer to the algorithms specified in [FIDOEd256Ed512Algorithm].
- **x5c**: The AIK certificate used for the attestation and its certificate chain, in X.509 encoding.
- **ecdaaKeyId**: The identifier of the ECDAA-Issuer public key. This is the `BigIntegerToB` encoding of the component `c` as defined in section 3.3, step 3.5 in [FIDOEd2aaAlgorithm].
- **sig**: The attestation signature, in the form of a TPMT_SIGNATURE structure as specified in [TPMv2-Part2] section 11.3.4.
- **certInfo**: The TPMS_ATTEST structure over which the above signature was computed, as specified in [TPMv2-Part2] section 10.12.8.
- **pubArea**: The TPMT_PUBLIC structure (see [TPMv2-Part2] section 12.2.4) used by the TPM to represent the credential public key.

**Signing procedure**

Let authenticatorData denote the authenticator data for the attestation, and let clientDataHash denote the hash of the serialized client data.

Concatenate authenticatorData and clientDataHash to form attToBeSigned.

Generate a signature using the procedure specified in [TPMv2-Part3] Section 18.2, using the attestation private key and setting the qualifyingData parameter to attToBeSigned.

Set the public area of the credential public key, the certInfo field to the output parameter of the same name, and the sig field to the signature obtained from the above procedure.

**Verification procedure**

Verify that the given attestation statement is valid CBOR conforming to the syntax defined above.
Let authenticatorData denote the authenticator data claimed to have been used for the attestation, and let clientDataHash denote the hash of the serialized client data.

Verify that the public key specified by the parameters and unique fields of pubArea is identical to the public key contained in the attestation data inside authenticatorData.

Concatenate authenticatorData and clientDataHash to form attToBeSigned.

Validate that certInfo is valid:

+ Verify that magic is set to TPM GENERATED_VALUE.
+ Verify that type is set to TPM_ST_ATTEST_CERTIFICATE.
+ Verify that extraData is set to attToBeSigned.

+ Verify that attested contains a TPMS CERTIFY_INFO structure, whose name field contains a valid Name for pubArea, as computed using the algorithm in the nameArea field of pubArea using the procedure specified in [TPMv2-Part1] section 16.

If x5c is present, this indicates that the attestation type is not ECDAA. In this case:

+ Verify the sig is a valid signature over certInfo using the attestation public key in x5c with the algorithm specified in alg.
+ Verify that x5c meets the requirements in 7.3.1 TPM attestation statement certificate requirements.
+ If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 14
  (id-fido-gen-ce-aaguid) verify that the value of this extension matches the AAGUID in authenticatorData.
+ If successful, return attestation type Privacy CA and trust path x5c.

If ecdaaKeyId is present, then the attestation type is ECDAA.

+ Perform ECDAA-Verify on sig to verify that it is a valid signature over certInfo (see [FIDO-EcdaaAlgorithm]).
+ If successful, return attestation type ECDAA and the identifier of the ECDAA-issuer public key ecdaaKeyId.

8.3.1. TPM attestation statement certificate requirements

TPM attestation certificate MUST have the following fields/extension:

* Version must be set to 3.
* Subject field MUST be set to empty.
* The Subject Alternative Name extension must be set as defined in [TPMv2-EK-Proile] section 3.2.9.
* The Extended Key Usage extension MUST contain the "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8) tcp-k-AIKCertificate(3)" OID.
* The Basic Constraints extension MUST have the CA component set to false.
* An Authority Information Access (AIA) extension with entry id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are both optional as the status of many attestation certificates is available through metadata services. See, for example, the FIDO Metadata Service [FIDOMetadataService].

7.4. Android Key Attestation Statement Format

When the authenticator in question is a platform-provided Authenticator on the Android "N" or later platform, the attestation statement is based on the Android key attestation. In these cases, the attestation statement is produced by a component operating in a secure operating environment, but the authenticator data for the attestation is produced outside this environment. The Relying Party is expected to check that the authenticator data claimed to have been used for the attestation is

Verify that attStmt is valid CBOR conforming to the syntax defined above, and perform CBOR decoding on it to extract the contained fields.

Verify that the public key specified by the parameters and unique fields of pubArea is identical to the credentialPublicKey in the attestatedCredentialData in authenticatorData.

Concatenate authenticatorData and clientDataHash to form attToBeSigned.

Validate that certInfo is valid:

+ Verify that magic is set to TPM GENERATED_VALUE.
+ Verify that type is set to TPM_ST_ATTEST_CERTIFICATE.
+ Verify that attested contains a TPMS CERTIFY_INFO structure, whose name field contains a valid Name for pubArea, as computed using the algorithm in the nameArea field of pubArea using the procedure specified in [TPMv2-Part1] section 16.

If x5c is present, this indicates that the attestation type is not ECDAA. In this case:

+ Verify the sig is a valid signature over certInfo using the attestation public key in x5c with the algorithm specified in alg.
+ Verify that x5c meets the requirements in 8.3.1 TPM attestation statement certificate requirements.
+ If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 14
  (id-fido-gen-ce-aaguid) verify that the value of this extension matches the AAGUID in authenticatorData.
+ If successful, return attestation type Privacy CA and attestation trust path x5c.

If ecdaaKeyId is present, then the attestation type is ECDAA.

+ Perform ECDAA-Verify on sig to verify that it is a valid signature over certInfo (see [FIDO-EcdaaAlgorithm]).
+ If successful, return attestation type ECDAA and the identifier of the ECDAA-issuer public key ecdaaKeyId.

8.4. Android Key Attestation Statement Format

When the authenticator in question is a platform-provided Authenticator on the Android "N" or later platform, the attestation statement is based on the Android key attestation. In these cases, the attestation statement is produced by a component operating in a secure operating environment, but the authenticator data for the attestation is produced outside this environment. The Relying Party is expected to check that the authenticator data claimed to have been used for the attestation is...
consistent with the fields of the attestation certificate's extension data.

Attestation statement format identifier

android-key

Attestation types supported

Basic

An Android key attestation statement consists simply of the 
Android attestation statement, which is a series of DER encoded 
X.509 certificates. See the Android developer documentation. Its 
syntax is defined as follows:

$$\text{attStmtType} \equiv \{
\begin{aligned}
\text{fmt: "android-key"}, \\
\text{attStmt: androidStmtFormat}
\end{aligned}
\}

androidStmtFormat = \text{bytes}

Syntax

Verifying a key attestation statement:

The authenticator produces sig by concatenating 
authenticatorData and clientDataHash, and signing the result 
using the credential private key. It sets alg to the algorithm 
of the signature format.

Verifying procedure

Verification is performed as follows:

+ Let authenticatorData denote the authenticator data claimed to 
have been used for the attestation, and let clientDataHash 
denote the hash of the serialized client data.
+ Verify that the public key in the first certificate in the 
series of certificates represented by the signature matches 
the credential public key in the attestation data field of 
authenticatorData.
+ Verify that in the attestation certificate extension 
data:
  o The value of the attestationChallenge field is identical 
to the concatenation of authenticatorData and 
clientDataHash.
  o The AuthorizationList.allApplications field is not 
present, since PublicKeyCredentials must be bound to the 
RP ID.
  o The value in the AuthorizationList.origin field is equal 
to KM_TAG_GENERATED.
  o If successful, return attestation type Basic with the trust 
path set to the entire attestation statement.

+ Verify that attStmt is valid CBOR conforming to the syntax 
declared above, and perform CBOR decoding on it to extract the 
contained fields.
+ Verify that the public key in the first certificate in the 
series of certificates represented by the signature matches 
the credentialPublicKey in the attestedCredentialData in 
authenticatorData.
+ Verify that in the attestation certificate extension data:
  o The value of the attestationChallenge field is identical 
to the concatenation of authenticatorData and 
clientDataHash.
  o The AuthorizationList.allApplications field is not 
present, since PublicKeyCredentials must be bound to the 
RP ID.
  o The value in the AuthorizationList.origin field is equal 
to KM_TAG_GENERATED.
  o If successful, return attestation type Basic with the attestation trust path set to the entire attestation statement.
7.5. Android SafetyNet Attestation Statement Format

When the authenticator in question is a platform-provided Authenticator on certain Android platforms, the attestation statement is based on the SafetyNet API. In this case the authenticator data is completely controlled by the caller of the SafetyNet API (typically an application running on the Android platform) and the attestation statement only provides some statements about the health of the platform and the identity of the calling application.

Attestation statement format identifier: `android-safetynet`

Attestation types supported: Basic

Syntax:
The syntax of an Android Attestation statement is defined as follows:

```plaintext
$sattStmtType = \{
fmt: "android-safetynet",
attStmt: safetynetStmtFormat
\}
safetynetStmtFormat = { 
ver: text,
response: bytes
}
```

The semantics of the above fields are as follows:

- `ver`: The version number of Google Play Services responsible for providing the SafetyNet API.
- `response`: The value returned by the above SafetyNet API. This value is a JWS [RFC7515] object (see SafetyNet online documentation) in Compact Serialization.
- `fmt`: The format of the attestation statement.

Signing procedure:
Let `attenticatorData` denote the authenticator data for the attestation, and `clientDataHash` denote the hash of the serialized client data.

Concateenate `attenticatorData` and `clientDataHash` to form `attToBeSigned`.

Request a SafetyNet attestation, providing `attToBeSigned` as the nonce value. Set response to the result, and ver to the version of Google Play Services running in the authenticator.

Verification procedure:
Verification is performed as follows:

+ Verify that the given attestation statement is valid CBOR conforming to the syntax defined above.
+ Verify that response is a valid SafetyNet response of version `ver`.
+ Verify that the nonce in the response is identical to the

8.5. Android SafetyNet Attestation Statement Format

When the authenticator in question is a platform-provided Authenticator on certain Android platforms, the attestation statement is based on the SafetyNet API. In this case the authenticator data is completely controlled by the caller of the SafetyNet API (typically an application running on the Android platform) and the attestation statement only provides some statements about the health of the platform and the identity of the calling application. This attestation does not provide information regarding provenance of the authenticator and its associated data. Therefore platform-provided authenticators should make use of the Android Key Attestation when available, even if the SafetyNet API is also present.

Attestation statement format identifier: `android-safetynet`

Attestation types supported: Basic

Syntax:
The syntax of an Android Attestation statement is defined as follows:

```plaintext
$sattStmtType = \{
fmt: "android-safetynet",
attStmt: safetynetStmtFormat
\}
safetynetStmtFormat = { 
ver: text,
response: bytes
}
```

The semantics of the above fields are as follows:

- `ver`: The version number of Google Play Services responsible for providing the SafetyNet API.
- `response`: The UTF-8 encoded result of the getJwsResult() call of the SafetyNet API. This value is a JWS [RFC7515] object (see SafetyNet online documentation) in Compact Serialization.
- `fmt`: The format of the attestation statement.

Signing procedure:
Let `attenticatorData` denote the authenticator data for the attestation, and `clientDataHash` denote the hash of the serialized client data.

Concateenate `attenticatorData` and `clientDataHash` to form `attToBeSigned`.

Request a SafetyNet attestation, providing `attToBeSigned` as the nonce value. Set response to the result, and ver to the version of Google Play Services running in the authenticator.

Verification procedure:
Given the verification procedure inputs `attStmt`, `attenticatorData` and `clientDataHash`, the verification procedure is as follows:

+ Verify that `attStmt` is valid CBOR conforming to the syntax defined above, and perform CBOR decoding on it to extract the contained fields.
+ Verify that response is a valid SafetyNet response of version `ver`.
+ Verify that the nonce in the response is identical to the
public key as x5c.

sig and set the attestation certificate of the attestation
set to the credential ID of the given credential. Set this as
challenge parameter set to tbsHash, and the key handle parameter
hash of the RP ID associated with the given credential, the
section 4.3, with the application parameter set to the SHA-256
attestation, and let clientDataHash denote the hash of the
attestation statements of Basic (i.e., without the user public
credential ID of the given credential, the challenge parameter set to tbsHash,
algorithm -7 ("ES256"), stop and return an error. Otherwise, 
attestation certificate of the attestation public key as x5c.
registration response message (i.e., without the user public key, key handle, and attestation certificates) as sig and set the attestation certificates of the attestation public key as x5c.

7.6. FIDO U2F Attestation Statement Format

This attestation statement format is used with FIDO U2F authenticators
using the formats defined in [FIDO-U2F-Message-Formats].

Attestation statement format identifier
fido-u2f

Attestation types supported
Basic, self attestation

Syntax
The syntax of a FIDO U2F attestation statement is defined as follows:

$attStmtType = \{
fmt: "fido-u2f",
attStmt: u2fStmtFormat
\}
u2fStmtFormat = \{
x5c: [ attestinCert: bytes, *(caCert: bytes) ],
sig: bytes
\}

The semantics of the above fields are as follows:

x5c
The elements of this array contain the attestation
certificate and its certificate chain, each encoded in
X.509 format. The attestation certificate must be the
first element in the array.

sig
The attestation signature.

Signing procedure
If the credential public key of the given credential is not of
algorithm -7 ("ES256"), stop and return an error.
Otherwise, let authData denote the authenticator data for the
attestation, and let clientDataHash denote the hash of the serialized client data.
If clientDataHash is 256 bits long, set tbsHash to this value.
Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
Generate a signature as specified in [FIDO-U2F-Message-Formats] section 4.3, with the application parameter set to the SHA-256 hash of the RP ID associated with the given credential, the challenge parameter set to tbsHash, and the key handle parameter set to the credential ID of the given credential. Set this as sig and set the attestation certificate of the attestation public key as x5c.

8.6. FIDO U2F Attestation Statement Format

This attestation statement format is used with FIDO U2F authenticators
using the formats defined in [FIDO-U2F-Message-Formats].

Attestation statement format identifier
fido-u2f

Attestation types supported
Basic Attestation, Self Attestation, Privacy CA

Syntax
The syntax of a FIDO U2F attestation statement is defined as follows:

$attStmtType = \{
fmt: "fido-u2f",
attStmt: u2fStmtFormat
\}
u2fStmtFormat = \{
x5c: [ attestinCert: bytes, *(caCert: bytes) ],
sig: bytes
\}

The semantics of the above fields are as follows:

x5c
The elements of this array contain the attestation certificate and its certificate chain, each encoded in X.509 format. The attestation certificate must be the first element in the array.

sig
The attestation signature. The signature was calculated over the raw U2F registration response message [FIDO-U2F-Message-Formats] received by the platform from the authenticator.

Signing procedure
If the credential public key of the given credential is not of algorithm -7 ("ES256"), stop and return an error. Otherwise, let authData denote the authenticator data for the attestation, and let clientDataHash denote the hash of the serialized client data.
If clientDataHash is 256 bits long, set tbsHash to this value.
Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
Generate a Registration Response Message as specified in [FIDO-U2F-Message-Formats] section 4.3, with the application parameter set to the SHA-256 hash of the RP ID associated with the given credential, the challenge parameter set to tbsHash, and the key handle parameter set to the credential ID of the given credential. Set the raw signature part of this Registration Response Message (i.e., without the user public key, key handle, and attestation certificates) as sig and set the attestation certificates of the attestation public key as x5c.
8. WebAuthn Extensions

The mechanism for generating public key credentials, as well as requesting and generating Authentication assertions, as defined in 4 Web Authentication API, can be extended to suit particular use cases. Each case is addressed by defining a registration extension and/or an authentication extension.

Every extension is a client extension, meaning that the extension involves communication with and processing by the client. Client extensions define the following steps and data:

* navigator.credentials.create() extension request parameters and response values for registration extensions and authentication extensions.
* navigator.credentials.get() extension request parameters and response values for authentication extensions.
* Client extension processing for registration extensions and authentication extensions.

When creating a public key credential or requesting an authentication assertion, a Relying Party can request the use of a set of extensions. These extensions will be invoked during the request operation if they are supported by the client and/or the authenticator. The Relying Party sends the client extension input for each extension in the get() call (for authentication extensions) or create() call (for registration extensions) to the client platform. The client platform performs client extension processing for registration extensions and authentication extensions.
extension identifiers are unique amongst themselves as a matter of "Registries for Web Authentication (WebAuthn)"). All registered extension identifiers SHOULD be registered per [WebAuthn-Registries] should be consulted for an up-to-date list of registered WebAuthn Extensions.

9.1. Extension Identifiers

Extensions are identified by a string, called an extension identifier, chosen by the extension author.

Extension identifiers SHOULD be registered per [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)": All registered extension identifiers are unique amongst themselves as a matter of course.

Unregistered extension identifiers should aim to be globally unique,

extension processing for each extension that it supports, and augments the client data as specified by each extension, by including the extension identifier and client extension output values. An extension can also be an authenticator extension, meaning that the extension involves communication with and processing by the authenticator. Authenticator extensions define the following steps and data:

- authenticatorMakeCredential request parameters and response values for registration extensions.
- authenticatorGetAssertion request parameters and response values for authentication extensions.

For authenticator extensions, as part of the client extension processing, the client also creates the CBOR authenticator extension input value as the value. The authenticator, in turn, performs additional processing for the extensions that it supports, and returns the CBOR authenticator extension output for each as specified by the extension. Part of the client extension processing for authenticator extensions is to use the authenticator extension output as an input to creating the client extension output.

All WebAuthn extensions are optional for both clients and authenticators. Thus, any extensions requested by a Relying Party may be ignored by the client browser or OS and not passed to the authenticator at all, or they may be ignored by the authenticator.

Ignoring an extension is never considered a failure in WebAuthn API processing, so when Relying Parties include extensions with any API calls, they must be prepared to handle cases where some or all of those extensions are ignored.

Clients wishing to support the widest possible range of extensions may choose to pass through any extensions that they do not recognize to authenticators, generating the authenticator extension input by simply encoding the client extension input value in CBOR. All WebAuthn extensions MUST be defined in such a way that this unregistered extension identifier and client extension output values.

The IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registries] should be consulted for an up-to-date list of registered WebAuthn Extensions.

8.1. Extension Identifiers

Extensions are identified by a string, called an extension identifier, chosen by the extension author.

Extension identifiers SHOULD be registered per [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)": All registered extension identifiers are unique amongst themselves as a matter of course.

Unregistered extension identifiers should aim to be globally unique,
e.g., by including the defining entity such as myCompany_extension.

All extension identifiers MUST be a maximum of 32 octets in length and
 MUST consist only of printable USASCII characters, excluding backslash
 and doublequote, i.e., VCHAR as defined in [RFC5324] but without %x22
 and %x5c. Implementations MUST match WebAuthn extension identifiers in
 a case-sensitive fashion.

Extensions that may exist in multiple versions should take care to
 include a version in their identifier. In effect, different versions
 are thus treated as different extensions, e.g., myCompany_extension_01
 and myCompany_extension_02.

