THE_URL:file://localhost/Users/jehodges/documents/work/standards/W3C/webauthn/index-master-3c5e383.html

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

W3C

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

Editor's Draft, 29 May 2018

This version:
https://w3c.github.io/webauthn/

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

Previous Versions:
https://www.w3.org/TR/2018/CR-webauthn-20180320/
https://www.w3.org/TR/2018/WD-webauthn-20180315/
https://www.w3.org/TR/2018/WD-webauthn-20180306/
https://www.w3.org/TR/2017/WD-webauthn-20171205/
https://www.w3.org/TR/2017/WD-webauthn-20170811/
https://www.w3.org/TR/2017/WD-webauthn-20170505/
https://www.w3.org/TR/2017/WD-webauthn-20170216/
https://www.w3.org/TR/2016/WD-webauthn-20161207/
https://www.w3.org/TR/2016/WD-webauthn-20160928/
https://www.w3.org/TR/2016/WD-webauthn-20160902/
https://www.w3.org/TR/2016/WD-webauthn-20160531/

Issue Tracking:
GitHub

Editors:
Dirk Balfanz (Google)
Alexei Czeskis (Google)
Jeff Hodges (PayPal)
J.C. Jones (Mozilla)
Michael B. Jones (Microsoft)
Akshay Kumar (Microsoft)
Angelo Liao (Microsoft)
Rolf Lindemann (Nok Nok Labs)
Emil Lundberg (Yubico)

Former Editors:
Vijay Bharadwaj (Microsoft)
Arnar Birgisson (Google)
Hubert Le Van Gong (PayPal)

Contributors:
Christiaan Brand (Google)
Adam Langley (Google)
Giridhar Mandyam (Qualcomm)
Mike West (Google)
Jeffrey Yasskin (Google)

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

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

THE_URL:file://localhost/Users/jehodges/documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.html

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

W3C

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

Editor's Draft, 29 May 2018

This version:
https://w3c.github.io/webauthn/

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

Previous Versions:
https://www.w3.org/TR/2018/CR-webauthn-20180320/
https://www.w3.org/TR/2018/WD-webauthn-20180315/
https://www.w3.org/TR/2018/WD-webauthn-20180306/
https://www.w3.org/TR/2017/WD-webauthn-20171205/
https://www.w3.org/TR/2017/WD-webauthn-20170811/
https://www.w3.org/TR/2017/WD-webauthn-20170505/
https://www.w3.org/TR/2017/WD-webauthn-20170216/
https://www.w3.org/TR/2016/WD-webauthn-20161207/
https://www.w3.org/TR/2016/WD-webauthn-20160928/
https://www.w3.org/TR/2016/WD-webauthn-20160902/
https://www.w3.org/TR/2016/WD-webauthn-20160531/

Issue Tracking:
GitHub

Editors:
Dirk Balfanz (Google)
Alexei Czeskis (Google)
Jeff Hodges (PayPal)
J.C. Jones (Mozilla)
Michael B. Jones (Microsoft)
Akshay Kumar (Microsoft)
Angelo Liao (Microsoft)
Rolf Lindemann (Nok Nok Labs)
Emil Lundberg (Yubico)

Former Editors:
Vijay Bharadwaj (Microsoft)
Arnar Birgisson (Google)
Hubert Le Van Gong (PayPal)

Contributors:
Christiaan Brand (Google)
Adam Langley (Google)
Giridhar Mandyam (Qualcomm)
Mike West (Google)
Jeffrey Yasskin (Google)

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

Copyright 2018 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 http://www.w3.org/TR/.

This document was published by the Web Authentication Working Group as an Editors' 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 an Editors' 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 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 February 2018 W3C Process Document.

Table of Contents

1. 1 Introduction
    1. 1.1 Use Cases
        1. 1.1.1 Registration
        2. 1.1.2 Authentication
        3. 1.1.3 Other use cases and configurations
2. 2 Conformance
    1. 2.1 User Agents
    2. 2.2 Authenticators
        1. 2.2.1 Backwards Compatibility with FIDO U2F
    3. 2.3 Relying Parties
    4. 2.4 All Conformance Classes
3. 3 Dependencies
4. 4 Terminology
5. 5 Web Authentication API
    1. 5.1 PublicKeyCredential Interface
        1. 5.1.1 CredentialCreationOptions Dictionary Extension
        2. 5.1.2 CredentialRequestOptions Dictionary Extension
        3. 5.1.3 Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method
        4. 5.1.4 Use an existing credential to make an assertion - PublicKeyCredential's [[Get]](options) method
            1. 5.1.4.1 PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
        5. 5.1.5 Store an existing credential - PublicKeyCredential's [[Store]](credential, sameOriginWithAncestors) method
        6. 5.1.6 Preventing silent access to an existing credential - PublicKeyCredential's

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 http://www.w3.org/TR/.

This document was published by the Web Authentication Working Group as an Editors' 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 an Editors' 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 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 February 2018 W3C Process Document.

Table of Contents

1. 1 Introduction
    1. 1.1 Use Cases
        1. 1.1.1 Registration
        2. 1.1.2 Authentication
        3. 1.1.3 Other use cases and configurations
2. 2 Conformance
    1. 2.1 User Agents
    2. 2.2 Authenticators
        1. 2.2.1 Backwards Compatibility with FIDO U2F
    3. 2.3 Relying Parties
    4. 2.4 All Conformance Classes
3. 3 Dependencies
4. 4 Terminology
5. 5 Web Authentication API
    1. 5.1 PublicKeyCredential Interface
        1. 5.1.1 CredentialCreationOptions Dictionary Extension
        2. 5.1.2 CredentialRequestOptions Dictionary Extension
        3. 5.1.3 Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method
        4. 5.1.4 Use an existing credential to make an assertion - PublicKeyCredential's [[Get]](options) method
            1. 5.1.4.1 PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
        5. 5.1.5 Store an existing credential - PublicKeyCredential's [[Store]](credential, sameOriginWithAncestors) method
        6. 5.1.6 Preventing silent access to an existing credential - PublicKeyCredential's

[[preventSilentAccess]](credential, sameOriginWithAncestors) method
7. 5.1.7 Availability of User-Verifying Platform Authenticator - PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable() method
2. 5.2 Authenticator Responses (interface AuthenticatorResponse)
1. 5.2.1 Information about Public Key Credential (interface AuthenticatorAttestationResponse)
2. 5.2.2 Web Authentication Assertion (interface AuthenticatorAssertionResponse)
3. 5.3 Parameters for Credential Generation (dictionary PublicKeyCredentialParameters)
4. 5.4 Options for Credential Creation (dictionary PublicKeyCredentialCreationOptions)
1. 5.4.1 Public Key Entity Description (dictionary PublicKeyCredentialEntity)
2. 5.4.2 RP Parameters for Credential Generation (dictionary PublicKeyCredentialRpEntity)
3. 5.4.3 User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity)
4. 5.4.4 Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)
5. 5.4.5 Authenticator Attachment enumeration (enum AuthenticatorAttachment)
6. 5.4.6 Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference)
5. 5.5 Options for Assertion Generation (dictionary PublicKeyCredentialRequestOptions)
6. 5.6 Abort operations with AbortSignal
7. 5.7 Authentication Extensions Client Inputs (typedef AuthenticationExtensionsClientInputs)
8. 5.8 Authentication Extensions Client Outputs (typedef AuthenticationExtensionsClientOutputs)
9. 5.9 Authentication Extensions Authenticator Inputs (typedef AuthenticationExtensionsAuthenticatorInputs)
10. 5.10 Supporting Data Structures
1. 5.10.1 Client data used in WebAuthn signatures (dictionary CollectedClientData)
2. 5.10.2 Credential Type enumeration (enum PublicKeyCredentialType)
3. 5.10.3 Credential Descriptor (dictionary PublicKeyCredentialDescriptor)
4. 5.10.4 Authenticator Transport enumeration (enum AuthenticatorTransport)
5. 5.10.5 Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)
6. 5.10.6 User Verification Requirement enumeration (enum UserVerificationRequirement)
6. 6 WebAuthn Authenticator Model
1. 6.1 Authenticator data
1. 6.1.1 Signature Counter Considerations
2. 6.1.2 FIDO U2F signature format compatibility
2. 6.2 Authenticator operations
1. 6.2.1 Lookup Credential Source by Credential ID algorithm
2. 6.2.2 The authenticatorMakeCredential operation
3. 6.2.3 The authenticatorGetAssertion operation
4. 6.2.4 The authenticatorCancel operation
3. 6.3 Attestation
1. 6.3.1 Attested credential data
1. 6.3.1.1 Examples of credentialPublicKey Values encoded in COSE_Key format
2. 6.3.2 Attestation Statement Formats
3. 6.3.3 Attestation Types
4. 6.3.4 Generating an Attestation Object
5. 6.3.5 Signature Formats for Packed Attestation, FIDO U2F Attestation, and Assertion Signatures
7. 7 Relying Party Operations
1. 7.1 Registering a new credential
2. 7.2 Verifying an authentication assertion
8. 8 Defined Attestation Statement Formats

[[preventSilentAccess]](credential, sameOriginWithAncestors) method
7. 5.1.7 Availability of User-Verifying Platform Authenticator - PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable() method
2. 5.2 Authenticator Responses (interface AuthenticatorResponse)
1. 5.2.1 Information about Public Key Credential (interface AuthenticatorAttestationResponse)
2. 5.2.2 Web Authentication Assertion (interface AuthenticatorAssertionResponse)
3. 5.3 Parameters for Credential Generation (dictionary PublicKeyCredentialParameters)
4. 5.4 Options for Credential Creation (dictionary PublicKeyCredentialCreationOptions)
1. 5.4.1 Public Key Entity Description (dictionary PublicKeyCredentialEntity)
2. 5.4.2 RP Parameters for Credential Generation (dictionary PublicKeyCredentialRpEntity)
3. 5.4.3 User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity)
4. 5.4.4 Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)
5. 5.4.5 Authenticator Attachment enumeration (enum AuthenticatorAttachment)
6. 5.4.6 Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference)
5. 5.5 Options for Assertion Generation (dictionary PublicKeyCredentialRequestOptions)
6. 5.6 Abort operations with AbortSignal
7. 5.7 Authentication Extensions Client Inputs (typedef AuthenticationExtensionsClientInputs)
8. 5.8 Authentication Extensions Client Outputs (typedef AuthenticationExtensionsClientOutputs)
9. 5.9 Authentication Extensions Authenticator Inputs (typedef AuthenticationExtensionsAuthenticatorInputs)
10. 5.10 Supporting Data Structures
1. 5.10.1 Client data used in WebAuthn signatures (dictionary CollectedClientData)
2. 5.10.2 Credential Type enumeration (enum PublicKeyCredentialType)
3. 5.10.3 Credential Descriptor (dictionary PublicKeyCredentialDescriptor)
4. 5.10.4 Authenticator Transport enumeration (enum AuthenticatorTransport)
5. 5.10.5 Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)
6. 5.10.6 User Verification Requirement enumeration (enum UserVerificationRequirement)
6. 6 WebAuthn Authenticator Model
1. 6.1 Authenticator data
1. 6.1.1 Signature Counter Considerations
2. 6.1.2 FIDO U2F signature format compatibility
2. 6.2 Authenticator operations
1. 6.2.1 Lookup Credential Source by Credential ID algorithm
2. 6.2.2 The authenticatorMakeCredential operation
3. 6.2.3 The authenticatorGetAssertion operation
4. 6.2.4 The authenticatorCancel operation
3. 6.3 Attestation
1. 6.3.1 Attested credential data
1. 6.3.1.1 Examples of credentialPublicKey Values encoded in COSE_Key format
2. 6.3.2 Attestation Statement Formats
3. 6.3.3 Attestation Types
4. 6.3.4 Generating an Attestation Object
5. 6.3.5 Signature Formats for Packed Attestation, FIDO U2F Attestation, and Assertion Signatures
7. 7 Relying Party Operations
1. 7.1 Registering a new credential
2. 7.2 Verifying an authentication assertion
8. 8 Defined Attestation Statement Formats

1. 8.1 Attestation Statement Format Identifiers
2. 8.2 Packed Attestation Statement Format
    1. 8.2.1 Packed attestation statement certificate requirements
3. 8.3 TPM Attestation Statement Format
    1. 8.3.1 TPM attestation statement certificate requirements
4. 8.4 Android Key Attestation Statement Format
5. 8.5 Android SafetyNet Attestation Statement Format
6. 8.6 FIDO U2F Attestation Statement Format
7. 8.7 None Attestation Statement Format
9. 9 WebAuthn Extensions
    1. 9.1 Extension Identifiers
    2. 9.2 Defining extensions
    3. 9.3 Extending request parameters
    4. 9.4 Client extension processing
    5. 9.5 Authenticator extension processing
10. 10 Defined Extensions
    1. 10.1 FIDO AppID Extension (appid)
    2. 10.2 Simple Transaction Authorization Extension (txAuthSimple)
    3. 10.3 Generic Transaction Authorization Extension (txAuthGeneric)
    4. 10.4 Authenticator Selection Extension (authnSel)
    5. 10.5 Supported Extensions Extension (exts)
    6. 10.6 User Verification Index Extension (uvi)
    7. 10.7 Location Extension (loc)
    8. 10.8 User Verification Method Extension (uvm)
    9. 10.9 Biometric Authenticator Performance Bounds Extension (biometricPerfBounds)
11. 11 IANA Considerations
    1. 11.1 WebAuthn Attestation Statement Format Identifier Registrations
    2. 11.2 WebAuthn Extension Identifier Registrations
    3. 11.3 COSE Algorithm Registrations
12. 12 Sample scenarios
    1. 12.1 Registration
    2. 12.2 Registration Specifically with User Verifying Platform Authenticator
    3. 12.3 Authentication
    4. 12.4 Aborting Authentication Operations
    5. 12.5 Decommissioning
13. 13 Security Considerations
    1. 13.1 Cryptographic Challenges
    2. 13.2 Attestation Security Considerations
        1. 13.2.1 Attestation Certificate Hierarchy
        2. 13.2.2 Attestation Certificate and Attestation Certificate CA Compromise
    3. 13.3 Security Benefits for Relying Parties
        1. 13.3.1 Considerations for Self and None Attestation Types and Ignoring Attestation
    4. 13.4 credentialId Unsigned
    5. 13.5 Browser Permissions Framework and Extensions
14. 14 Privacy Considerations
    1. 14.1 Attestation Privacy
    2. 14.2 Registration Ceremony Privacy
    3. 14.3 Authentication Ceremony Privacy
15. 15 Acknowledgements
16. Index
    1. Terms defined by this specification
    2. Terms defined by reference
17. References
    1. Normative References
    2. Informative References
18. IDL Index
19. Issues Index

1. Introduction

This section is not normative.

This specification defines an API enabling the creation and use of

1. 8.1 Attestation Statement Format Identifiers
2. 8.2 Packed Attestation Statement Format
    1. 8.2.1 Packed attestation statement certificate requirements
3. 8.3 TPM Attestation Statement Format
    1. 8.3.1 TPM attestation statement certificate requirements
4. 8.4 Android Key Attestation Statement Format
5. 8.5 Android SafetyNet Attestation Statement Format
6. 8.6 FIDO U2F Attestation Statement Format
7. 8.7 None Attestation Statement Format
9. 9 WebAuthn Extensions
    1. 9.1 Extension Identifiers
    2. 9.2 Defining extensions
    3. 9.3 Extending request parameters
    4. 9.4 Client extension processing
    5. 9.5 Authenticator extension processing
10. 10 Defined Extensions
    1. 10.1 FIDO AppID Extension (appid)
    2. 10.2 Simple Transaction Authorization Extension (txAuthSimple)
    3. 10.3 Generic Transaction Authorization Extension (txAuthGeneric)
    4. 10.4 Authenticator Selection Extension (authnSel)
    5. 10.5 Supported Extensions Extension (exts)
    6. 10.6 User Verification Index Extension (uvi)
    7. 10.7 Location Extension (loc)
    8. 10.8 User Verification Method Extension (uvm)
    9. 10.9 Biometric Authenticator Performance Bounds Extension (biometricPerfBounds)
11. 11 IANA Considerations
    1. 11.1 WebAuthn Attestation Statement Format Identifier Registrations
    2. 11.2 WebAuthn Extension Identifier Registrations
    3. 11.3 COSE Algorithm Registrations
12. 12 Sample scenarios
    1. 12.1 Registration
    2. 12.2 Registration Specifically with User Verifying Platform Authenticator
    3. 12.3 Authentication
    4. 12.4 Aborting Authentication Operations
    5. 12.5 Decommissioning
13. 13 Security Considerations
    1. 13.1 Cryptographic Challenges
    2. 13.2 Attestation Security Considerations
        1. 13.2.1 Attestation Certificate Hierarchy
        2. 13.2.2 Attestation Certificate and Attestation Certificate CA Compromise
    3. 13.3 Security Benefits for Relying Parties
        1. 13.3.1 Considerations for Self and None Attestation Types and Ignoring Attestation
    4. 13.4 credentialId Unsigned
    5. 13.5 Browser Permissions Framework and Extensions
14. 14 Privacy Considerations
    1. 14.1 Attestation Privacy
    2. 14.2 Registration Ceremony Privacy
    3. 14.3 Authentication Ceremony Privacy
15. 15 Acknowledgements
16. Index
    1. Terms defined by this specification
    2. Terms defined by reference
17. References
    1. Normative References
    2. Informative References
18. IDL Index
19. Issues Index

1. Introduction

This section is not normative.

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. 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 scoping is enforced jointly by conforming User Agents and authenticators. Additionally, privacy across Relying Parties is maintained; Relying Parties are not able to detect any properties, or even the existence, of credentials scoped to other Relying Parties.

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 present user's account (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 navigator.credentials.create() and navigator.credentials.get(). The former is used during Registration, and the latter during Authentication.

Broadly, compliant authenticators protect public key credentials, and interact with user agents to implement the Web Authentication API. Some authenticators MAY run on the same computing device (e.g., smart phone, tablet, desktop PC) as the user agent is running on. For instance, such an authenticator might consist of a Trusted Execution Environment (TEE) applet, a Trusted Platform Module (TPM), or a Secure Element (SE) integrated into the computing device in conjunction with some 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 12 Sample scenarios.

1.1.1. Registration

* 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."

1.1.2. Authentication

* On a laptop or desktop:
  + User pairs their phone with the laptop or desktop via Bluetooth.
  + User navigates to example.com in a browser and initiates signing in.
  + User gets a message from the browser, "Please complete this action on your phone."
* Next, on their phone:

strong, attested, scoped, public key-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 scoping is enforced jointly by conforming User Agents and authenticators. Additionally, privacy across Relying Parties is maintained; Relying Parties are not able to detect any properties, or even the existence, of credentials scoped to other Relying Parties.

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 present user's account (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 navigator.credentials.create() and navigator.credentials.get(). The former is used during Registration, and the latter during Authentication.

Broadly, compliant authenticators protect public key credentials, and interact with user agents to implement the Web Authentication API. Some authenticators MAY run on the same computing device (e.g., smart phone, tablet, desktop PC) as the user agent is running on. For instance, such an authenticator might consist of a Trusted Execution Environment (TEE) applet, a Trusted Platform Module (TPM), or a Secure Element (SE) integrated into the computing device in conjunction with some 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 12 Sample scenarios.

1.1.1. Registration

* 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."

1.1.2. Authentication

* On a laptop or desktop:
  + User pairs their phone with the laptop or desktop via Bluetooth.
  + User navigates to example.com in a browser and initiates signing in.
  + User 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 selects this prompt / notification.
- + 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.

* Now, back on the laptop:
  - + Web page shows that the selected user is signed in, and navigates to the signed-in page.

1.1.3. Other use cases and configurations

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 their phone.
* A user obtains a discrete, roaming authenticator, such as a "fob" with USB or USB+NFC/BLE connectivity options, loads example.com in their browser on a laptop or phone, and is guided though a flow to create and register a credential on the fob.
* A Relying Party prompts the user for their authorization gesture in order to authorize a single transaction, such as a payment or other financial transaction.

2. Conformance

This specification defines three conformance classes. Each of these classes is specified so that conforming members of the class are secure against non-conforming or hostile members of the other classes.

2.1. User Agents

A 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.

2.2.1. Backwards Compatibility with FIDO U2F

Authenticators that only support the 8.6 FIDO U2F Attestation Statement Format have no mechanism to store a user handle, so the returned userHandle will always be null.

2.3. Relying Parties

A Relying Party MUST behave as described in 7 Relying Party Operations to obtain all the security benefits offered by this specification. See 13.3 Security Benefits for Relying Parties for further discussion of this.

2.4. All Conformance Classes

All CBOR encoding performed by the members of the above conformance

+ User sees a discrete prompt or notification, "Sign in to example.com."
+ User selects this prompt / notification.
+ 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.
* Now, back on the laptop:
+ Web page shows that the selected user is signed in, and navigates to the signed-in page.

1.1.3. Other use cases and configurations

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 their phone.
* A user obtains a discrete, roaming authenticator, such as a "fob" with USB or USB+NFC/BLE connectivity options, loads example.com in their browser on a laptop or phone, and is guided though a flow to create and register a credential on the fob.
* A Relying Party prompts the user for their authorization gesture in order to authorize a single transaction, such as a payment or other financial transaction.

2. Conformance

This specification defines three conformance classes. Each of these classes is specified so that conforming members of the class are secure against non-conforming or hostile members of the other classes.

2.1. User Agents

A 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.

2.2.1. Backwards Compatibility with FIDO U2F

Authenticators that only support the 8.6 FIDO U2F Attestation Statement Format have no mechanism to store a user handle, so the returned userHandle will always be null.

2.3. Relying Parties

A Relying Party MUST behave as described in 7 Relying Party Operations to obtain all the security benefits offered by this specification. See 13.3 Security Benefits for Relying Parties for further discussion of this.

2.4. All Conformance Classes

All CBOR encoding performed by the members of the above conformance

classes MUST be done using the CTAP2 canonical CBOR encoding form. All decoders of the above conformance classes SHOULD reject CBOR that is not validly encoded in the CTAP2 canonical CBOR encoding form and SHOULD reject messages with duplicate map keys.

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 CTAP2 canonical CBOR encoding form of the Compact Binary Object Representation (CBOR) [RFC7049], as defined in [FIDO-CTAP].

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 [DOM4].

ECMAScript
%ArrayBuffer% is defined in [ECMAScript].

HTML
The concepts of relevant settings object, origin, opaque origin, and is a registrable domain suffix of or is equal to are defined in [HTML52].

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.

FIDO AppID
The algorithms for determining the FacetID of a calling application and determining if a caller's FacetID is authorized for an AppID (used only in the AppID extension) are defined by [FIDO-APPID].

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].

4. Terminology

Assertion
See Authentication Assertion.

classes MUST be done using the CTAP2 canonical CBOR encoding form. All decoders of the above conformance classes SHOULD reject CBOR that is not validly encoded in the CTAP2 canonical CBOR encoding form and SHOULD reject messages with duplicate map keys.

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 CTAP2 canonical CBOR encoding form of the Compact Binary Object Representation (CBOR) [RFC7049], as defined in [FIDO-CTAP].

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 [DOM4].

ECMAScript
%ArrayBuffer% is defined in [ECMAScript].

HTML
The concepts of relevant settings object, origin, opaque origin, and is a registrable domain suffix of or is equal to are defined in [HTML52].

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.

FIDO AppID
The algorithms for determining the FacetID of a calling application and determining if a caller's FacetID is authorized for an AppID (used only in the AppID extension) are defined by [FIDO-APPID].

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].

4. Terminology

Assertion
See Authentication Assertion.

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 6.3 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 a 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 an authenticatorGetAssertion operation.

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

Authenticator

A cryptographic entity used by a WebAuthn Client to (i) generate a public key credential and register it with a Relying Party, and (ii) authenticate by potentially verifying the user, and then cryptographically signing and returning, in the form of an Authentication Assertion, a challenge and other data presented by a Relying Party (in concert with the WebAuthn Client).

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 [ISOBiometricVocabulary].

Biometric Authenticator

Any authenticator that implements biometric recognition.

Ceremony

The concept of a ceremony [Ceremony] is an extension of the concept of a network protocol, with human nodes alongside

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 6.3 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 a 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 an authenticatorGetAssertion operation.

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

Authenticator

A cryptographic entity used by a WebAuthn Client to (i) generate a public key credential and register it with a Relying Party, and (ii) authenticate by potentially verifying the user, and then cryptographically signing and returning, in the form of an Authentication Assertion, a challenge and other data presented by a Relying Party (in concert with the WebAuthn Client).

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 [ISOBiometricVocabulary].

Biometric Authenticator

Any authenticator that implements biometric recognition.

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 WebAuthn Client, 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 credential private key given only an RP ID, possibly with user assistance (e.g., by providing the user a pick list of credentials associated with the RP ID). By definition, the private key is always exclusively controlled by the authenticator. In the case of a Client-side-resident Credential Private Key, the authenticator might offload storage of wrapped 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, by having the Relying Party store any necessary state. Note: [FIDO-UAF-AUTHNR-CMDS] includes guidance on encryption techniques under "Security Guidelines".

Relying Parties do not need to distinguish these two Credential ID forms.

Credential Public Key
User Public Key

The public key portion of a Relying Party-specific credential key pair, generated by an authenticator and returned to a Relying Party at registration time (see also public key credential). The private key portion of the credential key pair is known as the credential private key. Note that in the case of self attestation, the credential key pair is also used as the attestation key pair, see self attestation for details.

Note: The credential public key is referred to as the user public key in FIDO UAF [UAFProtocol], and in FIDO U2F [FIDO-U2F-Message-Formats] and some parts of this specification that relate to it.

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 WebAuthn Client, 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 credential private key given only an RP ID, possibly with user assistance (e.g., by providing the user a pick list of credentials associated with the RP ID). By definition, the private key is always exclusively controlled by the authenticator. In the case of a Client-side-resident Credential Private Key, the authenticator might offload storage of wrapped 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, by having the Relying Party store any necessary state. Note: [FIDO-UAF-AUTHNR-CMDS] includes guidance on encryption techniques under "Security Guidelines".

Relying Parties do not need to distinguish these two Credential ID forms.

Credential Public Key
User Public Key

The public key portion of a Relying Party-specific credential key pair, generated by an authenticator and returned to a Relying Party at registration time (see also public key credential). The private key portion of the credential key pair is known as the credential private key. Note that in the case of self attestation, the credential key pair is also used as the attestation key pair, see self attestation for details.

Note: The credential public key is referred to as the user public key in FIDO UAF [UAFProtocol], and in FIDO U2F [FIDO-U2F-Message-Formats] and some parts of this specification that relate to it.

Human Palatability

An identifier that is human-palatable is intended to be rememberable and reproducible by typical human users, in contrast to identifiers that are, for example, randomly generated sequences of bits [EduPersonObjectClassSpec].

Public Key Credential Source

A credential source ([CREDENTIAL-MANAGEMENT-1]) used by an authenticator to generate authentication assertions. A public key credential source consists of a struct with the following items:

type

whose value is of PublicKeyCredentialType, defaulting to public-key.

id

A Credential ID.

privateKey

The credential private key.

rpId

The Relying Party Identifier, for the Relying Party this public key credential source is associated with.

userHandle

The user handle associated when this public key credential source was created. This item is nullable.

otherUI

Optional other information used by the authenticator to inform its UI. For example, this might include the user's displayName.

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 credential public key 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:

"Authentication information" (possibly including a private key)

Public key credential source

"Signed value"

Authentication assertion

[RFC4949] "credential"

Credential public key or attestation object

Human Palatability

An identifier that is human-palatable is intended to be rememberable and reproducible by typical human users, in contrast to identifiers that are, for example, randomly generated sequences of bits [EduPersonObjectClassSpec].

Public Key Credential Source

A credential source ([CREDENTIAL-MANAGEMENT-1]) used by an authenticator to generate authentication assertions. A public key credential source consists of a struct with the following items:

type

whose value is of PublicKeyCredentialType, defaulting to public-key.

id

A Credential ID.

privateKey

The credential private key.

rpId

The Relying Party Identifier, for the Relying Party this public key credential source is associated with.

userHandle

The user handle associated when this public key credential source was created. This item is nullable.

otherUI

Optional other information used by the authenticator to inform its UI. For example, this might include the user's displayName.

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 credential public key 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:

"Authentication information" (possibly including a private key)

Public key credential source

"Signed value"

Authentication assertion

[RFC4949] "credential"

Credential public key or attestation object

At registration time, the authenticator creates an asymmetric key pair, and stores its private key portion and information from the Relying Party into a public key credential source. The public key portion is returned to the Relying Party, who then stores it in conjunction with the present user's account. Subsequently, only that 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.

Rate Limiting

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.

Registration

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 includes employing a test of user presence or user verification.

Relying Party

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.

Relying Party Identifier
RP ID

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 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 5.1.3 Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method and 5.1.4 Use an existing credential to make an assertion - 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, [RFC6265]). Please note that this is a greater relaxation of "same-origin" restrictions than what document.domain's setter provides.

At registration time, the authenticator creates an asymmetric key pair, and stores its private key portion and information from the Relying Party into a public key credential source. The public key portion is returned to the Relying Party, who then stores it in conjunction with the present user's account. Subsequently, only that 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.

Rate Limiting

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.

Registration

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 includes employing a test of user presence or user verification.

Relying Party

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.

Relying Party Identifier
RP ID

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 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 5.1.3 Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method and 5.1.4 Use an existing credential to make an assertion - 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, [RFC6265]). 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 gesture and technical process where a user interacts with an authenticator by (typically) simply touching it (other modalities may also exist), yielding a boolean result. Note that this does not constitute user verification because a user presence test, by definition, is not capable of biometric recognition, nor does it involve the presentation of a shared secret such as a password or PIN.

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 Handle

The user handle is specified by a Relying Party and is a unique identifier for a user account with that Relying Party. A user handle is an opaque byte sequence with a maximum size of 64 bytes.

The user handle 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 under the same user handle.

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) [ISOBiometricVocabulary]. 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
UP

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

User Verified
UV

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. A WebAuthn Client is an intermediary entity typically implemented in the user agent (in whole, or in part). Conceptually, it underlies the Web Authentication API and embodies the implementation of the [[Create]](origin, options, sameOriginWithAncestors) and [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) internal methods. It is responsible for both marshalling the inputs for the underlying authenticator operations, and for returning the results of the latter operations to the Web Authentication API's callers.

5. 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

Test of User Presence

A test of user presence is a simple form of authorization gesture and technical process where a user interacts with an authenticator by (typically) simply touching it (other modalities may also exist), yielding a boolean result. Note that this does not constitute user verification because a user presence test, by definition, is not capable of biometric recognition, nor does it involve the presentation of a shared secret such as a password or PIN.