9 Defined Extensions defines an initial set of extensions and their
 identifiers. See the IANA "WebAuthn Extension Identifier" registry
 established by [WebAuthn-Registries] for an up-to-date list of
 registered WebAuthn Extension Identifiers.

8.2. Defining extensions

A definition of an extension must specify an extension identifier, a
 client extension input argument to be sent via the get() or create() call,
 the client extension processing rules, and a client extension output value.
 If the extension communicates with the authenticator (meaning it is an authenticator extension), it must also specify the
 CBOR authenticator extension input argument sent via the
 authenticatorGetAssertion or authenticatorMakeCredential call, the
 authenticator extension processing rules, and the CBOR authenticator
 extension output value.

Any client extension that is processed by the client MUST return
 a client extension output value so that the Relying Party knows that the
 extension was honored by the client. Similarly, any extension that
 requires authenticator processing MUST return an authenticator
 extension output to let the Relying Party know that the extension was
 honored by the authenticator, if an extension does not otherwise
 require any result values, it SHOULD be defined as returning a JSON
 Boolean client extension output result, set to true to signify that the
 extension was understood and processed. Likewise, any authenticator
 extension that does not otherwise require any result values MUST return
 a value and SHOULD return a CBOR Boolean authenticator extension output
 result, set to true to signify that the extension was understood and
 processed.

8.3. Extending request parameters

An extension defines one or two request arguments. The client extension
 input, which is a value that can be encoded in JSON, is passed from the
 Relying Party to the client in the get() or create() call, while the
 CBOR authenticator extension input is passed from the client to the
 authenticator for authenticator extensions during the processing of
 these calls.

A Relying Party simultaneously requests the use of an extension and
 sets its client extension input by including an entry in the extensions
 option to the create() or get() call. The entry key is the extension
 identifier and the value is the client input.

Extension definitions MUST specify the valid values for their client
 extension input. Clients SHOULD ignore extensions with an invalid
 identifier such as myCompany_extension.

10 Defined Extensions defines an initial set of extensions and their
 identifiers. See the IANA "WebAuthn Extension Identifier" registry
 established by [WebAuthn-Registries] for an up-to-date list of
 registered WebAuthn Extension Identifiers.

9.2. Defining extensions

A definition of an extension must specify an extension identifier, a
 client extension input argument to be sent via the get() or create() call,
 the client extension processing rules, and a client extension output value. If the extension communicates with the authenticator (meaning it is an authenticator extension), it must also specify the
 CBOR authenticator extension input argument sent via the
 authenticatorGetAssertion or authenticatorMakeCredential call, the
 authenticator extension processing rules, and the CBOR authenticator
 extension output value.

Any client extension that is processed by the client MUST return
 a client extension output value so that the Relying Party knows that the
 extension was honored by the client. Similarly, any extension that
 requires authenticator processing MUST return an authenticator
 extension output to let the Relying Party know that the extension was
 honored by the authenticator, if an extension does not otherwise
 require any result values, it SHOULD be defined as returning a JSON
 Boolean client extension output result, set to true to signify that the
 extension was understood and processed. Likewise, any authenticator
 extension that does not otherwise require any result values MUST return
 a value and SHOULD return a CBOR Boolean authenticator extension output
 result, set to true to signify that the extension was understood and
 processed.

9.3. Extending request parameters

An extension defines one or two request arguments. The client extension
 input, which is a value that can be encoded in JSON, is passed from the
 Relying Party to the client in the get() or create() call, while the
 CBOR authenticator extension input is passed from the client to the
 authenticator for authenticator extensions during the processing of
 these calls.

A Relying Party simultaneously requests the use of an extension and
 sets its client extension input by including an entry in the extensions
 option to the create() or get() call. The entry key is the extension
 identifier and the value is the client input.

Extension definitions MUST specify the valid values for their client
 extension input. Clients SHOULD ignore extensions with an invalid
 identifier such as myCompany_extension.

All extension identifiers MUST be a maximum of 32 octets in length and
 MUST consist only of printable USASCII characters, excluding backslash
 and doublequote, i.e., VCHAR as defined in [RFC5324] but without %x22
 and %x5c. Implementations MUST match WebAuthn extension identifiers in
 a case-sensitive fashion.

Extensions that may exist in multiple versions should take care to
 include a version in their identifier. In effect, different versions
 are thus treated as different extensions, e.g., myCompany_extension_01
 and myCompany_extension_02.

9 Defined Extensions defines an initial set of extensions and their
 identifiers. See the IANA "WebAuthn Extension Identifier" registry
 established by [WebAuthn-Registries] for an up-to-date list of
 registered WebAuthn Extension Identifiers.

8.2. Defining extensions

A definition of an extension must specify an extension identifier, a
 client extension input argument to be sent via the get() or create() call,
 the client extension processing rules, and a client extension output value.
 If the extension communicates with the authenticator (meaning it is an authenticator extension), it must also specify the
 CBOR authenticator extension input argument sent via the
 authenticatorGetAssertion or authenticatorMakeCredential call, the
 authenticator extension processing rules, and the CBOR authenticator
 extension output value.

Any client extension that is processed by the client MUST return
 a client extension output value so that the Relying Party knows that the
 extension was honored by the client. Similarly, any extension that
 requires authenticator processing MUST return an authenticator
 extension output to let the Relying Party know that the extension was
 honored by the authenticator, if an extension does not otherwise
 require any result values, it SHOULD be defined as returning a JSON
 Boolean client extension output result, set to true to signify that the
 extension was understood and processed. Likewise, any authenticator
 extension that does not otherwise require any result values MUST return
 a value and SHOULD return a CBOR Boolean authenticator extension output
 result, set to true to signify that the extension was understood and
 processed.

8.3. Extending request parameters

An extension defines one or two request arguments. The client extension
 input, which is a value that can be encoded in JSON, is passed from the
 Relying Party to the client in the get() or create() call, while the
 CBOR authenticator extension input is passed from the client to the
 authenticator for authenticator extensions during the processing of
 these calls.

A Relying Party simultaneously requests the use of an extension and
 sets its client extension input by including an entry in the extensions
 option to the create() or get() call. The entry key is the extension
 identifier and the value is the client input.

Extension definitions MUST specify the valid values for their client
 extension input. Clients SHOULD ignore extensions with an invalid
 identifier such as myCompany_extension.

All extension identifiers MUST be a maximum of 32 octets in length and
 MUST consist only of printable USASCII characters, excluding backslash
 and doublequote, i.e., VCHAR as defined in [RFC5324] but without %x22
 and %x5c. Implementations MUST match WebAuthn extension identifiers in
 a case-sensitive fashion.

Extensions that may exist in multiple versions should take care to
 include a version in their identifier. In effect, different versions
 are thus treated as different extensions, e.g., myCompany_extension_01
 and myCompany_extension_02.

9 Defined Extensions defines an initial set of extensions and their
 identifiers. See the IANA "WebAuthn Extension Identifier" registry
 established by [WebAuthn-Registries] for an up-to-date list of
 registered WebAuthn Extension Identifiers.

8.2. Defining extensions

A definition of an extension must specify an extension identifier, a
 client extension input argument to be sent via the get() or create() call,
 the client extension processing rules, and a client extension output value.
 If the extension communicates with the authenticator (meaning it is an authenticator extension), it must also specify the
 CBOR authenticator extension input argument sent via the
 authenticatorGetAssertion or authenticatorMakeCredential call, the
 authenticator extension processing rules, and the CBOR authenticator
 extension output value.

Any client extension that is processed by the client MUST return
 a client extension output value so that the Relying Party knows that the
 extension was honored by the client. Similarly, any extension that
 requires authenticator processing MUST return an authenticator
 extension output to let the Relying Party know that the extension was
 honored by the authenticator, if an extension does not otherwise
 require any result values, it SHOULD be defined as returning a JSON
 Boolean client extension output result, set to true to signify that the
 extension was understood and processed. Likewise, any authenticator
 extension that does not otherwise require any result values MUST return
 a value and SHOULD return a CBOR Boolean authenticator extension output
 result, set to true to signify that the extension was understood and
 processed.

8.3. Extending request parameters

An extension defines one or two request arguments. The client extension
 input, which is a value that can be encoded in JSON, is passed from the
 Relying Party to the client in the get() or create() call, while the
 CBOR authenticator extension input is passed from the client to the
 authenticator for authenticator extensions during the processing of
 these calls.

A Relying Party simultaneously requests the use of an extension and
 sets its client extension input by including an entry in the extensions
 option to the create() or get() call. The entry key is the extension
 identifier and the value is the client input.

Extension definitions MUST specify the valid values for their client
 extension input. Clients SHOULD ignore extensions with an invalid
client extension input. If an extension does not require any parameters from the Relying Party, it SHOULD be defined as taking a Boolean client argument, set to true to signify that the extension is requested by the Relying Party.

Extensions that only affect client processing need not specify the authenticator extension input. Extensions that have authenticator processing MUST specify the method of computing the authenticator extension input from the client extension input. For extensions that do not require input parameters and are defined as taking a Boolean client extension input, this method SHOULD consist of passing an authenticator extension input value of true (CBOR major type 7, value 21).

Note: Extensions should aim to define authenticator arguments that are as small as possible. Some authenticators communicate over low-bandwidth links such as Bluetooth Low-Energy or NFC.

8.4. Client extension processing

Extensions may define additional processing requirements on the client platform during the creation of credentials or the generation of an assertion. The client extension input for each extension is used as an input to this client processing. Supported client extensions are recorded as a dictionary in the client data with the key clientExtensions. For each such extension, the client adds an entry to this dictionary with the extension identifier as the key, and the extension’s client extension input as the value.

Likewise, the client extension outputs are represented as a dictionary in the clientExtensionResults with extension identifiers as keys, and the client extension output value of each processed authenticator extension as the value. Like the client extension input, the client extension output is a value that can be encoded in JSON.

Extensions that require authenticator processing MUST define the process by which the client extension input can be used to determine the CBOR authenticator extension input and the process by which the CBOR authenticator extension output can be used to determine the client extension output.

8.5. Authenticator extension processing

As specified in 5.1 Authenticator data, the CBOR authenticator extension input value of each processed authenticator extension is included in the extensions data part of the authenticator data. This part is a CBOR map, with CBOR extension identifier values as keys, and the CBOR authentication extension input value of each extension as the value.

Likewise, the extension output is represented in the authenticator data as a CBOR map with CBOR extension identifiers as keys, and the CBOR authenticator extension output value of each extension as the value.

The authenticator extension processing rules are used to create the authenticator extension output from the authenticator extension input, and possibly also other inputs, for each extension.

8.6. Example Extension

This section is not normative.

To illustrate the requirements above, consider a hypothetical registration extension and authentication extension “Geo”. This extension, if supported, enables a geolocation location to be returned from the authenticator or client to the Relying Party.

The extension identifier is chosen as webauthnExample_geo. The client extension input is the constant value true, since the extension does not require the Relying Party to pass any particular information to the client extension input. If an extension does not require any parameters from the Relying Party, it SHOULD be defined as taking a Boolean client argument, set to true to signify that the extension is requested by the Relying Party.

Extensions that only affect client processing need not specify the authenticator extension input. Extensions that have authenticator processing MUST specify the method of computing the authenticator extension input from the client extension input. For extensions that do not require input parameters and are defined as taking a Boolean client extension input, this method SHOULD consist of passing an authenticator extension input value of true (CBOR major type 7, value 21).

Note: Extensions should aim to define authenticator arguments that are as small as possible. Some authenticators communicate over low-bandwidth links such as Bluetooth Low-Energy or NFC.

9.4. Client extension processing

Extensions may define additional processing requirements on the client platform during the creation of credentials or the generation of an assertion. The client extension input for each extension is used as an input to this client processing. Supported client extensions are recorded as a dictionary in the client data with the key clientExtensions. For each such extension, the client adds an entry to this dictionary with the extension identifier as the key, and the client extension input as the value.

Likewise, the client extension outputs are represented as a dictionary in the result of getClientExtensionResults() with extension identifiers as keys, and the client extension output value of each processed authenticator extension as the value. Like the client extension input, the client extension output is a value that can be encoded in JSON.

Extensions that require authenticator processing MUST define the process by which the client extension input can be used to determine the CBOR authenticator extension input and the process by which the CBOR authenticator extension output can be used to determine the client extension output.

9.5. Authenticator extension processing

The CBOR authenticator extension input value of each processed authenticator extension is included in the extensions data part of the authenticator request. This part is a CBOR map, with CBOR extension identifier values as keys, and the CBOR authenticator extension input value of each extension as the value.

Likewise, the extension output is represented in the authenticator data as a CBOR map with CBOR extension identifiers as keys, and the CBOR authenticator extension output value of each extension as the value.

The authenticator extension processing rules are used to create the authenticator extension output from the authenticator extension input, and possibly also other inputs, for each extension.

9.6. Example Extension

This section is not normative.

To illustrate the requirements above, consider a hypothetical registration extension and authentication extension “Geo”. This extension, if supported, enables a geolocation location to be returned from the authenticator or client to the Relying Party.

The extension identifier is chosen as webauthnExample_geo. The client extension input is the constant value true, since the extension does not require the Relying Party to pass any particular information to the Relying Party.
client, other than that it requests the use of the extension. The Relying Party sets this value in its request for an assertion:

```javascript
var assertionPromise = navigator.credentials.get({
  challenge: "SGFulFNvbG8gc2hvdCBuMC4=",
  allowCredentials: [], /* Empty filter */
  extensions: { 'webauthnExample_geo': true }
});
```

This authentication extension allows Relying Parties that have previously registered a credential using the legacy FIDO JavaScript APIs to request an assertion. Specifically, this extension allows Relying Parties to specify an appid (FIDO-APPID) to overwrite the otherwise computed rpId. This extension is only valid if used during the get() call; other usage will result in client error.

```javascript
Extension identifier: appid
```

This section defines the initial set of extensions to be registered in the IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registries]. These are recommended for implementation by user agents targeting broad interoperability.

9.1. FIDO AppId Extension (appid)

This authentication extension allows Relying Parties that have previously registered a credential using the legacy FIDO JavaScript APIs to request an assertion. Specifically, this extension allows Relying Parties to specify an appid (FIDO-APPID) to overwrite the otherwise computed rpId. This extension is only valid if used during the get() call; other usage will result in client error.

```javascript
Extension identifier: appid
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```

The extension also requires the client to set the authenticator parameter to the fixed value true.