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 Handle

The user handle is specified by a Relying Party and is a unique identifier for a user account with that Relying Party. A user handle is an opaque byte sequence with a maximum size of 64 bytes.

The user handle 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 under the same user handle.

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) [ISOBiometricVocabulary]. 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
UP

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

User Verified
UV

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. A WebAuthn Client is an intermediary entity typically implemented in the user agent (in whole, or in part). Conceptually, it underlies the Web Authentication API and embodies the implementation of the [[Create]](origin, options, sameOriginWithAncestors) and [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) internal methods. It is responsible for both marshalling the inputs for the underlying authenticator operations, and for returning the results of the latter operations to the Web Authentication API's callers.

5. 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. 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 that implements) a user interface for management. Such an interface MAY be used, for example, to reset 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 6.2 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 by 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 only 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.

```
[SecureContext, Exposed=Window]
interface PublicKeyCredential : Credential {
    [SameObject] readonly attribute ArrayBuffer              rawId;
    [SameObject] readonly attribute AuthenticatorResponse    response;
    AuthenticationExtensionsClientOutputs getClientExtensionResults();
};
```

id

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.

rawId

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. 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 that implements) a user interface for management. Such an interface MAY be used, for example, to reset 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 6.2 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 by 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 only 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.

[SecureContext, Exposed=Window]
interface PublicKeyCredential : Credential {
    [SameObject] readonly attribute ArrayBuffer   rawId;
    [SameObject] readonly attribute AuthenticatorResponse   response;
    AuthenticationExtensionsClientOutputs getClientExtensionResults();
};

id

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.

rawId

```
            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.

        getClientExtensionResults()
            This operation returns the value of [[clientExtensionsResults]],
            which is a map containing extension identifier -> client
            extension output entries produced by the extension's client
            extension processing.

        [[type]]
            The PublicKeyCredential interface object's [[type]] internal
            slot's value is the string "public-key".

            Note: This is reflected via the type attribute getter inherited
            from Credential.

        [[discovery]]
            The PublicKeyCredential interface object's [[discovery]]
            internal slot's value is "remote".

        [[identifier]]
            This internal slot contains the credential ID, chosen by the
            platform with help from the authenticator. The credential ID is
            used to look up credentials for use, and is therefore expected
            to be globally unique with high probability across all
            credentials of the same type, across all authenticators.

            Note: This API does not constrain the format or length of this
            identifier, except that it MUST be sufficient for the platform
            to uniquely select a key. For example, an authenticator without
            on-board storage may create identifiers containing a credential
            private key wrapped with a symmetric key that is burned into the
            authenticator.

        [[clientExtensionsResults]]
            This internal slot contains the results of processing client
            extensions requested by the Relying Party upon the Relying
            Party's invocation of either navigator.credentials.create() or
            navigator.credentials.get().

        PublicKeyCredential's interface object inherits Credential's
        implementation of [[CollectFromCredentialStore]](origin, options,
        sameOriginWithAncestors), and defines its own implementation of
        [[Create]](origin, options, sameOriginWithAncestors),
        [[DiscoverFromExternalSource]](origin, options,
        sameOriginWithAncestors), and [[Store]](credential,
        sameOriginWithAncestors).

         5.1.1. CredentialCreationOptions Dictionary Extension

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

         5.1.2. CredentialRequestOptions Dictionary Extension

        To support obtaining assertions via navigator.credentials.get(), this
        document extends the CredentialRequestOptions dictionary as follows:
      partial dictionary CredentialRequestOptions {
```

```
            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.

        getClientExtensionResults()
            This operation returns the value of [[clientExtensionsResults]],
            which is a map containing extension identifier -> client
            extension output entries produced by the extension's client
            extension processing.

        [[type]]
            The PublicKeyCredential interface object's [[type]] internal
            slot's value is the string "public-key".

            Note: This is reflected via the type attribute getter inherited
            from Credential.

        [[discovery]]
            The PublicKeyCredential interface object's [[discovery]]
            internal slot's value is "remote".

        [[identifier]]
            This internal slot contains the credential ID, chosen by the
            platform with help from the authenticator. The credential ID is
            used to look up credentials for use, and is therefore expected
            to be globally unique with high probability across all
            credentials of the same type, across all authenticators.

            Note: This API does not constrain the format or length of this
            identifier, except that it MUST be sufficient for the platform
            to uniquely select a key. For example, an authenticator without
            on-board storage may create identifiers containing a credential
            private key wrapped with a symmetric key that is burned into the
            authenticator.

        [[clientExtensionsResults]]
            This internal slot contains the results of processing client
            extensions requested by the Relying Party upon the Relying
            Party's invocation of either navigator.credentials.create() or
            navigator.credentials.get().

        PublicKeyCredential's interface object inherits Credential's
        implementation of [[CollectFromCredentialStore]](origin, options,
        sameOriginWithAncestors), and defines its own implementation of
        [[Create]](origin, options, sameOriginWithAncestors),
        [[DiscoverFromExternalSource]](origin, options,
        sameOriginWithAncestors), and [[Store]](credential,
        sameOriginWithAncestors).

         5.1.1. CredentialCreationOptions Dictionary Extension

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

         5.1.2. CredentialRequestOptions Dictionary Extension

        To support obtaining assertions via navigator.credentials.get(), this
        document extends the CredentialRequestOptions dictionary as follows:
      partial dictionary CredentialRequestOptions {
```

```
    PublicKeyCredentialRequestOptions      publicKey;
};
```

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

PublicKeyCredential's interface object's implementation of the [[Create]](origin, options, sameOriginWithAncestors) internal method [CREDENTIAL-MANAGEMENT-1] allows Relying Party scripts to call 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
: This argument is the relevant settings object's origin, as determined by the calling create() implementation.

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

sameOriginWithAncestors
: 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.create().

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

1. Assert: options.publicKey is present.
2. If sameOriginWithAncestors is false, return a "NotAllowedError" DOMException.
   Note: This "sameOriginWithAncestors" restriction aims to address the concern raised in the Origin Confusion section of [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script access to Web Authentication functionality, e.g., when running in a secure context framed document that is 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.
3. Let options be the value of options.publicKey.
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 a timer lifetimeTimer to this adjusted value. If the timeout member of options is not present, then set lifetimeTimer to a platform-specific default.
5. Let callerOrigin be origin. 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.rp.id

```
    PublicKeyCredentialRequestOptions      publicKey;
};
```

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

PublicKeyCredential's interface object's implementation of the [[Create]](origin, options, sameOriginWithAncestors) internal method [CREDENTIAL-MANAGEMENT-1] allows Relying Party scripts to call 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
: This argument is the relevant settings object's origin, as determined by the calling create() implementation.

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

sameOriginWithAncestors
: 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.create().

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

1. Assert: options.publicKey is present.
2. If sameOriginWithAncestors is false, return a "NotAllowedError" DOMException.
   Note: This "sameOriginWithAncestors" restriction aims to address the concern raised in the Origin Confusion section of [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script access to Web Authentication functionality, e.g., when running in a secure context framed document that is 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.
3. Let options be the value of options.publicKey.
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 a timer lifetimeTimer to this adjusted value. If the timeout member of options is not present, then set lifetimeTimer to a platform-specific default.
5. Let callerOrigin be origin. 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.rp.id

```
          Is present
                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.

          Is not present
                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.
        9. For each current of options.pubKeyCredParams:
           1. If current.type does not contain a PublicKeyCredentialType
              supported by this implementation, then continue.
           2. Let alg be current.alg.
           3. Append the pair of current.type and alg to
              credTypesAndPubKeyAlgs.
       10. If credTypesAndPubKeyAlgs is empty and options.pubKeyCredParams is
          not empty, return a DOMException whose name is "NotSupportedError",
          and terminate this algorithm.
       11. Let clientExtensions be a new map and let authenticatorExtensions
          be a new map.
       12. 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 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,
              continue.
           5. Set authenticatorExtensions[extensionId] to the base64url
              encoding of authenticatorExtensionInput.
       13. 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.

          tokenBinding
                The status of Token Binding between the client and the
                callerOrigin, as well as the Token Binding ID associated
                with callerOrigin, if one is available.

       14. Let clientDataJSON be the JSON-serialized client data constructed
          from collectedClientData.
       15. Let clientDataHash be the hash of the serialized client data
          represented by clientDataJSON.
       16. 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.
       17. Start lifetimeTimer.
       18. Let issuedRequests be a new ordered set.
       19. For each authenticator that becomes available on this platform
          during the lifetime of lifetimeTimer, do the following:
          The definitions of "lifetime of" and "becomes available" are
          intended to represent how devices are hot-plugged into (USB) or
          discovered by (NFC) browsers, and are underspecified. Resolving
```

```
          Is present
                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.

          Is not present
                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.
        9. For each current of options.pubKeyCredParams:
           1. If current.type does not contain a PublicKeyCredentialType
              supported by this implementation, then continue.
           2. Let alg be current.alg.
           3. Append the pair of current.type and alg to
              credTypesAndPubKeyAlgs.
       10. If credTypesAndPubKeyAlgs is empty and options.pubKeyCredParams is
          not empty, return a DOMException whose name is "NotSupportedError",
          and terminate this algorithm.
       11. Let clientExtensions be a new map and let authenticatorExtensions
          be a new map.
       12. 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 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,
              continue.
           5. Set authenticatorExtensions[extensionId] to the base64url
              encoding of authenticatorExtensionInput.
       13. 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.

          tokenBinding
                The status of Token Binding between the client and the
                callerOrigin, as well as the Token Binding ID associated
                with callerOrigin, if one is available.

       14. Let clientDataJSON be the JSON-serialized client data constructed
          from collectedClientData.
       15. Let clientDataHash be the hash of the serialized client data
          represented by clientDataJSON.
       16. 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.
       17. Let issuedRequests be a new ordered set.
       18. Let authenticators represent a set of platform-specific handles,
          where each value identifies an authenticator presently available on
          this platform at a given instant.
          Note: What qualifies an authenticator as "available" is
          intentionally unspecified; this is meant to represent how
          authenticators can be hot-plugged into (e.g., via USB) or
```

```
   this with good definitions or some other means will be addressed by
   resolving Issue #613.

    1. If options.authenticatorSelection is present:
         1. If options.authenticatorSelection.authenticatorAttachment
            is present and its value is not equal to authenticator's
            attachment modality, continue.
         2. If options.authenticatorSelection.requireResidentKey is
            set to true and the authenticator is not capable of
            storing a Client-Side-Resident Credential Private Key,
            continue.
         3. If options.authenticatorSelection.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 credential creation, a Boolean value, as
       follows. If options.authenticatorSelection.userVerification

         is set to required
                Let userVerification be true.

         is set to preferred
                If the authenticator

                    is capable of user verification
                           Let userVerification be true.

                    is not capable of user verification
                           Let userVerification be false.

         is set to discouraged
                Let userVerification be false.

    3. Let userPresence be a Boolean value set to the inverse of
       userVerification.
    4. Let excludeCredentialDescriptorList be a new list.
    5. For each credential descriptor C in
       options.excludeCredentials:
         1. If C.transports is not empty, and authenticator is
            connected over a transport not mentioned in C.transports,
            the client MAY continue.
         2. Otherwise, Append C to excludeCredentialDescriptorList.

    6. Invoke the authenticatorMakeCredential operation on
       authenticator with clientDataHash, options.rp, options.user,

       options.authenticatorSelection.requireResidentKey,
       userPresence, userVerification, credTypesAndPubKeyAlgs,
       excludeCredentialDescriptorList, and authenticatorExtensions
```

```
       discovered (e.g., via NFC or Bluetooth) by the client by various
       mechanisms.
   19. Start lifetimeTimer.
   20. While lifetimeTimer has not expired, perform the following actions
       depending upon lifetimeTimer and the state and response for each
       authenticator in authenticators:

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

        If the options.signal is present and its aborted flag is set to
               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 an authenticator becomes available on this platform,

             1. If options.authenticatorSelection is present:
                  1. If
                     options.authenticatorSelection.authenticatorAttachme
                     nt is present and its value is not equal to
                     authenticator's attachment modality, continue.
                  2. If options.authenticatorSelection.requireResidentKey
                     is set to true and the authenticator is not capable
                     of storing a Client-Side-Resident Credential Private
                     Key, continue.
                  3. If options.authenticatorSelection.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 credential creation, a Boolean value, as
                follows. If
                options.authenticatorSelection.userVerification

                  is set to required
                         Let userVerification be true.

                  is set to preferred
                         If the authenticator

                             is capable of user verification
                                    Let userVerification be true.

                             is not capable of user verification
                                    Let userVerification be false.

                  is set to discouraged
                         Let userVerification be false.

             3. Let userPresence be a Boolean value set to the inverse of
                userVerification.
             4. Let excludeCredentialDescriptorList be a new list.
             5. For each credential descriptor C in
                options.excludeCredentials:
                  1. If C.transports is not empty, and authenticator is
                     connected over a transport not mentioned in
                     C.transports, the client MAY continue.
                  2. Otherwise, Append C to
                     excludeCredentialDescriptorList.
             6. Invoke the authenticatorMakeCredential operation on
                authenticator with clientDataHash, options.rp,
                options.user,
                options.authenticatorSelection.requireResidentKey,
                userPresence, userVerification, credTypesAndPubKeyAlgs,
                excludeCredentialDescriptorList, and
```

as parameters.

7. Append authenticator to issuedRequests.

20. While lifetimeTimer has not expired, 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 options.signal is present and its aborted flag is set to 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.
Note: Authenticators may return an indication of "the user cancelled the entire operation". How a user agent manifests this state to users is unspecified.

If any authenticator returns an error status equivalent to "InvalidStateError",

1. Remove authenticator from issuedRequests.
2. For each remaining authenticator in issuedRequests invoke the authenticatorCancel operation on authenticator and remove it from issuedRequests.
3. Return a DOMException whose name is "InvalidStateError" and terminate this algorithm.

Note: This error status is handled separately because the authenticator returns it only if excludeCredentialDescriptorList identifies a credential bound to the authenticator and the user has consented to the operation. Given this explicit consent, it is acceptable for this case to be distinguishable to the Relying Party.

If any authenticator returns an error status not equivalent to "InvalidStateError",
Remove authenticator from issuedRequests.

Note: This case does not imply user consent for the operation, so details about the error must be hidden from the Relying Party in order to prevent leak of potentially identifying information. See 14.2 Registration Ceremony Privacy for details.

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.

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

authenticatorExtensions as parameters.

7. Append authenticator to issuedRequests.

If an authenticator ceases to be available on this platform,
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.
Note: Authenticators may return an indication of "the user cancelled the entire operation". How a user agent manifests this state to users is unspecified.

If any authenticator returns an error status equivalent to "InvalidStateError",

1. Remove authenticator from issuedRequests.
2. For each remaining authenticator in issuedRequests invoke the authenticatorCancel operation on authenticator and remove it from issuedRequests.
3. Return a DOMException whose name is "InvalidStateError" and terminate this algorithm.

Note: This error status is handled separately because the authenticator returns it only if excludeCredentialDescriptorList identifies a credential bound to the authenticator and the user has consented to the operation. Given this explicit consent, it is acceptable for this case to be distinguishable to the Relying Party.

If any authenticator returns an error status not equivalent to "InvalidStateError",
Remove authenticator from issuedRequests.

Note: This case does not imply user consent for the operation, so details about the error must be hidden from the Relying Party in order to prevent leak of potentially identifying information. See 14.2 Registration Ceremony Privacy for details.

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.

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

```
      clientDataJSONResult
          whose value is the bytes of clientDataJSON.

      attestationConveyancePreferenceOption
          whose value is the value of
          options.attestation.

      clientExtensionResults
          whose value is an
          AuthenticationExtensionsClientOutputs 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. If
          credentialCreationData.attestationConveyancePreferen
          ceOption's value is

               "none"
                   Replace potentially uniquely
                   identifying information with
                   non-identifying versions of the
                   same:

                1. If the AAGUID in the attested
                   credential data is 16 zero bytes,
                   credentialCreationData.attestationObj
                   ectResult.fmt is "packed", and "x5c"
                   & "ecdaaKeyId" are both absent from
                   credentialCreationData.attestationObj
                   ectResult, then self attestation is
                   being used and no further action is
                   needed.
                2. Otherwise
                1. Replace the AAGUID in the attested
                   credential data with 16 zero bytes.
                2. Set the value of
                   credentialCreationData.attestationObj
                   ectResult.fmt to "none", and set the
                   value of
                   credentialCreationData.attestationObj
                   ectResult.attStmt to be an empty CBOR
                   map. (See 8.7 None Attestation
                   Statement Format and 6.3.4
                   Generating an Attestation Object).

               "indirect"
                   The client MAY replace the AAGUID
                   and attestation statement with a
                   more privacy-friendly and/or more
                   easily verifiable version of the
                   same data (for example, by
                   employing an Anonymization CA).

               "direct"
                   Convey the authenticator's AAGUID
                   and attestation statement,
                   unaltered, to the RP.

                @balfanz 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 an
                "AttestationNotPrivateError".
```

```
      clientDataJSONResult
          whose value is the bytes of clientDataJSON.

      attestationConveyancePreferenceOption
          whose value is the value of
          options.attestation.

      clientExtensionResults
          whose value is an
          AuthenticationExtensionsClientOutputs 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. If
          credentialCreationData.attestationConveyancePreferen
          ceOption's value is

               "none"
                   Replace potentially uniquely
                   identifying information with
                   non-identifying versions of the
                   same:

                1. If the AAGUID in the attested
                   credential data is 16 zero bytes,
                   credentialCreationData.attestationObj
                   ectResult.fmt is "packed", and "x5c"
                   & "ecdaaKeyId" are both absent from
                   credentialCreationData.attestationObj
                   ectResult, then self attestation is
                   being used and no further action is
                   needed.
                2. Otherwise
                1. Replace the AAGUID in the attested
                   credential data with 16 zero bytes.
                2. Set the value of
                   credentialCreationData.attestationObj
                   ectResult.fmt to "none", and set the
                   value of
                   credentialCreationData.attestationObj
                   ectResult.attStmt to be an empty CBOR
                   map. (See 8.7 None Attestation
                   Statement Format and 6.3.4
                   Generating an Attestation Object).

               "indirect"
                   The client MAY replace the AAGUID
                   and attestation statement with a
                   more privacy-friendly and/or more
                   easily verifiable version of the
                   same data (for example, by
                   employing an Anonymization CA).

               "direct"
                   Convey the authenticator's AAGUID
                   and attestation statement,
                   unaltered, to the RP.

                @balfanz 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 an
                "AttestationNotPrivateError".
```

```
         2. Let attestationObject be a new ArrayBuffer, created
           using global's %ArrayBuffer%, containing the bytes
           of credentialCreationData.attestationObjectResult's
           value.
         3. Let id be
           attestationObject.authData.attestedCredentialData.cr
           edentialId.
         4. Let pubKeyCred be a new PublicKeyCredential object
           associated with global whose fields are:

              [[identifier]]
                   id

              response
                   A new AuthenticatorAttestationResponse
                   object associated with global whose
                   fields are:

                 clientDataJSON
                      A new ArrayBuffer, created using
                      global's %ArrayBuffer%, containing
                      the bytes of
                      credentialCreationData.clientDataJ
                      SONResult.

                 attestationObject
                      attestationObject

              [[clientExtensionsResults]]
                   A new ArrayBuffer, created using
                   global's %ArrayBuffer%, containing the
                   bytes of
                   credentialCreationData.clientExtensionRe
                   sults.

         5. Return pubKeyCred.
      4. For each remaining authenticator in issuedRequests invoke
        the authenticatorCancel operation on authenticator and
        remove it from issuedRequests.
      5. Return constructCredentialAlg and terminate this
        algorithm.

21. Return a DOMException whose name is "NotAllowedError". In order to
   prevent information leak that could identify the user without
   consent, this step MUST NOT be executed before lifetimeTimer has
   expired. See 14.3 Authentication Ceremony Privacy for details.

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.

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

Relying Parties call navigator.credentials.get({publicKey:..., ...}) to
discover and use an existing public key credential, with the user's
consent. Relying Party script optionally specifies some criteria to
indicate what credential sources 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 6.2.3 The authenticatorGetAssertion operation to sign a Relying
Party-provided challenge and other collected data into an assertion,
which is used as a credential.

The get() implementation [CREDENTIAL-MANAGEMENT-1] calls
PublicKeyCredential.[[CollectFromCredentialStore]]() to collect any
```

```
         2. Let attestationObject be a new ArrayBuffer, created
           using global's %ArrayBuffer%, containing the bytes
           of credentialCreationData.attestationObjectResult's
           value.
         3. Let id be
           attestationObject.authData.attestedCredentialData.cr
           edentialId.
         4. Let pubKeyCred be a new PublicKeyCredential object
           associated with global whose fields are:

              [[identifier]]
                   id

              response
                   A new AuthenticatorAttestationResponse
                   object associated with global whose
                   fields are:

                 clientDataJSON
                      A new ArrayBuffer, created using
                      global's %ArrayBuffer%, containing
                      the bytes of
                      credentialCreationData.clientDataJ
                      SONResult.

                 attestationObject
                      attestationObject

              [[clientExtensionsResults]]
                   A new ArrayBuffer, created using
                   global's %ArrayBuffer%, containing the
                   bytes of
                   credentialCreationData.clientExtensionRe
                   sults.

         5. Return pubKeyCred.
      4. For each remaining authenticator in issuedRequests invoke
        the authenticatorCancel operation on authenticator and
        remove it from issuedRequests.
      5. Return constructCredentialAlg and terminate this
        algorithm.

21. Return a DOMException whose name is "NotAllowedError". In order to
   prevent information leak that could identify the user without
   consent, this step MUST NOT be executed before lifetimeTimer has
   expired. See 14.3 Authentication Ceremony Privacy for details.

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.

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

Relying Parties call navigator.credentials.get({publicKey:..., ...}) to
discover and use an existing public key credential, with the user's
consent. Relying Party script optionally specifies some criteria to
indicate what credential sources 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 6.2.3 The authenticatorGetAssertion operation to sign a Relying
Party-provided challenge and other collected data into an assertion,
which is used as a credential.

The get() implementation [CREDENTIAL-MANAGEMENT-1] calls
PublicKeyCredential.[[CollectFromCredentialStore]]() to collect any
```

credentials that should be available without user mediation (roughly, this specification's authorization gesture), and if it does not find exactly one of those, it then calls PublicKeyCredential.[[DiscoverFromExternalSource]]() to have the user select a credential source.

Since this specification requires an authorization gesture to create any credentials, the PublicKeyCredential.[[CollectFromCredentialStore]](origin, options, sameOriginWithAncestors) internal method inherits the default behavior of Credential.[[CollectFromCredentialStore]](), of returning an empty set.

5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

This internal method accepts three arguments:

origin
    This argument is the relevant settings object's origin, as determined by the calling get() implementation, i.e., CredentialsContainer's Request a Credential abstract operation.

options
    This argument is a CredentialRequestOptions object whose options.publicKey member contains a PublicKeyCredentialRequestOptions object specifying the desired attributes of the public key credential to discover.

sameOriginWithAncestors
    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().

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

1. Assert: options.publicKey is present.
2. If sameOriginWithAncestors is false, return a "NotAllowedError" DOMException.
   Note: This "sameOriginWithAncestors" restriction aims to address the concern raised in the Origin Confusion section of [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script access to Web Authentication functionality, e.g., when running in a secure context framed document that is 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.
3. Let options be the value of options.publicKey.
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 a timer lifetimeTimer to this adjusted value. If the timeout member of options is not present, then set lifetimeTimer to a platform-specific default.
5. Let callerOrigin be origin. 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

credentials that should be available without user mediation (roughly, this specification's authorization gesture), and if it does not find exactly one of those, it then calls PublicKeyCredential.[[DiscoverFromExternalSource]]() to have the user select a credential source.

Since this specification requires an authorization gesture to create any credentials, the PublicKeyCredential.[[CollectFromCredentialStore]](origin, options, sameOriginWithAncestors) internal method inherits the default behavior of Credential.[[CollectFromCredentialStore]](), of returning an empty set.

5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

This internal method accepts three arguments:

origin
    This argument is the relevant settings object's origin, as determined by the calling get() implementation, i.e., CredentialsContainer's Request a Credential abstract operation.

options
    This argument is a CredentialRequestOptions object whose options.publicKey member contains a PublicKeyCredentialRequestOptions object specifying the desired attributes of the public key credential to discover.

sameOriginWithAncestors
    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().

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

1. Assert: options.publicKey is present.
2. If sameOriginWithAncestors is false, return a "NotAllowedError" DOMException.
   Note: This "sameOriginWithAncestors" restriction aims to address the concern raised in the Origin Confusion section of [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script access to Web Authentication functionality, e.g., when running in a secure context framed document that is 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.
3. Let options be the value of options.publicKey.
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 a timer lifetimeTimer to this adjusted value. If the timeout member of options is not present, then set lifetimeTimer to a platform-specific default.
5. Let callerOrigin be origin. 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

/Users/jehodges/Documents/work/standards/W3C/webauthn/index-master-3c5e383.txt, Top line: 1446

```
          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,
              continue.
           5. Set authenticatorExtensions[extensionId] to the base64url
              encoding of authenticatorExtensionInput.
      10. Let collectedClientData be a new CollectedClientData instance whose
          fields are:

           type
                  The string "webauthn.get".

           challenge
                  The base64url encoding of options.challenge

           origin
                  The serialization of callerOrigin.

           tokenBinding
                  The status of Token Binding between the client and the
                  callerOrigin, as well as the Token Binding ID associated
                  with callerOrigin, if one is available.

      11. Let clientDataJSON be the JSON-serialized client data constructed
          from collectedClientData.
      12. Let clientDataHash be the hash of the serialized client data
          represented by clientDataJSON.
      13. 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.
      14. Let issuedRequests be a new ordered set.
      15. Let authenticator be a platform-specific handle whose value
          identifies an authenticator.
      16. Start lifetimeTimer.
      17. For each authenticator that becomes available on this platform
          during the lifetime of lifetimeTimer, perform the following steps:
          The definitions of "lifetime of" and "becomes available" are
          intended to represent how devices are hot-plugged into (USB) or
          discovered by (NFC) browsers, and are underspecified. Resolving
          this with good definitions or some other means will be addressed by
          resolving Issue #613.
```

/Users/jehodges/Documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.txt, Top line: 1458

```
          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,
              continue.
           5. Set authenticatorExtensions[extensionId] to the base64url
              encoding of authenticatorExtensionInput.
      10. Let collectedClientData be a new CollectedClientData instance whose
          fields are:

           type
                  The string "webauthn.get".

           challenge
                  The base64url encoding of options.challenge

           origin
                  The serialization of callerOrigin.

           tokenBinding
                  The status of Token Binding between the client and the
                  callerOrigin, as well as the Token Binding ID associated
                  with callerOrigin, if one is available.

      11. Let clientDataJSON be the JSON-serialized client data constructed
          from collectedClientData.
      12. Let clientDataHash be the hash of the serialized client data
          represented by clientDataJSON.
      13. 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.
      14. Let issuedRequests be a new ordered set.
      15. Let savedCredentialIds be a new map.
      16. Let authenticators represent a set of platform-specific handles,
          where each value identifies an authenticator presently available on
          this platform at a given instant.
          Note: What qualifies an authenticator as "available" is
          intentionally unspecified; this is meant to represent how
          authenticators can be hot-plugged into (e.g., via USB) or
          discovered (e.g., via NFC or Bluetooth) by the client by various
          mechanisms.
      17. Start lifetimeTimer.
      18. While lifetimeTimer has not expired, perform the following actions
          depending upon lifetimeTimer and the state and response for each
          authenticator in authenticators:

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

           If the user exercises a user-interface option to cancel the
```

```
        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.

             is set to preferred
                    If the authenticator

                  is capable of user verification
                         Let userVerification be true.

                  is not capable of user verification
                         Let userVerification be false.

             is set to discouraged
                    Let userVerification be false.

        3. Let userPresence be a Boolean value set to the inverse of
           userVerification.
        4. If options.allowCredentials

             is not empty

                1. Let allowCredentialDescriptorList be a new list.
                2. 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.
```

```
                process,
                For each authenticator in issuedRequests invoke the
                authenticatorCancel operation on authenticator and remove
                authenticator from issuedRequests. Return a DOMException
                whose name is "NotAllowedError".

            If the signal member is present and the aborted flag is set to
                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 issuedRequests is empty, options.allowCredentials is not empty,
                and no authenticator will become available for any public
                key credentials therein,
                Indicate to the user that the credential could not be
                found. When the user acknowledges the dialog, or once
                lifetimeTimer expires, return a DOMException whose name is
                "NotAllowedError".

                Note: One way in which the platform may determine that no
                authenticator will become available is by using the
                transports members of options.allowCredentials. For
                example, if all credentials only list internal, but all
                internal authenticators have been tried, then there is no
                possibility of satisfying the request. Alternatively, all
                credentials may require transports that the platform does
                not support.

            If an authenticator becomes available on this platform,

               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.

                    is set to preferred
                           If the authenticator

                         is capable of user verification
                                Let userVerification be true.

                         is not capable of user verification
                                Let userVerification be false.

                    is set to discouraged
                           Let userVerification be false.

               3. Let userPresence be a Boolean value set to the inverse of
                  userVerification.
               4. If options.allowCredentials

                    is not empty

                       1. Let allowCredentialDescriptorList be a new
                          list.
                       2. 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
```

```
          3. If allowCredentialDescriptorList is empty, continue.

          4. Let distinctTransports be a new ordered set.
          5. 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 6.2.3 The authenticatorGetAssertion
             operation for more information).

             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.

          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.

       is empty
          Using local configuration knowledge of the
          appropriate transport to use with authenticator,
          invoke the authenticatorGetAssertion operation on

          authenticator with rpId, clientDataHash,
          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
```

```
             list.
          3. If allowCredentialDescriptorList is empty,
             continue.
          4. Let distinctTransports be a new ordered set.
          5. If allowCredentialDescriptorList has exactly
             one value, set
             savedCredentialIds[authenticator] to
             allowCredentialDescriptorList[0].id's value
             (see here in 6.2.3 The
             authenticatorGetAssertion operation for more
             information).
          6. 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.
          7. 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.

       is empty
          Using local configuration knowledge of the
          appropriate transport to use with
          authenticator, invoke the
          authenticatorGetAssertion operation on
          authenticator with rpId, clientDataHash,
          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
```

```
                   credential it may possess that is bound to the
                   Relying Party, as identified by rpId.

             5. Append authenticator to issuedRequests.
   18. While lifetimeTimer has not expired, 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
              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.
                Note: Authenticators may return an indication of "the
                user cancelled the entire operation". How a user agent
                manifests this state to users is unspecified.

        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
                        savedCredentialId. Otherwise, set the value of
                        credentialIdResult to be the bytes of the
                        credential ID returned from the successful
                        authenticatorGetAssertion operation, as
                        defined in 6.2.3 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
                        If the authenticator returned a user handle,
                        set the value of userHandleResult to be the
                        bytes of the returned user handle. Otherwise,
                        set the value of userHandleResult to null.

                  clientExtensionResults
                        whose value is an
                        AuthenticationExtensionsClientOutputs object
```

```
                   asked to exercise any credential it may
                   possess that is bound to the Relying Party, as
                   identified by rpId.

             5. Append authenticator to issuedRequests.

        If an authenticator ceases to be available on this platform,
              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.
                Note: Authenticators may return an indication of "the
                user cancelled the entire operation". How a user agent
                manifests this state to users is unspecified.

        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 savedCredentialIds[authenticator] exists,
                        set the value of credentialIdResult to be the
                        bytes of savedCredentialIds[authenticator].
                        Otherwise, set the value of credentialIdResult
                        to be the bytes of the credential ID returned
                        from the successful authenticatorGetAssertion
                        operation, as defined in 6.2.3 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
                        If the authenticator returned a user handle,
                        set the value of userHandleResult to be the
                        bytes of the returned user handle. Otherwise,
                        set the value of userHandleResult to null.

                  clientExtensionResults
                        whose value is an
                        AuthenticationExtensionsClientOutputs 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
                          .

                   response
                          A new AuthenticatorAssertionResponse
                          object associated with global whose
                          fields are:

                        clientDataJSON
                               A new ArrayBuffer, created using
                               global's %ArrayBuffer%, containing
                               the bytes of
                               assertionCreationData.clientDataJS
                               ONResult.

                        authenticatorData
                               A new ArrayBuffer, created using
                               global's %ArrayBuffer%, containing
                               the bytes of
                               assertionCreationData.authenticato
                               rDataResult.

                        signature
                               A new ArrayBuffer, created using
                               global's %ArrayBuffer%, containing
                               the bytes of
                               assertionCreationData.signatureRes
                               ult.

                        userHandle
                               If
                               assertionCreationData.userHandleRe
                               sult is null, set this field to
                               null. Otherwise, set this field to
                               a new ArrayBuffer, created using
                               global's %ArrayBuffer%, containing
                               the bytes of
                               assertionCreationData.userHandleRe
                               sult.

                   [[clientExtensionsResults]]
                          A new ArrayBuffer, created using
                          global's %ArrayBuffer%, containing the
                          bytes of
                          assertionCreationData.clientExtensionRes
                          ults.

              2. Return pubKeyCred.
          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.
```

```
                    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
                          .

                   response
                          A new AuthenticatorAssertionResponse
                          object associated with global whose
                          fields are:

                        clientDataJSON
                               A new ArrayBuffer, created using
                               global's %ArrayBuffer%, containing
                               the bytes of
                               assertionCreationData.clientDataJS
                               ONResult.

                        authenticatorData
                               A new ArrayBuffer, created using
                               global's %ArrayBuffer%, containing
                               the bytes of
                               assertionCreationData.authenticato
                               rDataResult.

                        signature
                               A new ArrayBuffer, created using
                               global's %ArrayBuffer%, containing
                               the bytes of
                               assertionCreationData.signatureRes
                               ult.

                        userHandle
                               If
                               assertionCreationData.userHandleRe
                               sult is null, set this field to
                               null. Otherwise, set this field to
                               a new ArrayBuffer, created using
                               global's %ArrayBuffer%, containing
                               the bytes of
                               assertionCreationData.userHandleRe
                               sult.

                   [[clientExtensionsResults]]
                          A new ArrayBuffer, created using
                          global's %ArrayBuffer%, containing the
                          bytes of
                          assertionCreationData.clientExtensionRes
                          ults.

              2. Return pubKeyCred.
          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.
```

19. Return a DOMException whose name is "NotAllowedError". In order to prevent information leak that could identify the user without consent, this step MUST NOT be executed before lifetimeTimer has expired. See 14.3 Authentication Ceremony Privacy for details.

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.

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

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

1. Return a DOMException whose name is "NotSupportedError", and terminate this algorithm

5.1.6. Preventing silent access to an existing credential - PublicKeyCredential's [[preventSilentAccess]](credential, sameOriginWithAncestors) method

Calling the [[preventSilentAccess]](credential, sameOriginWithAncestors) method will have no effect on authenticators that require an authorization gesture, but setting that flag may potentially exclude authenticators that can operate without user intervention.

This internal method accepts no arguments.

5.1.7. 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. This assessment may include various factors, such as:

* Whether the user is running in private or incognito mode.
* Whether the user has configured the client to not create such credentials.
* Whether the user has previously expressed an unwillingness to create a new credential for this Relying Party, either through 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.

19. Return a DOMException whose name is "NotAllowedError". In order to prevent information leak that could identify the user without consent, this step MUST NOT be executed before lifetimeTimer has expired. See 14.3 Authentication Ceremony Privacy for details.

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.

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

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

1. Return a DOMException whose name is "NotSupportedError", and terminate this algorithm

5.1.6. Preventing silent access to an existing credential - PublicKeyCredential's [[preventSilentAccess]](credential, sameOriginWithAncestors) method

Calling the [[preventSilentAccess]](credential, sameOriginWithAncestors) method will have no effect on authenticators that require an authorization gesture, but setting that flag may potentially exclude authenticators that can operate without user intervention.

This internal method accepts no arguments.

5.1.7. 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. This assessment may include various factors, such as:

* Whether the user is running in private or incognito mode.
* Whether the user has configured the client to not create such credentials.
* Whether the user has previously expressed an unwillingness to create a new credential for this Relying Party, either through 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 cannot distinguish between the case where the user was unwilling to create a credential using one of the available user-verifying platform authenticators and the case where no user-verifying 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 performed but short enough that the dangling promise will still be resolved in a reasonably timely fashion.

```
partial interface PublicKeyCredential {
    static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();
};
```

5.2. Authenticator 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().

5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)

The AuthenticatorAttestationResponse interface represents 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.

```
[SecureContext, Exposed=Window]
interface AuthenticatorAttestationResponse : AuthenticatorResponse {
    [SameObject] readonly attribute ArrayBuffer      attestationObject;
};
```

clientDataJSON
: This attribute, inherited from AuthenticatorResponse, contains the JSON-serialized client data (see 6.3 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 credential public key. The contents of the attestation statement are determined by 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 6.3 Attestation, 6.3.4 Generating an Attestation Object, and Figure 3.

5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)

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 cannot distinguish between the case where the user was unwilling to create a credential using one of the available user-verifying platform authenticators and the case where no user-verifying 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 performed but short enough that the dangling promise will still be resolved in a reasonably timely fashion.

```
partial interface PublicKeyCredential {
    static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();
};
```

5.2. Authenticator 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().

5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)

The AuthenticatorAttestationResponse interface represents 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.

```
[SecureContext, Exposed=Window]
interface AuthenticatorAttestationResponse : AuthenticatorResponse {
    [SameObject] readonly attribute ArrayBuffer      attestationObject;
};
```

clientDataJSON
: This attribute, inherited from AuthenticatorResponse, contains the JSON-serialized client data (see 6.3 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 credential public key. The contents of the attestation statement are determined by 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 6.3 Attestation, 6.3.4 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.

```
[SecureContext, Exposed=Window]
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 5.10.1 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 authenticator data returned by the authenticator. See 6.1 Authenticator data.

signature, of type ArrayBuffer, readonly
: This attribute contains the raw signature returned from the authenticator. See 6.2.3 The authenticatorGetAssertion operation.

userHandle, of type ArrayBuffer, readonly, nullable
: This attribute contains the user handle returned from the authenticator, or null if the authenticator did not return a user handle. See 6.2.3 The authenticatorGetAssertion operation.

5.3. Parameters for Credential Generation (dictionary PublicKeyCredentialParameters)

```
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.

5.4. Options for Credential Creation (dictionary PublicKeyCredentialCreationOptions)

```
dictionary PublicKeyCredentialCreationOptions {
    required PublicKeyCredentialRpEntity         rp;
    required PublicKeyCredentialUserEntity       user;

    required BufferSource                             challenge;
    required sequence<PublicKeyCredentialParameters>  pubKeyCredParams;

    unsigned long                                timeout;
```

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.

```
[SecureContext, Exposed=Window]
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 5.10.1 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 authenticator data returned by the authenticator. See 6.1 Authenticator data.

signature, of type ArrayBuffer, readonly
: This attribute contains the raw signature returned from the authenticator. See 6.2.3 The authenticatorGetAssertion operation.

userHandle, of type ArrayBuffer, readonly, nullable
: This attribute contains the user handle returned from the authenticator, or null if the authenticator did not return a user handle. See 6.2.3 The authenticatorGetAssertion operation.

5.3. Parameters for Credential Generation (dictionary PublicKeyCredentialParameters)

```
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.

5.4. Options for Credential Creation (dictionary PublicKeyCredentialCreationOptions)

```
dictionary PublicKeyCredentialCreationOptions {
    required PublicKeyCredentialRpEntity         rp;
    required PublicKeyCredentialUserEntity       user;

    required BufferSource                             challenge;
    required sequence<PublicKeyCredentialParameters>  pubKeyCredParams;

    unsigned long                                timeout;
    sequence<PublicKeyCredentialDescriptor>      excludeCredentials = [];
    AuthenticatorSelectionCriteria               authenticatorSelection;
    AttestationConveyancePreference              attestation = "none";
    AuthenticationExtensionsClientInputs         extensions;
};
```

rp, of type PublicKeyCredentialRpEntity
This member contains data about the Relying Party responsible for the request.

Its value's name member is required.

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, displayName and id members are required.

challenge, of type BufferSource
This member contains a challenge intended to be used for generating the newly created credential's attestation object. See the 13.1 Cryptographic Challenges security consideration.

pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>
This member contains information about the desired properties of the credential to be created. The sequence is ordered from most preferred to least preferred. The platform makes a best-effort to create the most preferred credential that it can.

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

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.

authenticatorSelection, of type AuthenticatorSelectionCriteria
This member is intended for use by Relying Parties that wish to select the appropriate authenticators to participate in the create() 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 AuthenticationExtensionsClientInputs
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.

```
    sequence<PublicKeyCredentialDescriptor>      excludeCredentials = [];
    AuthenticatorSelectionCriteria               authenticatorSelection;
    AttestationConveyancePreference              attestation = "none";
    AuthenticationExtensionsClientInputs         extensions;
};
```

rp, of type PublicKeyCredentialRpEntity
This member contains data about the Relying Party responsible for the request.

Its value's name member is required.

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, displayName and id members are required.

challenge, of type BufferSource
This member contains a challenge intended to be used for generating the newly created credential's attestation object. See the 13.1 Cryptographic Challenges security consideration.

pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>
This member contains information about the desired properties of the credential to be created. The sequence is ordered from most preferred to least preferred. The platform makes a best-effort to create the most preferred credential that it can.

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

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.

authenticatorSelection, of type AuthenticatorSelectionCriteria
This member is intended for use by Relying Parties that wish to select the appropriate authenticators to participate in the create() 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 AuthenticationExtensionsClientInputs
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.

```
   5.4.1. 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.
dictionary PublicKeyCredentialEntity {
    required DOMString    name;
    USVString             icon;
};

  name, of type DOMString
         A human-readable name for the entity. Its function depends on
         what the PublicKeyCredentialEntity represents:

         + When inherited by PublicKeyCredentialRpEntity it is a
           human-friendly identifier for the Relying Party, intended only
           for display. For example, "ACME Corporation", "Wonderful
           Widgets, Inc." or "OAO Primerteh".
         + When inherited by PublicKeyCredentialUserEntity, it is a
           human-palatable identifier for a user account. It is intended
           only for display, and SHOULD allow the user to easily tell the
           difference between user accounts with similar displayNames.
           For example, "alexm", "alex.p.mueller@example.com" or
           "+14255551234". The Relying Party MAY let the user choose
           this, and MAY restrict the choice as needed or appropriate.
           For example, a Relying Party might choose to map
           human-palatable username account identifiers to the name
           member of PublicKeyCredentialUserEntity.

         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.

  icon, of type USVString
         A serialized URL which resolves to an image associated with the
         entity. For example, this could be a user's avatar or a Relying
         Party's logo. This URL MUST be an a priori authenticated URL.
         Authenticators MUST accept and store a 128-byte minimum length
         for an icon member's value. Authenticators MAY ignore an icon
         member's value if its length is greater than 128 bytes.

   5.4.2. RP Parameters for Credential Generation (dictionary
   PublicKeyCredentialRpEntity)

  The PublicKeyCredentialRpEntity dictionary is used to supply additional
  Relying Party attributes when creating a new credential.
dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
    DOMString      id;
};

  id, of type DOMString
         A unique identifier for the Relying Party entity, which sets the
         RP ID.

   5.4.3. User Account Parameters for Credential Generation (dictionary
   PublicKeyCredentialUserEntity)

  The PublicKeyCredentialUserEntity dictionary is used to supply
  additional user account attributes when creating a new credential.
dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
    required BufferSource   id;
    required DOMString      displayName;
};

  id, of type BufferSource
         The user handle of the user account entity.

  displayName, of type DOMString
         A human-friendly name for the user account, intended only for
         display. For example, "Alex P. Mller" or " ". The Relying
         Party SHOULD let the user choose this, and SHOULD NOT restrict
```

```
   5.4.1. 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.
dictionary PublicKeyCredentialEntity {
    required DOMString    name;
    USVString             icon;
};

  name, of type DOMString
         A human-readable name for the entity. Its function depends on
         what the PublicKeyCredentialEntity represents:

         + When inherited by PublicKeyCredentialRpEntity it is a
           human-friendly identifier for the Relying Party, intended only
           for display. For example, "ACME Corporation", "Wonderful
           Widgets, Inc." or "OAO Primerteh".
         + When inherited by PublicKeyCredentialUserEntity, it is a
           human-palatable identifier for a user account. It is intended
           only for display, and SHOULD allow the user to easily tell the
           difference between user accounts with similar displayNames.
           For example, "alexm", "alex.p.mueller@example.com" or
           "+14255551234". The Relying Party MAY let the user choose
           this, and MAY restrict the choice as needed or appropriate.
           For example, a Relying Party might choose to map
           human-palatable username account identifiers to the name
           member of PublicKeyCredentialUserEntity.

         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.

  icon, of type USVString
         A serialized URL which resolves to an image associated with the
         entity. For example, this could be a user's avatar or a Relying
         Party's logo. This URL MUST be an a priori authenticated URL.
         Authenticators MUST accept and store a 128-byte minimum length
         for an icon member's value. Authenticators MAY ignore an icon
         member's value if its length is greater than 128 bytes.

   5.4.2. RP Parameters for Credential Generation (dictionary
   PublicKeyCredentialRpEntity)

  The PublicKeyCredentialRpEntity dictionary is used to supply additional
  Relying Party attributes when creating a new credential.
dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
    DOMString      id;
};

  id, of type DOMString
         A unique identifier for the Relying Party entity, which sets the
         RP ID.

   5.4.3. User Account Parameters for Credential Generation (dictionary
   PublicKeyCredentialUserEntity)

  The PublicKeyCredentialUserEntity dictionary is used to supply
  additional user account attributes when creating a new credential.
dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
    required BufferSource   id;
    required DOMString      displayName;
};

  id, of type BufferSource
         The user handle of the user account entity.

  displayName, of type DOMString
         A human-friendly name for the user account, intended only for
         display. For example, "Alex P. Mller" or " ". The Relying
         Party SHOULD let the user choose this, and SHOULD NOT restrict
```

the choice more than necessary.

Authenticators MUST accept and store a 64-byte minimum length for a displayName member's value. Authenticators MAY truncate a displayName member's value to a length equal to or greater than 64 bytes.

5.4.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

Relying Parties may use the AuthenticatorSelectionCriteria dictionary to specify their requirements regarding authenticator attributes.

```
dictionary AuthenticatorSelectionCriteria {
    AuthenticatorAttachment      authenticatorAttachment;
    boolean                      requireResidentKey = false;
    UserVerificationRequirement  userVerification = "preferred";
};
```

authenticatorAttachment, of type AuthenticatorAttachment
If this member is present, eligible authenticators are filtered to only authenticators attached with the specified 5.4.5 Authenticator Attachment enumeration (enum AuthenticatorAttachment).

requireResidentKey, of type boolean, defaulting to false
This member describes the Relying Parties' requirements regarding availability of the Client-side-resident Credential Private Key. If the parameter is set to true, the authenticator MUST create a Client-side-resident Credential Private Key when creating a public key credential.

userVerification, of type UserVerificationRequirement, defaulting to "preferred"
This member describes the Relying Party's requirements regarding user verification for the create() operation. Eligible authenticators are filtered to only those capable of satisfying this requirement.

5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)

```
enum AuthenticatorAttachment {
    "platform",       // Platform attachment
    "cross-platform"  // Cross-platform attachment
};
```

Clients can 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 can use a variety of standardized cross-platform transport protocols such as Bluetooth (see 5.10.4 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.

* platform attachment - the respective authenticator is attached using platform-specific transports. Usually, authenticators of this class are non-removable from the platform. A public key credential bound to a platform authenticator is called a platform credential.
* cross-platform attachment - the respective authenticator is attached using cross-platform transports. Authenticators of this class are removable from, and can "roam" among, client platforms. A public key credential bound to a roaming authenticator is called a roaming credential.

This distinction is important because there are use-cases where only platform authenticators are acceptable to a Relying Party, and

the choice more than necessary.

Authenticators MUST accept and store a 64-byte minimum length for a displayName member's value. Authenticators MAY truncate a displayName member's value to a length equal to or greater than 64 bytes.

5.4.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

Relying Parties may use the AuthenticatorSelectionCriteria dictionary to specify their requirements regarding authenticator attributes.

dictionary AuthenticatorSelectionCriteria {
    AuthenticatorAttachment   authenticatorAttachment;
    boolean   requireResidentKey = false;
    UserVerificationRequirement userVerification = "preferred";
};

authenticatorAttachment, of type AuthenticatorAttachment
If this member is present, eligible authenticators are filtered to only authenticators attached with the specified 5.4.5 Authenticator Attachment enumeration (enum AuthenticatorAttachment).

requireResidentKey, of type boolean, defaulting to false
This member describes the Relying Parties' requirements regarding availability of the Client-side-resident Credential Private Key. If the parameter is set to true, the authenticator MUST create a Client-side-resident Credential Private Key when creating a public key credential.

userVerification, of type UserVerificationRequirement, defaulting to "preferred"
This member describes the Relying Party's requirements regarding user verification for the create() operation. Eligible authenticators are filtered to only those capable of satisfying this requirement.

5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)

enum AuthenticatorAttachment {
    "platform",   // Platform attachment
    "cross-platform" // Cross-platform attachment
};

Clients can 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 can use a variety of standardized cross-platform transport protocols such as Bluetooth (see 5.10.4 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.

* platform attachment - the respective authenticator is attached using platform-specific transports. Usually, authenticators of this class are non-removable from the platform. A public key credential bound to a platform authenticator is called a platform credential.
* cross-platform attachment - the respective authenticator is attached using cross-platform transports. Authenticators of this class are removable from, and can "roam" among, client platforms. A public key credential bound to a roaming authenticator is called a roaming credential.

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 platform credential 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 from a given client for the first time, they may be asked to use a roaming credential which was originally registered with the Relying Party using a different client.

Note: An attachment modality selection option is available only in the [[Create]](origin, options, sameOriginWithAncestors) operation. The Relying Party may use it to, for example, ensure the user has a roaming credential for authenticating using other clients; or to specifically register a platform credential for easier reauthentication using a particular client. The [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) operation has no attachment modality selection option, so the Relying Party should accept any of the user's registered credentials. The client and user will then use whichever is available and convenient at the time.

5.4.6. Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference)

Relying Parties may use AttestationConveyancePreference to specify their preference regarding attestation conveyance during credential generation.

```
enum AttestationConveyancePreference {
    "none",
    "indirect",
    "direct"
};
```

* none - indicates that the Relying Party is not interested in authenticator attestation. For example, in order to potentially avoid having to obtain user consent to relay identifying information to the Relying Party, or to save a roundtrip to an Attestation CA.
  This is the default value.
* indirect - indicates that the Relying Party prefers an attestation conveyance yielding verifiable attestation statements, but allows the client to decide how to obtain such attestation statements. The client MAY replace the authenticator-generated attestation statements with attestation statements generated by an Anonymization CA, in order to protect the user's privacy, or to assist Relying Parties with attestation verification in a heterogeneous ecosystem.
  Note: There is no guarantee that the Relying Party will obtain a verifiable attestation statement in this case. For example, in the case that the authenticator employs self attestation.
* direct - indicates that the Relying Party wants to receive the attestation statement as generated by the authenticator.

5.5. Options for Assertion Generation (dictionary 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 = [];
    UserVerificationRequirement          userVerification = "preferred";
    AuthenticationExtensionsClientInputs extensions;
};
```

challenge, of type BufferSource
    This member represents a challenge that the selected

conversely ones where only roaming authenticators are employed. As a concrete example of the former, a platform credential 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 from a given client for the first time, they may be asked to use a roaming credential which was originally registered with the Relying Party using a different client.

Note: An attachment modality selection option is available only in the [[Create]](origin, options, sameOriginWithAncestors) operation. The Relying Party may use it to, for example, ensure the user has a roaming credential for authenticating using other clients; or to specifically register a platform credential for easier reauthentication using a particular client. The [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) operation has no attachment modality selection option, so the Relying Party should accept any of the user's registered credentials. The client and user will then use whichever is available and convenient at the time.

5.4.6. Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference)

Relying Parties may use AttestationConveyancePreference to specify their preference regarding attestation conveyance during credential generation.

enum AttestationConveyancePreference {
    "none",
    "indirect",
    "direct"
};

* none - indicates that the Relying Party is not interested in authenticator attestation. For example, in order to potentially avoid having to obtain user consent to relay identifying information to the Relying Party, or to save a roundtrip to an Attestation CA.
  This is the default value.
* indirect - indicates that the Relying Party prefers an attestation conveyance yielding verifiable attestation statements, but allows the client to decide how to obtain such attestation statements. The client MAY replace the authenticator-generated attestation statements with attestation statements generated by an Anonymization CA, in order to protect the user's privacy, or to assist Relying Parties with attestation verification in a heterogeneous ecosystem.
  Note: There is no guarantee that the Relying Party will obtain a verifiable attestation statement in this case. For example, in the case that the authenticator employs self attestation.
* direct - indicates that the Relying Party wants to receive the attestation statement as generated by the authenticator.

5.5. Options for Assertion Generation (dictionary 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 = [];
    UserVerificationRequirement   userVerification = "preferred";
    AuthenticationExtensionsClientInputs extensions;
};

challenge, of type BufferSource
    This member represents a challenge that the selected

authenticator signs, along with other data, when producing an authentication assertion. See the 13.1 Cryptographic Challenges security consideration.

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 CredentialsContainer object's relevant settings object's 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).

userVerification, of type UserVerificationRequirement, defaulting to "preferred"
This member describes the Relying Party's requirements regarding user verification for the get() operation. Eligible authenticators are filtered to only those capable of satisfying this requirement.

extensions, of type AuthenticationExtensionsClientInputs
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.

5.6. Abort operations with AbortSignal

Developers are encouraged to leverage the AbortController to manage the [[Create]](origin, options, sameOriginWithAncestors) and [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) operations. See DOM 3.3 Using AbortController and AbortSignal objects in APIs section for detailed instructions.

Note: DOM 3.3 Using AbortController and AbortSignal objects in APIs section specifies that web platform APIs integrating with the AbortController must reject the promise immediately once the aborted flag is set. Given the complex inheritance and parallelization structure of the [[Create]](origin, options, sameOriginWithAncestors) and [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) methods, the algorithms for the two APIs fulfills this requirement by checking the aborted flag in three places. In the case of [[Create]](origin, options, sameOriginWithAncestors), the aborted flag is checked first in Credential Management 1 2.5.4 Create a Credential immediately before calling [[Create]](origin, options, sameOriginWithAncestors), then in 5.1.3 Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method right before authenticator sessions start, and finally during authenticator sessions. The same goes for [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors).

The visibility and focus state of the Window object determines whether the [[Create]](origin, options, sameOriginWithAncestors) and [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) operations should continue. When the Window object associated with the [Document loses focus, [[Create]](origin, options, sameOriginWithAncestors) and [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) operations SHOULD be aborted.

authenticator signs, along with other data, when producing an authentication assertion. See the 13.1 Cryptographic Challenges security consideration.

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 CredentialsContainer object's relevant settings object's 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).

userVerification, of type UserVerificationRequirement, defaulting to "preferred"
This member describes the Relying Party's requirements regarding user verification for the get() operation. Eligible authenticators are filtered to only those capable of satisfying this requirement.

extensions, of type AuthenticationExtensionsClientInputs
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.

5.6. Abort operations with AbortSignal

Developers are encouraged to leverage the AbortController to manage the [[Create]](origin, options, sameOriginWithAncestors) and [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) operations. See DOM 3.3 Using AbortController and AbortSignal objects in APIs section for detailed instructions.

Note: DOM 3.3 Using AbortController and AbortSignal objects in APIs section specifies that web platform APIs integrating with the AbortController must reject the promise immediately once the aborted flag is set. Given the complex inheritance and parallelization structure of the [[Create]](origin, options, sameOriginWithAncestors) and [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) methods, the algorithms for the two APIs fulfills this requirement by checking the aborted flag in three places. In the case of [[Create]](origin, options, sameOriginWithAncestors), the aborted flag is checked first in Credential Management 1 2.5.4 Create a Credential immediately before calling [[Create]](origin, options, sameOriginWithAncestors), then in 5.1.3 Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method right before authenticator sessions start, and finally during authenticator sessions. The same goes for [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors).

The visibility and focus state of the Window object determines whether the [[Create]](origin, options, sameOriginWithAncestors) and [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) operations should continue. When the Window object associated with the [Document loses focus, [[Create]](origin, options, sameOriginWithAncestors) and [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) operations SHOULD be aborted.

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.

5.7. Authentication Extensions Client Inputs (typedef AuthenticationExtensionsClientInputs)

```
dictionary AuthenticationExtensionsClientInputs {
};
```

This is a dictionary containing the client extension input values for zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.

5.8. Authentication Extensions Client Outputs (typedef AuthenticationExtensionsClientOutputs)

```
dictionary AuthenticationExtensionsClientOutputs {
};
```

This is a dictionary containing the client extension output values for zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.

5.9. Authentication Extensions Authenticator Inputs (typedef AuthenticationExtensionsAuthenticatorInputs)

```
typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs
;
```

This is a dictionary containing the authenticator extension input values for zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.

5.10. Supporting Data Structures

The public key credential type uses certain data structures that are specified in supporting specifications. These are as follows.

5.10.1. Client data used in WebAuthn signatures (dictionary CollectedClientData)

The client data represents the contextual bindings of both the Relying Party and the client platform. It is a key-value mapping whose keys are strings. Values can be any type that has a valid encoding in JSON. Its structure is defined by the following Web IDL.

Note: The CollectedClientData may be extended in the future. Therefore it's critical when parsing to be tolerant of unknown keys and of any reordering of the keys.

```
dictionary CollectedClientData {
    required DOMString           type;
    required DOMString           challenge;
    required DOMString           origin;
    TokenBinding                 tokenBinding;
};

dictionary TokenBinding {
    required TokenBindingStatus status;
    DOMString id;
};

enum TokenBindingStatus { "present", "supported", "not-supported" };
```

The type member contains the string "webauthn.create" when creating new credentials, and "webauthn.get" when getting an assertion from an existing credential. The purpose of this member is to prevent certain types of signature confusion attacks (where an attacker substitutes one legitimate signature for another).

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.

5.7. Authentication Extensions Client Inputs (typedef AuthenticationExtensionsClientInputs)

dictionary AuthenticationExtensionsClientInputs {
};

This is a dictionary containing the client extension input values for zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.

5.8. Authentication Extensions Client Outputs (typedef AuthenticationExtensionsClientOutputs)

dictionary AuthenticationExtensionsClientOutputs {
};

This is a dictionary containing the client extension output values for zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.

5.9. Authentication Extensions Authenticator Inputs (typedef AuthenticationExtensionsAuthenticatorInputs)

typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs
;

This is a dictionary containing the authenticator extension input values for zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.

5.10. Supporting Data Structures

The public key credential type uses certain data structures that are specified in supporting specifications. These are as follows.

5.10.1. Client data used in WebAuthn signatures (dictionary CollectedClientData)

The client data represents the contextual bindings of both the Relying Party and the client platform. It is a key-value mapping whose keys are strings. Values can be any type that has a valid encoding in JSON. Its structure is defined by the following Web IDL.

Note: The CollectedClientData may be extended in the future. Therefore it's critical when parsing to be tolerant of unknown keys and of any reordering of the keys.

dictionary CollectedClientData {
    required DOMString   type;
    required DOMString   challenge;
    required DOMString   origin;
    TokenBinding   tokenBinding;
};

dictionary TokenBinding {
    required TokenBindingStatus status;
    DOMString id;
};

enum TokenBindingStatus { "present", "supported", "not-supported" };

The type member contains the string "webauthn.create" when creating new credentials, and "webauthn.get" when getting an assertion from an existing credential. The purpose of this member is to prevent certain types of signature confusion attacks (where an attacker substitutes one legitimate signature for another).

The challenge member contains the base64url encoding of the challenge provided by the RP. See the 13.1 Cryptographic Challenges security consideration.

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

The tokenBinding member contains information about the state of the Token Binding protocol used when communicating with the Relying Party. The status member is one of:

- not-supported: when the client does not support token binding.
- supported: the client supports token binding, but it was not negotiated when communicating with the Relying Party.
- present: token binding was used when communicating with the Relying Party. In this case, the id member MUST be present and MUST be a base64url encoding of the Token Binding ID that was used.

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 SHA-256) of the JSON-serialized client data, as constructed by the client.

5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)

```
enum PublicKeyCredentialType {
    "public-key"
};
```

This enumeration defines the valid credential types. It is an extension point; values can 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".

5.10.3. 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 public key 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 public key credential the caller is referring to.

The id member contains the credential ID of the public key credential the caller is referring to.

The OPTIONAL transports member contains a hint as to how the client might communicate with the managing authenticator of the public key credential the caller is referring to.

5.10.4. Authenticator Transport enumeration (enum AuthenticatorTransport)

```
enum AuthenticatorTransport {
    "usb",
```

The challenge member contains the base64url encoding of the challenge provided by the RP. See the 13.1 Cryptographic Challenges security consideration.

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

The tokenBinding member contains information about the state of the Token Binding protocol used when communicating with the Relying Party. The status member is one of:

- not-supported: when the client does not support token binding.
- supported: the client supports token binding, but it was not negotiated when communicating with the Relying Party.
- present: token binding was used when communicating with the Relying Party. In this case, the id member MUST be present and MUST be a base64url encoding of the Token Binding ID that was used.

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 SHA-256) of the JSON-serialized client data, as constructed by the client.

5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)

enum PublicKeyCredentialType {
    "public-key"
};

This enumeration defines the valid credential types. It is an extension point; values can 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".

5.10.3. 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 public key 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 public key credential the caller is referring to.

The id member contains the credential ID of the public key credential the caller is referring to.

The OPTIONAL transports member contains a hint as to how the client might communicate with the managing authenticator of the public key credential the caller is referring to.

5.10.4. Authenticator Transport enumeration (enum AuthenticatorTransport)

enum AuthenticatorTransport {
    "usb",

```
    "nfc",
    "ble",
    "internal"
};

   Authenticators may implement various transports for communicating with
   clients. This enumeration defines hints as to how clients might
   communicate with a particular authenticator in order to obtain an
   assertion for a specific credential. Note that these hints represent
   the Relying Party's best belief as to how an authenticator may be
   reached. A Relying Party may obtain a list of transports hints from
   some attestation statement formats or via some out-of-band mechanism;
   it is outside the scope of this specification to define that mechanism.
     * usb - the respective authenticator can be contacted over removable
       USB.
     * nfc - the respective authenticator can be contacted over Near Field
       Communication (NFC).
     * ble - the respective authenticator can be contacted over Bluetooth
       Smart (Bluetooth Low Energy / BLE).
     * internal - the respective authenticator is contacted using a
       platform-specific transport. These authenticators are not removable
       from the platform.