```javascript
allowCredentials: [], /* Empty filter */
```
Supports a richer visual appearance.

This registration extension and authentication extension allows images to be used as transaction authorization prompts as well. This allows authenticators without a font rendering engine to be used and also supports a richer visual appearance.

### 9.3. Generic Transaction Authorization Extension (txAuthGeneric)

- **Extension identifier**: txAuthGeneric
- **Extension input**: A single JSON string prompt.
- **Extension output**: A single JSON string, representing the prompt as displayed on the client.
- **Authenticator extension input**: None.
- **Authenticator extension output**: The authenticator may insert line breaks if needed.
- **Authenticator extension processing**: The authenticator MUST display the prompt to the user before performing either user verification or test of user presence. The authenticator may insert line breaks if needed.

### 10.2. Simple Transaction Authorization Extension (txAuthSimple)

- **Extension identifier**: txAuthSimple
- **Extension input**: A single JSON string prompt.
- **Extension output**: None.
- **Extension processing**: None.

### 10.3. Generic Transaction Authorization Extension (txAuthGeneric)

- **Extension identifier**: txAuthGeneric
- **Extension input**: A single JSON string, representing the prompt as displayed on the client.
- **Extension output**: A single JSON string, representing the prompt as displayed on the client.
- **Extension processing**: None.

---

**Client extension input**: A single JSON string specifying a FIDO appid.

**Client extension processing**: If rpId is present, reject promise with a DOMException whose name is "NotAllowedError", and terminate this algorithm.

**Authenticator extension input**: None.

**Authenticator extension processing**: None.

**Authenticator extension output**: None.

**Authenticator extension processing**: None, except creating the authenticator extension input from the client extension input.

**Authenticator extension output**: None.

This registration extension and authentication extension allows for a simple form of transaction authorization. A Relying Party can specify a prompt string, intended for display on a trusted device on the authenticator.

**Extension identifier**: txAuthSimple

**Client extension input**: A single JSON string prompt.

**Client extension processing**: None, except creating the authenticator extension input from the client extension input.

**Client extension output**: None.

**Authenticator extension input**: The client extension input encoded as a CBOR text string (major type 3).

**Authenticator extension processing**: The authenticator MUST display the prompt to the user before performing either user verification or test of user presence. The authenticator may insert line breaks if needed.

**Authenticator extension output**: A single CBOR string, representing the prompt as displayed (including any eventual line breaks).

**Authenticator extension processing**: The authenticator MUST display the prompt to the user before performing either user verification or test of user presence. The authenticator may insert line breaks if needed.

**Authenticator extension output**: A single CBOR string, representing the prompt as displayed (including any eventual line breaks).

**Authenticator extension processing**: The authenticator MUST display the prompt to the user before performing either user verification or test of user presence. The authenticator may insert line breaks if needed.

This registration extension and authentication extension allows images to be used as transaction authorization prompts as well. This allows authenticators without a font rendering engine to be used and also supports a richer visual appearance.
Extension identifier
txAuthGeneric

Client extension input
A CBOR map defined as follows:

txAuthGenericArg = {
"content": text,
"contentType": text,
"mime-type": text
}

Client extension processing
None, except creating the authenticator extension input from the client extension input.

Client extension output
Returns the base64url encoding of the authenticator extension output value as a JSON string.

Authenticator extension input
The client extension input encoded as a CBOR map.

Authenticator extension processing
The authenticator MUST display the content to the user before performing either user verification or test of user presence.

The authenticator may add other information below the content.
No changes are allowed to the content itself, i.e., inside content boundary box.

Authenticator extension output
The hash value of the content which was displayed. The authenticator MUST use the same hash algorithm as it uses for the signature itself.

10.4. Authenticator Selection Extension (authnSel)

This registration extension allows a Relying Party to guide the selection of the authenticator that will be leveraged when creating the credential. It is intended primarily for Relying Parties that wish to tightly control the experience around credential creation.

Extension identifier
authnSel

Client extension input
A sequence of AAGUIDs:

typedef sequence<AAGUID> AuthenticatorSelectionList;

each AAGUID corresponds to an authenticator model that is acceptable to the Relying Party for this credential creation.

The list is ordered by decreasing preference.

An AAGUID is defined as an array containing the globally unique identifier of the authenticator model being sought.

typedef BufferSource AAGUID;

Client extension processing
This extension can only be used during create(). If the client supports the Authenticator Selection Extension, it MUST use the first available authenticator whose AAGUID is present in the AuthenticatorSelectionList. If none of the available authenticators match a provided AAGUID, the client MUST select an authenticator from among the available authenticators to generate the credential.

Client extension output
9.6. Supported Extensions Extension (exts)

This registration extension enables the Relying Party to determine which extensions the authenticator supports.

Extension identifier
- exts

Client extension input
- The Boolean value true to indicate that this extension is requested by the Relying Party.

Client extension processing
- None, except creating the authenticator extension input from the client extension input.

Client extension output
- Returns the list of supported extensions as a JSON array of extension identifier strings.

Authenticator extension input
- The Boolean value true, encoded in CBOR (major type 7, value 21).

Authenticator extension processing
- The authenticator sets the authenticator extension output to be a list of extensions that the authenticator supports, as defined below. This extension can be added to attestation objects.

Authenticator extension output
- The SupportedExtensions extension is a list (CBOR array) of extension identifier strings.

9.6. User Verification Index Extension (uvi)

This registration extension and authentication extension enables use of a user verification index.

Extension identifier
- uvi

Client extension input
- The Boolean value true to indicate that this extension is requested by the Relying Party.

Client extension processing
- None, except creating the authenticator extension input from the client extension input.

Client extension output
- Returns a JSON string containing the base64url encoding of the authenticator extension output.

Authenticator extension input
- The Boolean value true, encoded in CBOR (major type 7, value 21).

Authenticator extension processing
The authenticator sets the authenticator extension output to be a user verification index indicating the method used by the user to authorize the operation, as defined below. This extension can be added to attestation objects and assertions.

**Authenticator extension output**

The user verification index (UVI) is a value uniquely identifying a user verification data record. The UVI is encoded as a CBOR byte string (type 0x58). Each UVI value MUST be specific to the related key (in order to provide unlinkability). It also must contain sufficient entropy that makes guessing impractical. UVI values MUST NOT be reused by the Authenticator (for other biometric data or users).

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

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

Example for authenticator data containing one UVI extension

---

| 81 | -- [\{RP ID\}] hash (32 bytes) |
| 00 00 00 01 | -- (initial) signature counter |
| A1 | -- extension: CBOR map of one element |
| 63 | -- Key 1: CBOR text string of 3 byte |
| s | -- "uvI" [=UTF-8 encoded=] string |
| 58 20 | -- Value 1: CBOR byte string with bx |
| 20 bytes | |
| 00 43 B8 E3 BE 27 95 8C | -- the UVI value itself |
| 28 D5 74 BF 46 8A 85 CF | |
| 46 9A 14 FD E5 16 69 31 | |
| DA 4B CF FF C1 BB 11 32 | |
| 82 | |

**9.7. Location Extension (loc)**

The location registration extension and authentication extension provides the client device’s current location to the WebAuthn Relying Party.

---

**Extension identifier**

loc

**Client extension input**

The Boolean value true to indicate that this extension is requested by the Relying Party.

**Client extension processing**

None, except creating the authenticator extension input from the client extension input.

**Client extension output**

Returns a JSON object that encodes the location information in the authenticator extension output as a Coordinates value, as defined by The W3C Geolocation API Specification.
364f Authenticator extension input
364g The Boolean value true, encoded in CBOR (major type 7, value 21).

365f Authenticator extension processing
365g If the authenticator does not support the extension, then the authenticator MUST ignore the extension request. If the authenticator accepts the extension, then the authenticator SHOULD only add this extension data to a packed attestation or assertion.

366f Authenticator extension output
366g If the authenticator accepts the extension request, then authenticator extension output SHOULD provide location data in the form of a CBOR-encoded map, with the first value being the extension identifier and the second being an array of returned values. The array elements SHOULD be derived from (key, value) pairings for each location attribute that the authenticator supports. The following is an example of location data where the returned array is comprised of a (longitude, latitude, altitude) triplet, following the coordinate representation defined in the W3C Geolocation API Specification.

367f 9.8. User Verification Method Extension (uvm)
368f Returns a JSON array of 3-element arrays of numbers that encodes the factors in the authenticator extension output

-- Authenticator extension input
-- The Boolean value true, encoded in CBOR (major type 7, value 21).

-- Authenticator extension processing
If the authenticator does not support the extension, then the authenticator MUST ignore the extension request. If the authenticator accepts the extension, then the authenticator SHOULD only add this extension data to a packed attestation or assertion.

-- Authenticator extension output
If the authenticator accepts the extension request, then authenticator extension output SHOULD provide location data in the form of a CBOR-encoded map, with the first value being the extension identifier and the second being an array of returned values. The array elements SHOULD be derived from (key, value) pairings for each location attribute that the authenticator supports. The following is an example of location data where the returned array is comprised of a (longitude, latitude, altitude) triplet, following the coordinate representation defined in the W3C Geolocation API Specification.
Authenticator extension input
The Boolean value true, encoded in CBOR (major type 7, value 21).

Authenticator extension processing
- The authenticator sets the authenticator extension output to be a user verification index indicating the method used by the user to authorize the operation, as defined below. This extension can be added to attestation objects and assertions.

Authenticator extension output
The authenticator can report up to 3 different user verification methods (factors) used in a single authentication instance, using the CBOR syntax defined below:

```
uvmFormat = [ 1*3 uvmEntry ]
uvmEntry = [ useVerificationMethod: uint .size 4,
keyProtectionType: uint .size 2,
matcherProtectionType: uint .size 2 ]
```

The semantics of the fields in each uvmEntry are as follows:
- useVerificationMethod: CBOR integer for User Verification Method
- keyProtectionType: CBOR short for Key Protection Type
- matcherProtectionType: CBOR short for Matcher Protection Type

Example for authenticator data containing one UVM extension for a multi-factor authentication instance where 2 factors were used:

```
  -- [RP ID] = hash (32 bytes)
  81 00 00 00 01  -- (initial) signature counter
  75 76 6d 69 73  -- "uvm" [UTF-8 encoded] string
  82 02 02 02 02 02 -- [extension: CBOR map of one element]
  63 02 02 02 02 02 -- Subitem 1: CBOR short for Key Protection Type TEE
  63 02 02 02 02 02 -- Subitem 2: CBOR short for Matcher Protection Type So
```

Example for authenticator data containing one UVM extension for a multi-factor authentication instance where 2 factors were used:

```
  -- [RP ID] = hash (32 bytes)
  81 00 00 00 01  -- (initial) signature counter
  75 76 6d 69 73  -- "uvm" [UTF-8 encoded] string
  82 02 02 02 02 02 -- [extension: CBOR map of one element]
  63 02 02 02 02 02 -- Subitem 1: CBOR short for Key Protection Type TEE
  63 02 02 02 02 02 -- Subitem 2: CBOR short for Matcher Protection Type So
```
10. IANA Considerations

10.1. WebAuthn Attestation Statement Format Identifier Registrations

This section registers the attestation statement formats defined in Section 8 Defined Attestation Statement Formats in the IANA "WebAuthn Attestation Statement Format Identifier" registry established by [WebAuthn-Registrars].

- WebAuthn Attestation Statement Format Identifier: packed
  
  * Description: The "packed" attestation statement format is a WebAuthn-optimized format for attestation data. It uses a very compact but still extensible encoding method. This format is implementable by authenticators with limited resources (e.g., secure elements).

- Specification Document: Section 7.2 Packed Attestation Statement Format of this specification

- WebAuthn Attestation Statement Format Identifier: tpm

  * Description: The TPM attestation statement format returns an attestation statement in the same format as the packed attestation statement format, although the the rawData and signature fields are computed differently.

- Specification Document: Section 7.3 TPM Attestation Statement Format of this specification

- WebAuthn Attestation Statement Format Identifier: android-key

  * Description: Platform-provided authenticators based on Android versions "N", and later, may provide this proprietary "hardware attestation" statement.

- Specification Document: Section 7.4 Android Key Attestation Statement Format of this specification

- WebAuthn Attestation Statement Format Identifier: android-safetynet

  * Description: Android-based, platform-provided authenticators may produce an attestation statement based on the Android SafetyNet API.

- Specification Document: Section 7.5 Android SafetyNet Attestation Statement Format of this specification

- WebAuthn Attestation Statement Format Identifier: fido-u2f

  * Description: Used with FIDO U2F authentication form.

- Specification Document: Section 7.6 FIDO U2F Attestation Statement Format of this specification

10.2. WebAuthn Extension Identifier Registrations

This section registers the extension identifier values defined in Section 9 WebAuthn Extensions in the IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registrars].

- WebAuthn Extension Identifier: appid

  * Description: This extension identifier allows Relying Parties that have previously registered a credential using the legacy FIDO JavaScript APIs to request an assertion.

- Specification Document: Section 9.1 FIDO AppId Extension (appid) of this specification

- WebAuthn Extension Identifier: txAuthSimple

  * Description: This registration extension and authentication extension allows for a simple form of transaction authorization, A WebAuthn Relying Party can specify a prompt string, intended for display on a trusted device on the authenticator.

- Specification Document: Section 9.2 Simple Transaction Authorization Extension (txAuthSimple) of this specification

- WebAuthn Extension Identifier: txAuthGeneric

  * Description: This registration extension and authentication extension allows images to be used as transaction authorization prompts as well. This allows authenticators without a font rendering engine to be used and also supports a richer visual appearance than accomplished with the webauthn.txauth.simple extension.

- Specification Document: Section 9.3 Generic Transaction Authorization Extension (txAuthGeneric) of this specification

- WebAuthn Extension Identifier: authnSel

  * Description: This registration extension and authentication extension allows users to select which authenticator to use.

- Specification Document: Section 9.4 Generic Transaction Authorization Extension (txAuthGeneric) of this specification

11. IANA Considerations

11.1. WebAuthn Attestation Statement Format Identifier Registrations

This section registers the attestation statement formats defined in Section 8 Defined Attestation Statement Formats in the IANA "WebAuthn Attestation Statement Format Identifier" registry established by [WebAuthn-Registrars].

- WebAuthn Extension Statement Format Identifier: packed

  * Description: The "packed" attestation statement format is a WebAuthn-optimized format for attestation data. It uses a very compact but still extensible encoding method. This format is implementable by authenticators with limited resources (e.g., secure elements).

- Specification Document: Section 8.2 Packed Attestation Statement Format of this specification

- WebAuthn Extension Statement Format Identifier: tpm

  * Description: The TPM attestation statement format returns an attestation statement in the same format as the packed attestation statement format, although the the rawData and signature fields are computed differently.

- Specification Document: Section 8.3 TPM Attestation Statement Format of this specification

- WebAuthn Extension Statement Format Identifier: android-key

  * Description: Platform-provided authenticators based on Android versions "N", and later, may provide this proprietary "hardware attestation" statement.

- Specification Document: Section 8.4 Android Key Attestation Statement Format of this specification

- WebAuthn Extension Statement Format Identifier: android-safetynet

  * Description: Android-based, platform-provided authenticators may produce an attestation statement based on the Android SafetyNet API.

- Specification Document: Section 8.5 Android SafetyNet Attestation Statement Format of this specification

- WebAuthn Extension Statement Format Identifier: fido-u2f

  * Description: Used with FIDO U2F authentication form.

- Specification Document: Section 8.6 FIDO U2F Attestation Statement Format of this specification

11.2. WebAuthn Extension Identifier Registrations

This section registers the extension identifier values defined in Section 9 WebAuthn Extensions in the IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registrars].

- WebAuthn Extension Identifier: appid

  * Description: This extension identifier allows Relying Parties that have previously registered a credential using the legacy FIDO JavaScript APIs to request an assertion.

- Specification Document: Section 10.1 FIDO AppId Extension (appid) of this specification

- WebAuthn Extension Identifier: txAuthSimple

  * Description: This registration extension and authentication extension allows for a simple form of transaction authorization.

- Specification Document: Section 10.2 Simple Transaction Authorization Extension (txAuthSimple) of this specification

- WebAuthn Extension Identifier: txAuthGeneric

  * Description: This registration extension and authentication extension allows images to be used as transaction authorization prompts as well. This allows authenticators without a font rendering engine to be used and also supports a richer visual appearance than accomplished with the webauthn.txauth.simple extension.

- Specification Document: Section 10.3 Generic Transaction Authorization Extension (txAuthGeneric) of this specification

- WebAuthn Extension Identifier: authnSel
Recommended: No

* Description: This registration extension enables the Relying Party to determine which extensions the authenticator supports. The extension data is a list (CBOR array) of extension identifiers encoded as UTF-8 Strings. This extension is added automatically by the authenticator. This extension can be added to attestation statements.

* Specification Document: Section 10.6 User Verification Index Extension (uvi) of this specification

11.3. COSE Algorithm Registrations

This section registers identifiers for RSASSA-PKCS1-v1_5 [RFC8017] algorithms using SHA-2 and SHA-1 hash functions in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG].

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Reference</th>
<th>Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS256</td>
<td>-256</td>
<td>Section 8.2 of [RFC8017]</td>
<td>No</td>
</tr>
<tr>
<td>RS384</td>
<td>-258</td>
<td>Section 8.2 of [RFC8017]</td>
<td>No</td>
</tr>
<tr>
<td>RS512</td>
<td>-259</td>
<td>Section 8.2 of [RFC8017]</td>
<td>No</td>
</tr>
<tr>
<td>RS384</td>
<td>-258</td>
<td>Section 8.2 of [RFC8017]</td>
<td>No</td>
</tr>
<tr>
<td>RS512</td>
<td>-259</td>
<td>Section 8.2 of [RFC8017]</td>
<td>No</td>
</tr>
</tbody>
</table>

This section registers identifiers for RSASSA-PKCS1-v1_5 [RFC8017] algorithms using SHA-2 and SHA-1 hash functions in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG]. It also registers identifiers for ECDSA algorithms:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Reference</th>
<th>Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS256</td>
<td>-256</td>
<td>Section 8.2 of [RFC8017]</td>
<td>No</td>
</tr>
<tr>
<td>RS384</td>
<td>-258</td>
<td>Section 8.2 of [RFC8017]</td>
<td>No</td>
</tr>
<tr>
<td>RS512</td>
<td>-259</td>
<td>Section 8.2 of [RFC8017]</td>
<td>No</td>
</tr>
</tbody>
</table>

This section registers identifiers for RSASSA-PKCS1-v1_5 [RFC8017] algorithms using SHA-2 and SHA-1 hash functions in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG]. It also registers identifiers for ECDSA algorithms:
The sample code for generating and registering a new key follows:

```javascript
var publicKey = {
    ... // code for generating and registering a new key...

    // Server side code...
    ... // code for generating and registering a new key...

    // Client side code...
    ... // code for generating and registering a new key...

    // Other relevant code...
    ... // code for generating and registering a new key...
};
```

As was the case in earlier sections, this flow focuses on a use case involving an external first-factor authenticator with its own display.

One example of such an authenticator would be a smart phone. Other authenticator types are also supported by this API, subject to implementation by the platform. For instance, this flow also works without modification for the case of an authenticator that is embedded in the client platform. The flow also works for the case of an authenticator without its own display (similar to a smart card) subject to specific implementation considerations. Specifically, the client platform needs to display any prompts that would otherwise be shown by the authenticator, and the authenticator needs to allow the client platform to enumerate all the authenticator’s credentials so that the client can have information to show appropriate prompts.

### 11. Sample scenarios

This is the first-time flow, in which a new credential is created and registered with the server. In this flow, the Relying Party does not have a preference for platform authenticator or roaming authenticators.

1. The user visits example.com, which serves up a script. At this point, the user may already be logged in using a legacy username and password, or additional authenticator, or other means acceptable to the Relying Party. Or the user may be in the process of creating a new account.
2. The Relying Party script runs the code snippet below.
3. The client platform searches for and locates the authenticator.
4. The client platform connects to the authenticator, performing any pairing actions if necessary.
5. The authenticator shows appropriate UI for the user to select the authenticator on which the new credential will be created, and obtains a biometric or other authorization gesture from the user.
6. The authenticator returns a response to the client platform, which in turn returns a response to the Relying Party script. If the user declined to select an authenticator or provide authorization, an appropriate error is returned.
7. If a new credential was created, + The Relying Party script sends the newly generated credential public key to the server, along with additional information such as attestation regarding the provenance and characteristics of the authenticator.
+ The server stores the public key in its database and associates it with the user as well as with the characteristics of authentication indicated by attestation, also storing a friendly name for later use.
+ The script may store data such as the credential ID in local storage, to improve future UX by narrowing the choice of credential for the user.

The sample code for generating and registering a new key follows:

```javascript
if ((PublicKeyCredential) /* Platform not capable. Handle error. */) {
    var publicKey = {
        ... // code for generating and registering a new key...
    };
    // Other relevant code...
    ... // code for generating and registering a new key...
}
```
then(function (userIntent) {

  if (!PublicKeyCredential) { /* Platform not capable of the API. Handle error. */
      return Promise.reject("Platform not supported");
  }

  // Relying Party:
  rp: {
    name: "Acme"
  },

  // User:
  user: {
    id: "098237235409872"
  },

  // Name:
  name: "John.p.smith@example.com",

  // Icon:
  icon: "https://pics.acme.com/00/p/A8jjppPb.png"

  // This Relying Party will either accept an ES256 or RS256 credential, but
  // prefers an ES256 credential.
  pubKeyCredParams: [
    
      // "public-key",
      type: "public-key",

      // "RS256" as registered in the IANA COSE Algorithms registry
      alg: -257 // Value registered by this specification for "RS256"

  ],

  // Time out:
  timeout: 60000, // 1 minute

  // Extensions:
  extensions: {
    "webauthn:location": true // Include location information
  },

  // Relying Party:
  rp: {
    name: "Acme"
  },

  // User:
  user: {
    id: "098237235409872"
  },

  // Name:
  name: "John.p.smith@example.com",

  // Icon:
  icon: "https://pics.acme.com/00/p/A8jjppPb.png"

  // This Relying Party will either accept an ES256 or RS256 credential, but
  // prefers an ES256 credential.
  pubKeyCredParams: [
    
      // "public-key",
      type: "public-key",

      // "ES256" as registered in the IANA COSE Algorithms registry
      alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry

  ],

  // Time out:
  timeout: 60000, // 1 minute

  // Extensions:
  extensions: {
    "webauthn:location": true // Include location information
  },

  // Note: The following call will cause the authenticator to display UI.
  navigator.credentials.create( publicKey )
  .then(function (newCredentialInfo) {
    
      // Send new credential info to server for verification and registration.
      navigator.credentials.get( newCredentialInfo )
       .then(function (newCredential) {
         
        // No acceptable authenticator or user refused consent. Handle appropriately
        },

      }),

  // 11.2. Registration Specifically with Platform Authenticator

  // This is flow for when the Relying Party is specifically interested in
  creating a public key credential with a platform authenticator.

  // 1. The user visits example.com and clicks on the login button, which
  // redirects the user to login.example.com.
  1. The user visits example.com and clicks on the login button, which
      redirects the user to login.example.com.

  // 2. The user enters a username and password to log in. After successful
  // login, the user is redirected back to example.com.
  2. The user enters a username and password to log in. After successful
    login, the user is redirected back to example.com.

  // 3. The Relying Party script runs the code snippet below.
  3. The Relying Party script runs the code snippet below.

  // 4. The user agent asks the user if they are willing to register
  // with the Relying Party using an available platform authenticator.
  4. The user agent asks the user if they are willing to register
    with the Relying Party using an available platform authenticator.

  // 5. If the user is not willing, terminate this flow.
  5. If the user is not willing, terminate this flow.

  // 6. The user is shown appropriate UI and guided in creating a
  // credential using one of the available platform authenticators. Upon
  // successful credential creation, the RP script conveys the new
  // credential to the server.
  6. The user is shown appropriate UI and guided in creating a
    credential using one of the available platform authenticators. Upon
    successful credential creation, the RP script conveys the new
    credential to the server.

  // If (PublicKeyCredential) {
    // Platform not capable of the API. Handle error.
  // }

  // 11.2. Registration Specifically with User Verifying Platform Authenticator

  // This is flow for when the Relying Party is specifically interested in
  creating a public key credential with a user-verifying platform
  authenticator.

  // 1. The user visits example.com and clicks on the login button, which
  // redirects the user to login.example.com.
  1. The user visits example.com and clicks on the login button, which
    redirects the user to login.example.com.

  // 2. The user enters a username and password to log in. After successful
  // login, the user is redirected back to example.com.
  2. The user enters a username and password to log in. After successful
    login, the user is redirected back to example.com.

  // 3. The Relying Party script runs the code snippet below.
  3. The Relying Party script runs the code snippet below.

  // 4. The user agent asks the user if they are willing to register
  // with the Relying Party using an available platform authenticator.
  4. The user agent asks the user if they are willing to register
    with the Relying Party using an available platform authenticator.

  // 5. If the user is not willing, terminate this flow.
  5. If the user is not willing, terminate this flow.

  // 6. The user is shown appropriate UI and guided in creating a
  // credential using one of the available platform authenticators. Upon
  // successful credential creation, the RP script conveys the new
  // credential to the server.
  6. The user is shown appropriate UI and guided in creating a
    credential using one of the available platform authenticators. Upon
    successful credential creation, the RP script conveys the new
    credential to the server.

  // If (PublicKeyCredential) {
    // Platform not capable of the API. Handle error.
  // }
// The challenge must be produced by the server, see the Security

11.3. Authentication

This is the flow when a user with an already registered credential visits a website and wants to authenticate using the credential. This is the flow when a user with an already registered credential visits a website and wants to authenticate using the credential.

1. The user visits example.com, which serves up a script.

2. The script asks the client platform for an Authentication

   Assertion, providing as much information as possible to narrow the choice of acceptable credentials for the user. The user may be obtained from data that was stored locally after registration, or by user behavior such as prompting the user for a username.

3. The Relying Party script runs one of the code snippets below.

4. The client platform searches for and locates the authenticator with some information on the origin that is requesting these keys.

5. The client platform connects to the authenticator, performing any pairing actions if necessary.

6. The authenticator presents the user with a notification that their attention is required. On opening the notification, the user is shown a friendly selection menu of acceptable credentials using the account information provided when creating the credentials, along with some information on the origin that is requesting these keys.

7. The authenticator obtains a biometric or other authorization gesture from the user.

8. The authenticator returns a response to the client platform, which in turn returns a response to the Relying Party script. If the user declined to select a credential or provide an authorization, an appropriate error is returned.

9. If an assertion was successfully generated and returned,

   + The script sends the assertion to the server.
   + The server examines the assertion, extracts the credential ID, looks up the registered credential public key if it is database, and verifies the assertion’s signature assertion. If valid, it looks up the identity associated with the assertion’s credential ID; that identity is now authenticated.
   
   If the credential ID is not recognized by the server (e.g., it has been deregistered due to inactivity) then the authentication has failed; each Relying Party will handle this in its own way.

   + The server now does whatever it would otherwise do upon successful authentication — return a success page, set authentication cookies, etc.

If the Relying Party script does not have any hints available (e.g., from locally stored data) to help it narrow the list of credentials, then the sample code for performing such an authentication might look like this:

```javascript
if ((PublicKeyCredential) /* Platform not capable. Handle error. */) var options = { challenge: new TextEncoder().encode("climb a mountain"),
  }
```

```javascript
else {
  return navigator.credentials.create({"publicKey": publicKeyOptions, // Send new credential info to server for verification and registration.
    }).then(function(newCredentialInfo) {
      // Something went wrong. Handle appropriately.
    });

  if (userIntent) {
    if (userIntent) {
      return navigator.credentials.create({"publicKey": publicKeyOptions, // Create and register credentials.
        }).then(function(newCredentialInfo) {
          // Send new credential info to server for verification and registration.
        });
    } else {
        // Record that the user does not intend to use a platform authenticator
        // and default the user to a password-based flow in the future.
      }).catch( function(err) {
        // Something went wrong. Handle appropriately.
      });
    }
  } else {
    // If the user has affirmed willingness to register with RP using an ava
    // ilable platform authenticator
    if (userIntent) {
      var publicKeyOptions = { /* Public key credential creation options.
        */
        // Create and register credentials.
        return navigator.credentials.create({"publicKey": publicKeyOptions, // Send new credential info to server for verification and registration.
          });
    } else {
      // Record that the user does not intend to use a platform authenticator
      // and default the user to a password-based flow in the future.
    }
    }
  } else {
    // Something went wrong. Handle appropriately.
  });
```

```javascript
// If the user has affirmed willingness to register with RP using an ava
// ilable platform authenticator
if (userIntent) {
  var publicKeyOptions = { /* Public key credential creation options.
    */
    // Create and register credentials.
    return navigator.credentials.create({"publicKey": publicKeyOptions, // Send new credential info to server for verification and registration.
      });
  } else {
    // Record that the user does not intend to use a platform authenticator
    // and default the user to a password-based flow in the future.
  }
  }
```
11.4. Decommissioning

On the other hand, if the Relying Party script has some hints to help it narrow the list of credentials, then the sample code for performing such an authentication might look like the following. Note that this sample also demonstrates how to use the extension for transaction authorization.

navigator.credentials.get({ "publicKey": options }).then(function (assertion) {
    // Send assertion to server for verification
    // Send assertion to server for verification
    }).catch(function (err) {
        // No acceptable credential or user refused consent. Handle appropriately.
    });

On the other hand, if the Relying Party script has some hints to help it narrow the list of credentials, then the sample code for performing such an authentication might look like the following. Note that this sample also demonstrates how to use the extension for transaction authorization.

if (!PublicKeyCredential) { /* Platform not capable. Handle error. */

var encoder = new TextEncoder();
var acceptableCredential1 = {
    type: "public-key",
    id: encoder.encode("!!!!!!!hi there!!!!!!!")
};
var acceptableCredential2 = {
    type: "public-key",
    id: encoder.encode("roses are red, violets are blue\n")
};

var options = {
    challenge: encoder.encode("climb a mountain"),
    timeout: 60000, // 1 minute
    allowCredentials: [acceptableCredential1, acceptableCredential2]
};

navigator.credentials.get({ "publicKey": options }).then(function (assertion) {
    // Send assertion to server for verification
    }).catch(function (err) {
        // No acceptable credential or user refused consent. Handle appropriately.
    });

11.4. Decommissioning

The below example shows how a developer may use the AbortSignal parameter to abort a credential registration operation. A similar procedure applies to an authentication operation.

const authAbortController = new AbortController;
const authAbortSignal = authAbortController.signal;

authAbortSignal.onabort = function () {
    // Once the page knows the abort started, inform user it is attempting to abort.
};

var options = {
    // A list of options.
    publicKey: options,
    signal: authAbortSignal
};
The following are possible situations in which decommissioning a credential might be desired. Note that all of these are handled on the server side and do not require the API to be specified here.

- Possibility #1 -- user reports the credential as lost.
  +User goes to server.example.net, authenticates and follows a link to report a lost/stolen device.
  +Server returns a page showing the list of registered credentials with friendly names as configured during registration.
  +User selects a credential and the server deletes it from its database.
  +In future, the Relying Party script does not specify this credential in any list of acceptable credentials, and assertions signed by this credential are rejected.

- Possibility #2 -- server deregisters the credential due to inactivity.
  +Server deletes credential from its database during maintenance activity.
  +In future, the Relying Party script does not specify this credential in any list of acceptable credentials, and assertions signed by this credential are rejected.

- Possibility #3 -- user deletes the credential from the device.
  +User employs a device-specific method (e.g., device settings UI) to delete a credential from their device.
  +From this point on, this credential will not appear in any selection prompts, and no assertions can be generated with it.

+ Sometimes later, the server deregisters this credential due to inactivity.

12. Acknowledgements

We thank the following for their contributions to, and thorough review of, this specification: Richard Barnes, Dominic Bartr, Domenic Denicola, Rahul Ghosh, Brad Hill, Jing Jin, Angelo Liao, Anne van Kesteren, Ian Kilpatrick, Giridhar Mandyam, Azel Nennker, Kimberly Paulhamus, Adam Powers, Yaron Sheffer, Mike West, Jeffrey Yasskin, Boris Zbarsky.
Terms defined by this specification

* aaguid, in 6.3.1
* AAGUID, in 10.4
* alg, in 6.3
* allowCredentials, in 5.5
* Assertion, in 4
* assertion signature, in 6
* attachment, in 9
* attachment mode, in 9
* Attestation, in 4
* attachment mode, in 9.4.5
* Attestation Certificate, in 4
* Attestation Conveyance, in 5.4.6
* attestation trust path, in 6.3.2
* Attestation Certificate, in 4
* Attestation Conveyance, in 5.4.6
* Attestation Conveyance Preference Option, in 5.1.3
* assertion key pair, in 4
* public key, in 4.3
* assertion signature, in 6
* attestation object, in 6.3
* attestation object, in 5.1.3
* public key, in 4.3
* attestation object, in 6.3
* attestation object, in 6.3
* attestation objectResult, in 5.1.3
* trust path, in 6.3.2
* Attestation Certificate, in 4
* attestation trust path, in 6.3.2
* Attestation Conveyance, in 5.4.6
* attestation trust path, in 6.3.2
* Attestation Certificate, in 4
* Attestation Conveyance, in 5.4.6
* attestation trust path, in 6.3.2
* Attestation Certificate, in 4
* Attestation Conveyance, in 5.4.6
* attachment type, in 5.3
* attachment type, in 6.3
* Attested credential data, in 6.3.1
* attestedCredentialData, in 6.3.1
* authExtensions, in 6.3.1
* authentication, in 4
* Authentication Assertion, in 4
* authentication, in 5
* attachment type, in 6.3
* Attestation Certificate, in 4
* attachment type, in 6.3
* Attestation Certificate, in 4
* attachment type, in 6.3
* Attestation Conveyance, in 5.4.6
* attachment type, in 6.3
* Attestation Conveyance, in 5.4.6
* attachment type, in 6.3
* Attestation Conveyance, in 5.4.6
* attachment type, in 6.3
* Attestation Conveyance, in 5.4.6
* attachment type, in 6.3
* Attestation Conveyance, in 5.4.6
* attachment type, in 6.3
** User Present, in 3
** User Present, in 3
* User Consent, in 3
* usb, in 4.7.4
* origin, in 4.7.1
* "plat", in 4.4.4
* plat, in 4.4.4
* platform attachment, in 4.4.4
* platform authenticators, in 4.4.4
* Privacy CA, in 5.3.3
* publicKeyCredParams, in 4.4

* publicKey
  + dict-member for CredentialCreationOptions, in 4.1.1
  + dict-member for CredentialRequestOptions, in 4.1.2
  + publicKey, in 4.5.2
  + PublicKey Credential, in 3
  + PublicKeyCredential, in 4.1
  + PublicKeyCredential Descriptor, in 4.7.3
  + PublicKeyCredential Entity, in 4.4.1
  + PublicKeyCredential Parameters, in 4.3
  + PublicKeyCredentialRequestOptions, in 4.5
  + PublicKeyCredentialRequestType, in 4.7.2
  + PublicKeyCredentialUserEntity, in 4.4.2
  + Rate Limiting, in 3
  + rawId, in 4.1
  + Registration, in 3
  + registration extension, in 8
  + Relying Party, in 3
  + Relaying Party Identifier, in 3
  + response, in 4.1
  + rk, in 4.4.3
  + roaming authenticators, in 4.4.4
  + rp, in 5.5.6
  + rpid, in 4.5
  + RP ID, in 3
  + Self Attestation, in 5.3.3
  + signature, in 4.2.2
  + Signing procedure, in 5.3.2
  + Test of User Presence, in 3
  + transports, in 4.7.3
  + [[type]], in 4.1
  + type
  + dict-member for PublicKeyCredentialParameters, in 4.3
  + dict-member for PublicKeyCredentialDescriptor, in 4.7.3
  + UP, in 3
  + usb, in 4.7.4
  + user, in 4.4
  + User Consent, in 3
  + User Present, in 3

* timeout

+ dict-member for MakePublicKeyCredentialOptions, in 4.4
+ dict-member for PublicKeyCredentialRequestOptions, in 4.5
+ tokenBindingId, in 4.7.1
+ transports, in 4.7.3
+ type
+ dict-member for PublicKeyCredentialParameters, in 4.3
+ dict-member for PublicKeyCredentialDescriptor, in 4.7.3
+ UP, in 3
+ usb, in 4.7.4
+ user, in 4.4
+ User Consent, in 3
+ User Present, in 3

* identifier of the ECDAA-Issuer public key, in 7.2
* PlatformAuthenticatorAvailable(), in 4.1.9
* JSON-serialized client data, in 4.7.1
* MakePublicKeyCredentialOptions, in 4.4
* name, in 4.4.1
* nfc, in 4.7.4
* origin, in 4.7.1
* "plat", in 4.4.4
* plat, in 4.4.4
* platform attachment, in 4.4.4
* platform authenticators, in 4.4.4
* Privacy CA, in 5.3.3
* publicKeyCredParams, in 4.4

+ dict-member for CredentialRequestOptions, in 4.1.2
+ dict-member for CredentialCreationOptions, in 4.1.1
+ dict-member for MakePublicKeyCredentialOptions, in 4.4
+ dict-member for CredentialRequestOptions, in 4.5
+ dict-member for PublicKeyCredentialOptions, in 4.3
+ dict-member for MakePublicKeyCredentialOptions, in 4.4
+ dict-member for PublicKeyCredentialRequestOptions, in 4.5
+ dict-member for PublicKeyCredentialRequestOptions, in 5.5
+ PublicKey Credential, in 4
+ publicKey, in 5.8.2
+ PublicKey Credential, in 4.1
+ PublicKeyCredentialType, in 4.7.2
+ PublicKeyCredentialEntity, in 4.4.1
+ PublicKeyCredentialUserEntity, in 5.4.3
+ Rate Limiting, in 3
+ rawId, in 4.1
+ Registration, in 3
+ registration extension, in 8
+ Relying Party, in 3
+ Relaying Party Identifier, in 3
+ response, in 4.1
+ rk, in 4.4.3
+ roaming authenticators, in 4.4.4
+ rp, in 5.5.6
+ rpid, in 4.5
+ RP ID, in 3
+ Self Attestation, in 5.3.3
+ signature, in 4.2.2
+ Signing procedure, in 5.3.2
+ Test of User Presence, in 3

* identifier of the ECDAA-Issuer public key, in 8.2
* PlatformAuthenticatorAvailable(), in 5.4.6
* isUserVerifyingPlatformAuthenticatorAvailable(), in 5.1.6
* JSON-serialized client data, in 5.8.1
* MakePublicKeyCredentialOptions, in 5.4
* managing authenticator, in 4
* name, in 5.4.1
* nfc, in 5.8.4
* none, in 5.4.6
* origin, in 5.8.1
* platform, in 5.4.5
* "platform", in 5.4.5
* platform attachment, in 5.4.5
* platform authenticators, in 5.4.5
* "preferred", in 5.8.6
* preferred, in 5.8.6
* Privacy CA, in 6.3.3
* publicKeyCredParams, in 5.4
* PublicKey
+ dict-member for CredentialCreationOptions, in 5.1.1
+ dict-member for CredentialRequestOptions, in 5.1.2
+ publicKey, in 5.8.2
+ PublicKey Credential, in 4
+ PublicKeyCredential, in 5.1
+ PublicKeyCredentialDescriptor, in 5.8.3
+ PublicKeyCredentialEntity, in 5.4.1
+ PublicKeyCredentialParameters, in 5.3
+ PublicKeyCredentialRequestOptions, in 5.5
+ PublicKeyCredentialRequestType, in 5.4.2
+ PublicKey Credential Source, in 4
+ PublicKeyCredentialType, in 5.8.2
+ PublicKeyCredentialUserEntity, in 5.4.3
+ Rate Limiting, in 4
+ rawId, in 5.1
+ Registration, in 4
+ registration extension, in 9
+ Relaying Party, in 4
+ Relaying Party Identifier, in 4
+ "required", in 5.8.6
+ required, in 5.8.6
+ requireResidentKey, in 5.4.4
+ response, in 5.1
+ roaming authenticators, in 5.4.5
+ rp, in 5.4
+ rpid, in 5.5
+ RP ID, in 4
+ "rpidHash", in 6.1
+ Self Attestation, in 6.3.3
+ signature, in 5.2.2
+ Signature Counter, in 6.1.1
+ signatureResult, in 5.1.4.1
+ signCount, in 6.1
+ Signing procedure, in 6.3.2
+ [Store] credential, sameOriginWithAncestors, in 5.1.5
+ Test of User Presence, in 4
+ timeout

+ dict-member for MakePublicKeyCredentialOptions, in 5.4
+ dict-member for PublicKeyCredentialRequestOptions, in 5.5
+ tokenBindingId, in 5.8.1
+ transports, in 5.8.3
+ [type], in 5.1
+ type
+ dict-member for PublicKeyCredentialParameters, in 5.3
+ dict-member for CollectedClientData, in 5.8.3
+ dict-member for PublicKeyCredentialDescriptor, in 5.8.3
+ UP, in 4
+ usb, in 5.8.4
+ user, in 5.4
+ User Consent, in 4

77/109
Terms defined by reference

* [CREDENTIAL-MANAGEMENT-1] defines the following terms:
  - Credential
  - CredentialCreationOptions
  - CredentialRequestOptions
  - CredentialsContainer
  - (CollectFromCredentialStore)(options)
  - (Store)(credential)

  + [[discovery]]
  + [[type]]
  + create()

+ get()
+ id
+ remote

+ type

* [ECMAScript] defines the following terms:
  + %arraybuffer%
  + internal slot

+ stringify
+ internal method

* [ENCODING] defines the following terms:
  + utf-8 encode

* [HTML] defines the following terms:
  + ascii serialization of an origin
  + dom manipulation task source
  + effective domain

+ global object
  + in parallel
+ is a registrable domain suffix of or is equal to
+ is not a registrable domain suffix of and is not equal to
+ origin
+ promise
+ relevant settings object
+ task

* [HTML52] defines the following terms:
  + userVerification, in 3
  + User Verified, in 3
  + UV, in 3
  + xplat, in 4.4.4
  + xplat, in 4.4.4

  + Proof
  + relevant settings object
  + origin
  + is not a registrable domain suffix of and is not equal to
  + is a registrable domain suffix of or is equal to

+ in parallel
+ global object
  + in parallel
+ is a registrable domain suffix of or is equal to
+ is not a registrable domain suffix of and is not equal to
+ origin
+ promise
+ relevant settings object
+ task

* [HTML52] defines the following terms:
  + User Handle, in 5.2.2
  + UserHandle, in 4
  + UserHandleResult, in 5.1.4.1
  + User Present, in 4
  + userVerification
  + dict-member for AuthenticatorSelectionCriteria, in 5.4.4
  + dict-member for PublicKeyCredentialRequestOptions, in 5.5

* User Verification, in 4
  + UserVerificationRequirement, in 5.8.6
  + UV, in 4
  + Verification procedure, in 6.3.2
  + verification procedure inputs, in 6.3.2

* Web Authentication API, in 5
  + WebAuthn Client, in 4

Terms defined by reference

* [CREDENTIAL-MANAGEMENT-1] defines the following terms:
  + Credential
  + CredentialCreationOptions
  + CredentialRequestOptions
  + CredentialsContainer
  + (CollectFromCredentialStore)(options)
  + (Store)(credential)

  + [[discovery]]
  + [[type]]
  + create()

+ get()
+ id
+ remote

+ same-origin with its ancestors
  + signal (for CredentialCreationOptions)
  + signal (for CredentialRequestOptions)
  + store()

+ type

* [DOM] defines the following terms:
  + AbortController
  + aborted flag
  + document

* [ECMAScript] defines the following terms:
  + %arraybuffer%
  + internal slot

+ stringify
+ internal method

* [ENCODING] defines the following terms:
  + utf-8 encode

* [FETCH] defines the following terms:
  + window

* [HTML] defines the following terms:
  + ascii serialization of an origin
  + dom manipulation task source

+ effective domain
+ environment settings object
+ global object

+ is a registrable domain suffix of or is equal to
+ is not a registrable domain suffix of and is not equal to
+ origin

+ relevant settings object
IDL Index

[SecureContext]
interface PublicKeyCredential : Credential {
  [SameObject] readonly attribute ArrayBuffer rawid;
  [SameObject] readonly attribute AuthenticatorResponse response;
  [SameObject] readonly attribute AuthenticationExtensions clientExtensionResults;
};

[SecureContext]
interface PublicKeyCredentialRequestOptions {
  PublicKeyCredentialRequestOptions publicKey;
};

[SecureContext]
interface PublicKeyCredentialCreationOptions {
  MakePublicKeyCredentialOptions publicKey;
};

[SecureContext]
interface PublicKeyCredentialRequestOptions {
  PublicKeyCredentialRequestOptions publicKey;
};

[SecureContext]
interface PublicKeyCredentialOptions {
  [Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();
};

[SecureContext]
interface AuthenticatorResponse {
  [SameObject] readonly attribute ArrayBuffer clientDataJSON;
};

[SecureContext]
interface AuthenticatorAttestationResponse : AuthenticatorResponse {
  [SameObject] readonly attribute ArrayBuffer attestationObject;
};

[SecureContext, Exposed=Window]
interface PublicKeyCredential : Credential {
  [SameObject] readonly attribute ArrayBuffer rawid;
  [SameObject] readonly attribute AuthenticatorResponse response;
  [SameObject] readonly attribute AuthenticationExtensions clientExtensionResults;
};

[SecureContext, Exposed=Window]
interface PublicKeyCredentialRequestOptions {
  PublicKeyCredentialRequestOptions publicKey;
};

[SecureContext, Exposed=Window]
interface PublicKeyCredentialOptions {
  [Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();
};

[SecureContext]
interface AuthenticatorResponse {
  [SameObject] readonly attribute ArrayBuffer clientDataJSON;
};

[SecureContext]
interface AuthenticatorAttestationResponse : AuthenticatorResponse {
  [SameObject] readonly attribute ArrayBuffer attestationObject;
};

[SecureContext, Exposed=Window]
interface PublicKeyCredentialOptions {
  [Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();
};

[SecureContext, Exposed=Window]
interface AuthenticatorAttestationResponse : AuthenticatorResponse {
  [SameObject] readonly attribute ArrayBuffer attestationObject;
};
dictionary CollectedClientData {
        typedef record<DOMString, any> AuthenticationExtensions;
    }

AuthenticationExtensions extensions;

sequence<PublicKeyCredentialDescriptor> allowCredentials = [];

USVString rpId;
unsigned long timeout;
required BufferSource challenge;

dictionary PublicKeyCredentialRequestOptions {
    required PublicKeyCredentialType type;
    required COSEAlgorithmIdentifier alg;
    enum AuthenticatorAttachment {
        "direct", "indirect",
    } authenticatorAttachment;
    boolean requireUserVerification = false;
    boolean requireResidentKey = false;
    DOMString id;
    USVString icon;
}

interface AuthenticatorAssertionResponse : AuthenticatorResponse {
    [SecureContext]
    [SameObject] readonly attribute ArrayBuffer authenticatorData;
    [SameObject] readonly attribute ArrayBuffer userHandle;
}

dictionary PublicKeyCredentialParameters {
    required PublicKeyCredentialUserEntity user;
    required PublicKeyCredentialUserEntity rp;
    required PublicKeyCredentialUserEntity step;
    sequence<PublicKeyCredentialParameters> pubKeyCredParams;
    unsigned long timeout;
    sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
    AuthenticatorSelectionCriteria authenticatorSelection;
    AttestationConveyancePreference attestation = "none";
    AuthenticationExtensions extensions;
}

dictionary PublicKeyCredentialEntity {
    DOMString id;
    USVString icon;
    dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
        DOMString displayName;
    }
    AuthenticatorSelectionCriteria authenticatorSelection;
    unsigned long timeout;
    required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
    required BufferSource challenge;
    required PublicKeyCredentialUserEntity user;
    required PublicKeyCredentialEntity rp;
    required COSEAlgorithmIdentifier alg;
    [SameObject] readonly attribute ArrayBuffer signature;
    [SameObject] readonly attribute ArrayBuffer authenticatorData;
}

dictionary PublicKeyCredentialRequestOptions {
    required BufferSource challenge;
    required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
    unsigned long timeout;
    sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
    AuthenticatorSelectionCriteria authenticatorSelection;
    AttestationConveyancePreference attestation = "none";
    AuthenticationExtensions extensions;
}

dictionary MakePublicKeyCredentialOptions {
    required DOMString rp;
    required DOMString name;
    required DOMString icon;
    AttestationConveyancePreference attestation = "none";
    AuthenticatorSelectionCriteria authenticatorSelection;
}

dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
    DOMString id;
    required DOMString name;
    AttestationConveyancePreference attestation = "none";
    AuthenticatorSelectionCriteria authenticatorSelection;
    unsigned long timeout;
    required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
    required BufferSource challenge;
    required PublicKeyCredentialUserEntity user;
    required PublicKeyCredentialEntity rp;
    required COSEAlgorithmIdentifier alg;
    [SameObject] readonly attribute ArrayBuffer signature;
    [SameObject] readonly attribute ArrayBuffer authenticatorData;
}

interface AuthenticatorAssertionResponse : AuthenticatorResponse {
    [SecureContext]
    [SameObject] readonly attribute ArrayBuffer authenticatorData;
    [SameObject] readonly attribute ArrayBuffer userHandle;
}

dictionary PublicKeyCredentialParameters {
    required PublicKeyCredentialUserEntity user;
    required PublicKeyCredentialUserEntity step;
    sequence<PublicKeyCredentialParameters> pubKeyCredParams;
    unsigned long timeout;
    sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
    AuthenticatorSelectionCriteria authenticatorSelection;
    AttestationConveyancePreference attestation = "none";
    AuthenticationExtensions extensions;
}

dictionary PublicKeyCredentialRequestOptions {
    required BufferSource challenge;
    required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
    unsigned long timeout;
    sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
    AuthenticatorSelectionCriteria authenticatorSelection;
    AttestationConveyancePreference attestation = "none";
    AuthenticationExtensions extensions;
}

dictionary MakePublicKeyCredentialOptions {
    required PublicKeyCredentialUserEntity user;
    required PublicKeyCredentialUserEntity step;
    sequence<PublicKeyCredentialParameters> pubKeyCredParams;
    unsigned long timeout;
    sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
    AuthenticatorSelectionCriteria authenticatorSelection;
    AttestationConveyancePreference attestation = "none";
    AuthenticationExtensions extensions;
}
typedef long COSEAlgorithmIdentifier;

typedef BufferSource AAGUID;

typedef long COSEAlgorithmIdentifier;

typedef sequence<AuthenticatorSelectionList>

typedef sequence<AAGUID> AuthenticatorSelectionList;
typedf BufferSource AAGUID;

#base64url-encoding

/* 4.1. PublicKeyCredential Interface */

4.1.3. Create a new credential - PublicKeyCredential's

[[Create](options) method (2)

4.1.3. Create a new credential - PublicKeyCredential's

* 4.1.3. Create a new credential - PublicKeyCredential's

"public-key"

"preferred",

"discouraged"

"usb",

"nfc",

"ble"

}];

/* 5.1. PublicKeyCredential Interface */

enum PublicKeyCredentialType {

"public-key",

"preferred",

"discouraged"

};

dictionary PublicKeyCredentialDescriptor {

required PublicKeyCredentialType type;

required BufferSource id;

sequence<AuthenticatorTransport> transports;

};

enum AuthenticatorTransport {

"usb",

"nfc",

"ble"

};

typedf long COSEAlgorithmIdentifier;

eenum UserVerificationRequirement {

"required",

"preferred",

"discouraged"

};

typedf sequence<AAGUID> AuthenticatorSelectionList;
typedf BufferSource AAGUID;

Issues Index

The definitions of "lifetime of" and "becomes available" are intended to represent how devices are hotplugged into (USB) or discovered by (NFC) browsers, and are under-specified. Resolving this with good definitions or some other means will be addressed by resolving Issue #613. RET

need to define "blinding". See also #462
https://github.com/w3c/webauthn/issues/694#RET

@ballanz wishes to add to the "direct" case: If the authenticator violates the privacy requirements of the attestation type it is using, the client SHOULD terminate this algorithm with a "AttestationNotPrivateError", RET

The definitions of "lifetime of" and "becomes available" are intended to represent how devices are hotplugged into (USB) or discovered by (NFC) browsers, and are under-specified. Resolving this with good definitions or some other means will be addressed by resolving Issue #613. RET

The foregoing step may be incorrect, in that we are attempting to create savedCredentialId here and use it later below, and we do not have a global in which to allocate a place for it. Perhaps this is good enough? addendum: @jcjones feels the above step is likely good enough. RET

have a global in which to allocate a place for it. Perhaps this is good enough? addendum: @jcjones feels the above step is likely good enough. RET

403. RET

The WHATWG HTML WG is discussing whether to provide a hook when a browsing context gains or loses focuses. If a hook is provided, the above paragraph will be updated to include the hook. See WHATWG HTML WG issue #2711 for more details. RET

581.

*base64url-encoding

/* 5.1. PublicKeyCredential Interface */

5.1.3. Create a new credential - PublicKeyCredential's

[[Create](origin, options, sameOriginWithAncestors) method (2)
4.1.4. Use an existing credential to make an assertion -