    5.10.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)

typedef long COSEAlgorithmIdentifier;

   A COSEAlgorithmIdentifier's value is a number identifying a
   cryptographic algorithm. The algorithm identifiers SHOULD be values
   registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],
   for instance, -7 for "ES256" and -257 for "RS256".

    5.10.6. User Verification Requirement enumeration (enum
    UserVerificationRequirement)

enum UserVerificationRequirement {
    "required",
    "preferred",
    "discouraged"
};

   A Relying Party may require user verification for some of its
   operations but not for others, and may use this type to express its
   needs.

   The value required indicates that the Relying Party requires user
   verification for the operation and will fail the operation if the
   response does not have the UV flag set.

   The value preferred indicates that the Relying Party prefers user
   verification for the operation if possible, but will not fail the
   operation if the response does not have the UV flag set.

   The value discouraged indicates that the Relying Party does not want
   user verification employed during the operation (e.g., in the interest
   of minimizing disruption to the user interaction flow).

6. WebAuthn Authenticator Model

   The Web Authentication API implies a specific abstract functional model
   for an authenticator. This section describes that 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
   5 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
```

```
    "nfc",
    "ble"

};

   Authenticators may communicate with clients using a variety of
   transports. This enumeration defines a hint as to how clients might
   communicate with a particular authenticator in order to obtain an
   assertion for a specific credential. Note that these hints represent
   the Relying Party's best belief as to how an authenticator may be
   reached. A Relying Party may obtain a list of transports hints from
   some attestation statement formats or via some out-of-band mechanism;
   it is outside the scope of this specification to define that mechanism.
     * usb - the respective authenticator can be contacted over USB.

     * nfc - the respective authenticator can be contacted over Near Field
       Communication (NFC).
     * ble - the respective authenticator can be contacted over Bluetooth
       Smart (Bluetooth Low Energy / BLE).




    5.10.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)

typedef long COSEAlgorithmIdentifier;

   A COSEAlgorithmIdentifier's value is a number identifying a
   cryptographic algorithm. The algorithm identifiers SHOULD be values
   registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],
   for instance, -7 for "ES256" and -257 for "RS256".

    5.10.6. User Verification Requirement enumeration (enum
    UserVerificationRequirement)

enum UserVerificationRequirement {
    "required",
    "preferred",
    "discouraged"
};

   A Relying Party may require user verification for some of its
   operations but not for others, and may use this type to express its
   needs.

   The value required indicates that the Relying Party requires user
   verification for the operation and will fail the operation if the
   response does not have the UV flag set.

   The value preferred indicates that the Relying Party prefers user
   verification for the operation if possible, but will not fail the
   operation if the response does not have the UV flag set.

   The value discouraged indicates that the Relying Party does not want
   user verification employed during the operation (e.g., in the interest
   of minimizing disruption to the user interaction flow).

6. WebAuthn Authenticator Model

   The Web Authentication API implies a specific abstract functional model
   for an authenticator. This section describes that 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
   5 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 authenticators communicate with the client platform, unless they are necessary for interoperability with Relying Parties. For instance, this abstract model does not define protocols for connecting authenticators to clients over transports such as USB or NFC. Similarly, this abstract model does not define specific error codes or methods of returning them; however, it does define error behavior in terms of the needs of the client. Therefore, specific error codes are mentioned as a means of showing which error conditions must be distinguishable (or not) from each other in order to enable a compliant and secure client implementation.

Relying Parties may influence authenticator selection, if they deem necessary, by stipulating various authenticator characteristics when creating credentials and/or when generating assertions, through use of credential creation options or assertion generation options, respectively. The algorithms underlying the WebAuthn API marshal these options and pass them to the applicable authenticator operations defined below.

In this abstract model, the authenticator provides key management and cryptographic signatures. It can be embedded in the WebAuthn client or housed in a separate device entirely. The authenticator itself can contain a cryptographic module which operates at a higher security level than the rest of the authenticator. This is particularly important for authenticators that are embedded in the WebAuthn client, as in those cases this cryptographic module (which may, for example, be a TPM) could be considered more trustworthy than the rest of the authenticator.

Each authenticator stores a credentials map, a map from (rpId, [userHandle]) to public key credential source.

Additionally, each authenticator has an AAGUID, which is a 128-bit identifier indicating 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 high probability) from the AAGUIDs of all other types of authenticators. The AAGUID for a given type of authenticator SHOULD be randomly generated to ensure this. 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 observed and added at different levels of the stack as a signature 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 in two: Those added by the RP or the client, referred to as client data; and those added by the authenticator, referred to as the 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 the authenticator should have the flexibility to add contextual bindings as needed.
 * The design aims to reuse as much as possible of existing encoding

the Relying Party. However, it does not define the details of how authenticators communicate with the client platform, unless they are necessary for interoperability with Relying Parties. For instance, this abstract model does not define protocols for connecting authenticators to clients over transports such as USB or NFC. Similarly, this abstract model does not define specific error codes or methods of returning them; however, it does define error behavior in terms of the needs of the client. Therefore, specific error codes are mentioned as a means of showing which error conditions must be distinguishable (or not) from each other in order to enable a compliant and secure client implementation.

Relying Parties may influence authenticator selection, if they deem necessary, by stipulating various authenticator characteristics when creating credentials and/or when generating assertions, through use of credential creation options or assertion generation options, respectively. The algorithms underlying the WebAuthn API marshal these options and pass them to the applicable authenticator operations defined below.

In this abstract model, the authenticator provides key management and cryptographic signatures. It can be embedded in the WebAuthn client or housed in a separate device entirely. The authenticator itself can contain a cryptographic module which operates at a higher security level than the rest of the authenticator. This is particularly important for authenticators that are embedded in the WebAuthn client, as in those cases this cryptographic module (which may, for example, be a TPM) could be considered more trustworthy than the rest of the authenticator.

Each authenticator stores a credentials map, a map from (rpId, [userHandle]) to public key credential source.

Additionally, each authenticator has an AAGUID, which is a 128-bit identifier indicating 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 high probability) from the AAGUIDs of all other types of authenticators. The AAGUID for a given type of authenticator SHOULD be randomly generated to ensure this. 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 observed and added at different levels of the stack as a signature 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 in two: Those added by the RP or the client, referred to as client data; and those added by the authenticator, referred to as the 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 the 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. An attestation signature provides cryptographic proof of certain properties of 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 attestation 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 possessing a particular credential private key 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 desired since authenticators can be devices with limited 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.
     + 0 means the user is not present.
  * Bit 1: Reserved for future use (RFU1).
  * Bit 2: User Verified (UV) result.
     + 1 means the user is verified.
     + 0 means the user is not verified.
  * Bits 3-5: Reserved for future use (RFU2).
  * Bit 6: Attested credential data included (AT).
     + Indicates whether the authenticator added attested credential data.
  * Bit 7: Extension data included (ED).
     + Indicates if the authenticator data has extensions.

signCount 4 Signature counter, 32-bit unsigned big-endian integer.
attestedCredentialData variable (if present) attested credential data

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. An attestation signature provides cryptographic proof of certain properties of 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 attestation 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 possessing a particular credential private key 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 desired since authenticators can be devices with limited 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.
     + 0 means the user is not present.
  * Bit 1: Reserved for future use (RFU1).
  * Bit 2: User Verified (UV) result.
     + 1 means the user is verified.
     + 0 means the user is not verified.
  * Bits 3-5: Reserved for future use (RFU2).
  * Bit 6: Attested credential data included (AT).
     + Indicates whether the authenticator added attested credential data.
  * Bit 7: Extension data included (ED).
     + Indicates if the authenticator data has extensions.

signCount 4 Signature counter, 32-bit unsigned big-endian integer.
attestedCredentialData variable (if present) attested credential data

(if present). See 6.3.1 Attested credential data for details. Its length depends on the length of the credential ID and credential public key being attested.

extensions variable (if present) Extension-defined authenticator data. This is a CBOR [RFC7049] map with extension identifiers as keys, and authenticator extension outputs as values. See 9 WebAuthn Extensions for details.

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.

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 supplied by the client, and that the RP ID is a registrable 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 an authenticator specific gesture. The RFU bits SHALL be set to zero.

For attestation signatures, the authenticator MUST set the AT flag and include the attestedCredentialData. For authentication signatures, the AT flag MUST NOT be set and the attestedCredentialData 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.
[fido-signature-formats-figure1.svg] Authenticator data layout.

Note that the authenticator data describes its own length: If the AT and ED flags are not set, it is always 37 bytes long. The attested credential data (which is only present if the AT flag is set) describes its own length. If the ED flag is set, then the total length is 37 bytes plus the length of the attested credential data, plus the length of the CBOR map that follows.

6.1.1. Signature Counter Considerations

Authenticators MUST implement a signature counter feature. The signature counter is incremented for each successful authenticatorGetAssertion operation by some positive value, and its value is returned to the Relying Party within the authenticator data. The signature counter's purpose is to aid Relying Parties in detecting cloned authenticators. Clone detection is more important for authenticators with limited protection measures.

An Relying Party stores the signature counter of the most recent authenticatorGetAssertion operation. Upon a new authenticatorGetAssertion operation, the Relying Party compares the stored signature counter value with the new signCount value returned in the assertion's authenticator data. If this new signCount value is less than or equal to the stored value, a cloned authenticator may exist, or the authenticator may be malfunctioning.

Detecting a signature counter mismatch does not indicate whether the current operation was performed by a cloned authenticator or the original authenticator. Relying Parties should address this situation appropriately relative to their individual situations, i.e., their risk tolerance.

(if present). See 6.3.1 Attested credential data for details. Its length depends on the length of the credential ID and credential public key being attested.

extensions variable (if present) Extension-defined authenticator data. This is a CBOR [RFC7049] map with extension identifiers as keys, and authenticator extension outputs as values. See 9 WebAuthn Extensions for details.

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.

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 supplied by the client, and that the RP ID is a registrable 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 an authenticator specific gesture. The RFU bits SHALL be set to zero.

For attestation signatures, the authenticator MUST set the AT flag and include the attestedCredentialData. For authentication signatures, the AT flag MUST NOT be set and the attestedCredentialData 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.
[fido-signature-formats-figure1.svg] Authenticator data layout.

Note that the authenticator data describes its own length: If the AT and ED flags are not set, it is always 37 bytes long. The attested credential data (which is only present if the AT flag is set) describes its own length. If the ED flag is set, then the total length is 37 bytes plus the length of the attested credential data, plus the length of the CBOR map that follows.

6.1.1. Signature Counter Considerations

Authenticators MUST implement a signature counter feature. The signature counter is incremented for each successful authenticatorGetAssertion operation by some positive value, and its value is returned to the Relying Party within the authenticator data. The signature counter's purpose is to aid Relying Parties in detecting cloned authenticators. Clone detection is more important for authenticators with limited protection measures.

An Relying Party stores the signature counter of the most recent authenticatorGetAssertion operation. Upon a new authenticatorGetAssertion operation, the Relying Party compares the stored signature counter value with the new signCount value returned in the assertion's authenticator data. If this new signCount value is less than or equal to the stored value, a cloned authenticator may exist, or the authenticator may be malfunctioning.

Detecting a signature counter mismatch does not indicate whether the current operation was performed by a cloned authenticator or the original authenticator. Relying Parties should address this situation appropriately relative to their individual situations, i.e., their risk tolerance.

Authenticators:

- should implement per-RP ID signature counters. This prevents the signature counter value from being shared between Relying Parties and being possibly employed as a correlation handle for the user. Authenticators may implement a global signature counter, i.e., on a per-authenticator basis, but this is less privacy-friendly for users.
- should ensure that the signature counter value does not accidentally decrease (e.g., due to hardware failures).

6.1.2. FIDO U2F signature format compatibility

The format for assertion signatures, which sign over the concatenation of an authenticator data structure and the hash of the serialized client data, are compatible with the FIDO U2F authentication signature format (see Section 5.4 of [FIDO-U2F-Message-Formats]).

This is because the first 37 bytes of the signed data in a FIDO U2F authentication response message constitute a valid authenticator data structure, and the remaining 32 bytes are the hash of the serialized client data. In this authenticator data structure, the rpIdHash is the FIDO U2F application parameter, all flags except UP are always zero, and the attestedCredentialData and extensions are never present. FIDO U2F authentication signatures can therefore be verified by the same procedure as other assertion signatures generated by the authenticatorMakeCredential operation.

6.2. Authenticator operations

A WebAuthn 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.

6.2.1. Lookup Credential Source by Credential ID algorithm

The result of looking up a credential id credentialId in an authenticator authenticator is the result of the following algorithm:

1. If authenticator can decrypt credentialId into a public key credential source credSource:
   1. Set credSource.id to credentialId.
   2. Return credSource.
2. For each public key credential source credSource of authenticator's credentials map:
   1. If credSource.id is credentialId, return credSource.
3. Return null.

6.2.2. The authenticatorMakeCredential operation

It takes the following input parameters:

hash
: The hash of the serialized client data, provided by the client.

rpEntity
: The Relying Party's PublicKeyCredentialRpEntity.

userEntity
: The user account's PublicKeyCredentialUserEntity, containing the user handle given by the Relying Party.

requireResidentKey
: The authenticatorSelection.requireResidentKey value given by the Relying Party.

Authenticators:

- should implement per-RP ID signature counters. This prevents the signature counter value from being shared between Relying Parties and being possibly employed as a correlation handle for the user. Authenticators may implement a global signature counter, i.e., on a per-authenticator basis, but this is less privacy-friendly for users.
- should ensure that the signature counter value does not accidentally decrease (e.g., due to hardware failures).

6.1.2. FIDO U2F signature format compatibility

The format for assertion signatures, which sign over the concatenation of an authenticator data structure and the hash of the serialized client data, are compatible with the FIDO U2F authentication signature format (see Section 5.4 of [FIDO-U2F-Message-Formats]).

This is because the first 37 bytes of the signed data in a FIDO U2F authentication response message constitute a valid authenticator data structure, and the remaining 32 bytes are the hash of the serialized client data. In this authenticator data structure, the rpIdHash is the FIDO U2F application parameter, all flags except UP are always zero, and the attestedCredentialData and extensions are never present. FIDO U2F authentication signatures can therefore be verified by the same procedure as other assertion signatures generated by the authenticatorMakeCredential operation.

6.2. Authenticator operations

A WebAuthn 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.

6.2.1. Lookup Credential Source by Credential ID algorithm

The result of looking up a credential id credentialId in an authenticator authenticator is the result of the following algorithm:

1. If authenticator can decrypt credentialId into a public key credential source credSource:
   1. Set credSource.id to credentialId.
   2. Return credSource.
2. For each public key credential source credSource of authenticator's credentials map:
   1. If credSource.id is credentialId, return credSource.
3. Return null.

6.2.2. The authenticatorMakeCredential operation

It takes the following input parameters:

hash
: The hash of the serialized client data, provided by the client.

rpEntity
: The Relying Party's PublicKeyCredentialRpEntity.

userEntity
: The user account's PublicKeyCredentialUserEntity, containing the user handle given by the Relying Party.

requireResidentKey
: The authenticatorSelection.requireResidentKey value given by the Relying Party.

requireUserPresence
A Boolean value provided by the client, which in invocations from a WebAuthn Client's [[Create]](origin, options, sameOriginWithAncestors) method is always set to the inverse of requireUserVerification.

requireUserVerification
The effective user verification requirement for credential creation, a Boolean value provided by the client.

credTypesAndPubKeyAlgs
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.

excludeCredentialDescriptorList
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. excludeCredentialDescriptorList contains a list of known credentials.

extensions
A CBOR map from extension identifiers to their authenticator extension inputs, created by the client based on the extensions requested by the Relying Party, if any.

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

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. Check if at least one of the specified combinations of PublicKeyCredentialType and cryptographic parameters in credTypesAndPubKeyAlgs is supported. If not, return an error code equivalent to "NotSupportedError" and terminate the operation.
3. For each descriptor of excludeCredentialDescriptorList:
    1. If looking up descriptor.id in this authenticator returns non-null, and the returned item's RP ID and type match rpEntity.id and excludeCredentialDescriptorList.type respectively, then obtain user consent for creating a new credential. The method of obtaining user consent MUST include a test of user presence. If the user

        confirms consent to create a new credential
        return an error code equivalent to "InvalidStateError" and terminate the operation.

        does not consent to create a new credential
        return an error code equivalent to "NotAllowedError" and terminate the operation.

4. If requireResidentKey is true and the authenticator cannot store 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 rpEntity.id, rpEntity.name, userEntity.name and userEntity.displayName, if possible.
   If requireUserVerification is true, the method of obtaining user

/Users/jehodges/Documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.txt, Top line: 2835

requireUserPresence
A Boolean value provided by the client, which in invocations from a WebAuthn Client's [[Create]](origin, options, sameOriginWithAncestors) method is always set to the inverse of requireUserVerification.

requireUserVerification
The effective user verification requirement for credential creation, a Boolean value provided by the client.

credTypesAndPubKeyAlgs
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.

excludeCredentialDescriptorList
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. excludeCredentialDescriptorList contains a list of known credentials.

extensions
A CBOR map from extension identifiers to their authenticator extension inputs, created by the client based on the extensions requested by the Relying Party, if any.

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

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. Check if at least one of the specified combinations of PublicKeyCredentialType and cryptographic parameters in credTypesAndPubKeyAlgs is supported. If not, return an error code equivalent to "NotSupportedError" and terminate the operation.
3. For each descriptor of excludeCredentialDescriptorList:
    1. If looking up descriptor.id in this authenticator returns non-null, and the returned item's RP ID and type match rpEntity.id and excludeCredentialDescriptorList.type respectively, then obtain user consent for creating a new credential. The method of obtaining user consent MUST include a test of user presence. If the user

        confirms consent to create a new credential
        return an error code equivalent to "InvalidStateError" and terminate the operation.

        does not consent to create a new credential
        return an error code equivalent to "NotAllowedError" and terminate the operation.

4. If requireResidentKey is true and the authenticator cannot store 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 rpEntity.id, rpEntity.name, userEntity.name and userEntity.displayName, if possible.
   If requireUserVerification is true, the method of obtaining user

```
          consent MUST include user verification.
          If requireUserPresence is true, the method of obtaining user
          consent MUST include a test of user presence.
          If the user does not consent or if user verification fails, return
          an error code equivalent to "NotAllowedError" and terminate the
          operation.
       7. Once user consent has been obtained, generate a new credential
          object:
            1. Let (publicKey, privateKey) be a new pair of cryptographic
               keys using the combination of PublicKeyCredentialType and
               cryptographic parameters represented by the first item in
               credTypesAndPubKeyAlgs that is supported by this
               authenticator.
            2. Let userHandle be userEntity.id.
            3. Let credentialSource be a new public key credential source
               with the fields:

                type
                        public-key.

                privateKey
                        privateKey

                rpId
                        rpEntity.id

                userHandle
                        userHandle

                otherUI
                        Any other information the authenticator chooses to
                        include.

            4. If requireResidentKey is true or the authenticator chooses to
               create a Client-side-resident Credential Private Key:
                 1. Let credentialId be a new credential id.
                 2. Set credentialSource.id to credentialId.
                 3. Let credentials be this authenticator's credentials map.
                 4. Set credentials[(rpEntity.id, userHandle)] to
                    credentialSource.
            5. Otherwise:
                 1. Let credentialId be the result of serializing and
                    encrypting credentialSource so that only this
                    authenticator can decrypt it.
       8. If any error occurred while creating the new credential object,
          return an error code equivalent to "UnknownError" and terminate the
          operation.
       9. Let processedExtensions be the result of authenticator extension
          processing for each supported extension identifier -> authenticator
          extension input in extensions.
      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.

           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.
```

```
          consent MUST include user verification.
          If requireUserPresence is true, the method of obtaining user
          consent MUST include a test of user presence.
          If the user does not consent or if user verification fails, return
          an error code equivalent to "NotAllowedError" and terminate the
          operation.
       7. Once user consent has been obtained, generate a new credential
          object:
            1. Let (publicKey, privateKey) be a new pair of cryptographic
               keys using the combination of PublicKeyCredentialType and
               cryptographic parameters represented by the first item in
               credTypesAndPubKeyAlgs that is supported by this
               authenticator.
            2. Let userHandle be userEntity.id.
            3. Let credentialSource be a new public key credential source
               with the fields:

                type
                        public-key.

                privateKey
                        privateKey

                rpId
                        rpEntity.id

                userHandle
                        userHandle

                otherUI
                        Any other information the authenticator chooses to
                        include.

            4. If requireResidentKey is true or the authenticator chooses to
               create a Client-side-resident Credential Private Key:
                 1. Let credentialId be a new credential id.
                 2. Set credentialSource.id to credentialId.
                 3. Let credentials be this authenticator's credentials map.
                 4. Set credentials[(rpEntity.id, userHandle)] to
                    credentialSource.
            5. Otherwise:
                 1. Let credentialId be the result of serializing and
                    encrypting credentialSource so that only this
                    authenticator can decrypt it.
       8. If any error occurred while creating the new credential object,
          return an error code equivalent to "UnknownError" and terminate the
          operation.
       9. Let processedExtensions be the result of authenticator extension
          processing for each supported extension identifier -> authenticator
          extension input in extensions.
      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.

           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 procedure specified in 6.3.4 Generating an Attestation Object using an authenticator-chosen attestation statement format, authenticatorData, and hash. For more details on attestation, see 6.3 Attestation.

On successful completion of this operation, the authenticator returns the attestation object to the client.

6.2.3. The authenticatorGetAssertion operation

It takes the following input parameters:

rpId
    The caller's RP ID, as determined by the user agent and the client.

hash
    The hash of the serialized client data, provided by the client.

allowCredentialDescriptorList
    An optional list of PublicKeyCredentialDescriptors describing credentials acceptable to the Relying Party (possibly filtered by the client), if any.

requireUserPresence
    A Boolean value provided by the client, which in invocations from a WebAuthn Client's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method is always set to the inverse of requireUserVerification.

requireUserVerification
    The effective user verification requirement for assertion, a Boolean value provided by the client.

extensions
    A CBOR map from extension identifiers to their authenticator extension inputs, created by the client based on the extensions requested by the Relying Party, if any.

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 "UnknownError" and terminate the operation.
2. Let credentialOptions be a new empty set of public key credential sources.
3. If allowCredentialDescriptorList was supplied, then for each descriptor of allowCredentialDescriptorList:
    1. Let credSource be the result of looking up descriptor.id in this authenticator.
    2. If credSource is not null, append it to credentialOptions.
4. Otherwise (allowCredentialDescriptorList was not supplied), for each key -> credSource of this authenticator's credentials map, append credSource to credentialOptions.
5. Remove any items from credentialOptions whose rpId is not equal to rpId.
6. If credentialOptions is now empty, return an error code equivalent to "NotAllowedError" and terminate the operation.
7. Prompt the user to select a public key credential source selectedCredential from credentialOptions. Obtain user consent for using selectedCredential. The prompt for obtaining this consent may be shown by the authenticator if it has its own output capability, or by the user agent otherwise.
   If requireUserVerification is true, the method of obtaining user consent MUST include user verification.

13. Return the attestation object for the new credential created by the procedure specified in 6.3.4 Generating an Attestation Object using an authenticator-chosen attestation statement format, authenticatorData, and hash. For more details on attestation, see 6.3 Attestation.

On successful completion of this operation, the authenticator returns the attestation object to the client.

6.2.3. The authenticatorGetAssertion operation

It takes the following input parameters:

rpId
    The caller's RP ID, as determined by the user agent and the client.

hash
    The hash of the serialized client data, provided by the client.

allowCredentialDescriptorList
    An optional list of PublicKeyCredentialDescriptors describing credentials acceptable to the Relying Party (possibly filtered by the client), if any.

requireUserPresence
    A Boolean value provided by the client, which in invocations from a WebAuthn Client's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method is always set to the inverse of requireUserVerification.

requireUserVerification
    The effective user verification requirement for assertion, a Boolean value provided by the client.

extensions
    A CBOR map from extension identifiers to their authenticator extension inputs, created by the client based on the extensions requested by the Relying Party, if any.

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 "UnknownError" and terminate the operation.
2. Let credentialOptions be a new empty set of public key credential sources.
3. If allowCredentialDescriptorList was supplied, then for each descriptor of allowCredentialDescriptorList:
    1. Let credSource be the result of looking up descriptor.id in this authenticator.
    2. If credSource is not null, append it to credentialOptions.
4. Otherwise (allowCredentialDescriptorList was not supplied), for each key -> credSource of this authenticator's credentials map, append credSource to credentialOptions.
5. Remove any items from credentialOptions whose rpId is not equal to rpId.
6. If credentialOptions is now empty, return an error code equivalent to "NotAllowedError" and terminate the operation.
7. Prompt the user to select a public key credential source selectedCredential from credentialOptions. Obtain user consent for using selectedCredential. The prompt for obtaining this consent may be shown by the authenticator if it has its own output capability, or by the user agent otherwise.
   If requireUserVerification is true, the method of obtaining user consent MUST include user verification.

If requireUserPresence is true, the method of obtaining user consent MUST include a test of user presence. If the user does not consent, return an error code equivalent to "NotAllowedError" and terminate the operation.

8. Let processedExtensions be the result of authenticator extension processing for each supported extension identifier -> authenticator extension input in extensions.
9. Increment the RP ID-associated signature counter or the global signature counter value, depending on which approach is implemented by the authenticator, by some positive value.
10. Let authenticatorData be the byte array specified in 6.1 Authenticator data including processedExtensions, if any, as the extensions and excluding attestedCredentialData.
11. Let signature be the assertion signature of the concatenation authenticatorData || hash using the privateKey of selectedCredential as shown in Figure 2, below. A simple, undelimited concatenation is safe to use here because the authenticator data describes its own length. The hash of the serialized client data (which potentially has a variable length) is always the last element.
    [fido-signature-formats-figure2.svg] Generating an assertion signature.
12. If any error occurred while generating the assertion signature, return an error code equivalent to "UnknownError" and terminate the operation.
13. Return to the user agent:
    + selectedCredential.id, if either a list of credentials (i.e., allowCredentialDescriptorList) of length 2 or greater was supplied by the client, or no such list was supplied.
      Note: If, within allowCredentialDescriptorList, the client supplied exactly one credential and it was successfully employed, then its credential ID is not returned since the client already knows it. This saves transmitting these bytes over what may be a constrained connection in what is likely a common case.
    + authenticatorData
    + signature
    + selectedCredential.userHandle
      Note: the returned userHandle value may be null, see: userHandleResult.

If the authenticator cannot find any credential corresponding to the specified Relying Party that matches the specified criteria, it terminates the operation and returns an error.

6.2.4. The authenticatorCancel operation

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 authenticatorGetAssertion operation currently in progress.

6.3. Attestation

Authenticators MUST also provide some form of attestation. The basic requirement is that the authenticator can produce, for each credential 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

If requireUserPresence is true, the method of obtaining user consent MUST include a test of user presence. If the user does not consent, return an error code equivalent to "NotAllowedError" and terminate the operation.

8. Let processedExtensions be the result of authenticator extension processing for each supported extension identifier -> authenticator extension input in extensions.
9. Increment the RP ID-associated signature counter or the global signature counter value, depending on which approach is implemented by the authenticator, by some positive value.
10. Let authenticatorData be the byte array specified in 6.1 Authenticator data including processedExtensions, if any, as the extensions and excluding attestedCredentialData.
11. Let signature be the assertion signature of the concatenation authenticatorData || hash using the privateKey of selectedCredential as shown in Figure 2, below. A simple, undelimited concatenation is safe to use here because the authenticator data describes its own length. The hash of the serialized client data (which potentially has a variable length) is always the last element.
    [fido-signature-formats-figure2.svg] Generating an assertion signature.
12. If any error occurred while generating the assertion signature, return an error code equivalent to "UnknownError" and terminate the operation.
13. Return to the user agent:
    + selectedCredential.id, if either a list of credentials (i.e., allowCredentialDescriptorList) of length 2 or greater was supplied by the client, or no such list was supplied.
      Note: If, within allowCredentialDescriptorList, the client supplied exactly one credential and it was successfully employed, then its credential ID is not returned since the client already knows it. This saves transmitting these bytes over what may be a constrained connection in what is likely a common case.
    + authenticatorData
    + signature
    + selectedCredential.userHandle
      Note: the returned userHandle value may be null, see: userHandleResult.

If the authenticator cannot find any credential corresponding to the specified Relying Party that matches the specified criteria, it terminates the operation and returns an error.

6.2.4. The authenticatorCancel operation

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 authenticatorGetAssertion operation currently in progress.

6.3. Attestation

Authenticators MUST also provide some form of attestation. The basic requirement is that the authenticator can produce, for each credential 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 attested credential data) and the attestation statement is illustrated in figure 3, below.

[fido-attestation-structures.svg] Attestation object layout illustrating the included authenticator data (containing attested credential data) and the attestation statement.

This figure illustrates only the packed attestation statement format. Several additional attestation statement formats are defined in 8 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 itself 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 attestation statement formats. This specification supports a variety of such formats in an extensible way, as defined in 6.3.2 Attestation Statement 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 6.3.3 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 8.2 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 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.

6.3.1. Attested credential data

Attested credential data is a variable-length byte array added to the authenticator data when generating an attestation object for a given credential. It has the following format:

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 attested credential data) and the attestation statement is illustrated in figure 3, below.

[fido-attestation-structures.svg] Attestation object layout illustrating the included authenticator data (containing attested credential data) and the attestation statement.

This figure illustrates only the packed attestation statement format. Several additional attestation statement formats are defined in 8 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 itself 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 attestation statement formats. This specification supports a variety of such formats in an extensible way, as defined in 6.3.2 Attestation Statement 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 6.3.3 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 8.2 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 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.

6.3.1. Attested credential data

Attested credential data is a variable-length byte array added to the authenticator data when generating an attestation object for a given credential. It has the following format:

Name Length (in bytes) Description
aaguid 16 The AAGUID of the authenticator.
credentialIdLength 2 Byte length L of Credential ID, 16-bit unsigned big-endian integer.
credentialId L Credential ID
credentialPublicKey variable The credential public key encoded in COSE_Key format, as defined in Section 7 of [RFC8152], using the CTAP2 canonical CBOR encoding form. The COSE_Key-encoded credential public key MUST contain the optional "alg" parameter and MUST NOT contain any other optional parameters. The "alg" parameter MUST contain a COSEAlgorithmIdentifier value. The encoded credential public key MUST also contain any additional required parameters stipulated by the relevant key type specification, i.e., required for the key type "kty" and algorithm "alg" (see Section 8 of [RFC8152]).

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 attested credential data.

6.3.1.1. Examples of credentialPublicKey Values encoded in COSE_Key format

This section provides examples of COSE_Key-encoded Elliptic Curve and RSA public keys for the ES256, PS256, and RS256 signature algorithms. These examples adhere to the rules defined above for the credentialPublicKey value, and are presented in [CDDL] for clarity.

[RFC8152] Section 7 defines the general framework for all COSE_Key-encoded keys. Specific key types for specific algorithms are defined in other sections of [RFC8152] as well as in other specifications, as noted below.

Below is an example of a COSE_Key-encoded Elliptic Curve public key in EC2 format (see [RFC8152] Section 13.1), on the P-256 curve, to be used with the ES256 signature algorithm (ECDSA w/ SHA-256, see [RFC8152] Section 8.1):

```
{
  1:   2,  ; kty: EC2 key type
  3:  -7,  ; alg: ES256 signature algorithm
 -1:   1,  ; crv: P-256 curve
 -2:   x,  ; x-coordinate as byte string 32 bytes in length
           ; e.g., in hex: 65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108d
e439c08551d
 -3:   y   ; y-coordinate as byte string 32 bytes in length
           ; e.g., in hex: 1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9e
ecd0084d19c
}
```

Below is the above Elliptic Curve public key encoded in the CTAP2 canonical CBOR encoding form, whitespace and line breaks are included here for clarity and to match the [CDDL] presentation above:

```
A5
   01  02

   03  26

   20  01

   21  58 20   65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108de439c08551d

   22  58 20   1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9eecd0084d19c
```

Below is an example of a COSE_Key-encoded 2048-bit RSA public key (see [RFC8230] Section 4), to be used with the PS256 signature algorithm (RSASSA-PSS with SHA-256, see [RFC8230] Section 2):

```
{
  1:   3,  ; kty: RSA key type
  3: -37,  ; alg: PS256
 -1:   n,  ; n:   RSA modulus n byte string 256 bytes in length
           ;      e.g., in hex (middle bytes elided for brevity): DB5F651550...6
```

Name Length (in bytes) Description
aaguid 16 The AAGUID of the authenticator.
credentialIdLength 2 Byte length L of Credential ID, 16-bit unsigned big-endian integer.
credentialId L Credential ID
credentialPublicKey variable The credential public key encoded in COSE_Key format, as defined in Section 7 of [RFC8152], using the CTAP2 canonical CBOR encoding form. The COSE_Key-encoded credential public key MUST contain the optional "alg" parameter and MUST NOT contain any other optional parameters. The "alg" parameter MUST contain a COSEAlgorithmIdentifier value. The encoded credential public key MUST also contain any additional required parameters stipulated by the relevant key type specification, i.e., required for the key type "kty" and algorithm "alg" (see Section 8 of [RFC8152]).

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 attested credential data.

6.3.1.1. Examples of credentialPublicKey Values encoded in COSE_Key format

This section provides examples of COSE_Key-encoded Elliptic Curve and RSA public keys for the ES256, PS256, and RS256 signature algorithms. These examples adhere to the rules defined above for the credentialPublicKey value, and are presented in [CDDL] for clarity.

[RFC8152] Section 7 defines the general framework for all COSE_Key-encoded keys. Specific key types for specific algorithms are defined in other sections of [RFC8152] as well as in other specifications, as noted below.

Below is an example of a COSE_Key-encoded Elliptic Curve public key in EC2 format (see [RFC8152] Section 13.1), on the P-256 curve, to be used with the ES256 signature algorithm (ECDSA w/ SHA-256, see [RFC8152] Section 8.1):

```
{
  1:   2,  ; kty: EC2 key type
  3:  -7,  ; alg: ES256 signature algorithm
 -1:   1,  ; crv: P-256 curve
 -2:   x,  ; x-coordinate as byte string 32 bytes in length
           ; e.g., in hex: 65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108d
e439c08551d
 -3:   y   ; y-coordinate as byte string 32 bytes in length
           ; e.g., in hex: 1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9e
ecd0084d19c
}
```

Below is the above Elliptic Curve public key encoded in the CTAP2 canonical CBOR encoding form, whitespace and line breaks are included here for clarity and to match the [CDDL] presentation above:

```
A5
   01  02

   03  26

   20  01

   21  58 20   65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108de439c08551d

   22  58 20   1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9eecd0084d19c
```

Below is an example of a COSE_Key-encoded 2048-bit RSA public key (see [RFC8230] Section 4), to be used with the PS256 signature algorithm (RSASSA-PSS with SHA-256, see [RFC8230] Section 2):

```
{
  1:   3,  ; kty: RSA key type
  3: -37,  ; alg: PS256
 -1:   n,  ; n:   RSA modulus n byte string 256 bytes in length
           ;      e.g., in hex (middle bytes elided for brevity): DB5F651550...6
DC6548ACC3
 -2:   e   ; e:   RSA public exponent e byte string 3 bytes in length
          ;     e.g., in hex: 010001
}
```

Below is an example of the same COSE_Key-encoded RSA public key as above, to be used with the RS256 signature algorithm (RSASSA-PKCS1-v1_5 with SHA-256, see 11.3 COSE Algorithm Registrations):

```
{
  1:   3,  ; kty: RSA key type
  3:-257,  ; alg: RS256
 -1:   n,  ; n:   RSA modulus n byte string 256 bytes in length
          ;     e.g., in hex (middle bytes elided for brevity): DB5F651550...6
DC6548ACC3
 -2:   e   ; e:   RSA public exponent e byte string 3 bytes in length
          ;     e.g., in hex: 010001
}
```

6.3.2. 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:
* Supported attestation types:
* Syntax: The syntax of an attestation statement produced in this format, defined using [CDDL] for the extension point $attStmtFormat defined in 6.3.4 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 the following verification procedure inputs:
    + attStmt: The attestation statement structure
    + authenticatorData: The authenticator data claimed to have been used for the attestation
    + clientDataHash: The hash of the serialized client data

  The procedure returns either:
    + An error indicating that the attestation is invalid, or
    + The attestation type, and the trust path. This attestation trust path is either empty (in case of self attestation), an identifier of an ECDAA-Issuer public key (in the case of ECDAA), or a set of X.509 certificates.

The initial list of specified attestation statement formats is in 8 Defined Attestation Statement Formats.

6.3.3. Attestation Types

WebAuthn supports multiple attestation types:

Basic Attestation (Basic)
: In the case of basic attestation [UAFProtocol], 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 14.1 Attestation Privacy for further information.

Self Attestation (Self)
: 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 credential private key to create the attestation signature. Authenticators without meaningful protection measures for an attestation private key typically use this attestation type.

Attestation CA (AttCA)

DC6548ACC3
 -2:   e   ; e: RSA public exponent e byte string 3 bytes in length
          ;   e.g., in hex: 010001
}

Below is an example of the same COSE_Key-encoded RSA public key as above, to be used with the RS256 signature algorithm (RSASSA-PKCS1-v1_5 with SHA-256, see 11.3 COSE Algorithm Registrations):

{
  1:   3, ; kty: RSA key type
  3:-257, ; alg: RS256
 -1:   n, ; n: RSA modulus n byte string 256 bytes in length
          ;   e.g., in hex (middle bytes elided for brevity): DB5F651550...6
DC6548ACC3
 -2:   e   ; e: RSA public exponent e byte string 3 bytes in length
          ;   e.g., in hex: 010001
}

6.3.2. 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:
* Supported attestation types:
* Syntax: The syntax of an attestation statement produced in this format, defined using [CDDL] for the extension point $attStmtFormat defined in 6.3.4 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 the following verification procedure inputs:
    + attStmt: The attestation statement structure
    + authenticatorData: The authenticator data claimed to have been used for the attestation
    + clientDataHash: The hash of the serialized client data

  The procedure returns either:
    + An error indicating that the attestation is invalid, or
    + The attestation type, and the trust path. This attestation trust path is either empty (in case of self attestation), an identifier of an ECDAA-Issuer public key (in the case of ECDAA), or a set of X.509 certificates.

The initial list of specified attestation statement formats is in 8 Defined Attestation Statement Formats.

6.3.3. Attestation Types

WebAuthn supports multiple attestation types:

Basic Attestation (Basic)
In the case of basic attestation [UAFProtocol], 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 14.1 Attestation Privacy for further information.

Self Attestation (Self)
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 credential private key to create the attestation signature. Authenticators without meaningful protection measures for an attestation private key typically use this attestation type.

Attestation CA (AttCA)

In this case, an authenticator is based on a Trusted Platform Module (TPM) and holds an authenticator-specific "endorsement key" (EK). This key is used to securely communicate with a trusted third party, the Attestation CA [TCG-CMCProfile-AIKCertEnroll] (formerly known as a "Privacy CA"). The authenticator can generate multiple attestation identity key pairs (AIK) and requests an Attestation CA to issue an AIK certificate for each. Using this approach, such an authenticator can limit the exposure of the EK (which is a global correlation handle) to Attestation CA(s). AIKs can be requested for each authenticator-generated public key credential individually, and conveyed to Relying Parties as attestation certificates.

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]).

No attestation statement (None)
In this case, no attestation information is available.

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: { * tstr => any } ; Map is filled in by each
concrete attStmtType
                  )

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

6.3.5. Signature Formats for Packed Attestation, FIDO U2F Attestation, and Assertion Signatures

* For COSEAlgorithmIdentifier -7 (ES256), and other ECDSA-based

In this case, an authenticator is based on a Trusted Platform Module (TPM) and holds an authenticator-specific "endorsement key" (EK). This key is used to securely communicate with a trusted third party, the Attestation CA [TCG-CMCProfile-AIKCertEnroll] (formerly known as a "Privacy CA"). The authenticator can generate multiple attestation identity key pairs (AIK) and requests an Attestation CA to issue an AIK certificate for each. Using this approach, such an authenticator can limit the exposure of the EK (which is a global correlation handle) to Attestation CA(s). AIKs can be requested for each authenticator-generated public key credential individually, and conveyed to Relying Parties as attestation certificates.

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]).

No attestation statement (None)
In this case, no attestation information is available.

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: { * tstr => any } ; Map is filled in by each
concrete attStmtType
                  )

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

6.3.5. Signature Formats for Packed Attestation, FIDO U2F Attestation, and Assertion Signatures

* For COSEAlgorithmIdentifier -7 (ES256), and other ECDSA-based

algorithms, a signature value is encoded as an ASN.1 DER Ecdsa-Sig-Value, as defined in [RFC3279] section 2.2.3.

```
 Example:
 30 44                                ; SEQUENCE (68 Bytes)
     02 20                            ; INTEGER (32 Bytes)
     |  3d 46 28 7b 8c 6e 8c 8c  26 1c 1b 88 f2 73 b0 9a
     |  32 a6 cf 28 09 fd 6e 30  d5 a7 9f 26 37 00 8f 54
     02 20                            ; INTEGER (32 Bytes)
     |  4e 72 23 6e a3 90 a9 a1  7b cf 5f 7a 09 d6 3a b2
     |  17 6c 92 bb 8e 36 c0 41  98 a2 7b 90 9b 6e 8f 13
```

Note: As CTAP1/U2F devices are already producing signatures values in this format, CTAP2 devices will also produce signatures values in the same format, for consistency reasons. It is recommended that any new attestation formats defined not use ASN.1 encodings, but instead represent signatures as equivalent fixed-length byte arrays without internal structure, using the same representations as used by COSE signatures as defined in [RFC8152] and [RFC8230].

* For COSEAlgorithmIdentifier -257 (RS256), sig contains the signature generated using the RSASSA-PKCS1-v1_5 signature scheme defined in section 8.2.1 in [RFC8017] with SHA-256 as the hash function. The signature is not ASN.1 wrapped.
* For COSEAlgorithmIdentifier -37 (PS256), sig contains the signature generated using the RSASSA-PSS signature scheme defined in section 8.1.1 in [RFC8017] with SHA-256 as the hash function. The signature is not ASN.1 wrapped.

7. Relying Party Operations

Upon successful execution of create() or get(), the Relying Party's script receives 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 an AuthenticatorAttestationResponse structure response and an AuthenticationExtensionsClientOutputs structure clientExtensionResults, as part of a registration ceremony, a Relying Party MUST proceed as follows:

1. Let JSONtext be the result of running UTF-8 decode on the value of response.clientDataJSON.
   Note: Using any implementation of UTF-8 decode is acceptable as long as it yields the same result as that yielded by the UTF-8 decode algorithm. In particular, any leading byte order mark (BOM) MUST be stripped.
2. Let C, the client data claimed as collected during the credential creation, be the result of running an implementation-specific JSON parser on JSONtext.
   Note: C may be any implementation-specific data structure representation, as long as C's components are referenceable, as required by this algorithm.
3. Verify that the value of C.type is webauthn.create.
4. Verify that the value of C.challenge matches the challenge that was sent to the authenticator in the create() call.
5. Verify that the value of C.origin matches the Relying Party's origin.
6. Verify that the value of C.tokenBinding.status matches the state of Token Binding for the TLS connection over which the assertion was obtained. If Token Binding was used on that TLS connection, also verify that C.tokenBinding.id matches the base64url encoding of the Token Binding ID for the connection.
7. Compute the hash of response.clientDataJSON using SHA-256.
8. Perform CBOR decoding on the attestationObject field of the AuthenticatorAttestationResponse structure to obtain the

algorithms, a signature value is encoded as an ASN.1 DER Ecdsa-Sig-Value, as defined in [RFC3279] section 2.2.3.

 Example:
 30 44   ; SEQUENCE (68 Bytes)
     02 20   ; INTEGER (32 Bytes)
     | 3d 46 28 7b 8c 6e 8c 8c 26 1c 1b 88 f2 73 b0 9a
     | 32 a6 cf 28 09 fd 6e 30 d5 a7 9f 26 37 00 8f 54
     02 20   ; INTEGER (32 Bytes)
     | 4e 72 23 6e a3 90 a9 a1 7b cf 5f 7a 09 d6 3a b2
     | 17 6c 92 bb 8e 36 c0 41 98 a2 7b 90 9b 6e 8f 13

Note: As CTAP1/U2F devices are already producing signatures values in this format, CTAP2 devices will also produce signatures values in the same format, for consistency reasons. It is recommended that any new attestation formats defined not use ASN.1 encodings, but instead represent signatures as equivalent fixed-length byte arrays without internal structure, using the same representations as used by COSE signatures as defined in [RFC8152] and [RFC8230].

* For COSEAlgorithmIdentifier -257 (RS256), sig contains the signature generated using the RSASSA-PKCS1-v1_5 signature scheme defined in section 8.2.1 in [RFC8017] with SHA-256 as the hash function. The signature is not ASN.1 wrapped.
* For COSEAlgorithmIdentifier -37 (PS256), sig contains the signature generated using the RSASSA-PSS signature scheme defined in section 8.1.1 in [RFC8017] with SHA-256 as the hash function. The signature is not ASN.1 wrapped.

7. Relying Party Operations

Upon successful execution of create() or get(), the Relying Party's script receives 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 an AuthenticatorAttestationResponse structure response and an AuthenticationExtensionsClientOutputs structure clientExtensionResults, as part of a registration ceremony, a Relying Party MUST proceed as follows:

1. Let JSONtext be the result of running UTF-8 decode on the value of response.clientDataJSON.
   Note: Using any implementation of UTF-8 decode is acceptable as long as it yields the same result as that yielded by the UTF-8 decode algorithm. In particular, any leading byte order mark (BOM) MUST be stripped.
2. Let C, the client data claimed as collected during the credential creation, be the result of running an implementation-specific JSON parser on JSONtext.
   Note: C may be any implementation-specific data structure representation, as long as C's components are referenceable, as required by this algorithm.
3. Verify that the value of C.type is webauthn.create.
4. Verify that the value of C.challenge matches the challenge that was sent to the authenticator in the create() call.
5. Verify that the value of C.origin matches the Relying Party's origin.
6. Verify that the value of C.tokenBinding.status matches the state of Token Binding for the TLS connection over which the assertion was obtained. If Token Binding was used on that TLS connection, also verify that C.tokenBinding.id matches the base64url encoding of the Token Binding ID for the connection.
7. Compute the hash of response.clientDataJSON using SHA-256.
8. 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.

9. Verify that the RP ID hash in authData is indeed the SHA-256 hash of the RP ID expected by the RP.
10. Verify that the User Present bit of the flags in authData is set.
11. If user verification is required for this registration, verify that the User Verified bit of the flags in authData is set.
12. Verify that the values of the client extension outputs in clientExtensionResults and the authenticator extension outputs in the extensions in authData are as expected, considering the client extension input values that were given as the extensions option in the create() call. In particular, any extension identifier values in the clientExtensionResults and the extensions in authData MUST be also be present as extension identifier values in the extensions member of options, i.e., no extensions are present that were not requested. In the general case, the meaning of "are as expected" is specific to the Relying Party and which extensions are in use.
    Note: Since all extensions are OPTIONAL for both the client and the authenticator, the Relying Party MUST be prepared to handle cases where none or not all of the requested extensions were acted upon.
13. Determine the attestation statement format by performing a USASCII case-sensitive match on fmt against the set of supported WebAuthn Attestation Statement Format Identifier values. The up-to-date list of registered WebAuthn Attestation Statement Format Identifier values is maintained in the in the IANA registry of the same name [WebAuthn-Registries].
14. Verify that attStmt is a correct attestation statement, conveying a valid attestation signature, by using the attestation statement format fmt's verification procedure given attStmt, authData and the hash of the serialized client data computed in step 7.
    Note: Each attestation statement format specifies its own verification procedure. See 8 Defined Attestation Statement Formats for the initially-defined formats, and [WebAuthn-Registries] for the up-to-date list.
15. If validation is successful, obtain a list of acceptable trust anchors (attestation root certificates or ECDAA-Issuer public keys) for that attestation type and attestation statement format fmt, 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.
16. Assess the attestation trustworthiness using the outputs of the verification procedure in step 14, as follows:
    + If self attestation was used, check if self attestation is acceptable under Relying Party policy.
    + If ECDAA was used, verify that the identifier of the ECDAA-Issuer public key used is included in the set of acceptable trust anchors obtained in step 15.
    + Otherwise, use the X.509 certificates returned by the verification procedure to verify that the attestation public key correctly chains up to an acceptable root certificate.
17. Check that the credentialId is not yet registered to any other user. If registration is requested for a credential that is already registered to a different user, the Relying Party SHOULD fail this registration ceremony, or it MAY decide to accept the registration, e.g. while deleting the older registration.
18. If the attestation statement attStmt verified successfully and is found to be trustworthy, then register the new credential with the account that was denoted in the options.user passed to create(), by associating it with the credentialId and credentialPublicKey in the attestedCredentialData in authData, as appropriate for the Relying Party's system.
19. If the attestation statement attStmt successfully verified but is not trustworthy per step 16 above, the Relying Party SHOULD fail the registration ceremony.
    NOTE: However, if permitted by policy, the Relying Party MAY register the credential ID and credential public key but treat the credential as one with self attestation (see 6.3.3 Attestation Types). If doing so, the Relying Party is asserting there is no cryptographic proof that the public key credential has been

attestation statement format fmt, the authenticator data authData, and the attestation statement attStmt.

9. Verify that the RP ID hash in authData is indeed the SHA-256 hash of the RP ID expected by the RP.
10. Verify that the User Present bit of the flags in authData is set.
11. If user verification is required for this registration, verify that the User Verified bit of the flags in authData is set.
12. Verify that the values of the client extension outputs in clientExtensionResults and the authenticator extension outputs in the extensions in authData are as expected, considering the client extension input values that were given as the extensions option in the create() call. In particular, any extension identifier values in the clientExtensionResults and the extensions in authData MUST be also be present as extension identifier values in the extensions member of options, i.e., no extensions are present that were not requested. In the general case, the meaning of "are as expected" is specific to the Relying Party and which extensions are in use.
    Note: Since all extensions are OPTIONAL for both the client and the authenticator, the Relying Party MUST be prepared to handle cases where none or not all of the requested extensions were acted upon.
13. Determine the attestation statement format by performing a USASCII case-sensitive match on fmt against the set of supported WebAuthn Attestation Statement Format Identifier values. The up-to-date list of registered WebAuthn Attestation Statement Format Identifier values is maintained in the in the IANA registry of the same name [WebAuthn-Registries].
14. Verify that attStmt is a correct attestation statement, conveying a valid attestation signature, by using the attestation statement format fmt's verification procedure given attStmt, authData and the hash of the serialized client data computed in step 7.
    Note: Each attestation statement format specifies its own verification procedure. See 8 Defined Attestation Statement Formats for the initially-defined formats, and [WebAuthn-Registries] for the up-to-date list.
15. If validation is successful, obtain a list of acceptable trust anchors (attestation root certificates or ECDAA-Issuer public keys) for that attestation type and attestation statement format fmt, 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.
16. Assess the attestation trustworthiness using the outputs of the verification procedure in step 14, as follows:
    + If self attestation was used, check if self attestation is acceptable under Relying Party policy.
    + If ECDAA was used, verify that the identifier of the ECDAA-Issuer public key used is included in the set of acceptable trust anchors obtained in step 15.
    + Otherwise, use the X.509 certificates returned by the verification procedure to verify that the attestation public key correctly chains up to an acceptable root certificate.
17. Check that the credentialId is not yet registered to any other user. If registration is requested for a credential that is already registered to a different user, the Relying Party SHOULD fail this registration ceremony, or it MAY decide to accept the registration, e.g. while deleting the older registration.
18. If the attestation statement attStmt verified successfully and is found to be trustworthy, then register the new credential with the account that was denoted in the options.user passed to create(), by associating it with the credentialId and credentialPublicKey in the attestedCredentialData in authData, as appropriate for the Relying Party's system.
19. If the attestation statement attStmt successfully verified but is not trustworthy per step 16 above, the Relying Party SHOULD fail the registration ceremony.
    NOTE: However, if permitted by policy, the Relying Party MAY register the credential ID and credential public key but treat the credential as one with self attestation (see 6.3.3 Attestation Types). If doing so, the Relying Party is asserting there is no cryptographic proof that the public key credential has been

generated by a particular authenticator model. See [FIDOSecRef] and [UAFProtocol] for a more detailed discussion.

Verification of attestation objects requires that the Relying Party has a trusted method of determining acceptable trust anchors in step 15 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.

7.2. Verifying an authentication assertion

When verifying a given PublicKeyCredential structure (credential) and an AuthenticationExtensionsClientOutputs structure clientExtensionResults, as part of an authentication ceremony, the Relying Party MUST proceed as follows:

1. If the allowCredentials option was given when this authentication ceremony was initiated, verify that credential.id identifies one of the public key credentials that were listed in allowCredentials.
2. If credential.response.userHandle is present, verify that the user identified by this value is the owner of the public key credential identified by credential.id.
3. 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.
4. Let cData, authData and sig denote the value of credential's response's clientDataJSON, authenticatorData, and signature respectively.
5. Let JSONtext be the result of running UTF-8 decode on the value of cData.
   Note: Using any implementation of UTF-8 decode is acceptable as long as it yields the same result as that yielded by the UTF-8 decode algorithm. In particular, any leading byte order mark (BOM) MUST be stripped.
6. Let C, the client data claimed as used for the signature, be the result of running an implementation-specific JSON parser on JSONtext.
   Note: C may be any implementation-specific data structure representation, as long as C's components are referenceable, as required by this algorithm.
7. Verify that the value of C.type is the string webauthn.get.
8. Verify that the value of C.challenge matches the challenge that was sent to the authenticator in the PublicKeyCredentialRequestOptions passed to the get() call.
9. Verify that the value of C.origin matches the Relying Party's origin.
10. Verify that the value of C.tokenBinding.status matches the state of Token Binding for the TLS connection over which the attestation was obtained. If Token Binding was used on that TLS connection, also verify that C.tokenBinding.id matches the base64url encoding of the Token Binding ID for the connection.
11. Verify that the rpIdHash in authData is the SHA-256 hash of the RP ID expected by the Relying Party.
12. Verify that the User Present bit of the flags in authData is set.
13. If user verification is required for this assertion, verify that the User Verified bit of the flags in authData is set.
14. Verify that the values of the client extension outputs in clientExtensionResults and the authenticator extension outputs in the extensions in authData are as expected, considering the client extension input values that were given as the extensions option in the get() call. In particular, any extension identifier values in the clientExtensionResults and the extensions in authData MUST be also be present as extension identifier values in the extensions member of options, i.e., no extensions are present that were not requested. In the general case, the meaning of "are as expected" is specific to the Relying Party and which extensions are in use.
    Note: Since all extensions are OPTIONAL for both the client and the authenticator, the Relying Party MUST be prepared to handle cases where none or not all of the requested extensions were acted upon.

generated by a particular authenticator model. See [FIDOSecRef] and [UAFProtocol] for a more detailed discussion.

Verification of attestation objects requires that the Relying Party has a trusted method of determining acceptable trust anchors in step 15 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.

7.2. Verifying an authentication assertion

When verifying a given PublicKeyCredential structure (credential) and an AuthenticationExtensionsClientOutputs structure clientExtensionResults, as part of an authentication ceremony, the Relying Party MUST proceed as follows:

1. If the allowCredentials option was given when this authentication ceremony was initiated, verify that credential.id identifies one of the public key credentials that were listed in allowCredentials.
2. If credential.response.userHandle is present, verify that the user identified by this value is the owner of the public key credential identified by credential.id.
3. 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.
4. Let cData, authData and sig denote the value of credential's response's clientDataJSON, authenticatorData, and signature respectively.
5. Let JSONtext be the result of running UTF-8 decode on the value of cData.
   Note: Using any implementation of UTF-8 decode is acceptable as long as it yields the same result as that yielded by the UTF-8 decode algorithm. In particular, any leading byte order mark (BOM) MUST be stripped.
6. Let C, the client data claimed as used for the signature, be the result of running an implementation-specific JSON parser on JSONtext.
   Note: C may be any implementation-specific data structure representation, as long as C's components are referenceable, as required by this algorithm.
7. Verify that the value of C.type is the string webauthn.get.
8. Verify that the value of C.challenge matches the challenge that was sent to the authenticator in the PublicKeyCredentialRequestOptions passed to the get() call.
9. Verify that the value of C.origin matches the Relying Party's origin.
10. Verify that the value of C.tokenBinding.status matches the state of Token Binding for the TLS connection over which the attestation was obtained. If Token Binding was used on that TLS connection, also verify that C.tokenBinding.id matches the base64url encoding of the Token Binding ID for the connection.
11. Verify that the rpIdHash in authData is the SHA-256 hash of the RP ID expected by the Relying Party.
12. Verify that the User Present bit of the flags in authData is set.
13. If user verification is required for this assertion, verify that the User Verified bit of the flags in authData is set.
14. Verify that the values of the client extension outputs in clientExtensionResults and the authenticator extension outputs in the extensions in authData are as expected, considering the client extension input values that were given as the extensions option in the get() call. In particular, any extension identifier values in the clientExtensionResults and the extensions in authData MUST be also be present as extension identifier values in the extensions member of options, i.e., no extensions are present that were not requested. In the general case, the meaning of "are as expected" is specific to the Relying Party and which extensions are in use.
    Note: Since all extensions are OPTIONAL for both the client and the authenticator, the Relying Party MUST be prepared to handle cases where none or not all of the requested extensions were acted upon.

15. Let hash be the result of computing a hash over the cData using SHA-256.
16. Using the credential public key looked up in step 3, verify that sig is a valid signature over the binary concatenation of authData and hash.
    Note: This verification step is compatible with signatures generated by FIDO U2F authenticators. See 6.1.2 FIDO U2F signature format compatibility.
17. If the signature counter value authData.signCount is nonzero or the value stored in conjunction with credential's id attribute is nonzero, then run the following sub-step:
    + If the signature counter value authData.signCount is

      greater than the signature counter value stored in conjunction with credential's id attribute.
      Update the stored signature counter value, associated with credential's id attribute, to be the value of authData.signCount.

      less than or equal to the signature counter value stored in conjunction with credential's id attribute.
      This is a signal that the authenticator may be cloned, i.e. at least two copies of the credential private key may exist and are being used in parallel. Relying Parties should incorporate this information into their risk scoring. Whether the Relying Party updates the stored signature counter value in this case, or not, or fails the authentication ceremony or not, is Relying Party-specific.

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

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 an attestation statement format identifier, chosen by the author of the attestation statement format.

Attestation statement format identifiers SHOULD be registered per [WebAuthn-Registries] "Registries 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 [RFC5234] but without %x22 and %x5c.

Note: This means attestation statement format identifiers based on domain names MUST incorporate only LDH Labels [RFC5890].

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.

15. Let hash be the result of computing a hash over the cData using SHA-256.
16. Using the credential public key looked up in step 3, verify that sig is a valid signature over the binary concatenation of authData and hash.
    Note: This verification step is compatible with signatures generated by FIDO U2F authenticators. See 6.1.2 FIDO U2F signature format compatibility.
17. If the signature counter value authData.signCount is nonzero or the value stored in conjunction with credential's id attribute is nonzero, then run the following sub-step:
    + If the signature counter value authData.signCount is

      greater than the signature counter value stored in conjunction with credential's id attribute.
      Update the stored signature counter value, associated with credential's id attribute, to be the value of authData.signCount.

      less than or equal to the signature counter value stored in conjunction with credential's id attribute.
      This is a signal that the authenticator may be cloned, i.e. at least two copies of the credential private key may exist and are being used in parallel. Relying Parties should incorporate this information into their risk scoring. Whether the Relying Party updates the stored signature counter value in this case, or not, or fails the authentication ceremony or not, is Relying Party-specific.

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

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 an attestation statement format identifier, chosen by the author of the attestation statement format.

Attestation statement format identifiers SHOULD be registered per [WebAuthn-Registries] "Registries 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 [RFC5234] but without %x22 and %x5c.

Note: This means attestation statement format identifiers based on domain names MUST incorporate only LDH Labels [RFC5890].

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].

8.2. 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:

```
 $$attStmtType //= (
                       fmt: "packed",
                       attStmt: packedStmtFormat
                   )

 packedStmtFormat = {
                        alg: COSEAlgorithmIdentifier,
                        sig: bytes,
                        x5c: [ attestnCert: bytes, * (caCert: bytes) ]
                    } //
                    {
                        alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261
for ED512)
                        sig: bytes,
                        ecdaaKeyId: bytes
                    } //
                    {
                        alg: COSEAlgorithmIdentifier
                        sig: bytes,
                    }
```

The semantics of the fields are as follows:

alg
A COSEAlgorithmIdentifier containing the identifier of the algorithm used to generate the attestation signature.

sig
A byte string containing the attestation signature.

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 BigNumberToB encoding of the component "c" of the ECDAA-Issuer public key as defined section 3.3, step 3.5 in [FIDOEcdaaAlgorithm].

Signing procedure
The signing procedure for this attestation statement format is similar to the procedure for generating assertion signatures.

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

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].

8.2. 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:

 $$attStmtType //= (
                       fmt: "packed",
                       attStmt: packedStmtFormat
)

 packedStmtFormat = {
                        alg: COSEAlgorithmIdentifier,
                        sig: bytes,
                        x5c: [attestnCert: bytes, * (caCert: bytes)]
                    } //
                    {
                        alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261
for ED512)
                        sig: bytes,
                        ecdaaKeyId: bytes
                    } //
                    {
                        alg: COSEAlgorithmIdentifier
                        sig: bytes,
                    }

The semantics of the fields are as follows:

alg
A COSEAlgorithmIdentifier containing the identifier of the algorithm used to generate the attestation signature.

sig
A byte string containing the attestation signature.

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 BigNumberToB encoding of the component "c" of the ECDAA-Issuer public key as defined section 3.3, step 3.5 in [FIDOEcdaaAlgorithm].

Signing procedure
The signing procedure for this attestation statement format is similar to the procedure for generating assertion signatures.

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

```
              serialized client data.
           2. If Basic or AttCA attestation is in use, the authenticator
              produces the sig by concatenating authenticatorData and
              clientDataHash, and signing the result using an attestation
              private key selected through an authenticator-specific
              mechanism. It sets x5c to the certificate chain of the
              attestation public key and alg to the algorithm of the
              attestation private key.
           3. If ECDAA is in use, the authenticator produces sig by
              concatenating authenticatorData and clientDataHash, and
              signing the result using ECDAA-Sign (see section 3.5 of
              [FIDOEcdaaAlgorithm]) after selecting an ECDAA-Issuer public
              key related to the ECDAA signature private key through an
              authenticator-specific mechanism (see [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
              not ECDAA. In this case:
                 o Verify that sig is a valid signature over the
                   concatenation of authenticatorData and clientDataHash
                   using the attestation public key in x5c with the
                   algorithm specified in alg.
                 o Verify that x5c meets the requirements in 8.2.1 Packed
                   attestation statement certificate requirements.
                 o 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.
                 o If successful, return attestation type Basic and
                   attestation trust path x5c.
           3. If ecdaaKeyId is present, then the attestation type is ECDAA.
              In this case:
                 o Verify that sig is a valid signature over the
                   concatenation of authenticatorData and clientDataHash
                   using ECDAA-Verify with ECDAA-Issuer public key
                   identified by ecdaaKeyId (see [FIDOEcdaaAlgorithm]).
                 o If successful, return attestation type ECDAA and
                   attestation trust path ecdaaKeyId.
           4. If neither x5c nor ecdaaKeyId is present, self attestation is
              in use.
                 o Validate that alg matches the algorithm of the
                   credentialPublicKey in authenticatorData.
                 o Verify that sig is a valid signature over the
                   concatenation of authenticatorData and clientDataHash
                   using the credential public key with alg.
                 o If successful, return attestation type Self and empty
                   attestation trust path.

    8.2.1. Packed attestation statement certificate requirements

   The attestation certificate MUST have the following fields/extensions:
     * Version MUST be set to 3 (which is indicated by an ASN.1 INTEGER
       with value 2).
     * Subject field MUST be set to:

```

```
              serialized client data.
           2. If Basic or AttCA attestation is in use, the authenticator
              produces the sig by concatenating authenticatorData and
              clientDataHash, and signing the result using an attestation
              private key selected through an authenticator-specific
              mechanism. It sets x5c to the certificate chain of the
              attestation public key and alg to the algorithm of the
              attestation private key.
           3. If ECDAA is in use, the authenticator produces sig by
              concatenating authenticatorData and clientDataHash, and
              signing the result using ECDAA-Sign (see section 3.5 of
              [FIDOEcdaaAlgorithm]) after selecting an ECDAA-Issuer public
              key related to the ECDAA signature private key through an
              authenticator-specific mechanism (see [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
              not ECDAA. In this case:
                 o Verify that sig is a valid signature over the
                   concatenation of authenticatorData and clientDataHash
                   using the attestation public key in x5c with the
                   algorithm specified in alg.
                 o Verify that x5c meets the requirements in 8.2.1 Packed
                   attestation statement certificate requirements.
                 o 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.
                 o If successful, return attestation type Basic and
                   attestation trust path x5c.
           3. If ecdaaKeyId is present, then the attestation type is ECDAA.
              In this case:
                 o Verify that sig is a valid signature over the
                   concatenation of authenticatorData and clientDataHash
                   using ECDAA-Verify with ECDAA-Issuer public key
                   identified by ecdaaKeyId (see [FIDOEcdaaAlgorithm]).
                 o If successful, return attestation type ECDAA and
                   attestation trust path ecdaaKeyId.
           4. If neither x5c nor ecdaaKeyId is present, self attestation is
              in use.
                 o Validate that alg matches the algorithm of the
                   credentialPublicKey in authenticatorData.
                 o Verify that sig is a valid signature over the
                   concatenation of authenticatorData and clientDataHash
                   using the credential public key with alg.
                 o If successful, return attestation type Self and empty
                   attestation trust path.

    8.2.1. Packed attestation statement certificate requirements

   The attestation certificate MUST have the following fields/extensions:
     * Version MUST be set to 3 (which is indicated by an ASN.1 INTEGER
       with value 2).
     * Subject field MUST be set to:

            Subject-C
                   ISO 3166 code specifying the country where the
                   Authenticator vendor is incorporated (PrintableString)

            Subject-O
                   Legal name of the Authenticator vendor (UTF8String)

            Subject-OU
                   Literal string "Authenticator Attestation" (UTF8String)

            Subject-CN
                   A UTF8String of the vendor's choosing

         * 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 a
           16-byte OCTET STRING. The extension MUST NOT be marked as critical.
           Note that an X.509 Extension encodes the DER-encoding of the value
           in an OCTET STRING. Thus, the AAGUID must be wrapped in two OCTET
           STRINGS to be valid. Here is a sample, encoded Extension structure:
    30 21                                    -- SEQUENCE
      06 0b 2b 06 01 04 01 82 e5 1c 01 01 04  -- 1.3.6.1.4.1.45724.1.1.4
      04 12                                  -- OCTET STRING
        04 10                                -- OCTET STRING
          cd 8c 39 5c 26 ed ee de            -- AAGUID
          65 3b 00 79 7d 03 ca 3c

         * 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 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
              AttCA, ECDAA

       Syntax
              The syntax of a TPM Attestation statement is as follows:

        $$attStmtType // = (
                              fmt: "tpm",
                              attStmt: tpmStmtFormat
                          )

        tpmStmtFormat = {
                            ver: "2.0",
                            (
                                alg: COSEAlgorithmIdentifier,
                                x5c: [ aikCert: bytes, * (caCert: bytes) ]
                            ) //
                            (
                                alg:  COSEAlgorithmIdentifier, (-260 for ED256 / -26
    1 for ED512)
                                ecdaaKeyId: bytes
                            ),
                            sig: bytes,
                            certInfo: bytes,
                            pubArea: bytes
                        }
```

/Users/jehodges/Documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.txt, Top line: 3815

```
            Subject-C
                   ISO 3166 code specifying the country where the
                   Authenticator vendor is incorporated (PrintableString)

            Subject-O
                   Legal name of the Authenticator vendor (UTF8String)

            Subject-OU
                   Literal string "Authenticator Attestation" (UTF8String)

            Subject-CN
                   A UTF8String of the vendor's choosing

         * 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 a
           16-byte OCTET STRING. The extension MUST NOT be marked as critical.
           Note that an X.509 Extension encodes the DER-encoding of the value
           in an OCTET STRING. Thus, the AAGUID must be wrapped in two OCTET
           STRINGS to be valid. Here is a sample, encoded Extension structure:
    30 21                                    -- SEQUENCE
      06 0b 2b 06 01 04 01 82 e5 1c 01 01 04  -- 1.3.6.1.4.1.45724.1.1.4
      04 12                                  -- OCTET STRING
        04 10                                -- OCTET STRING
          cd 8c 39 5c 26 ed ee de            -- AAGUID
          65 3b 00 79 7d 03 ca 3c

         * 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 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
              AttCA, ECDAA

       Syntax
              The syntax of a TPM Attestation statement is as follows:

        $$attStmtType // = (
                              fmt: "tpm",
                              attStmt: tpmStmtFormat
                          )

        tpmStmtFormat = {
                            ver: "2.0",
                            (
                                alg: COSEAlgorithmIdentifier,
                                x5c: [ aikCert: bytes, * (caCert: bytes) ]
                            ) //
                            (
                                alg:  COSEAlgorithmIdentifier, (-260 for ED256 / -26
    1 for ED512)
                                ecdaaKeyId: bytes
                            ),
                            sig: bytes,
                            certInfo: bytes,
                            pubArea: bytes
                        }
```

The semantics of the above fields are as follows:

ver
The version of the TPM specification to which the signature conforms.

alg
A COSEAlgorithmIdentifier containing the identifier of the algorithm used to generate the attestation signature.

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 BigNumberToB encoding of the component "c" as defined section 3.3, step 3.5 in [FIDOEcdaaAlgorithm].

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 extraData parameter to the digest of attToBeSigned using the hash algorithm corresponding to the "alg" signature algorithm. (For the "RS256" algorithm, this would be a SHA-256 digest.)

Set the pubArea field to 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
Given the verification procedure inputs attStmt, authenticatorData 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 the public key specified by the parameters and unique fields of pubArea is identical to the credentialPublicKey in the attestedCredentialData in authenticatorData.

Concatenate authenticatorData and clientDataHash to form attToBeSigned.

Validate that certInfo is valid:

The semantics of the above fields are as follows:

ver
The version of the TPM specification to which the signature conforms.

alg
A COSEAlgorithmIdentifier containing the identifier of the algorithm used to generate the attestation signature.

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 BigNumberToB encoding of the component "c" as defined section 3.3, step 3.5 in [FIDOEcdaaAlgorithm].

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 extraData parameter to the digest of attToBeSigned using the hash algorithm corresponding to the "alg" signature algorithm. (For the "RS256" algorithm, this would be a SHA-256 digest.)

Set the pubArea field to 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
Given the verification procedure inputs attStmt, authenticatorData 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 the public key specified by the parameters and unique fields of pubArea is identical to the credentialPublicKey in the attestedCredentialData 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_CERTIFY.
          + Verify that extraData is set to the hash of attToBeSigned
            using the hash algorithm employed in "alg".
          + Verify that attested contains a TPMS_CERTIFY_INFO structure as
            specified in [TPMv2-Part2] section 10.12.3, whose name field
            contains a valid Name for pubArea, as computed using the
            algorithm in the nameAlg field of pubArea using the procedure
            specified in [TPMv2-Part1] section 16.
          + Note that the remaining fields in the "Standard Attestation
            Structure" [TPMv2-Part1] section 31.2, i.e., qualifiedSigner,
            clockInfo and firmwareVersion are ignored. These fields MAY be
            used as an input to risk engines.

          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 1 4
            (id-fido-gen-ce-aaguid) verify that the value of this
            extension matches the aaguid in authenticatorData.
          + If successful, return attestation type AttCA 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 [FIDOEcdaaAlgorithm]).
          + 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/extensions:
       * 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-Profile] section 3.2.9.
       * The Extended Key Usage extension MUST contain the
         "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8)
         tcg-kp-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].

   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 running 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
```

```
          + Verify that magic is set to TPM_GENERATED_VALUE.
          + Verify that type is set to TPM_ST_ATTEST_CERTIFY.
          + Verify that extraData is set to the hash of attToBeSigned
            using the hash algorithm employed in "alg".
          + Verify that attested contains a TPMS_CERTIFY_INFO structure as
            specified in [TPMv2-Part2] section 10.12.3, whose name field
            contains a valid Name for pubArea, as computed using the
            algorithm in the nameAlg field of pubArea using the procedure
            specified in [TPMv2-Part1] section 16.
          + Note that the remaining fields in the "Standard Attestation
            Structure" [TPMv2-Part1] section 31.2, i.e., qualifiedSigner,
            clockInfo and firmwareVersion are ignored. These fields MAY be
            used as an input to risk engines.

          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 1 4
            (id-fido-gen-ce-aaguid) verify that the value of this
            extension matches the aaguid in authenticatorData.
          + If successful, return attestation type AttCA 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 [FIDOEcdaaAlgorithm]).
          + 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/extensions:
       * 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-Profile] section 3.2.9.
       * The Extended Key Usage extension MUST contain the
         "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8)
         tcg-kp-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].

   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 running 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

Syntax

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:

```
 $$attStmtType //= (
                      fmt: "android-key",
                      attStmt: androidStmtFormat
                  )

 androidStmtFormat = {
                      alg: COSEAlgorithmIdentifier,
                      sig: bytes,
                      x5c: [ credCert: bytes, * (caCert: bytes) ]
                    }
```

Signing procedure

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

Request an Android Key Attestation by calling keyStore.getCertificateChain(myKeyUUID) providing clientDataHash as the challenge value (e.g., by using setAttestationChallenge). Set x5c to the returned value.

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.

Verification procedure

Given the verification procedure inputs attStmt, authenticatorData 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 sig is a valid signature over the concatenation of authenticatorData and clientDataHash using the public key in the first certificate in x5c with the algorithm specified in alg.
+ Verify that the public key in the first certificate in in x5c 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 clientDataHash.
    o The AuthorizationList.allApplications field is not present, since PublicKeyCredential must be bound to the RP ID.
    o The value in the AuthorizationList.origin field is equal to KM_TAG_GENERATED.
    o The value in the AuthorizationList.purpose field is equal to KM_PURPOSE_SIGN.
+ If successful, return attestation type Basic with the attestation trust path set to x5c.

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

Basic

Syntax

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:

```
 $$attStmtType //= (
                      fmt: "android-key",
                      attStmt: androidStmtFormat
                  )

 androidStmtFormat = {
                      alg: COSEAlgorithmIdentifier,
                      sig: bytes,
                      x5c: [ credCert: bytes, * (caCert: bytes) ]
                    }
```

Signing procedure

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

Request an Android Key Attestation by calling keyStore.getCertificateChain(myKeyUUID) providing clientDataHash as the challenge value (e.g., by using setAttestationChallenge). Set x5c to the returned value.

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.

Verification procedure

Given the verification procedure inputs attStmt, authenticatorData 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 sig is a valid signature over the concatenation of authenticatorData and clientDataHash using the public key in the first certificate in x5c with the algorithm specified in alg.
+ Verify that the public key in the first certificate in in x5c 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 clientDataHash.
    o The AuthorizationList.allApplications field is not present, since PublicKeyCredential must be bound to the RP ID.
    o The value in the AuthorizationList.origin field is equal to KM_TAG_GENERATED.
    o The value in the AuthorizationList.purpose field is equal to KM_PURPOSE_SIGN.
+ If successful, return attestation type Basic with the attestation trust path set to x5c.

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:

       $$attStmtType //= (
                             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.

      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, perform
             SHA-256 hash of the concatenated string, and let the result of
             the hash 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,
             authenticatorData 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
               SHA-256 hash of the concatenation of authenticatorData and
               clientDataHash.
             + Verify that the attestation certificate is issued to the
               hostname "attest.android.com" (see SafetyNet online
               documentation).
             + Verify that the ctsProfileMatch attribute in the payload of
               response is true.
             + If successful, return attestation type Basic with the
```

```
      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:

       $$attStmtType //= (
                             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.

      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, perform
             SHA-256 hash of the concatenated string, and let the result of
             the hash 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,
             authenticatorData 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
               SHA-256 hash of the concatenation of authenticatorData and
               clientDataHash.
             + Verify that the attestation certificate is issued to the
               hostname "attest.android.com" (see SafetyNet online
               documentation).
             + Verify that the ctsProfileMatch attribute in the payload of
               response is true.
             + If successful, return attestation type Basic with the
```

attestation trust path set to the above attestation certificate.

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, AttCA

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

```
 $$attStmtType //= (
                   fmt: "fido-u2f",
                   attStmt: u2fStmtFormat
               )

 u2fStmtFormat = {
                 x5c: [ attestnCert: bytes ],
                 sig: bytes
             }
```

The semantics of the above fields are as follows:

x5c
A single element array containing the attestation certificate in X.509 format.

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 authenticatorData denote the authenticator data for the attestation, and let clientDataHash denote the hash of the serialized client data. (Since SHA-256 is used to hash the serialized client data, clientDataHash will be 32 bytes long.)

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 clientDataHash, 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.

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. Check that x5c has exactly one element and let attCert be that element. Let certificate public key be the public key conveyed by attCert. If certificate public key is not an Elliptic Curve

attestation trust path set to the above attestation certificate.

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, AttCA

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

 $$attStmtType //= (
                   fmt: "fido-u2f",
                   attStmt: u2fStmtFormat
)

 u2fStmtFormat = {
                 x5c: [attestnCert: bytes],
                 sig: bytes
             }

The semantics of the above fields are as follows:

x5c
A single element array containing the attestation certificate in X.509 format.

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 authenticatorData denote the authenticator data for the attestation, and let clientDataHash denote the hash of the serialized client data. (Since SHA-256 is used to hash the serialized client data, clientDataHash will be 32 bytes long.)

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 clientDataHash, 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.

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. Check that x5c has exactly one element and let attCert be that element. Let certificate public key be the public key conveyed by attCert. If certificate public key is not an Elliptic Curve

```
          (EC) public key over the P-256 curve, terminate this algorithm
          and return an appropriate error.
       3. Extract the claimed rpIdHash from authenticatorData, and the
          claimed credentialId and credentialPublicKey from
          authenticatorData.attestedCredentialData.
       4. Convert the COSE_KEY formatted credentialPublicKey (see
          Section 7 of [RFC8152]) to Raw ANSI X9.62 public key format
          (see ALG_KEY_ECC_X962_RAW in Section 3.6.2 Public Key
          Representation Formats of [FIDO-Registry]).
            o Let x be the value corresponding to the "-2" key
              (representing x coordinate) in credentialPublicKey, and
              confirm its size to be of 32 bytes. If size differs or
              "-2" key is not found, terminate this algorithm and
              return an appropriate error.
            o Let y be the value corresponding to the "-3" key
              (representing y coordinate) in credentialPublicKey, and
              confirm its size to be of 32 bytes. If size differs or
              "-3" key is not found, terminate this algorithm and
              return an appropriate error.
            o Let publicKeyU2F be the concatenation 0x04 || x || y.
              Note: This signifies uncompressed ECC key format.
       5. Let verificationData be the concatenation of (0x00 || rpIdHash
          || clientDataHash || credentialId || publicKeyU2F) (see
          Section 4.3 of [FIDO-U2F-Message-Formats]).
       6. Verify the sig using verificationData and certificate public
          key per [SEC1].
       7. If successful, return attestation type Basic with the
          attestation trust path set to x5c.

  8.7. None Attestation Statement Format

   The none attestation statement format is used to replace any
   authenticator-provided attestation statement when a Relying Party
   indicates it does not wish to receive attestation information, see
   5.4.6 Attestation Conveyance Preference enumeration (enum
   AttestationConveyancePreference).

   Attestation statement format identifier
          none

   Attestation types supported
          None

   Syntax
          The syntax of a none attestation statement is defined as
          follows:

    $$attStmtType //= (
                          fmt: "none",
                          attStmt: emptyMap
                      )

    emptyMap = {}

   Signing procedure
          Return the fixed attestation statement defined above.

   Verification procedure
          Return attestation type None with an empty trust path.

9. WebAuthn Extensions

   The mechanism for generating public key credentials, as well as
   requesting and generating Authentication assertions, as defined in 5
   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
```

```
          (EC) public key over the P-256 curve, terminate this algorithm
          and return an appropriate error.
       3. Extract the claimed rpIdHash from authenticatorData, and the
          claimed credentialId and credentialPublicKey from
          authenticatorData.attestedCredentialData.
       4. Convert the COSE_KEY formatted credentialPublicKey (see
          Section 7 of [RFC8152]) to Raw ANSI X9.62 public key format
          (see ALG_KEY_ECC_X962_RAW in Section 3.6.2 Public Key
          Representation Formats of [FIDO-Registry]).
            o Let x be the value corresponding to the "-2" key
              (representing x coordinate) in credentialPublicKey, and
              confirm its size to be of 32 bytes. If size differs or
              "-2" key is not found, terminate this algorithm and
              return an appropriate error.
            o Let y be the value corresponding to the "-3" key
              (representing y coordinate) in credentialPublicKey, and
              confirm its size to be of 32 bytes. If size differs or
              "-3" key is not found, terminate this algorithm and
              return an appropriate error.
            o Let publicKeyU2F be the concatenation 0x04 || x || y.
              Note: This signifies uncompressed ECC key format.
       5. Let verificationData be the concatenation of (0x00 || rpIdHash
          || clientDataHash || credentialId || publicKeyU2F) (see
          Section 4.3 of [FIDO-U2F-Message-Formats]).
       6. Verify the sig using verificationData and certificate public
          key per [SEC1].
       7. If successful, return attestation type Basic with the
          attestation trust path set to x5c.

  8.7. None Attestation Statement Format

   The none attestation statement format is used to replace any
   authenticator-provided attestation statement when a Relying Party
   indicates it does not wish to receive attestation information, see
   5.4.6 Attestation Conveyance Preference enumeration (enum
   AttestationConveyancePreference).

   Attestation statement format identifier
          none

   Attestation types supported
          None

   Syntax
          The syntax of a none attestation statement is defined as
          follows:

    $$attStmtType //= (
                          fmt: "none",
                          attStmt: emptyMap
                      )

    emptyMap = {}

   Signing procedure
          Return the fixed attestation statement defined above.

   Verification procedure
          Return attestation type None with an empty trust path.

9. WebAuthn Extensions

   The mechanism for generating public key credentials, as well as
   requesting and generating Authentication assertions, as defined in 5
   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.
* 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 requested 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 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 extension request parameters and response values for registration extensions.
* authenticatorGetAssertion extension request parameters and response values for authentication extensions.
* Authenticator extension processing for registration extensions and authentication extensions.

For authenticator extensions, as part of the client extension processing, the client also creates the CBOR authenticator extension input value for each extension (often based on the corresponding client extension input value), and passes them to the authenticator in the create() call (for registration extensions) or the get() call (for authentication extensions). These authenticator extension input values are represented in CBOR and passed as name-value pairs, with the extension identifier as the name, and the corresponding authenticator extension input 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 in CBOR. All WebAuthn extensions MUST be defined in such a way that this implementation choice does not endanger the user's security or privacy. For instance, if an extension requires client processing, it could be defined in a manner that ensures such a nave pass-through will produce a semantically invalid authenticator extension input value, resulting in the extension being ignored by the authenticator. Since all extensions are OPTIONAL, this will not cause a functional failure in the API operation. Likewise, clients can choose to produce a client extension output value for an extension that it does not understand by encoding the authenticator extension output value into JSON, provided that the CBOR output uses only types present in JSON.

extensions define the following steps and data:

* navigator.credentials.create() extension request parameters and response values for registration 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 requested 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 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 extension request parameters and response values for registration extensions.
* authenticatorGetAssertion extension request parameters and response values for authentication extensions.
* Authenticator extension processing for registration extensions and authentication extensions.

For authenticator extensions, as part of the client extension processing, the client also creates the CBOR authenticator extension input value for each extension (often based on the corresponding client extension input value), and passes them to the authenticator in the create() call (for registration extensions) or the get() call (for authentication extensions). These authenticator extension input values are represented in CBOR and passed as name-value pairs, with the extension identifier as the name, and the corresponding authenticator extension input 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 in CBOR. All WebAuthn extensions MUST be defined in such a way that this implementation choice does not endanger the user's security or privacy. For instance, if an extension requires client processing, it could be defined in a manner that ensures such a nave pass-through will produce a semantically invalid authenticator extension input value, resulting in the extension being ignored by the authenticator. Since all extensions are OPTIONAL, this will not cause a functional failure in the API operation. Likewise, clients can choose to produce a client extension output value for an extension that it does not understand by encoding the authenticator extension output value into JSON, provided that the CBOR output uses only types present in JSON.

When clients choose to pass through extensions they do not recognize, the JavaScript values in the client extension inputs are converted to CBOR values in the authenticator extension inputs. When the JavaScript value is an %ArrayBuffer%, it is converted to a CBOR byte array. When the JavaScript value is a non-integer number, it is converted to a 64-bit CBOR floating point number. Otherwise, when the JavaScript type corresponds to a JSON type, the conversion is done using the rules defined in Section 4.2 of [RFC7049] (Converting from JSON to CBOR), but operating on inputs of JavaScript type values rather than inputs of JSON type values. Once these conversions are done, canonicalization of the resulting CBOR MUST be performed using the CTAP2 canonical CBOR encoding form.

Likewise, when clients receive outputs from extensions they have passed through that they do not recognize, the CBOR values in the authenticator extension outputs are converted to JavaScript values in the client extension outputs. When the CBOR value is a byte string, it is converted to a JavaScript %ArrayBuffer% (rather than a base64url-encoded string). Otherwise, when the CBOR type corresponds to a JSON type, the conversion is done using the rules defined in Section 4.1 of [RFC7049] (Converting from CBOR to JSON), but producing outputs of JavaScript type values rather than outputs of JSON type values.

Note that some clients may choose to implement this pass-through capability under a feature flag. Supporting this capability can facilitate innovation, allowing authenticators to experiment with new extensions and Relying Parties to use them before there is explicit support for them in clients.

The IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registries] can 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, 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 [RFC5234] 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

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

When clients choose to pass through extensions they do not recognize, the JavaScript values in the client extension inputs are converted to CBOR values in the authenticator extension inputs. When the JavaScript value is an %ArrayBuffer%, it is converted to a CBOR byte array. When the JavaScript value is a non-integer number, it is converted to a 64-bit CBOR floating point number. Otherwise, when the JavaScript type corresponds to a JSON type, the conversion is done using the rules defined in Section 4.2 of [RFC7049] (Converting from JSON to CBOR), but operating on inputs of JavaScript type values rather than inputs of JSON type values. Once these conversions are done, canonicalization of the resulting CBOR MUST be performed using the CTAP2 canonical CBOR encoding form.

Likewise, when clients receive outputs from extensions they have passed through that they do not recognize, the CBOR values in the authenticator extension outputs are converted to JavaScript values in the client extension outputs. When the CBOR value is a byte string, it is converted to a JavaScript %ArrayBuffer% (rather than a base64url-encoded string). Otherwise, when the CBOR type corresponds to a JSON type, the conversion is done using the rules defined in Section 4.1 of [RFC7049] (Converting from CBOR to JSON), but producing outputs of JavaScript type values rather than outputs of JSON type values.

Note that some clients may choose to implement this pass-through capability under a feature flag. Supporting this capability can facilitate innovation, allowing authenticators to experiment with new extensions and Relying Parties to use them before there is explicit support for them in clients.

The IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registries] can 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, 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 [RFC5234] 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

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 extension input.

```
var assertionPromise = navigator.credentials.get({
    publicKey: {
        // The challenge must be produced by the server, see the Security Consid
erations
        challenge: new Uint8Array([4,99,22 /* 29 more random bytes generated by
the server */]),
        extensions: {
            "webauthnExample_foobar": 42
        }
    }
});
```

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 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 value set to true, 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 the extension is used as an input to this client processing. For each supported client extension, the client adds an entry to the clientExtensions map with the extension identifier as the key, and the extension's client extension input as

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 extension input.

var assertionPromise = navigator.credentials.get({
    publicKey: {
        // The challenge must be produced by the server, see the Security Consid
erations
        challenge: new Uint8Array([4,99,22 /* 29 more random bytes generated by
the server */]),
        extensions: {
            "webauthnExample_foobar": 42
        }
    }
});

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 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 value set to true, 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 the extension is used as an input to this client processing. For each supported client extension, the client adds an entry to the clientExtensions map 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 result of getClientExtensionResults() with extension identifiers as keys, and the client extension output value of each 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 parameter of the authenticatorMakeCredential and authenticatorGetAssertion operations. The extensions parameter is a CBOR map where each key is an extension identifier and the corresponding value is the authenticator extension input for that extension.

Likewise, the extension output is represented in the extensions part of the authenticator data. The extensions part of the authenticator data is a CBOR map where each key is an extension identifier and the corresponding value is the authenticator extension output for that extension.

For each supported extension, the authenticator extension processing rule for that extension is used create the authenticator extension output from the authenticator extension input and possibly also other inputs.

10. Defined Extensions

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.

10.1. FIDO AppID Extension (appid)

This client extension allows Relying Parties that have previously registered a credential using the legacy FIDO JavaScript APIs to request an assertion. The FIDO APIs use an alternative identifier for relying parties called an AppID [FIDO-APPID], and any credentials created using those APIs will be bound to that identifier. Without this extension, they would need to be re-registered in order to be bound to an RP ID.

This extension does not allow FIDO-compatible credentials to be created. Thus, credentials created with WebAuthn are not backwards compatible with the FIDO JavaScript APIs.

Extension identifier
      appid

Client extension input
      A single USVString specifying a FIDO AppID.