```javascript
PublicKeyCredential's [[DiscoverFromExternalSource]](options)
```

4.1.3. Create a new credential - PublicKeyCredential

```javascript
PublicKeyCredential's [[Create]](options)
```

4.1.2. Verifying an authentication assertion

```javascript
PublicKeyCredential's
```

#cborReferenced in:

* 4.1.3. Create a new credential - PublicKeyCredential's

### WebAuthn Authenticator model

```javascript
PublicKeyCredential's [[Create]](options)
```

4.2. Verifying an authentication assertion

```javascript
PublicKeyCredential's
```

* 5.1. Authenticator data (2)

### WebAuthn Extensions (2) (3)

* 5.4. Client extension processing (2)

* 5.5. Authenticator extension processing (2) (3) (4) (5)

#attestationReferenced in:

* 5.3. Attestation

### WebAuthn Authenticator model (2)

```javascript
5.3. Attestation (2) (3) (4)
```

#attestation-certificateReferenced in:

* 3. Terminology (2)

```javascript
* 7.3.1. TPM attestation statement certificate requirements
```

#attestation-key-pairReferenced in:

* 3. Terminology (2)

* 5.3. Attestation

### WebAuthn Authenticator model (2)

```javascript
* 5.3. Attestation
```

#attestation-private-keyReferenced in:

* 5. WebAuthn Authenticator model

* 5.3. Attestation

### WebAuthn Authenticator model (2)

```javascript
* 5.3. Attestation
```

#authenticationReferenced in:

* 5.3. Attestation

### WebAuthn Authenticator model (2)

```javascript
* 5.3. Attestation
```

#authentication-assertionReferenced in:

* 5.1. Introduction (2)

* 5.3. Authentication

### WebAuthn Authenticator model (2)

```javascript
* 5.3. Authentication
```

#authentication-assertion-Referenced in:

* 5.1. Introduction (2)

* 5.2. Verifying an authentication assertion

### WebAuthn Authenticator model (2)

```javascript
* 5.2. Verifying an authentication assertion
```

#attestationReferenced in:

* 5.3. Terminology (2)

```javascript
* 5.3. Attestation
```

#attestation-certificateReferenced in:

* 5.3. Terminology (2)

```javascript
* 7.3.1. TPM attestation statement certificate requirements
```

#attestation-key-pairReferenced in:

* 5.3. Terminology (2)

```javascript
* 5.3. Attestation
```

#attestation-private-keyReferenced in:

* 5. WebAuthn Authenticator model

* 5.3. Attestation

### WebAuthn Authenticator model (2)

```javascript
* 5.3. Attestation
```

#authenticationReferenced in:

* 5.1. Introduction (2)

* 5.3. Authentication

### WebAuthn Authenticator model (2)

```javascript
* 5.3. Authentication
```

#authentication-assertionReferenced in:

* 5.1. Introduction (2)

* 5.2. Verifying an authentication assertion (2) (3)

### WebAuthn Authenticator model (2)

```javascript
* 5.2. Verifying an authentication assertion (2) (3)
```

#authentication-assertionReferenced in:

* 5.1. Introduction (2)

* 5.2. Verifying an authentication assertion

### WebAuthn Authenticator model (2)

```javascript
* 5.2. Verifying an authentication assertion
```

#attestationReferenced in:

* 5.3. Terminology (2)

```javascript
* 5.3. Attestation
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```

#attestation-conveyance-preference-referenced in:

* 5.6. WebAuthn Authenticator model (2)

```javascript
* 5.6.3. Attestation Conveyance Preference enumeration (enum)
```
5.2.1. The authenticatorMakeCredential operation

AuthenticatorSelectionCriteria) (2)

[[Create]](options) method

3. Terminology (2)

3. Terminology (2)

3. Terminology (2)

3. Terminology (2)

4.1.5. Platform Authenticator Availability - PublicKeyCredential's

isPlatformAuthenticatorAvailable() method (2) (3) (4)

4.1.2. Information about Public Key Credential (interface

AuthenticatorAttestationResponse) (2)

4.2.2. Web Authentication Assertion (interface

AuthenticatorAssertionResponse)

4.4.4. Authenticator Attachment enumeration (enum

AuthenticatorAttachment)

4.5. Options for Assertion Generation (dictionary

PublicKeyCredentialRequestOptions)

5.1. Authenticator data

5.1.1. WebAuthn Authenticator model (2) (3) (4) (5) (6)

5.1.2. The authenticatorMakeCredential operation

5.2.2. The authenticatorGetAssertion operation (2) (3)

5.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)

5.3.2. Attestation Statement Formats

5.3.4. Generating an Attestation Object (2)

5.3.5.1. Privacy

5.3.5.2. Attestation Certificate and Attestation Certificate CA

Compromise

6.1. Registering a new credential

7.2. Packed Attestation Statement Format

7.4. Android Key Attestation Statement Format

7.5. Android SafetyNet Attestation Statement Format

7.5.5. Supported Extensions Extension data

8.6. User Verification Index Extension (uvi)

8.7. Location Extension loc (3) (4)

8.8. User Verification Method Extension (uvm)

11. Sample scenarios

#authorization-gestureReferenced in:

1.1.1. Registration

1.1.2. Authentication

1.1.3. Other use cases and configurations

3. Terminology (2) (3) (4) (5) (6)

#biometric-recognitionReferenced in:

3. Terminology (2)

3. Terminology (2)

3. Terminology (2)

6.1. Registering a new credential

6.2. Verifying an authentication assertion

6.2.2. The authenticatorGetAssertion operation (2) (3)

7.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)

8.2. Packed Attestation Statement Format

8.4. Android Key Attestation Statement Format

8.5. Android SafetyNet Attestation Statement Format

9.10.6. User Verification Index Extension (uvi)

9.10.7. Location Extension loc (3) (4)

9.10.8. User Verification Method Extension (uvm)

12. Sample scenarios

#authorization-gestureReferenced in:

1.1.1. Registration

1.1.2. Authentication

1.1.3. Other use cases and configurations

4. Terminology (2) (3) (4) (5) (6)

5.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Get][options]] method (2)

#biometric-recognitionReferenced in:

8. Terminology (2)

8. Terminology (2)

8. Terminology (2)

8. Terminology (2)

6.1. Registering a new credential

7.1. Registering a new credential

7.2. Packed Attestation Statement Format

8.4. Android Key Attestation Statement Format

8.5. Android SafetyNet Attestation Statement Format

9.10.6. User Verification Index Extension (uvi)

9.10.7. Location Extension loc (3) (4)

9.10.8. User Verification Method Extension (uvm)

12. Sample scenarios

5.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Get][options]] method (2)

#biometric-recognitionReferenced in:

8. Terminology (2)

8. Terminology (2)

8. Terminology (2)

8. Terminology (2)

6.1. Registering a new credential

7.1. Registering a new credential

7.2. Packed Attestation Statement Format

8.4. Android Key Attestation Statement Format

8.5. Android SafetyNet Attestation Statement Format

9.10.6. User Verification Index Extension (uvi)

9.10.7. Location Extension loc (3) (4)

9.10.8. User Verification Method Extension (uvm)

12. Sample scenarios

5.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Get][options]] method (2)

#biometric-recognitionReferenced in:

8. Terminology (2)

8. Terminology (2)

8. Terminology (2)

8. Terminology (2)

6.1. Registering a new credential

7.1. Registering a new credential

7.2. Packed Attestation Statement Format

8.4. Android Key Attestation Statement Format

8.5. Android SafetyNet Attestation Statement Format

9.10.6. User Verification Index Extension (uvi)

9.10.7. Location Extension loc (3) (4)

9.10.8. User Verification Method Extension (uvm)

12. Sample scenarios

5.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Get][options]] method (2)

#biometric-recognitionReferenced in:

8. Terminology (2)

8. Terminology (2)

8. Terminology (2)

8. Terminology (2)
5.3.5.1. Privacy

5.3. Attestation (2) (3) (4) (5) (6)

5.2.2. The authenticatorGetAssertion operation (2) (3)

5.2.1. The authenticatorMakeCredential operation (2) (3) (4) (5) (6)

5.1. Authenticator data (2)

5. WebAuthn Authenticator model (2)

4.7.4. Authenticator Transport enumeration (enum AuthenticatorAttachment) (2) (3) (4)

4.4.4. Authenticator Attachment enumeration (enum PublicKeyCredentialEntity) (2) (3)


4.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[DiscoverFromExternalSource]](options) method (2)

4.1.3. Platform Authenticator Availability - PublicKeyCredential's [[DiscoverFromExternalSource]](options) method (2) (3)

4.1.2. Authenticator Responses (interface AuthenticatorResponse)

4.1.1. Information about PublicKey Credential (interface AuthenticatorResponse)

4.1. Information about PublicKey Credentials (dictionary PublicKeyCredential)


4.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Get]](options) method (2)

4.1.3. Create a new credential - PublicKeyCredential's [[CreateOrInitiate]](origin, options, sameOriginWithAncestors) method (2) (3)

4.1.2. Authenticator Responses (interface AuthenticatorResponse)

4.1.1. Information about PublicKey Credential (interface AuthenticatorResponse)

4.1. Availability of User-Verifying Platform Authenticator - PublicKeyCredential's

4.0.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.3. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRequestOptions)

4.0.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

4.0.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.3. Authenticator Attachment enumeration (enum PublicKeyCredentialEntity)

4.0.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRequestOptions)

4.0.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

4.0.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.3. Authenticator Attachment enumeration (enum PublicKeyCredentialEntity)

4.0.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRequestOptions)

4.0.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

4.0.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.3. Authenticator Attachment enumeration (enum PublicKeyCredentialEntity)

4.0.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRequestOptions)

4.0.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

4.0.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.3. Authenticator Attachment enumeration (enum PublicKeyCredentialEntity)

4.0.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRequestOptions)

4.0.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

4.0.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.3. Authenticator Attachment enumeration (enum PublicKeyCredentialEntity)

4.0.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRequestOptions)

4.0.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

4.0.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.3. Authenticator Attachment enumeration (enum PublicKeyCredentialEntity)

4.0.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRequestOptions)

4.0.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

4.0.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.3. Authenticator Attachment enumeration (enum PublicKeyCredentialEntity)

4.0.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRequestOptions)

4.0.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

4.0.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.3. Authenticator Attachment enumeration (enum PublicKeyCredentialEntity)

4.0.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRequestOptions)

4.0.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

4.0.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.3. Authenticator Attachment enumeration (enum PublicKeyCredentialEntity)

4.0.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRequestOptions)

4.0.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

4.0.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

4.0.3. Authenticator Attachment enumeration (enum PublicKeyCredentialEntity)
Compromise (2) (3) (4) (5) (6)

* 5. Relying Party Operations (2) (3) (4)
* 6.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)
* 6.1.1. Registration (2) (3) (4) (5) (6) (7) (8) (9)
* 6.1.2. Registration (2) (3) (4) (5) (6) (7) (8) (9)
* 6.1.3. Registration (2) (3) (4) (5) (6) (7) (8) (9)
* 6.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Create](origin, options, sameOriginWithAncestors)] method (2)
* 6.1.4.1. Use an existing credential to make an assertion - PublicKeyCredential's [[Create](origin, options, sameOriginWithAncestors)] method (2)

[2] 4.4. Options for Credential Creation (dictionary)

MakePublicKeyCredentialOptions


PublicKeyCredentialRequestOptions


#relying-party-identifierReferencedIn:

* 4. Web Authentication API
* 4.4. Options for Credential Creation (dictionary)
[4] MakePublicKeyCredentialOptions
[4] PublicKeyCredentialRequestOptions

#ro-idReferencedIn:

* 3. Terminology (2) (3) (4) (5) (6)
* 6. Web Authentication API (2) (3) (4) (5)
* 6.3.1. Create a new credential - PublicKeyCredential's [[Create](options)] method (2)
* 6.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Create](DiscoverFromExternalSource)](options) method (2)

[4] 4.4.1. Public Key Entity Description (dictionary)

PublicKeyCredentialPublicKeyEntity


[5] 5.2.1. The authenticatorMakeCredential operation (2) (3)
[5] 5.2.2. The authenticatorGetAssertion operation (2) (3)

[6] 6.2. Verifying an authentication assertion (2)
[7] 7.6. FIDO U2F Attestation Statement Format (2) (3)

[System-polyester,- credential]:

* 4.3.1. Privacy
* 4.3.2. Attestation Certificate and Attestation Certificate CA
* 4.3.3. RegistrationSpecificallyWithPlatformAuthenticator

[12] 11.2. Registration (2) (3) (4) (5)
[12] 11.3. Authentication (2) (3) (4) (5)

[4] 12.2. Registration Specifically with User Verifying Platform Authenticator (2)

[4] 12.3. Authentication (2) (3) (4) (5)


[System-polyester,- credential]:

* 4.1. Terminology
* 4.2. Web Authentication API

[5] 5.4. Options for Credential Creation (dictionary)

MakePublicKeyCredentialOptions

[5] 5.5. Options for Assertion Generation (dictionary)

PublicKeyCredentialRequestOptions


[9] 9.2. Defining request parameters (2) (3) (4) (5)
[9] 9.3. Extending request parameters (2) (3) (4) (5)
[9] 9.5. Authenticator Selection Extension (authnSel) (2) (3) (4) (5)
[9] 9.7. Location Extension (loc) (2)
[9] 9.7. Location Extension (loc) (2)
[9] 9.7. Location Extension (loc) (2)

[System-polyester,- credential]:

* 10.1. FIDO Appid Extension (appid) (2)
* 10.2. Simple Transaction Authorization Extension (txAuthSimple) (2)
* 10.3. Supported Extensions Extension (exts) (2)
* 10.3. Supported Extensions Extension (exts) (2)
* 10.4. Authenticator Selection Extension (authnSel) (2) (3)
* 10.5. Supported Extensions Extension (exts) (2)
* 10.5. Supported Extensions Extension (exts) (2)

[System-polyester,- credential]:

* 11.1. Registration (2) (3) (4) (5) (6) (7) (8) (9)
* 11.2. WebAuthn Extension Identifier Registrations (2)
* 11.2. WebAuthn Extension Identifier Registrations (2)

[System-polyester,- credential]:

* 12.1. Registration (2) (3) (4) (5) (6) (7)
* 12.2. Registration Specifically with User Verifying Platform Authenticator (2)

[System-polyester,- credential]:

* 12.3. Authentication (2) (3) (4) (5)
* 12.4. Decommissioning (2)
* 13.1. Cryptographic Challenges

[System-polyester,- credential]:

* 14. Termi...
9.3. Generic Transaction Authorization Extension (txAuthGeneric)

9.2. Simple Transaction Authorization Extension (txAuthSimple)

[[Create](options) method

4.1.3. Create a new credential - PublicKeyCredential's

3. Terminology (2) (3) (4) (5) (6)

1. Introduction

5.2.2. The authenticatorGetAssertion operation (2) (3)

5.3. Attestation (2)

5.3.2. Attestation Statement Formats

5.3.3. Attestation Types

5.3.4. Generating an Attestation Object

5.3.5.2. Attestation Certificate and Attestation Certificate CA Compromise (2)

6.1. Registering a new credential

6.2.1. The authenticatorMakeCredential operation

6.2.2. The authenticatorGetAssertion operation

6.2.3. Use of an existing credential to make an assertion - PublicKeyCredential’s {[Get][options] method

5.4.3. User Account Parameters for Credential Generation

5.4.4. Authenticator Selection Criteria (dictionary

5.5. Options for Assertion Generation (dictionary

5.6. Web Authentication Assertion (interface

5.6.1. The authenticatorGetAssertion operation

5.6.2. The authenticatorGetAssertion operation (2) (3) (4) (5)

6.2.2. The authenticatorGetAssertion operation (2) (3) (4) (5)

11. Sample scenarios

#test-of-user-presenceReferenced in:

8. WebAuthn Extensions (2)

5.1.6. Availability of User-Verifying Platform Authenticator - isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)

5.1.4.1. PublicKeyCredential's {[Create](origin, options, sameOriginWithAncestors) method

5.1.4. Use an existing credential to make an assertion - PublicKeyCredential’s {[Get][options] method

5.1.3. Create a new credential - PublicKeyCredential’s

5.1.2. The authenticatorMakeCredential operation

5.1.1. The authenticatorMakeCredential operation (2) (3) (4) (5)

5.1.0. The authenticatorCreateCredential operation

5.1. The authenticatorCreateCredential operation (2) (3) (4) (5) (6)

5.1.0.1. PublicKeyCredential’s {[Get][options] method

5.0.4. Authenticator Selection Criteria (dictionary

5.0.3. The authenticatorGetAssertion operation (2) (3) (4) (5)

5.0.2. Web Authentication Assertion (interface

5.0.1. The authenticatorGetAssertion operation

5.0.0. The authenticatorGetAssertion operation (2) (3) (4) (5)

5.0.0.1. PublicKeyCredential's {[Get][options] method

4.8. Attestation Conveyance Preference enumeration (enum

4.7. Supporting Data Structures

4.6. Web Authentication API

4.5. Options for Assertion Generation (dictionary

4.4. Public Key Credential common interface

4.3. Create a new credential - PublicKeyCredential’s

4.2. Web Authentication Assertion (interface

4.1.3. Create a new credential - PublicKeyCredential’s

4.1.2. Web Authentication Assertion (interface

4.1.1. Public Key Credential common interface

4.1. Create a new credential - PublicKeyCredential's

4.0. Options for Assertion Generation (dictionary

3. Terminology (2) (3) (4) (5) (6)

3. Introduction

#user-consentReferenced in:

#user-handleReferenced in:

#test-of-user-presenceReferenced in:

#user-verificationReferenced in:

#test-of-user-presenceReferenced in:
4.1. Use an existing credential to make an assertion

4.1.4. Use an existing credential to make an assertion - [Create](options) method

4.1.3. Create a new credential - PublicKeyCredential's

4.1. PublicKeyCredential Interface

5.1. Authenticator data

5.1. Authenticator data (2) (3)

5.1.4.1. PublicKeyCredential's

5.1.4.1. PublicKeyCredential's

5.1.3. Create a new credential - PublicKeyCredential's

5.1.2. The authenticatorMakeCredential operation (2) (3)

5.1.5. Store an existing credential - PublicKeyCredential's

5.1.5. Store an existing credential - PublicKeyCredential's

5.1.4. Use an existing credential to make an assertion - PublicKeyCredential's

5.1.4. Use an existing credential to make an assertion - PublicKeyCredential's

5.1.3. Create a new credential - PublicKeyCredential's

5.1.2. The authenticatorGetAssertion operation

5.1.1. The PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)

5.1.1. The PublicKeyCredential Interface

4. Terminology

4. Terminology

3. Terminology (2)

3. Terminology

1. Introduction (2) (3)

1. Introduction (2) (3)

PublicCredentialDescriptor

UserVerificationRequirement

UserVerificationRequirement

#uvReferenced in:

#user-verified

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client

#webauthn-client

#webauthn-clientReferenced in:

#webauthn-client
4.1. PublicKeyCredential Interface

* 4.1. PublicKeyCredential Interface

#dom-publickeycredential-discoverfromexternalsource-slotReferenced in:

5.1. PublicKeyCredential Interface

* 5.1. PublicKeyCredential Interface

#dom-publickeycredential-clientextensionsresults-slotReferenced in:

5.1. PublicKeyCredential Interface

* 5.1. PublicKeyCredential Interface

#dom-publickeycredential-identifier-slotReferenced in:

6.1. Registering a new credential - PublicKeyCredential

* 6.1. Registering a new credential

#credentialcreationdata-attestationobjectresultReferenced in:

7.1. Registering a new credential

* 7.1. Registering a new credential

#credentialcreationdata-clientdatasonresultReferenced in:

* 7.1. Registering a new credential

#credentialcreationdata-attestationconvaneypreferenceoptionReferenced in:

* 7.1.3. Create a new credential - PublicKeyCredential

#credentialcreationdata-clientextensionsresultsReferenced in:

* 7.1.3. Create a new credential - PublicKeyCredential

#dom-publickeycredential-colectfromcredentialstore-slotReferenced in:

* 7.1.4. Use an existing credential to make an assertion - PublicKeyCredential

#dom-publickeycredential-discoverfromexternalsource-slotReferenced in:

* 8.4. Client extension processing

* 8.4. Client extension processing

 PublicKeyCredential's [[DiscoverFromExternalSource]](options) method

 PublicKeyCredential's [[DiscoverFromExternalSource]](options) method (2) (3)

 PublicKeyCredential's [[DiscoverFromExternalSource]](options) method (2) (3)

 PublicKeyCredential's [[DiscoverFromExternalSource]](options) method (2) (3)
4.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Get]](options) method
4.5. Abort operations with AbortSignal (2) (3) (4) (5)

6.2.2. The authenticatorGetAssertion operation

*dom-publickeycredential-discoverfromexternalsource-origin-options-sameoriginwithancestors-originReferenced in:
  * 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

#effective-user-verification-requirement-for-assertionReferenced in:
  * 6.2.2. The authenticatorGetAssertion operation

#assertioncreationdata-credentialresultReferenced in:
  * 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

#assertioncreationdata-clientdatajsonReferenced in:
  * 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

#assertioncreationdata-clientdatasonsonresultReferenced in:
  * 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

#assertioncreationdata-assertiondataresultReferenced in:
  * 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

#assertioncreationdata-signatureresultReferenced in:
  * 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

#assertioncreationdata-userhandleresultReferenced in:
  * 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

#assertioncreationdata-clientextensionresultReferenced in:
  * 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

#authenticatorresponseReferenced in:
  * 5.1. PublicKeyCredential Interface (2)

* 5.1.3. Create a new credential - PublicKeyCredential's [[Get]](options) method

* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

#assertorresultReferenced in:
  * 5.2.2. The authenticatorGetAssertion operation

* 6.2.2. The authenticatorGetAssertion operation

* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
[4.1.1. Information about Public Key Credential (interface
AuthenticatorAttestationResponse) (2)
7. Relying Party Operations
7.1. Registering a new credential (2) (3)

#dictdef-makepublickeycredentialoptionsReferenced in:
4.1.3. Create a new credential - PublicKeyCredential's
    [[Create](options) method)
4.2.1. Information about Public Key Credential (interface
    AuthenticatorAttestationResponse)
4.2.1. Information about Public Key Credential (interface
    AuthenticatorAttestationResponse)
4.3. Parameters for Credential Generation (dictionary
    PublicKeyCredentialParameters)
4.3. Parameters for Credential Generation (dictionary
    PublicKeyCredentialParameters)
4.3. Parameters for Credential Generation (dictionary
    PublicKeyCredentialParameters)
4.3. Parameters for Credential Generation (dictionary
    PublicKeyCredentialParameters)
4.4. Options for Credential Creation (dictionary
    PublicKeyCredentialOptions)
4.4. Options for Credential Creation (dictionary
    PublicKeyCredentialOptions)
4.4. Options for Credential Creation (dictionary
    PublicKeyCredentialOptions)
4.4. Options for Credential Creation (dictionary
    PublicKeyCredentialOptions)

#dictdef-makepublickeycredentialparametersReferenced in:
4.2.1. Information about Public Key Credential (interface
    AuthenticatorAttestationResponse) (2)
4.3. Parameters for Credential Generation (dictionary
    PublicKeyCredentialParameters)
4.4. Options for Credential Creation (dictionary
    PublicKeyCredentialOptions)
4.4. Options for Credential Creation (dictionary
    PublicKeyCredentialOptions)

#dictdef-makepublickeycredentialparameters-typeReferenced in:
4.1.3. Create a new credential - PublicKeyCredential's
    [[Create](options) method)
4.2.1. Information about Public Key Credential (interface
    AuthenticatorAttestationResponse)
4.3. Parameters for Credential Generation (dictionary
    PublicKeyCredentialParameters)
4.3. Parameters for Credential Generation (dictionary
    PublicKeyCredentialParameters)
4.3. Parameters for Credential Generation (dictionary
    PublicKeyCredentialParameters)
4.3. Parameters for Credential Generation (dictionary
    PublicKeyCredentialParameters)
4.3. Parameters for Credential Generation (dictionary
    PublicKeyCredentialParameters)
4.4. Options for Credential Creation (dictionary
    PublicKeyCredentialOptions)
4.4. Options for Credential Creation (dictionary
    PublicKeyCredentialOptions)
4.4. Options for Credential Creation (dictionary
    PublicKeyCredentialOptions)
4.4. Options for Credential Creation (dictionary
    PublicKeyCredentialOptions)
4.4.2. User Account Parameters for Credential Generation

MakePublicKeyCredentialOptions)

PublicKeyCredentialRpEntity)

PublicKeyCredentialEntity) (2)

* 4.4.2. User Account Parameters for Credential Generation

MakePublicKeyCredentialOptions)

PublicKeyCredentialEntity) (2)

* 4.4.2. User Account Parameters for Credential Generation

MakePublicKeyCredentialOptions)

PublicKeyCredentialRpEntity)

PublicKeyCredentialEntity) (2) (3) (6)

* 5.1.3. Create a new credential - PublicKeyCredential's

Create [[origin, options, sameOriginWithAncestors] method (2) (3) (5) (6)) (6)}

* 5.4. Options for Credential Creation (dictionary

MakePublicKeyCredentialOptions)

PublicKeyCredentialEntity) (2) (3) (6)

* 5.4.2. RP Parameters for Credential Generation (dictionary

PublickeyCredentialEntity) (2)
#dictdef-publickeycredentialrpentityReferenced in:
* 5.4.1. Public Key Entity Creation (dictionary PublicKeyCredentialEntity)
* 5.4.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialEntity)
* 5.4.3. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialEntity)

#dictdef-publickeycredentialuserentityidReferenced in:
* 4.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

#dictdef-publickeycredentialuserentity-displaynameReferenced in:
* 4.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

#dictdef-authenticatorselectioncriteriaReferenced in:
* 5.4.3. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity)

#dom-publickeycredentialrpentity-idReferenced in:
* 5.4.4. Authenticator Selection Criteria (dictionary PublicKeyCredentialEntity)

#dom-publickeycredentialrpentity-displaynameReferenced in:
* 5.4.4. Authenticator Selection Criteria (dictionary PublicKeyCredentialEntity)
* 5.4.5. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity)

#dom-publickeycredentialrpentity-idReferenced in:
* 5.4.4. Authenticator Selection Criteria (dictionary PublicKeyCredentialEntity)

#dom-publickeycredentialrpentity-displaynameReferenced in:
* 5.4.4. Authenticator Selection Criteria (dictionary PublicKeyCredentialEntity)

#dom-publickeycredentialrpentity-displaynameReferenced in:
* 5.4.4. Authenticator Selection Criteria (dictionary PublicKeyCredentialEntity)

#dom-publickeycredentialrpentity-displaynameReferenced in:
* 5.4.4. Authenticator Selection Criteria (dictionary PublicKeyCredentialEntity)

#dom-authenticatortattachmentReferenced in:
* 5.4.4. Authenticator Selection Criteria (dictionary PublicKeyCredentialEntity)
4.4.4. Authenticator Attachment enumeration (enum)

#cross-platform-attachedReferenced in:

11.1. Registration

11.2. Registration Specifically with Platform Authenticator (2)

AuthenticatorAttachment

platform-authenticatorsReferenced in:

4.4.4. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

#cross-platform-attachedReferenced in:

4.4.4. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

#attestation-conveyanceReferenced in:

4.1.3. Create a new credential - PublicKeyCredential's

Create (options)

AuthenticatorSelectionCriteria

4.4.4. Authenticator Selection Criteria (dictionary)

AuthenticatorSelectionCriteria

#dom-authenticatorselectioncriteria-rkReferenced in:

4.4.4. Authenticator Selection Criteria (dictionary)

AuthenticatorSelectionCriteria

#dom-authenticatorselectioncriteria-uvReferenced in:

12.1. Registration

AuthenticatorAttachment (2)

#PLATFORM-AuthenticatorAvailability - PublicKeyCredential's

isPlatformAuthenticatorAvailable (method) (2) (3) (4) (5)

4.4.4. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

#platform-authenticatorsReferenced in:

5.4.5. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

#cross-platform-attachedReferenced in:

5.4.5. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

#attestation-conveyanceReferenced in:

4.1.3. Create a new credential - PublicKeyCredential's

Create (origin, options, sameOriginWithAncestors)

AuthenticatorSelectionCriteria

4.4.4. Authenticator Selection Criteria (dictionary)

AuthenticatorSelectionCriteria

#dom-authenticatorelectioncriteria-requireresidentkeyReferenced in:

5.4.3. Create a new credential - PublicKeyCredential's

Create (origin, options, sameOriginWithAncestors) (2)

5.4.4. Authenticator Selection Criteria (dictionary)

AuthenticatorSelectionCriteria

#6.2.1. The authenticatorMakeCredential operation

5.1.3. Create a new credential - PublicKeyCredential's

Create (origin, options, sameOriginWithAncestors) (2)

5.4.4. Authenticator Selection Criteria (dictionary)

AuthenticatorSelectionCriteria

#platform-authenticatorsReferenced in:

6.2.1. The authenticatorMakeCredential operation

5.4.4. Authenticator Selection Criteria (dictionary)

AuthenticatorSelectionCriteria (2)

5.4.5. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

#platform-authenticatorsReferenced in:

5.4.6. Availability of User-Verifying Platform Authenticator - PublicKeyCredential's

isUserVerifyingPlatformAuthenticatorAvailable (method) (2) (3) (4)

5.4.5. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

#roaming-authenticatorsReferenced in:

1.1.3. Other use cases and configurations

5.4.4. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

12.1. Registration

12.2. Registration Specifically with User Verifying Platform

Authenticator (2)

#platform-authenticatorsReferenced in:

5.4.7. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

#cross-platform-attachedReferenced in:

5.4.5. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

#attestation-conveyanceReferenced in:

5.4.6. Attestation Conveyance Preference enumeration (enum)

AttestationConveyancePreference

#enumdef-attestationconveyancepreferenceReferenced in:

5.4.6. Attestation Conveyance Preference enumeration (enum)

AttestationConveyancePreference

#enumdef-authenticatorattachmentReferenced in:

5.4.5. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

#platform-authenticatorsReferenced in:

5.4.6. Attestation Conveyance Preference enumeration (enum)

AttestationConveyancePreference

#enumdef-authenticatorattachmentReferenced in:

5.4.4. Authenticator Selection Criteria (dictionary)

AuthenticatorSelectionCriteria

5.4.5. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

5.4.6. Attestation Conveyance Preference enumeration (enum)

AttestationConveyancePreference

#enumdef-attestationconveyancepreferenceReferenced in:

5.4.5. Authenticator Attachment enumeration (enum)

AuthenticatorAttachment (2)

5.4.6. Attestation Conveyance Preference enumeration (enum)

AttestationConveyancePreference

5.4.6. Attestation Conveyance Preference enumeration (enum)

AttestationConveyancePreference

#enumdef-attestationconveyancepreference-indirectReferenced in:
4.5. Options for Assertion Generation (dictionary method)

5.1.3. Create a new credential - PublicKeyCredential's

5.1.4.1. PublicKeyCredential's

5.4.6. Attestation Conveyance Preference enumeration (enum)

5.5. Options for Assertion Generation (dictionary)

6.2. Verifying an authentication assertion

6.4.5. Options for Assertion Generation (dictionary)

7.2. Verifying an authentication assertion

8.1.4.1. PublicKeyCredential's

9.1. FIDO AppId Extension (appid)

10.1. FIDO AppId Extension (appid)

11. FIDO Authentification Extensions (appid)

12. FIDO Credential Request Options

13. Cryptographic Challenges

14. FIDO Attestation Conveyance Preference

15. FIDO Attestation Conveyance Preference

16. FIDO Attestation Conveyance Preference

17. FIDO Attestation Conveyance Preference

18. FIDO Attestation Conveyance Preference

19. FIDO Attestation Conveyance Preference

20. FIDO Attestation Conveyance Preference

21. FIDO Attestation Conveyance Preference

22. FIDO Attestation Conveyance Preference

23. FIDO Attestation Conveyance Preference

24. FIDO Attestation Conveyance Preference

25. FIDO Attestation Conveyance Preference

26. FIDO Attestation Conveyance Preference

27. FIDO Attestation Conveyance Preference

28. FIDO Attestation Conveyance Preference

29. FIDO Attestation Conveyance Preference

30. FIDO Attestation Conveyance Preference

31. FIDO Attestation Conveyance Preference

32. FIDO Attestation Conveyance Preference

33. FIDO Attestation Conveyance Preference

34. FIDO Attestation Conveyance Preference

35. FIDO Attestation Conveyance Preference

36. FIDO Attestation Conveyance Preference

37. FIDO Attestation Conveyance Preference

38. FIDO Attestation Conveyance Preference

39. FIDO Attestation Conveyance Preference

40. FIDO Attestation Conveyance Preference

41. FIDO Attestation Conveyance Preference

42. FIDO Attestation Conveyance Preference

43. FIDO Attestation Conveyance Preference

44. FIDO Attestation Conveyance Preference

45. FIDO Attestation Conveyance Preference

46. FIDO Attestation Conveyance Preference

47. FIDO Attestation Conveyance Preference

48. FIDO Attestation Conveyance Preference

49. FIDO Attestation Conveyance Preference

50. FIDO Attestation Conveyance Preference

51. FIDO Attestation Conveyance Preference

52. FIDO Attestation Conveyance Preference

53. FIDO Attestation Conveyance Preference

54. FIDO Attestation Conveyance Preference

55. FIDO Attestation Conveyance Preference

56. FIDO Attestation Conveyance Preference

57. FIDO Attestation Conveyance Preference

58. FIDO Attestation Conveyance Preference

59. FIDO Attestation Conveyance Preference

60. FIDO Attestation Conveyance Preference

61. FIDO Attestation Conveyance Preference

62. FIDO Attestation Conveyance Preference

63. FIDO Attestation Conveyance Preference

64. FIDO Attestation Conveyance Preference

65. FIDO Attestation Conveyance Preference

66. FIDO Attestation Conveyance Preference

67. FIDO Attestation Conveyance Preference

68. FIDO Attestation Conveyance Preference

69. FIDO Attestation Conveyance Preference

70. FIDO Attestation Conveyance Preference

71. FIDO Attestation Conveyance Preference

72. FIDO Attestation Conveyance Preference

73. FIDO Attestation Conveyance Preference

74. FIDO Attestation Conveyance Preference

75. FIDO Attestation Conveyance Preference

76. FIDO Attestation Conveyance Preference

77. FIDO Attestation Conveyance Preference

78. FIDO Attestation Conveyance Preference

79. FIDO Attestation Conveyance Preference

80. FIDO Attestation Conveyance Preference

81. FIDO Attestation Conveyance Preference

82. FIDO Attestation Conveyance Preference

83. FIDO Attestation Conveyance Preference

84. FIDO Attestation Conveyance Preference

85. FIDO Attestation Conveyance Preference

86. FIDO Attestation Conveyance Preference

87. FIDO Attestation Conveyance Preference

88. FIDO Attestation Conveyance Preference

89. FIDO Attestation Conveyance Preference

90. FIDO Attestation Conveyance Preference

91. FIDO Attestation Conveyance Preference

92. FIDO Attestation Conveyance Preference

93. FIDO Attestation Conveyance Preference

94. FIDO Attestation Conveyance Preference

95. FIDO Attestation Conveyance Preference

96. FIDO Attestation Conveyance Preference

97. FIDO Attestation Conveyance Preference

98. FIDO Attestation Conveyance Preference

99. FIDO Attestation Conveyance Preference
#dctdef-collectedclientdataReferenced in:
* 4.1.3. Create a new credential - PublicKeyCredential's
  [[Create](options) method
* 4.1.4. Use an existing credential to make an assertion -
  PublicKeyCredential's [[DiscoverFromExternalSource](options)] method
* 6.1. Client data used in WebAuthn signatures (dictionary
  CollectedClientData) (2)

#client-dataReferenced in:
* 4.2. Authenticator Responses (interface AuthenticatorResponse)
* 5. WebAuthn Authenticator model (2) (3) (4)
* 6.2. Verifying an authentication assertion
* 8. WebAuthn Extensions
* 8.4. Client extension processing
* 8.6. Example Extension

#dom-collectedclientdataReferenced in:
* 5.1.3. Create a new credential - PublicKeyCredential's
  [Create](options) method
* 5.1.4.1. PublicKeyCredential's
  [[DiscoverFromExternalSource](origin, options,
  sameOriginWithAncestors) method
* 5.1.4.2. PublicKeyCredential's
  [[DiscoverFromExternalSource](origin, options,
  sameOriginWithAncestors) method
* 5.8.1. Client data used in WebAuthn signatures (dictionary
  CollectedClientData)
* 7.1. Registering a new credential
* 7.2. Verifying an authentication assertion

#dom-collectedclientdata-originReferenced in:
* 5.1.3. Create a new credential - PublicKeyCredential's
  [Create](options) method
* 5.1.4.1. PublicKeyCredential's
  [[DiscoverFromExternalSource](origin, options,
  sameOriginWithAncestors) method
* 5.8.1. Client data used in WebAuthn signatures (dictionary
  CollectedClientData)
* 7.1. Registering a new credential
* 7.2. Verifying an authentication assertion

#dom-collectedclientdata-hashalgorithmReferenced in:
* 5.1.3. Create a new credential - PublicKeyCredential's
  [Create](options) method
* 5.1.4.1. PublicKeyCredential's
  [[DiscoverFromExternalSource](options)] method
* 5.2. Verifying an authentication assertion

#dom-collectedclientdata-tokenbindingidReferenced in:
* 5.1.3. Create a new credential - PublicKeyCredential's
  [Create](options) method
#enumdef-publickeycredentialtype

Referenced in:

* 5.1.3. Create a new credential - PublicKeyCredential's [[Create](origin, options, sameOriginWithAncestors) method (2)

* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource](origin, options, sameOriginWithAncestors) method

* 6.1. Registering a new credential

* 6.2. Verifying an authentication assertion

* 6.2.4. Client extension processing

* 6.2.7. Client data used in WebAuthn signatures (dictionary CollectedClientData)

* 6.3.4. Generating an Attestation Object

* 7.1. Registering a new credential

* 7.2. Verifying an authentication assertion

#dom-collectedclientdata-clientextensionsReferenced in:

* 8.6. FIDO U2F Attestation Statement Format

* 8.4. Android Key Attestation Statement Format

* 8.2. Packed Attestation Statement Format

* 6.3.2. Attestation Statement Formats (2)

* 6.2.1. The authenticatorMakeCredential operation

* 5.8.2.1. Information about Public Key Credential (interface AuthenticatorAssertionResponse) (2)

* 5.8.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)

* 5.8.1. Client data used in WebAuthn signatures (dictionary CollectedClientData)

* 7.1. Registering a new credential

* 7.2. Verifying an authentication assertion

* 9.4. Client extension processing

#dom-collectedclientdata-authenticatorextensionsReferenced in:

* 8.6. FIDO U2F Attestation Statement Format

* 8.4. Android Key Attestation Statement Format

* 8.2. Packed Attestation Statement Format

* 6.3.2. Attestation Statement Formats (2)

* 6.2.1. The authenticatorMakeCredential operation

* 5.8.2.1. Information about Public Key Credential (interface AuthenticatorAssertionResponse) (2)

* 5.8.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)

* 5.8.1. Client data used in WebAuthn signatures (dictionary CollectedClientData)

* 7.1. Registering a new credential

* 7.2. Verifying an authentication assertion

#collectedclientdata-json-serialized-client-dataReferenced in:

* 6.2. Verifying an authentication assertion

* 6.2.4. Client extension processing

* 6.2.7. Client data used in WebAuthn signatures (dictionary CollectedClientData)

* 6.3.4. Generating an Attestation Object

* 7.1. Registering a new credential

* 7.2. Verifying an authentication assertion

* 7.2.4. Client extension processing

* 7.2.7. Client data used in WebAuthn signatures (dictionary CollectedClientData)

* 7.3. TPM Attestation Statement Format (2)

* 6.1. Registering a new credential

* 6.2. Verifying an authentication assertion

* 8.4. Client extension processing

#collectedclientdata-authenticatorextensionsReferenced in:

* 8.2. Packed Attestation Statement Format

* 7.3. TPM Attestation Statement Format (2)

* 6.1. Registering a new credential

* 6.2. Verifying an authentication assertion

* 8.4. Client extension processing

* 8.6. FIDO U2F Attestation Statement Format

* 8.4. Android Key Attestation Statement Format

* 8.2. Packed Attestation Statement Format

* 6.3.2. Attestation Statement Formats (2)

* 6.2.1. The authenticatorMakeCredential operation

* 5.8.2.1. Information about Public Key Credential (interface AuthenticatorAssertionResponse) (2)

* 5.8.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)

* 5.8.1. Client data used in WebAuthn signatures (dictionary CollectedClientData)

* 7.1. Registering a new credential

* 7.2. Verifying an authentication assertion

* 9.4. Client extension processing
4.1.3. Create a new credential - PublicKeyCredential's

4.3. Parameters for Credential Generation (dictionary

5.1.3. Create a new credential - PublicKeyCredential's

5.3. Parameters for Credential Generation (dictionary

5.8.2. Credential Type enumeration (enum PublicKeyCredentialType)

5.8.3. Credential Descriptor (dictionary

6.2.1. The authenticatorMakeCredential operation (2) (3)

6.2.2. The authenticatorGetAssertion operation

6.2.4. Credential Type enumeration (enum PublicKeyCredentialType)
5.3.1. Attestation data

- 6. WebAuthn Authenticator model (2) (3) (4) (5) (6) (7) (8) (9)

#attestation-signature

- 7.1. Registering a new credential

#dom-userverificationrequirement

- 7.6. FIDO U2F Attestation Statement Format

### 8.6. FIDO U2F Attestation Statement Format

- 8.3. TPM Attestation Statement Format

#assertion-signature

- 5.2.2. The authenticatorGetAssertion operation (2) (3) (4) (5) (6)

#assertion-data

- 4.2.1. Information about PublicKeyCredential (interface

#AuthenticatorAttestationResponse

- 2. Web Authentication Assertion (interface

#AuthenticatorAssertionResponse

- 5. WebAuthn Authenticator model (2)

#attestation-signature

- 3. Terminology

- 5. WebAuthn Authenticator model (2) (3)

- 5.2.2. The authenticatorGetAssertion operation (2) (3) (4) (5) (6)

- 4.2.1. Information about PublicKeyCredential (interface

- 5.2.1. The authenticatorMakeCredential operation

- 5.2.2. The authenticatorGetAssertion operation (2) (3) (4)

- 5.3.1. Attestation data
5.1. Authenticator data

5.2. Attestation Statement Formats (2)

5.3.4. Generating an Attestation Object (2) (3)

5.3.3. Attestation Certificate Hierarchy

5.1.1. Registering a new credential (2) (3)

7.5. Android SafetyNet Attestation Statement Format

8.5. Authenticator extension processing (2)

8.6. Example Extension (2)

9.6. User Verification Index Extension (uvi)

9.8. User Verification Method Extension (uvm)

9.7. Location Extension (loc)

4.1.4. Use an existing credential to make an assertion - PublicKeyCredential's

5.8.6. User Verification Requirement enumeration (enum)

3. Terminology (2) (3)

5.1.4.1. PublicKeyCredential's

4.1.3. Create a new credential - PublicKeyCredential's

3. Terminology (2) (3)

5.8.6. User Verification Requirement enumeration (enum)

6.2.1. The authenticatorMakeCredential operation

6.2.2. The authenticatorGetAssertion operation

7.1. Registering a new credential (2)

8.3. TPM Attestation Statement Format

8.4. Android Key Attestation Statement Format

8.6. FIDO U2F Attestation Statement Format

9.2. Defining extensions

9.3. Terminology (2) (3)

9.4. [Create](options) method (2)