```
partial dictionary AuthenticationExtensionsClientInputs {
  USVString appid;
};
```

Client extension processing

1. If present in a create() call, return a "NotSupportedError" DOMException--this extension is only valid when requesting an assertion.

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 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 parameter of the authenticatorMakeCredential and authenticatorGetAssertion operations. The extensions parameter is a CBOR map where each key is an extension identifier and the corresponding value is the authenticator extension input for that extension.

Likewise, the extension output is represented in the extensions part of the authenticator data. The extensions part of the authenticator data is a CBOR map where each key is an extension identifier and the corresponding value is the authenticator extension output for that extension.

For each supported extension, the authenticator extension processing rule for that extension is used create the authenticator extension output from the authenticator extension input and possibly also other inputs.

10. Defined Extensions

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.

10.1. FIDO AppID Extension (appid)

This client extension allows Relying Parties that have previously registered a credential using the legacy FIDO JavaScript APIs to request an assertion. The FIDO APIs use an alternative identifier for relying parties called an AppID [FIDO-APPID], and any credentials created using those APIs will be bound to that identifier. Without this extension, they would need to be re-registered in order to be bound to an RP ID.

This extension does not allow FIDO-compatible credentials to be created. Thus, credentials created with WebAuthn are not backwards compatible with the FIDO JavaScript APIs.

Extension identifier
      appid

Client extension input
      A single USVString specifying a FIDO AppID.

partial dictionary AuthenticationExtensionsClientInputs {
  USVString appid;
};

Client extension processing

1. If present in a create() call, return a "NotSupportedError" DOMException--this extension is only valid when requesting an assertion.

2. Let facetId be the result of passing the caller's origin to the FIDO algorithm for determining the FacetID of a calling application.
3. Let appId be the extension input.
4. Pass facetId and appId to the FIDO algorithm for determining if a caller's FacetID is authorized for an AppID. If that algorithm rejects appId then return a "SecurityError" DOMException.
5. When building allowCredentialDescriptorList, if a U2F authenticator indicates that a credential is inapplicable (i.e. by returning SW_WRONG_DATA) then the client MUST retry with the U2F application parameter set to the SHA-256 hash of appId. If this results in an applicable credential, the client MUST include the credential in allowCredentialDescriptorList. The value of appId then replaces the rpId parameter of authenticatorGetAssertion.

Client extension output
Returns the value true to indicate to the RP that the extension was acted upon.

```
partial dictionary AuthenticationExtensionsClientOutputs {
  boolean appid;
};
```

Authenticator extension input
None.

Authenticator extension processing
None.

Authenticator extension output
None.

10.2. Simple Transaction Authorization Extension (txAuthSimple)

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 USVString prompt.

```
partial dictionary AuthenticationExtensionsClientInputs {
  USVString txAuthSimple;
};
```

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

Client extension output
Returns the authenticator extension output string UTF-8 decoded into a USVString.

```
partial dictionary AuthenticationExtensionsClientOutputs {
  USVString txAuthSimple;
};
```

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

```
CDDL:
txAuthSimpleInput = (tstr)
```

2. Let facetId be the result of passing the caller's origin to the FIDO algorithm for determining the FacetID of a calling application.
3. Let appId be the extension input.
4. Pass facetId and appId to the FIDO algorithm for determining if a caller's FacetID is authorized for an AppID. If that algorithm rejects appId then return a "SecurityError" DOMException.
5. When building allowCredentialDescriptorList, if a U2F authenticator indicates that a credential is inapplicable (i.e. by returning SW_WRONG_DATA) then the client MUST retry with the U2F application parameter set to the SHA-256 hash of appId. If this results in an applicable credential, the client MUST include the credential in allowCredentialDescriptorList. The value of appId then replaces the rpId parameter of authenticatorGetAssertion.

Client extension output
Returns the value true to indicate to the RP that the extension was acted upon.

partial dictionary AuthenticationExtensionsClientOutputs {
  boolean appid;
};

Authenticator extension input
None.

Authenticator extension processing
None.

Authenticator extension output
None.

10.2. Simple Transaction Authorization Extension (txAuthSimple)

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 USVString prompt.

partial dictionary AuthenticationExtensionsClientInputs {
  USVString txAuthSimple;
};

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

Client extension output
Returns the authenticator extension output string UTF-8 decoded into a USVString.

partial dictionary AuthenticationExtensionsClientOutputs {
  USVString txAuthSimple;
};

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

CDDL:
txAuthSimpleInput = (tstr)

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).

```
CDDL:
txAuthSimpleOutput = (tstr)
```

10.3. Generic Transaction Authorization Extension (txAuthGeneric)

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 JavaScript object defined as follows:

```
dictionary txAuthGenericArg {
    required USVString contentType;    // MIME-Type of the content, e.g., "image
/png"
    required ArrayBuffer content;
};

partial dictionary AuthenticationExtensionsClientInputs {
  txAuthGenericArg txAuthGeneric;
};
```

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

Client extension output
Returns the authenticator extension output value as an ArrayBuffer.

```
partial dictionary AuthenticationExtensionsClientOutputs {
  ArrayBuffer txAuthGeneric;
};
```

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

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).

CDDL:
txAuthSimpleOutput = (tstr)

10.3. Generic Transaction Authorization Extension (txAuthGeneric)

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 JavaScript object defined as follows:

dictionary txAuthGenericArg {
    required USVString contentType;   // MIME-Type of the content, e.g., "image
/png"
    required ArrayBuffer content;
};

partial dictionary AuthenticationExtensionsClientInputs {
  txAuthGenericArg txAuthGeneric;
};

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

Client extension output
Returns the authenticator extension output value as an ArrayBuffer.

partial dictionary AuthenticationExtensionsClientOutputs {
  ArrayBuffer txAuthGeneric;
};

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;

   partial dictionary AuthenticationExtensionsClientInputs {
     AuthenticatorSelectionList authnSel;
   };

          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
          Returns the value true to indicate to the RP that the extension
          was acted upon.

   partial dictionary AuthenticationExtensionsClientOutputs {
     boolean authnSel;
   };

     Authenticator extension input
          None.

     Authenticator extension processing
          None.

     Authenticator extension output
          None.

    10.5. 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.

   partial dictionary AuthenticationExtensionsClientInputs {
     boolean exts;
   };

     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 an array of
          extension identifier strings.
```

```
          authnSel

     Client extension input
          A sequence of AAGUIDs:

   typedef sequence<AAGUID> AuthenticatorSelectionList;

   partial dictionary AuthenticationExtensionsClientInputs {
     AuthenticatorSelectionList authnSel;
   };

          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
          Returns the value true to indicate to the RP that the extension
          was acted upon.

   partial dictionary AuthenticationExtensionsClientOutputs {
     boolean authnSel;
   };

     Authenticator extension input
          None.

     Authenticator extension processing
          None.

     Authenticator extension output
          None.

    10.5. 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.

   partial dictionary AuthenticationExtensionsClientInputs {
     boolean exts;
   };

     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 an array of
          extension identifier strings.
```

```
typedef sequence<USVString> AuthenticationExtensionsSupported;

partial dictionary AuthenticationExtensionsClientOutputs {
  AuthenticationExtensionsSupported exts;
};
```

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 (UTF-8 encoded) strings.

10.6. 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.

```
partial dictionary AuthenticationExtensionsClientInputs {
  boolean uvi;
};
```

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

Client extension output
: Returns the authenticator extension output as an ArrayBuffer.

```
partial dictionary AuthenticationExtensionsClientOutputs {
  ArrayBuffer uvi;
};
```

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 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).

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

typedef sequence<USVString> AuthenticationExtensionsSupported;

partial dictionary AuthenticationExtensionsClientOutputs {
  AuthenticationExtensionsSupported exts;
};

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 (UTF-8 encoded) strings.

10.6. 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.

partial dictionary AuthenticationExtensionsClientInputs {
  boolean uvi;
};

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

Client extension output
Returns the authenticator extension output as an ArrayBuffer.

partial dictionary AuthenticationExtensionsClientOutputs {
  ArrayBuffer uvi;
};

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 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).

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

As an example, the UVI could be computed as SHA256(KeyID || SHA256(rawUVI)), where || represents concatenation, and 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.

Example of authenticator data containing one UVI extension

```
...                                  -- [=RP ID=] hash (32 bytes)
81                                     -- UP and ED set
00 00 00 01                               -- (initial) signature counter
...                                  -- all public key alg etc.
A1                                      -- extension: CBOR map of one elemen
t
    63                                 -- Key 1: CBOR text string of 3 byte
s
        75 76 69                           -- "uvi" [=UTF-8 encoded=] string
    58 20                               -- Value 1: CBOR byte string with 0x
20 bytes
        43 B8 E3 BE 27 95 8C 28             -- the UVI value itself
        D5 74 BF 46 8A 85 CF 46
        9A 14 F0 E5 16 69 31 DA
        4B CF FF C1 BB 11 32 82
```

10.7. 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.

```
partial dictionary AuthenticationExtensionsClientInputs {
  boolean loc;
};
```

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

Client extension output
Returns a JavaScript object that encodes the location information in the authenticator extension output as a Coordinates value, as defined by [Geolocation-API].

```
partial dictionary AuthenticationExtensionsClientOutputs {
  Coordinates loc;
};
```

Authenticator extension input
The Boolean value true, encoded in CBOR (major type 7, value 21).

Authenticator extension processing
Determine the Geolocation value.

Authenticator extension output
A [Geolocation-API] Coordinates record encoded as a CBOR map. Values represented by the "double" type in JavaScript are represented as 64-bit CBOR floating point numbers. Per the Geolocation specification, the "latitude", "longitude", and "accuracy" values are required and other values such as "altitude" are optional.

As an example, the UVI could be computed as SHA256(KeyID || SHA256(rawUVI)), where || represents concatenation, and 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.

Example of authenticator data containing one UVI extension

```
...                                  -- [=RP ID=] hash (32 bytes)
81                                     -- UP and ED set
00 00 00 01                               -- (initial) signature counter
...                                  -- all public key alg etc.
A1                                      -- extension: CBOR map of one elemen
t
    63                                 -- Key 1: CBOR text string of 3 byte
s
        75 76 69                           -- "uvi" [=UTF-8 encoded=] string
    58 20                               -- Value 1: CBOR byte string with 0x
20 bytes
        43 B8 E3 BE 27 95 8C 28             -- the UVI value itself
        D5 74 BF 46 8A 85 CF 46
        9A 14 F0 E5 16 69 31 DA
        4B CF FF C1 BB 11 32 82
```

10.7. 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.

```
partial dictionary AuthenticationExtensionsClientInputs {
  boolean loc;
};
```

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

Client extension output
Returns a JavaScript object that encodes the location information in the authenticator extension output as a Coordinates value, as defined by [Geolocation-API].

```
partial dictionary AuthenticationExtensionsClientOutputs {
  Coordinates loc;
};
```

Authenticator extension input
The Boolean value true, encoded in CBOR (major type 7, value 21).

Authenticator extension processing
Determine the Geolocation value.

Authenticator extension output
A [Geolocation-API] Coordinates record encoded as a CBOR map. Values represented by the "double" type in JavaScript are represented as 64-bit CBOR floating point numbers. Per the Geolocation specification, the "latitude", "longitude", and "accuracy" values are required and other values such as "altitude" are optional.

10.8. User Verification Method Extension (uvm)

This registration extension and authentication extension enables use of a user verification method.

Extension identifier
: uvm

Client extension input
: The Boolean value true to indicate that this extension is requested by the Relying Party.

```
partial dictionary AuthenticationExtensionsClientInputs {
  boolean uvm;
};
```

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

Client extension output
: Returns a JSON array of 3-element arrays of numbers that encodes the factors in the authenticator extension output.

```
typedef sequence<unsigned long> UvmEntry;
typedef sequence<UvmEntry> UvmEntries;

partial dictionary AuthenticationExtensionsClientOutputs {
  UvmEntries uvm;
};
```

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 one or more user verification methods indicating the method(s) used by the user to authorize the operation, as defined below. This extension can be added to attestation objects and assertions.

Authenticator extension output
: Authenticators 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 = [
              userVerificationMethod: uint .size 4,
              keyProtectionType: uint .size 2,
              matcherProtectionType: uint .size 2
          ]
```

The semantics of the fields in each uvmEntry are as follows:

userVerificationMethod
: The authentication method/factor used by the authenticator to verify the user. Available values are defined in Section 3.1 User Verification Methods of [FIDO-Registry].

keyProtectionType
: The method used by the authenticator to protect the FIDO registration private key material. Available values are defined in Section 3.2 Key Protection Types of [FIDO-Registry].

matcherProtectionType
: The method used by the authenticator to protect the

10.8. User Verification Method Extension (uvm)

This registration extension and authentication extension enables use of a user verification method.

Extension identifier
: uvm

Client extension input
: The Boolean value true to indicate that this extension is requested by the Relying Party.

partial dictionary AuthenticationExtensionsClientInputs {
  boolean uvm;
};

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

Client extension output
: Returns a JSON array of 3-element arrays of numbers that encodes the factors in the authenticator extension output.

typedef sequence<unsigned long> UvmEntry;
typedef sequence<UvmEntry> UvmEntries;

partial dictionary AuthenticationExtensionsClientOutputs {
  UvmEntries uvm;
};

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 one or more user verification methods indicating the method(s) used by the user to authorize the operation, as defined below. This extension can be added to attestation objects and assertions.

Authenticator extension output
: Authenticators 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 = [
              userVerificationMethod: uint .size 4,
              keyProtectionType: uint .size 2,
              matcherProtectionType: uint .size 2
          ]

The semantics of the fields in each uvmEntry are as follows:

userVerificationMethod
: The authentication method/factor used by the authenticator to verify the user. Available values are defined in Section 3.1 User Verification Methods of [FIDO-Registry].

keyProtectionType
: The method used by the authenticator to protect the FIDO registration private key material. Available values are defined in Section 3.2 Key Protection Types of [FIDO-Registry].

matcherProtectionType
: The method used by the authenticator to protect the

```
                 matcher that performs user verification. Available values
                 are defined in Section 3.3 Matcher Protection Types of
                 [FIDO-Registry].

            If >3 factors can be used in an authentication instance the
            authenticator vendor MUST select the 3 factors it believes will
            be most relevant to the Server to include in the UVM.

            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                     -- UP and ED set
   00 00 00 01               -- (initial) signature counter
   ...                   -- all public key alg etc.
   A1                     -- extension: CBOR map of one element
       63                 -- Key 1: CBOR text string of 3 bytes
         75 76 6d         -- "uvm" [=UTF-8 encoded=] string
       82                 -- Value 1: CBOR array of length 2 indicating two factor
   usage
         83                -- Item 1: CBOR array of length 3
           02               -- Subitem 1: CBOR integer for User Verification Method
    Fingerprint
           04               -- Subitem 2: CBOR short for Key Protection Type TEE
           02               -- Subitem 3: CBOR short for Matcher Protection Type TE
   E
         83                -- Item 2: CBOR array of length 3
           04               -- Subitem 1: CBOR integer for User Verification Method
    Passcode
           01               -- Subitem 2: CBOR short for Key Protection Type Softwa
   re
           01               -- Subitem 3: CBOR short for Matcher Protection Type So
   ftware

     10.9. Biometric Authenticator Performance Bounds Extension
     (biometricPerfBounds)

      This registration extension allows Relying Parties to specify the
      desired performance bounds for selecting biometric authenticators as
      candidates to be employed in a registration ceremony.

      Extension identifier
            biometricPerfBounds

      Client extension input
            Biometric performance bounds:

   dictionary authenticatorBiometricPerfBounds{
       float FAR;
       float FRR;
       };

            The FAR is the maximum false acceptance rate for a biometric
            authenticator allowed by the Relying Party.

            The FAR is the maximum false rejection rate for a biometric
            authenticator allowed by the Relying Party.

      Client extension processing
            This extension can only be used during create(). If the client
            supports this extension, it MUST NOT use a biometric
            authenticator whose FAR or FRR does not match the bounds as
            provided. The client can obtain information about the biometric
            authenticator's performance from authoritative sources such as
            the FIDO Metadata Service [FIDOMetadataService] (see Sec. 3.2 of
            [FIDOUAFAuthenticatorMetadataStatements]).

      Client extension output
            Returns the JSON value true to indicate to the RP that the
```

```
                 matcher that performs user verification. Available values
                 are defined in Section 3.3 Matcher Protection Types of
                 [FIDO-Registry].

            If >3 factors can be used in an authentication instance the
            authenticator vendor MUST select the 3 factors it believes will
            be most relevant to the Server to include in the UVM.

            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                     -- UP and ED set
   00 00 00 01               -- (initial) signature counter
   ...                   -- all public key alg etc.
   A1                     -- extension: CBOR map of one element
       63                 -- Key 1: CBOR text string of 3 bytes
         75 76 6d         -- "uvm" [=UTF-8 encoded=] string
       82                 -- Value 1: CBOR array of length 2 indicating two factor
   usage
         83                -- Item 1: CBOR array of length 3
           02               -- Subitem 1: CBOR integer for User Verification Method
    Fingerprint
           04               -- Subitem 2: CBOR short for Key Protection Type TEE
           02               -- Subitem 3: CBOR short for Matcher Protection Type TE
   E
         83                -- Item 2: CBOR array of length 3
           04               -- Subitem 1: CBOR integer for User Verification Method
    Passcode
           01               -- Subitem 2: CBOR short for Key Protection Type Softwa
   re
           01               -- Subitem 3: CBOR short for Matcher Protection Type So
   ftware

     10.9. Biometric Authenticator Performance Bounds Extension
     (biometricPerfBounds)

      This registration extension allows Relying Parties to specify the
      desired performance bounds for selecting biometric authenticators as
      candidates to be employed in a registration ceremony.

      Extension identifier
            biometricPerfBounds

      Client extension input
            Biometric performance bounds:

   dictionary authenticatorBiometricPerfBounds{
       float FAR;
       float FRR;
       };

            The FAR is the maximum false acceptance rate for a biometric
            authenticator allowed by the Relying Party.

            The FAR is the maximum false rejection rate for a biometric
            authenticator allowed by the Relying Party.

      Client extension processing
            This extension can only be used during create(). If the client
            supports this extension, it MUST NOT use a biometric
            authenticator whose FAR or FRR does not match the bounds as
            provided. The client can obtain information about the biometric
            authenticator's performance from authoritative sources such as
            the FIDO Metadata Service [FIDOMetadataService] (see Sec. 3.2 of
            [FIDOUAFAuthenticatorMetadataStatements]).

      Client extension output
            Returns the JSON value true to indicate to the RP that the
```

extension was acted upon

Authenticator extension input
None.

Authenticator extension processing
None.

Authenticator extension output
None.

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-Registries].
* WebAuthn Attestation Statement Format Identifier: packed
* Description: The "packed" attestation statement format is a WebAuthn-optimized format for attestation. 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 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 rawData and signature fields are computed differently.
* Specification Document: Section 8.3 TPM Attestation Statement Format of this specification
* WebAuthn Attestation Statement Format Identifier: android-key
* Description: Platform-provided authenticators based on 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 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 8.5 Android SafetyNet Attestation Statement Format of this specification
* WebAuthn Attestation Statement Format Identifier: fido-u2f
* Description: Used with FIDO U2F authenticators
* 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-Registries].
* WebAuthn Extension Identifier: appid
* Description: This authentication extension 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. A WebAuthn Relying Party can specify a prompt string, intended for display on a trusted device on the authenticator
* Specification Document: Section 10.2 Simple Transaction Authorization Extension (txAuthSimple) of this specification
* WebAuthn Extension Identifier: txAuthGeneric
* Description: This registration extension and authentication

extension was acted upon

Authenticator extension input
None.

Authenticator extension processing
None.

Authenticator extension output
None.

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-Registries].
* WebAuthn Attestation Statement Format Identifier: packed
* Description: The "packed" attestation statement format is a WebAuthn-optimized format for attestation. 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 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 rawData and signature fields are computed differently.
* Specification Document: Section 8.3 TPM Attestation Statement Format of this specification
* WebAuthn Attestation Statement Format Identifier: android-key
* Description: Platform-provided authenticators based on 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 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 8.5 Android SafetyNet Attestation Statement Format of this specification
* WebAuthn Attestation Statement Format Identifier: fido-u2f
* Description: Used with FIDO U2F authenticators
* 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-Registries].
* WebAuthn Extension Identifier: appid
* Description: This authentication extension 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. A WebAuthn Relying Party can specify a prompt string, intended for display on a trusted device on the authenticator
* 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
* Description: This registration extension allows a WebAuthn Relying Party to guide the selection of the authenticator that will be leveraged when creating the credential. It is intended primarily for WebAuthn Relying Parties that wish to tightly control the experience around credential creation.
* Specification Document: Section 10.4 Authenticator Selection Extension (authnSel) of this specification
* WebAuthn Extension Identifier: exts
* 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 10.5 Supported Extensions Extension (exts) of this specification
* WebAuthn Extension Identifier: uvi
* Description: This registration extension and authentication extension enables use of a user verification index. The user verification index is a value uniquely identifying a user verification data record. The UVI data can be used by servers to understand whether an authentication was authorized by the exact same biometric data as the initial key generation. This allows the detection and prevention of "friendly fraud".
* Specification Document: Section 10.6 User Verification Index Extension (uvi) of this specification
* WebAuthn Extension Identifier: loc
* Description: The location registration extension and authentication extension provides the client device's current location to the WebAuthn relying party, if supported by the client device and subject to user consent.
* Specification Document: Section 10.7 Location Extension (loc) of this specification
* WebAuthn Extension Identifier: uvm
* Description: This registration extension and authentication extension enables use of a user verification method. The user verification method extension returns to the Webauthn relying party which user verification methods (factors) were used for the WebAuthn operation.
* Specification Document: Section 10.8 User Verification Method Extension (uvm) of this specification

11.3. COSE Algorithm Registrations

This section registers identifiers for the following ECDAA algorithms in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG]. Note that [WebAuthn-COSE-Algs] also registers RSASSA-PKCS1-v1_5 [RFC8017] algorithms using SHA-2 and SHA-1 hash functions in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG], such as registering -257 for "RS256".

* Name: ED256
* Value: TBD (requested assignment -260)
* Description: TPM_ECC_BN_P256 curve w/ SHA-256
* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
* Recommended: Yes
* Name: ED512
* Value: TBD (requested assignment -261)
* Description: ECC_BN_ISOP512 curve w/ SHA-512
* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
* Recommended: Yes

12. Sample scenarios

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
* Description: This registration extension allows a WebAuthn Relying Party to guide the selection of the authenticator that will be leveraged when creating the credential. It is intended primarily for WebAuthn Relying Parties that wish to tightly control the experience around credential creation.
* Specification Document: Section 10.4 Authenticator Selection Extension (authnSel) of this specification
* WebAuthn Extension Identifier: exts
* 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 10.5 Supported Extensions Extension (exts) of this specification
* WebAuthn Extension Identifier: uvi
* Description: This registration extension and authentication extension enables use of a user verification index. The user verification index is a value uniquely identifying a user verification data record. The UVI data can be used by servers to understand whether an authentication was authorized by the exact same biometric data as the initial key generation. This allows the detection and prevention of "friendly fraud".
* Specification Document: Section 10.6 User Verification Index Extension (uvi) of this specification
* WebAuthn Extension Identifier: loc
* Description: The location registration extension and authentication extension provides the client device's current location to the WebAuthn relying party, if supported by the client device and subject to user consent.
* Specification Document: Section 10.7 Location Extension (loc) of this specification
* WebAuthn Extension Identifier: uvm
* Description: This registration extension and authentication extension enables use of a user verification method. The user verification method extension returns to the Webauthn relying party which user verification methods (factors) were used for the WebAuthn operation.
* Specification Document: Section 10.8 User Verification Method Extension (uvm) of this specification

11.3. COSE Algorithm Registrations

This section registers identifiers for the following ECDAA algorithms in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG]. Note that [WebAuthn-COSE-Algs] also registers RSASSA-PKCS1-v1_5 [RFC8017] algorithms using SHA-2 and SHA-1 hash functions in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG], such as registering -257 for "RS256".

* Name: ED256
* Value: TBD (requested assignment -260)
* Description: TPM_ECC_BN_P256 curve w/ SHA-256
* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
* Recommended: Yes
* Name: ED512
* Value: TBD (requested assignment -261)
* Description: ECC_BN_ISOP512 curve w/ SHA-512
* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
* Recommended: Yes

12. Sample scenarios

This section is not normative.

In this section, we walk through some events in the lifecycle of a public key credential, along with the corresponding sample code for using this API. Note that this is an example flow and does not limit the scope of how the API can be used.

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.

12.1. Registration

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 credential 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:

```
if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }

var publicKey = {
  // The challenge must be produced by the server, see the Security Consideratio
ns
  challenge: new Uint8Array([21,31,105 /* 29 more random bytes generated by the
server */]),

  // Relying Party:
  rp: {
    name: "ACME Corporation"
  },
```

This section is not normative.

In this section, we walk through some events in the lifecycle of a public key credential, along with the corresponding sample code for using this API. Note that this is an example flow and does not limit the scope of how the API can be used.

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.

12.1. Registration

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 credential 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:

```
if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }

var publicKey = {
  // The challenge must be produced by the server, see the Security Consideratio
ns
  challenge: new Uint8Array([21,31,105 /* 29 more random bytes generated by the
server */]),

  // Relying Party:
  rp: {
    name: "ACME Corporation"
  },
    // User:
    user: {
      id: Uint8Array.from(window.atob("MIIBkzCCATigAwIBAjCCAZMwggE4oAMCAQIwggGTMII
="), c=>c.charCodeAt(0)),
      name: "alex.p.mueller@example.com",
      displayName: "Alex P. Mller",
      icon: "https://pics.example.com/00/p/aBjjjpqPb.png"
    },

    // This Relying Party will accept either an ES256 or RS256 credential, but
    // prefers an ES256 credential.
    pubKeyCredParams: [
      {
        type: "public-key",
        alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
      },
      {
        type: "public-key",
        alg: -257 // Value registered by this specification for "RS256"
      }
    ],

    timeout: 60000,  // 1 minute
    excludeCredentials: [], // No exclude list of PKCredDescriptors
    extensions: {"loc": true}  // Include location information
                                         // in attestation
  };

  // 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.
    }).catch(function (err) {
      // No acceptable authenticator or user refused consent. Handle appropriately
.
    });

    12.2. 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.
      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.
      4. The user agent asks the user whether they are willing to register
         with the Relying Party using an available platform authenticator.
      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.

  if (!window.PublicKeyCredential) { /* Platform not capable of the API. Handle er
ror. */ }

  PublicKeyCredential.isUserVerifyingPlatformAuthenticatorAvailable()
      .then(function (userIntent) {

          // 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
  });
```

/Users/jehodges/Documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.txt, Top line: 5285

```
    // User:
    user: {
      id: Uint8Array.from(window.atob("MIIBkzCCATigAwIBAjCCAZMwggE4oAMCAQIwggGTMII
="), c=>c.charCodeAt(0)),
      name: "alex.p.mueller@example.com",
      displayName: "Alex P. Mller",
      icon: "https://pics.example.com/00/p/aBjjjpqPb.png"
    },

    // This Relying Party will accept either an ES256 or RS256 credential, but
    // prefers an ES256 credential.
    pubKeyCredParams: [
      {
        type: "public-key",
        alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
      },
      {
        type: "public-key",
        alg: -257 // Value registered by this specification for "RS256"
      }
    ],

    timeout: 60000,  // 1 minute
    excludeCredentials: [], // No exclude list of PKCredDescriptors
    extensions: {"loc": true}  // Include location information
                                         // in attestation
  };

  // 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.
    }).catch(function (err) {
      // No acceptable authenticator or user refused consent. Handle appropriately
.
    });

    12.2. 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.
      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.
      4. The user agent asks the user whether they are willing to register
         with the Relying Party using an available platform authenticator.
      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.

  if (!window.PublicKeyCredential) { /* Platform not capable of the API. Handle er
ror. */ }

  PublicKeyCredential.isUserVerifyingPlatformAuthenticatorAvailable()
      .then(function (userIntent) {

          // 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
  });
```

```
    } else {

        // Record that the user does not intend to use a platform authentica
tor
        // and default the user to a password-based flow in the future.
    }

}).then(function (newCredentialInfo) {
    // Send new credential info to server for verification and registration.
}).catch( function(err) {
    // Something went wrong. Handle appropriately.
});
```

12.3. Authentication

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. This can be obtained from the data that was stored locally after registration, or by other means 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.
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 needed. 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 it is database, and verifies the assertion's authentication signature. 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:

```
if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }

var options = {
              // The challenge must be produced by the server, see the Securit
y Considerations
              challenge: new Uint8Array([4,101,15 /* 29 more random bytes gene
rated by the server */]),
              timeout: 60000,  // 1 minute
              allowCredentials: [{ type: "public-key" }]
            };

navigator.credentials.get({ "publicKey": options })
    .then(function (assertion) {
```

```
    } else {

        // Record that the user does not intend to use a platform authentica
tor
        // and default the user to a password-based flow in the future.
    }

}).then(function (newCredentialInfo) {
    // Send new credential info to server for verification and registration.
}).catch( function(err) {
    // Something went wrong. Handle appropriately.
});
```

12.3. Authentication

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. This can be obtained from the data that was stored locally after registration, or by other means 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.
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 needed. 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 it is database, and verifies the assertion's authentication signature. 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:

```
if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }

var options = {
              // The challenge must be produced by the server, see the Securit
y Considerations
              challenge: new Uint8Array([4,101,15 /* 29 more random bytes gene
rated by the server */]),
              timeout: 60000,  // 1 minute
              allowCredentials: [{ type: "public-key" }]
            };

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.
});
```

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 (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }

var encoder = new TextEncoder();
var acceptableCredential1 = {
    type: "public-key",
    id: encoder.encode("!!!!!!!hi there!!!!!!!\n")
};
var acceptableCredential2 = {
    type: "public-key",
    id: encoder.encode("roses are red, violets are blue\n")
};

var options = {
                // The challenge must be produced by the server, see the Securit
y Considerations
                challenge: new Uint8Array([8,18,33 /* 29 more random bytes gener
ated by the server */]),
                timeout: 60000,  // 1 minute
                allowCredentials: [acceptableCredential1, acceptableCredential2]
,
                extensions: { 'txAuthSimple':
                   "Wave your hands in the air like you just don't care" }
              };

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.
});
```