9.5. WebAuthn Authenticator model

9.6. The authenticatorCancel operation (2)

9.7. Defining extensions

9.8. User Verification Method Extension (uvm)

9.9. Example Extension (2)

10.5. User Verification Index Extension (uvi)

10.7. Location Extension (loc)

10.8. User Verification Method Extension (uvm)

10.8. Use an existing credential to make an assertion - PublicKeyCredential's

3. Terminology (2) (3)

5.1.3. Create a new credential - PublicKeyCredential's

4.1.1. Signature Counter Considerations (2) (3)

5.2.1. The authenticatorMakeCredential operation

6.2.2. The authenticatorGetAssertion operation

7.2. Verifying an authentication assertion (2) (3)

6.1. Authenticator data

6.2.1. The authenticatorMakeCredential operation

6.2.2. The authenticatorGetAssertion operation

6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6) (7) (8) (9) (10)

6.2.1. The authenticatorMakeCredential operation (2) (3) (4)

6.2.2. The authenticatorGetAssertion operation (2)

7.2. Verifying an authentication assertion (2) (3) (4) (5) (6)

4. Terminology (2) (3)

5.1.3. Create a new credential - PublicKeyCredential's

5.2.1. The authenticatorMakeCredential operation (2)

6.2.2. The authenticatorGetAssertion operation (2)

6.2.3. The authenticatorCancel operation (2)

5.1. Terminology (2) (3)

4.1.4. Use an existing credential to make an assertion - PublicKeyCredential's

[Create](from:source:options) method (2) (3) (4)

5. WebAuthn Authenticator model

5.1. Authenticator data
#attestation-data
Referenced in:
5.3.2. Attestation Statement Formats (2) (3)
5.3. Attestation (2) (3) (4) (5) (6) (7)
5.2.1. The authenticatorMakeCredential operation (2)
5.1. Authenticator data (2) (3) (4) (5) (6) (7)

#attestation-statement
Referenced in:
5.3.4. Generating an Attestation Object (2) (3) (4)
5.3.1. Attestation data
5.3. Attestation (2) (3)
5.2.1. The authenticatorMakeCredential operation (2)
5.1.3. Create a new credential - PublicKeyCredential's

#attestation-object
Referenced in:
5.3.4. Generating an Attestation Object (2) (3) (4)
5.3.1. Attestation data
5.3. Attestation (2) (3)
5.2.1. The authenticatorMakeCredential operation (2)
5.1.3. Create a new credential - PublicKeyCredential's

#authenticatorcancel
Referenced in:
5.8.4. Authenticator Transport enumeration (enum AuthenticatorAttestationResponse) (2) (3) (4) (5) (6) (7)
5.3.3. Attestation Types
5.3.2. Attestation Statement Formats (2) (3) (4)
5.3.1. Attested credential data
5.3. Attestation (2) (3)
5.2.1. The authenticatorMakeCredential operation (2)
5.1.4.1. PublicKeyCredential's

#aaguid
Referenced in:
5.4. Options for Credential Creation (dictionary AuthenticatorAttestationResponse)
5.3.3. Attestation Types
5.3.2. Attestation Statement Formats (2) (3) (4)
5.3.1. Attested credential data
5.3. Attestation (2) (3)
5.2.1. The authenticatorMakeCredential operation (2)
5.1.4.1. PublicKeyCredential's

#signature-counter
Referenced in:
5.1.4.1. PublicKeyCredential's
5.1.3. Create a new credential - PublicKeyCredential's
5.1.1. Signature Counter Considerations (2) (3)
5.1. WebAuthn Extensions
5.0.2. Web Authentication API

#attestation-statement
Referenced in:
5.4.6. Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)
5.3.2. Attestation Statement Formats (2) (3) (4)
5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse) (2) (3)
5.1.4.1. PublicKeyCredential's

#attestation-statement-format
Referenced in:
5.5.3.4. Generating an Attestation Object (2)
5.5.3.2. Attestation Statement Formats (2) (3) (4)
5.5.3.1. Attested credential data
5.5.3. Attestation (2) (3)
5.5.2.1. The authenticatorMakeCredential operation (2)
5.5.1. Authenticator data (2) (3) (4) (5) (6)

#attestation-type
Referenced in:
5.5.3.2. Attestation Statement Formats (2) (3) (4)
5.5.2. Attestation Statement Formats (2) (3) (4)
5.5.1. Authenticator data (2) (3) (4) (5)
5.5. Attestation (2) (3)
5.4. Options for Credential Creation (dictionary AuthenticatorAttestationResponse)

#attested-credential-data
Referenced in:
5.6.3.2. Attestation Statement Formats (2) (3) (4)
5.6.3.1. Attested credential data
5.6.3. Attestation (2) (3)
5.6.2.1. The authenticatorMakeCredential operation (2)
5.6.1. Authenticator data (2) (3) (4) (5)
5.6. Attestation (2) (3)
5.5.3.1. Attested credential data

#auguid
Referenced in:
5.1.4.1. PublicKeyCredential's
5.1.3. Create a new credential - PublicKeyCredential's
5.1.2. Web Authentication API
5.1.1. Signature Counter Considerations (2) (3)
5.1. WebAuthn Extensions
5.0.2. Web Authentication API

#attestation-cancel
Referenced in:
5.5.6.4. Authenticator Transport enumeration (enum AuthenticatorAttestationResponse)
5.5.2.1. The authenticatorMakeCredential operation (2)
5.5.1. Authenticator data (2) (3) (4) (5)
5.5. Authenticator (2) (3)
5.4. Options for Credential Creation (dictionary AuthenticatorAttestationResponse)

#authenticatorcancel
Referenced in:
5.6.3.2. Attestation Statement Formats (2) (3) (4)
5.6.3.1. Attested credential data
5.6.3. Attestation (2) (3)
5.6.2.1. The authenticatorMakeCredential operation (2)
5.6.1. Authenticator data (2) (3) (4) (5)
5.6. Authenticator (2) (3)
5.5.3.1. Attested credential data

#attestation-cancel
Referenced in:
5.7.3.1. Attested credential data
5.7.3. Attestation (2) (3)
5.7.2.1. The authenticatorMakeCredential operation (2)
5.7.1. Authenticator data (2) (3) (4) (5)
5.7. Authenticator (2) (3)
5.6.3.1. Attested credential data

#attestation-cancel
Referenced in:
5.8.2.1. The authenticatorRegisterCredential operation (2)
5.8.2. The authenticatorGetAssertion operation (2)
5.8.1. WebAuthn Extensions
5.7.2. Web Authentication API
5.7.1. WebAuthn Extensions
5.0.2. Web Authentication API
#self-attestation

Referenced in:

* 5.3.2. Attestation Statement Formats

#signing-procedure

Referenced in:

* 6.1. Authenticator data

#credentialid

Referenced in:

* 5.1.3. Create a new credential - PublicKeyCredential's
  [[Create]](origin, options, sameOriginWithAncestors) method

* 6.1. Authenticator data

* 7.1. Registering a new credential

* 8.4. Android Key Attestation Statement Format

#signing-procedure

Referenced in:

* 6.3.2. Attestation Statement Formats

#authenticator-data

Referenced in:

* 7.2. Packed Attestation Statement Format

* 7.3. TPM Attestation Statement Format

* 7.4. Android Key Attestation Statement Format (2)

* 7.5. Android SafetyNet Attestation Statement Format

* 7.6. FIDO U2F Attestation Statement Format

#basic-attestation

Referenced in:

* 5.3.5.1. Privacy

#self-attestation

Referenced in:

* 6.1. Registering a new credential (2) (3)

* 7.2. Packed Attestation Statement Format (2)

* 7.6. FIDO U2F Attestation Statement Format

#basic-attestation

Referenced in:

* 6.3.5.1. Privacy

* 5.4.6. Attestation Conveyance Preference enumeration (enum
  AttestationConveyancePreference)

* 7.1. Registering a new credential

* 8.2. Packed Attestation Statement Format

* 8.3. TPM Attestation Statement Format

* 5.4.6. Attestation Conveyance Preference enumeration (enum
  AttestationConveyancePreference)

* 5.3.3. Attestation Types

* 5.3.3.1. Attestation ConveyancePreference

* 5.3.3.2. Attestation Certificate and Attestation Certificate CA

#self-attestation

Referenced in:

* 4. Terminology (2) (3) (4)

* 5.4.6. Attestation Conveyance Preference enumeration (enum
  AttestationConveyancePreference)

* 5.3. Attestation (2)

* 6.3.2. Attestation Statement Formats

* 6.3.3. Attestation Types

* 6.3.6.2. Attestation Certificate and Attestation Certificate CA

Compromise

Referenced in:

* 7.1. Registering a new credential (2) (3)

* 8.2. Packed Attestation Statement Format (2)

* 8.6. FIDO U2F Attestation Statement Format
#privacy-caReferenced in:

* 5.3.5.1. Privacy

#elliptic-curve-based-direct-anonymous-attestationReferenced in:

* 5.3.5.1. Privacy

#ecdaaReferenced in:

* 5.3.2. Attestation Statement Formats
* 5.3.3. Attestation Types
* 5.3.5.2. Attestation Certificate and Attestation Certificate CA
* Compromise
* 7.3. Registered a new credential
* 7.3.2. Packed Attestation Statement Format (2)
* 7.3.3. TPM Attestation Statement Format (2)

#attestation-statement-format-identifierReferenced in:

* 5.3.2. Attestation Statement Formats
* 5.3.4. Generating an Attestation Object

#Identifier-of-the-ecdaa-issuer-public-keyReferenced in:

* 6.1. Registering a new credential
* 7.2. Packed Attestation Statement Format
* 7.3. TPM Attestation Statement Format (2)

#ecdaa-issuer-public-keyReferenced in:

* 5.3.2. Attestation Statement Formats
* 5.3.5.1. Privacy
* 6.1. Registering a new credential
* 7.2. Packed Attestation Statement Format (2) (3)

#registration-extensionReferenced in:

* 5.1.3. Create a new credential - PublicKeyCredential's
[[Create]](options) method
* 5.6. WebAuthn Extensions (2) (3) (4) (5) (6)
* 5.8. Example Extension
* 5.9. Simple Transaction Authorization Extension (txAuthSimple)
* 5.10. Generic Transaction Authorization Extension (txAuthGeneric)
* 5.11. Authenticator Selection Extension (authnSel)
* 5.12. Supported Extensions Extension (exts)
* 5.13. User Verification Index Extension (uvi)
* 5.14. Location Extension (loc)
* 5.15. User Verification Method Extension (uvm)
* 5.16. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
* 5.17. Privacy

#authentication-extensionReferenced in:

* 4.1.4. Use an existing credential to make an assertion - PublicKeyCredential's
[[DiscoverFromExternalSource]](options) method
* 6.8. WebAuthn Extensions (2) (3) (4) (5) (6)
* 6.9. Example Extension
* 6.10. FIDO Agpd Extension (appid)
* 6.11. Simple Transaction Authorization Extension (txAuthSimple)
* 6.13. User Verification Index Extension (uvi)
* 6.14. Location Extension (loc)
* 6.15. User Verification Method Extension (uvm)
* 6.16. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
* 6.17. Privacy

#client-extensionReferenced in:

* 4.1.3. Create a new credential - PublicKeyCredential's
[[Create]](options) method

#privacy-caReferenced in:

* 5.1.3. Create a new credential - PublicKeyCredential's
[[Create]](origin, options, sameOriginWithAncestors) method
* 5.4.6. Attestation Conveyance Preference enumeration (enum)
* 5.3.5.1. Privacy

#elliptic-curve-based-direct-anonymous-attestationReferenced in:

* 6.3.5.1. Privacy

#ecdaaReferenced in:

* 5.3.2. Attestation Statement Formats
* 5.3.3. Attestation Types
* 5.3.5.2. Attestation Certificate and Attestation Certificate CA
* Compromise
* 7.1. Registering a new credential
* 7.2. Packed Attestation Statement Format (2)
* 7.3. TPM Attestation Statement Format (2) (3)

#attestation-statement-format-identifierReferenced in:

* 5.3.2. Attestation Statement Formats
* 5.3.4. Generating an Attestation Object

#Identifier-of-the-ecdaa-issuer-public-keyReferenced in:

* 6.1. Registering a new credential
* 7.2. Packed Attestation Statement Format
* 7.3. TPM Attestation Statement Format (2)

#ecdaa-issuer-public-keyReferenced in:

* 6.3.2. Attestation Statement Formats
* 6.3.5.1. Privacy
* 7.1. Registering a new credential
* 8.2. Packed Attestation Statement Format (2) (3)

#registration-extensionReferenced in:

* 5.1.3. Create a new credential - PublicKeyCredential's
[[Create]](origin, options, sameOriginWithAncestors) method
* 5.6. WebAuthn Extensions (2) (3) (4) (5) (6)
* 6.8. Example Extension
* 10.2. Simple Transaction Authorization Extension (txAuthSimple)
* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
* 10.4. Authenticator Selection Extension (authnSel)
* 10.5. Supported Extensions Extension (exts)
* 10.6. User Verification Index Extension (uvi)
* 10.7. Location Extension (loc)
* 10.8. User Verification Method Extension (uvm)
* 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
* 10.9. Privacy

#authentication-extensionReferenced in:

* 5.1.4.1. PublicKeyCredential's
[[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
* 9. WebAuthn Extensions (2) (3) (4) (5) (6)
* 9.6. Example Extension
* 10.1. FIDO Agpd Extension (appid)
* 10.2. Simple Transaction Authorization Extension (txAuthSimple)
* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
* 10.6. User Verification Index Extension (uvi)
* 10.7. Location Extension (loc)
* 10.8. User Verification Method Extension (uvm)
* 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
* 10.9. Privacy
* 4.1.3. Create a new credential - PublicKeyCredential’s
[Create](options) method
* 8. WebAuthn Extensions
* 8.2. Defining extensions
* 8.4. Client extension processing

#authenticator-extensionReferenced in:
* 4.1.3. Create a new credential - PublicKeyCredential’s
[Create](options) method
* 8. WebAuthn Extensions (2) (3)
* 8.2. Defining extensions (2)
* 8.3. Extending request parameters
* 8.5. Authenticator extension processing

#extension-identifierReferenced in:
* 4.1. PublicKeyCredential Interface
* 4.1.3. Create a new credential - PublicKeyCredential’s
[Create](options) method
* 4.1.4. Use an existing credential to make an assertion - PublicKeyCredential’s
[DiscoverFromExternalSource](options) method
* 5.1.3. Create a new credential - PublicKeyCredential’s
[DiscoverFromExternalSource](options) method
* 8. WebAuthn Extensions (2) (3)
* 8.2. Defining extensions
* 8.3. Extending request parameters
* 8.5. Authenticator extension processing

#client-extension-inputReferenced in:
* 7. WebAuthn Extensions (2) (3)
* 8. WebAuthn Extensions (2) (3) (4) (5) (6)
* 8.4. Client extension processing (2) (3) (4)
* 8.6. Example Extension

#authenticator-extension-inputReferenced in:
* 8. WebAuthn Extensions (2) (3) (4) (5)
* 8.2. Defining extensions
* 8.3. Extending request parameters (2) (3) (4) (5) (6)
* 8.4. Client extension processing (2) (3) (4)
* 8.5. Authenticator extension processing (2) (3)

#client-extension-processingReferenced in:
* 4.1. PublicKeyCredential Interface
* 4.1.3. Create a new credential - PublicKeyCredential’s
[Create](options) method (2)
* 4.1.4. Use an existing credential to make an assertion - PublicKeyCredential’s
[DiscoverFromExternalSource](options) method (2)
* 8. WebAuthn Extensions (2) (3) (4)
* 8.2. Defining extensions
* 8.4. Client extension processing

#client-extension-outputReferenced in:
* 4.1. PublicKeyCredential Interface
* 4.1.3. Create a new credential - PublicKeyCredential’s
[Create](options) method (2)
9.4. Authenticator Selection Extension (authnSel)

#typedefdef-aaguidReferenced in:
#typedefdef-authenticatorselectionlistReferenced in:

9.8. User Verification Method Extension (uvm)

* 9.8. User Verification Method Extension (uvm)
* 9.7. Location Extension (loc)

8. WebAuthn Extensions

* 8. WebAuthn Extensions

9. WebAuthn Extensions

* 9. WebAuthn Extensions

9.6. Example Extension

#authenticator-extension-processingReferenced in:

6.2.1. The authenticatorMakeCredential operation

* 6.2.1. The authenticatorMakeCredential operation

9.6. Example Extension

#authenticator-extension-outputReferenced in:

6.2.2. The authenticatorGetAssertion operation

* 6.2.2. The authenticatorGetAssertion operation

6.1. Authenticator data

* 6.1. Authenticator data

9.5. Authenticator extension processing

* 9.5. Authenticator extension processing

9.6. Example Extension

* 9.6. Example Extension

9.5. Supported Extensions Extension (exts)

* 9.5. Supported Extensions Extension (exts)

9.6. User Verification Index Extension (uvi)

* 9.6. User Verification Index Extension (uvi)

9.8. User Verification Method Extension (uvm)

* 9.8. User Verification Method Extension (uvm)

#typedefdef-authenticatorselectionlistReferenced in:

* 9.4. Authenticator Selection Extension (authnSel)

9.4. Client extension processing (2) (3)

* 9.4. Client extension processing (2) (3)

9.6. Example Extension

#authenticator-extension-processingReferenced in:

9.6. Example Extension

* 9.6. Example Extension

9.4. Client extension processing

* 9.4. Client extension processing

9.2. Defining extensions

* 9.2. Defining extensions

8.4. Client extension processing

* 8.4. Client extension processing

9.5. Supported Extensions Extension (exts)

* 9.5. Supported Extensions Extension (exts)

9.6. Example Extension

#authenticator-extension-outputReferenced in:

9.5. Authenticator extension processing

* 9.5. Authenticator extension processing

6.2.1. The authenticatorGetAssertion operation

* 6.2.1. The authenticatorGetAssertion operation

6.1. Authenticator data

* 6.1. Authenticator data

9.5. Authenticator extension processing

* 9.5. Authenticator extension processing

9.6. Example Extension

* 9.6. Example Extension

9.5. Supported Extensions Extension (exts)

* 9.5. Supported Extensions Extension (exts)

9.6. User Verification Index Extension (uvi)

* 9.6. User Verification Index Extension (uvi)

9.8. User Verification Method Extension (uvm)

* 9.8. User Verification Method Extension (uvm)

#typedefdef-authenticatorselectionlistReferenced in:

* 9.4. Authenticator Selection Extension (authnSel)

9.4. Client extension processing (2) (3)

* 9.4. Client extension processing (2) (3)

9.6. Example Extension

#authenticator-extension-processingReferenced in:

9.6. Example Extension

* 9.6. Example Extension

9.4. Client extension processing

* 9.4. Client extension processing

9.2. Defining extensions

* 9.2. Defining extensions