12.4. Aborting Authentication Operations

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 ab
ort.
}

var options = {
    // A list of options.
}

navigator.credentials.create({
    publicKey: options,
    signal: authAbortSignal})
    .then(function (attestation) {
        // Register the user.
    }).catch(function (error) {
        if (error == "AbortError") {
            // Inform user the credential hasn't been created.
            // Let the server know a key hasn't been created.
        }
    });
```

    // 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 (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }

var encoder = new TextEncoder();
var acceptableCredential1 = {
    type: "public-key",
    id: encoder.encode("!!!!!!!hi there!!!!!!!\n")
};
var acceptableCredential2 = {
    type: "public-key",
    id: encoder.encode("roses are red, violets are blue\n")
};

var options = {
                // The challenge must be produced by the server, see the Securit
y Considerations
                challenge: new Uint8Array([8,18,33 /* 29 more random bytes gener
ated by the server */]),
                timeout: 60000, // 1 minute
                allowCredentials: [acceptableCredential1, acceptableCredential2]
,
                extensions: { 'txAuthSimple':
                   "Wave your hands in the air like you just don't care" }
              };

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.
});

12.4. Aborting Authentication Operations

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 ab
ort.
}

var options = {
    // A list of options.
}

navigator.credentials.create({
    publicKey: options,
    signal: authAbortSignal})
    .then(function (attestation) {
        // Register the user.
    }).catch(function (error) {
        if (error == "AbortError") {
            // Inform user the credential hasn't been created.
            // Let the server know a key hasn't been created.
        }
    });

```
// Assume widget shows up whenever authentication occurs.
if (widget == "disappear") {
  authAbortController.abort();
}
```

12.5. Decommissioning

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 need support from the API 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 the 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.
    + Sometime later, the server deregisters this credential due to inactivity.

13. Security Considerations

This specification defines a Web API and a cryptographic peer-entity authentication protocol. The Web Authentication API allows Web developers (i.e., "authors") to utilize the Web Authentication protocol in their registration and authentication ceremonies. The entities comprising the Web Authentication protocol endpoints are user-controlled authenticators and a Relying Party's computing environment hosting the Relying Party's web application. In this model, the user agent, together with the WebAuthn Client, comprise an intermediary between authenticators and Relying Parties. Additionally, authenticators can attest to Relying Parties as to their provenance.

At this time, this specification does not feature detailed security considerations. However, the [FIDOSecRef] document provides a security analysis which is overall applicable to this specification. Also, the [FIDOAuthnrSecReqs] document suite defines authenticator security characteristics which are overall applicable for WebAuthn authenticators.

The below subsections comprise the current Web Authentication-specific security considerations.

13.1. Cryptographic Challenges

As a cryptographic protocol, Web Authentication is dependent upon randomized challenges to avoid replay attacks. Therefore, both challenge's and challenge's value MUST be randomly generated by Relying Parties in an environment they trust (e.g., on the server-side), and the returned challenge value in the client's response MUST match what was generated. This SHOULD be done in a fashion that does not rely upon a client's behavior, e.g., the Relying Party SHOULD store the challenge temporarily until the operation is complete. Tolerating a mismatch will compromise the security of the protocol.

// Assume widget shows up whenever authentication occurs.
if (widget == "disappear") {
  authAbortController.abort();
}

12.5. Decommissioning

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 need support from the API 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 the 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.
    + Sometime later, the server deregisters this credential due to inactivity.

13. Security Considerations

This specification defines a Web API and a cryptographic peer-entity authentication protocol. The Web Authentication API allows Web developers (i.e., "authors") to utilize the Web Authentication protocol in their registration and authentication ceremonies. The entities comprising the Web Authentication protocol endpoints are user-controlled authenticators and a Relying Party's computing environment hosting the Relying Party's web application. In this model, the user agent, together with the WebAuthn Client, comprise an intermediary between authenticators and Relying Parties. Additionally, authenticators can attest to Relying Parties as to their provenance.

At this time, this specification does not feature detailed security considerations. However, the [FIDOSecRef] document provides a security analysis which is overall applicable to this specification. Also, the [FIDOAuthnrSecReqs] document suite defines authenticator security characteristics which are overall applicable for WebAuthn authenticators.

The below subsections comprise the current Web Authentication-specific security considerations.

13.1. Cryptographic Challenges

As a cryptographic protocol, Web Authentication is dependent upon randomized challenges to avoid replay attacks. Therefore, both challenge's and challenge's value MUST be randomly generated by Relying Parties in an environment they trust (e.g., on the server-side), and the returned challenge value in the client's response MUST match what was generated. This SHOULD be done in a fashion that does not rely upon a client's behavior, e.g., the Relying Party SHOULD store the challenge temporarily until the operation is complete. Tolerating a mismatch will compromise the security of the protocol.

In order to prevent replay attacks, the challenges MUST contain enough entropy to make guessing them infeasible. Challenges SHOULD therefore be at least 16 bytes long.

13.2. Attestation Security Considerations

13.2.1. 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.

13.2.2. Attestation Certificate and Attestation Certificate CA Compromise

When an intermediate CA or a root CA used for issuing attestation certificates is compromised, WebAuthn authenticator attestation keys are still safe although their certificates can no longer be trusted. A WebAuthn Authenticator manufacturer that has recorded the public attestation keys for their devices can issue new attestation certificates for these keys from a new intermediate CA or from a new root CA. If the root CA changes, the Relying Parties MUST update their trusted root certificates accordingly.

A WebAuthn Authenticator attestation certificate MUST be revoked by the issuing CA if its key has been 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 RogueList (i.e., the list of revoked authenticators) maintained by the related ECDAA-Issuer. The Relying Party SHOULD verify whether an authenticator belongs to the RogueList when performing ECDAA-Verify (see section 3.6 in [FIDOEcdaaAlgorithm]). For example, the FIDO Metadata Service [FIDOMetadataService] provides one way to access such information.

13.3. Security Benefits for Relying Parties

The main benefits offered to Relying Parties by this specification include:

1. Users and accounts can be secured using widely compatible, easy-to-use multi-factor authentication.
2. The Relying Party does not need to provision authenticator hardware to its users. Instead, each user can independently obtain any

In order to prevent replay attacks, the challenges MUST contain enough entropy to make guessing them infeasible. Challenges SHOULD therefore be at least 16 bytes long.

13.2. Attestation Security Considerations

13.2.1. 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.

13.2.2. Attestation Certificate and Attestation Certificate CA Compromise

When an intermediate CA or a root CA used for issuing attestation certificates is compromised, WebAuthn authenticator attestation keys are still safe although their certificates can no longer be trusted. A WebAuthn Authenticator manufacturer that has recorded the public attestation keys for their devices can issue new attestation certificates for these keys from a new intermediate CA or from a new root CA. If the root CA changes, the Relying Parties MUST update their trusted root certificates accordingly.

A WebAuthn Authenticator attestation certificate MUST be revoked by the issuing CA if its key has been 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 RogueList (i.e., the list of revoked authenticators) maintained by the related ECDAA-Issuer. The Relying Party SHOULD verify whether an authenticator belongs to the RogueList when performing ECDAA-Verify (see section 3.6 in [FIDOEcdaaAlgorithm]). For example, the FIDO Metadata Service [FIDOMetadataService] provides one way to access such information.

13.3. Security Benefits for Relying Parties

The main benefits offered to Relying Parties by this specification include:

1. Users and accounts can be secured using widely compatible, easy-to-use multi-factor authentication.
2. The Relying Party does not need to provision authenticator hardware to its users. Instead, each user can independently obtain any

conforming authenticator and use that same authenticator with any number of Relying Parties. The Relying Party can optionally enforce requirements on authenticators' security properties by inspecting the attestation statements returned from the authenticators.

3. Registration and authentication ceremonies are resistant to man-in-the-middle attacks.
4. The Relying Party can automatically support multiple types of user verification - for example PIN, biometrics and/or future methods - with little or no code change, and can let each user decide which they prefer to use via their choice of authenticator.
5. The Relying Party does not need to store additional secrets in order to gain the above benefits.

As stated in the Conformance section, the Relying Party MUST behave as described in 7 Relying Party Operations to obtain all of the above security benefits. However, one notable use case that departs slightly from this is described in the next section.

13.3.1. Considerations for Self and None Attestation Types and Ignoring Attestation

When registering a new credential, the Relying Party MAY choose to accept an attestation statement of type Self or None, or to not verify the attestation statement. In all of these cases the Relying Party loses much of benefit (3) listed above, but retains the other benefits.

In these cases it is possible for a man-in-the-middle attacker - for example, a malicious client or script - to replace the credential public key to be registered, and subsequently tamper with future authentication assertions bound for the same Relying Party and passing through the same attacker. Accepting these types of attestation statements therefore constitutes a leap of faith. In cases where registration was accomplished securely, subsequent authentication ceremonies remain resistant to man-in-the-middle attacks, i.e., benefit (3) is retained. Note, however, that such an attack would be easy to detect and very difficult to maintain, since any authentication ceremony that the same attacker does not or cannot tamper with would always fail.

The Relying Party SHOULD consider the above in its threat model when deciding its policy on what attestation types to accept or whether to ignore attestation.

Note: The default attestation type is None, since the above issues will likely not be a major concern in most Relying Parties' threat models. For example, the man-in-the-middle attack described above is more difficult than a man-in-the-middle attack against a Relying Party that only uses conventional password authentication.

13.4. credentialId Unsigned

The credential ID is not signed. This is not a problem because all that would happen if an authenticator returns the wrong credential ID, or if an attacker intercepts and manipulates the credential ID, is that the Relying Party would not look up the correct credential public key with which to verify the returned signed authenticator data (a.k.a., assertion), and thus the interaction would end in an error.

13.5. Browser Permissions Framework and Extensions

Web Authentication API implementations should leverage the browser permissions framework as much as possible when obtaining user permissions for certain extensions. An example is the location extension (see 10.7 Location Extension (loc)), implementations of which should make use of the existing browser permissions framework for the Geolocation API.

14. Privacy Considerations

The privacy principles in [FIDO-Privacy-Principles] also apply to this

conforming authenticator and use that same authenticator with any number of Relying Parties. The Relying Party can optionally enforce requirements on authenticators' security properties by inspecting the attestation statements returned from the authenticators.

3. Registration and authentication ceremonies are resistant to man-in-the-middle attacks.
4. The Relying Party can automatically support multiple types of user verification - for example PIN, biometrics and/or future methods - with little or no code change, and can let each user decide which they prefer to use via their choice of authenticator.
5. The Relying Party does not need to store additional secrets in order to gain the above benefits.

As stated in the Conformance section, the Relying Party MUST behave as described in 7 Relying Party Operations to obtain all of the above security benefits. However, one notable use case that departs slightly from this is described in the next section.

13.3.1. Considerations for Self and None Attestation Types and Ignoring Attestation

When registering a new credential, the Relying Party MAY choose to accept an attestation statement of type Self or None, or to not verify the attestation statement. In all of these cases the Relying Party loses much of benefit (3) listed above, but retains the other benefits.

In these cases it is possible for a man-in-the-middle attacker - for example, a malicious client or script - to replace the credential public key to be registered, and subsequently tamper with future authentication assertions bound for the same Relying Party and passing through the same attacker. Accepting these types of attestation statements therefore constitutes a leap of faith. In cases where registration was accomplished securely, subsequent authentication ceremonies remain resistant to man-in-the-middle attacks, i.e., benefit (3) is retained. Note, however, that such an attack would be easy to detect and very difficult to maintain, since any authentication ceremony that the same attacker does not or cannot tamper with would always fail.

The Relying Party SHOULD consider the above in its threat model when deciding its policy on what attestation types to accept or whether to ignore attestation.

Note: The default attestation type is None, since the above issues will likely not be a major concern in most Relying Parties' threat models. For example, the man-in-the-middle attack described above is more difficult than a man-in-the-middle attack against a Relying Party that only uses conventional password authentication.

13.4. credentialId Unsigned

The credential ID is not signed. This is not a problem because all that would happen if an authenticator returns the wrong credential ID, or if an attacker intercepts and manipulates the credential ID, is that the Relying Party would not look up the correct credential public key with which to verify the returned signed authenticator data (a.k.a., assertion), and thus the interaction would end in an error.

13.5. Browser Permissions Framework and Extensions

Web Authentication API implementations should leverage the browser permissions framework as much as possible when obtaining user permissions for certain extensions. An example is the location extension (see 10.7 Location Extension (loc)), implementations of which should make use of the existing browser permissions framework for the Geolocation API.

14. Privacy Considerations

The privacy principles in [FIDO-Privacy-Principles] also apply to this

specification.

14.1. Attestation Privacy

Attestation keys can be used to track users or link various online identities of the same user together. This can 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 key(s) (called Basic Attestation). This will anonymize the user at the risk of not being able to revoke a particular attestation key if its private key is compromised.
  [UAFProtocol] requires that at least 100,000 devices share the same attestation certificate in order to produce sufficiently large groups. This may serve as guidance about suitable batch sizes.
* A WebAuthn authenticator may be capable of dynamically generating different attestation keys (and requesting related certificates) per-origin (similar to the Attestation CA approach). For example, an authenticator can ship with a master attestation key (and certificate), and combined with a cloud-operated Anonymization CA, can dynamically generate per-origin attestation keys and attestation certificates.
  Note: In various places outside this specification, the term "Privacy CA" is used to refer to what is termed here as an Anonymization CA. Because the Trusted Computing Group (TCG) also used the term "Privacy CA" to refer to what the TCG now refers to as an Attestation CA (ACA) [TCG-CMCProfile-AIKCertEnroll], and the envisioned functionality of an Anonymization CA is not firmly established, we are using the term Anonymization CA here to try to mitigate confusion in the specific context of this specification.
* A WebAuthn Authenticator can implement Elliptic Curve based direct anonymous attestation (see [FIDOEcdaaAlgorithm]). Using this scheme, the authenticator generates a blinded attestation signature. This allows the Relying Party to verify the signature using the ECDAA-Issuer public key, but the attestation signature does not serve as a global correlation handle.

14.2. Registration Ceremony Privacy

In order to protect users from being identified without consent, implementations of the [[Create]](origin, options, sameOriginWithAncestors) method need to take care to not leak information that could enable a malicious Relying Party to distinguish between these cases, where "excluded" means that at least one of the credentials listed by the Relying Party in excludeCredentials is bound to the authenticator:

* No authenticators are present.
* At least one authenticator is present, and at least one present authenticator is excluded.

If the above cases are distinguishable, information is leaked by which a malicious Relying Party could identify the user by probing for which credentials are available. For example, one such information leak is if the client returns a failure response as soon as an excluded authenticator becomes available. In this case - especially if the excluded authenticator is a platform authenticator - the Relying Party could detect that the ceremony was canceled before the timeout and before the user could feasibly have canceled it manually, and thus conclude that at least one of the credentials listed in the excludeCredentials parameter is available to the user.

The above is not a concern, however, if the user has consented to create a new credential before a distinguishable error is returned, because in this case the user has confirmed intent to share the information that would be leaked.

14.3. Authentication Ceremony Privacy

In order to protect users from being identified without consent, implementations of the [[DiscoverFromExternalSource]](origin, options,

specification.

14.1. Attestation Privacy

Attestation keys can be used to track users or link various online identities of the same user together. This can 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 key(s) (called Basic Attestation). This will anonymize the user at the risk of not being able to revoke a particular attestation key if its private key is compromised.
  [UAFProtocol] requires that at least 100,000 devices share the same attestation certificate in order to produce sufficiently large groups. This may serve as guidance about suitable batch sizes.
* A WebAuthn authenticator may be capable of dynamically generating different attestation keys (and requesting related certificates) per-origin (similar to the Attestation CA approach). For example, an authenticator can ship with a master attestation key (and certificate), and combined with a cloud-operated Anonymization CA, can dynamically generate per-origin attestation keys and attestation certificates.
  Note: In various places outside this specification, the term "Privacy CA" is used to refer to what is termed here as an Anonymization CA. Because the Trusted Computing Group (TCG) also used the term "Privacy CA" to refer to what the TCG now refers to as an Attestation CA (ACA) [TCG-CMCProfile-AIKCertEnroll], and the envisioned functionality of an Anonymization CA is not firmly established, we are using the term Anonymization CA here to try to mitigate confusion in the specific context of this specification.
* A WebAuthn Authenticator can implement Elliptic Curve based direct anonymous attestation (see [FIDOEcdaaAlgorithm]). Using this scheme, the authenticator generates a blinded attestation signature. This allows the Relying Party to verify the signature using the ECDAA-Issuer public key, but the attestation signature does not serve as a global correlation handle.

14.2. Registration Ceremony Privacy

In order to protect users from being identified without consent, implementations of the [[Create]](origin, options, sameOriginWithAncestors) method need to take care to not leak information that could enable a malicious Relying Party to distinguish between these cases, where "excluded" means that at least one of the credentials listed by the Relying Party in excludeCredentials is bound to the authenticator:

* No authenticators are present.
* At least one authenticator is present, and at least one present authenticator is excluded.

If the above cases are distinguishable, information is leaked by which a malicious Relying Party could identify the user by probing for which credentials are available. For example, one such information leak is if the client returns a failure response as soon as an excluded authenticator becomes available. In this case - especially if the excluded authenticator is a platform authenticator - the Relying Party could detect that the ceremony was canceled before the timeout and before the user could feasibly have canceled it manually, and thus conclude that at least one of the credentials listed in the excludeCredentials parameter is available to the user.

The above is not a concern, however, if the user has consented to create a new credential before a distinguishable error is returned, because in this case the user has confirmed intent to share the information that would be leaked.

14.3. Authentication Ceremony Privacy

In order to protect users from being identified without consent, implementations of the [[DiscoverFromExternalSource]](origin, options,

sameOriginWithAncestors) method need to take care to not leak information that could enable a malicious Relying Party to distinguish between these cases, where "named" means that the credential is listed by the Relying Party in allowCredentials:

* A named credential is not available.
* A named credential is available, but the user does not consent to use it.

If the above cases are distinguishable, information is leaked by which a malicious Relying Party could identify the user by probing for which credentials are available. For example, one such information leak is if the client returns a failure response as soon as the user denies consent to proceed with an authentication ceremony. In this case the Relying Party could detect that the ceremony was canceled by the user and not the timeout, and thus conclude that at least one of the credentials listed in the allowCredentials parameter is available to the user.

15. Acknowledgements

We thank the following people for their reviews of, and contributions to, this specification: Yuriy Ackermann, James Barclay, Richard Barnes, Dominic Battr, John Bradley, Domenic Denicola, Rahul Ghosh, Brad Hill, Jing Jin, Wally Jones, Ian Kilpatrick, Axel Nennker, Yoshikazu Nojima, Kimberly Paulhamus, Adam Powers, Yaron Sheffer, Anne van Kesteren, Johan Verrept, and Boris Zbarsky.

We thank Anthony Nadalin, John Fontana, and Richard Barnes for their contributions as co-chairs of the Web Authentication Working Group.

We also thank Wendy Seltzer, Samuel Weiler, and Harry Halpin for their contributions as our W3C Team Contacts.

Index

Terms defined by this specification

* aaguid, in 6.3.1
* AAGUID, in 10.4
* alg, in 5.3
* allowCredentials, in 5.5
* Anonymization CA, in 14.1
* appid
    + dict-member for AuthenticationExtensionsClientInputs, in 10.1
    + dict-member for AuthenticationExtensionsClientOutputs, in 10.1
* AppID, in 10.1
* Assertion, in 4
* assertion signature, in 6
* attachment modality, in 5.4.5
* AttCA, in 6.3.3
* Attestation, in 4
* attestation, in 5.4
* Attestation CA, in 6.3.3
* Attestation Certificate, in 4
* Attestation Conveyance, in 5.4.6
* AttestationConveyancePreference, in 5.4.6
* attestationConveyancePreferenceOption, in 5.1.3
* attestation key pair, in 4
* attestationObject, in 5.2.1
* attestation object, in 6.3
* attestationObjectResult, in 5.1.3
* attestation private key, in 4
* attestation public key, in 4
* attestation signature, in 6
* attestation statement, in 6.3
* attestation statement format, in 6.3
* attestation statement format identifier, in 8.1
* attestation trust path, in 6.3.2
* attestation type, in 6.3

sameOriginWithAncestors) method need to take care to not leak information that could enable a malicious Relying Party to distinguish between these cases, where "named" means that the credential is listed by the Relying Party in allowCredentials:

* A named credential is not available.
* A named credential is available, but the user does not consent to use it.

If the above cases are distinguishable, information is leaked by which a malicious Relying Party could identify the user by probing for which credentials are available. For example, one such information leak is if the client returns a failure response as soon as the user denies consent to proceed with an authentication ceremony. In this case the Relying Party could detect that the ceremony was canceled by the user and not the timeout, and thus conclude that at least one of the credentials listed in the allowCredentials parameter is available to the user.

15. Acknowledgements

We thank the following people for their reviews of, and contributions to, this specification: Yuriy Ackermann, James Barclay, Richard Barnes, Dominic Battr, John Bradley, Domenic Denicola, Rahul Ghosh, Brad Hill, Jing Jin, Wally Jones, Ian Kilpatrick, Axel Nennker, Yoshikazu Nojima, Kimberly Paulhamus, Adam Powers, Yaron Sheffer, Anne van Kesteren, Johan Verrept, and Boris Zbarsky.

We thank Anthony Nadalin, John Fontana, and Richard Barnes for their contributions as co-chairs of the Web Authentication Working Group.

We also thank Wendy Seltzer, Samuel Weiler, and Harry Halpin for their contributions as our W3C Team Contacts.

Index

Terms defined by this specification

* aaguid, in 6.3.1
* AAGUID, in 10.4
* alg, in 5.3
* allowCredentials, in 5.5
* Anonymization CA, in 14.1
* appid
    + dict-member for AuthenticationExtensionsClientInputs, in 10.1
    + dict-member for AuthenticationExtensionsClientOutputs, in 10.1
* AppID, in 10.1
* Assertion, in 4
* assertion signature, in 6
* attachment modality, in 5.4.5
* AttCA, in 6.3.3
* Attestation, in 4
* attestation, in 5.4
* Attestation CA, in 6.3.3
* Attestation Certificate, in 4
* Attestation Conveyance, in 5.4.6
* AttestationConveyancePreference, in 5.4.6
* attestationConveyancePreferenceOption, in 5.1.3
* attestation key pair, in 4
* attestationObject, in 5.2.1
* attestation object, in 6.3
* attestationObjectResult, in 5.1.3
* attestation private key, in 4
* attestation public key, in 4
* attestation signature, in 6
* attestation statement, in 6.3
* attestation statement format, in 6.3
* attestation statement format identifier, in 8.1
* attestation trust path, in 6.3.2
* attestation type, in 6.3

* Attested credential data, in 6.3.1
* attestedCredentialData, in 6.1
* authDataExtensions, in 6.1
* Authentication, in 4
* Authentication Assertion, in 4
* authentication extension, in 9
* AuthenticationExtensionsAuthenticatorInputs
    + (typedef), in 5.9
    + definition of, in 5.9
* AuthenticationExtensionsClientInputs
    + (dictionary), in 5.7
    + definition of, in 5.7
* AuthenticationExtensionsClientOutputs
    + (dictionary), in 5.8
    + definition of, in 5.8
* AuthenticationExtensionsSupported, in 10.5
* Authenticator, in 4
* AuthenticatorAssertionResponse, in 5.2.2
* AuthenticatorAttachment, in 5.4.5
* authenticatorAttachment, in 5.4.4
* AuthenticatorAttestationResponse, in 5.2.1
* authenticatorBiometricPerfBounds, in 10.9
* authenticatorCancel, in 6.2.4
* authenticator data, in 6.1
* authenticatorData, in 5.2.2
* authenticator data claimed to have been used for the attestation, in 6.3.2
* authenticator data for the attestation, in 6.3.2
* authenticatorDataResult, in 5.1.4.1
* authenticator extension, in 9
* authenticator extension input, in 9.3
* authenticator extension output, in 9.5
* Authenticator extension processing, in 9.5
* authenticatorGetAssertion, in 6.2.3
* authenticatorMakeCredential, in 6.2.2
* Authenticator Model, in 6
* Authenticator operations, in 6.2
* AuthenticatorResponse, in 5.2
* authenticatorSelection, in 5.4
* AuthenticatorSelectionCriteria, in 5.4.4
* AuthenticatorSelectionList, in 10.4
* authenticator session, in 6.2
* AuthenticatorTransport, in 5.10.4
* authnSel
    + dict-member for AuthenticationExtensionsClientInputs, in 10.4
    + dict-member for AuthenticationExtensionsClientOutputs, in 10.4
* Authorization Gesture, in 4
* Base64url Encoding, in 3
* Basic, in 6.3.3
* Basic Attestation, in 6.3.3
* Biometric Authenticator, in 4
* Biometric Recognition, in 4
* ble, in 5.10.4
* CBOR, in 3
* Ceremony, in 4
* challenge
    + dict-member for CollectedClientData, in 5.10.1
    + dict-member for PublicKeyCredentialCreationOptions, in 5.4
    + dict-member for PublicKeyCredentialRequestOptions, in 5.5
* Client, in 4
* client data, in 5.10.1
* clientDataJSON, in 5.2
* clientDataJSONResult
    + dfn for assertionCreationData, in 5.1.4.1
    + dfn for credentialCreationData, in 5.1.3
* client extension, in 9
* client extension input, in 9.3
* client extension output, in 9.4
* Client extension processing, in 9.4

/Users/jehodges/Documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.txt, Top line: 5845

* Attested credential data, in 6.3.1
* attestedCredentialData, in 6.1
* authDataExtensions, in 6.1
* Authentication, in 4
* Authentication Assertion, in 4
* authentication extension, in 9
* AuthenticationExtensionsAuthenticatorInputs
    + (typedef), in 5.9
    + definition of, in 5.9
* AuthenticationExtensionsClientInputs
    + (dictionary), in 5.7
    + definition of, in 5.7
* AuthenticationExtensionsClientOutputs
    + (dictionary), in 5.8
    + definition of, in 5.8
* AuthenticationExtensionsSupported, in 10.5
* Authenticator, in 4
* AuthenticatorAssertionResponse, in 5.2.2
* AuthenticatorAttachment, in 5.4.5
* authenticatorAttachment, in 5.4.4
* AuthenticatorAttestationResponse, in 5.2.1
* authenticatorBiometricPerfBounds, in 10.9
* authenticatorCancel, in 6.2.4
* authenticator data, in 6.1
* authenticatorData, in 5.2.2
* authenticator data claimed to have been used for the attestation, in 6.3.2
* authenticator data for the attestation, in 6.3.2
* authenticatorDataResult, in 5.1.4.1
* authenticator extension, in 9
* authenticator extension input, in 9.3
* authenticator extension output, in 9.5
* Authenticator extension processing, in 9.5
* authenticatorGetAssertion, in 6.2.3
* authenticatorMakeCredential, in 6.2.2
* Authenticator Model, in 6
* Authenticator operations, in 6.2
* AuthenticatorResponse, in 5.2
* authenticatorSelection, in 5.4
* AuthenticatorSelectionCriteria, in 5.4.4
* AuthenticatorSelectionList, in 10.4
* authenticator session, in 6.2
* AuthenticatorTransport, in 5.10.4
* authnSel
    + dict-member for AuthenticationExtensionsClientInputs, in 10.4
    + dict-member for AuthenticationExtensionsClientOutputs, in 10.4
* Authorization Gesture, in 4
* Base64url Encoding, in 3
* Basic, in 6.3.3
* Basic Attestation, in 6.3.3
* Biometric Authenticator, in 4
* Biometric Recognition, in 4
* ble, in 5.10.4
* CBOR, in 3
* Ceremony, in 4
* challenge
    + dict-member for CollectedClientData, in 5.10.1
    + dict-member for PublicKeyCredentialCreationOptions, in 5.4
    + dict-member for PublicKeyCredentialRequestOptions, in 5.5
* Client, in 4
* client data, in 5.10.1
* clientDataJSON, in 5.2
* clientDataJSONResult
    + dfn for assertionCreationData, in 5.1.4.1
    + dfn for credentialCreationData, in 5.1.3
* client extension, in 9
* client extension input, in 9.3
* client extension output, in 9.4
* Client extension processing, in 9.4

```
       * clientExtensionResults
            + dfn for assertionCreationData, in 5.1.4.1
            + dfn for credentialCreationData, in 5.1.3
       * [[clientExtensionsResults]], in 5.1
       * Client-Side, in 4
       * client-side credential private key storage, in 4
       * Client-side-resident Credential Private Key, in 4
       * CollectedClientData, in 5.10.1
       * [[CollectFromCredentialStore]](origin, options,
         sameOriginWithAncestors), in 5.1.4
       * Conforming User Agent, in 4
       * content, in 10.3
       * contentType, in 10.3
       * COSEAlgorithmIdentifier
            + (typedef), in 5.10.5
            + definition of, in 5.10.5
       * [[Create]](origin, options, sameOriginWithAncestors), in 5.1.3
       * Credential ID, in 4
       * credentialId, in 6.3.1
       * credentialIdLength, in 6.3.1
       * credentialIdResult, in 5.1.4.1
       * credential key pair, in 4
       * credential private key, in 4
       * Credential Public Key, in 4
       * credentialPublicKey, in 6.3.1
       * credentials map, in 6
       * "cross-platform", in 5.4.5
       * cross-platform attached, in 5.4.5
       * cross-platform attachment, in 5.4.5
       * DAA, in 6.3.3
       * direct, in 5.4.6
       * "discouraged", in 5.10.6
       * [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors), in 5.1.4.1
       * [[discovery]], in 5.1
       * displayName, in 5.4.3
       * ECDAA, in 6.3.3
       * ECDAA-Issuer public key, in 8.2
       * effective user verification requirement for assertion, in 5.1.4.1
       * effective user verification requirement for credential creation, in
         5.1.3
       * Elliptic Curve based Direct Anonymous Attestation, in 6.3.3
       * excludeCredentials, in 5.4
       * extension identifier, in 9.1
       * extensions
            + dict-member for PublicKeyCredentialCreationOptions, in 5.4
            + dict-member for PublicKeyCredentialRequestOptions, in 5.5
       * exts
            + dict-member for AuthenticationExtensionsClientInputs, in 10.5
            + dict-member for AuthenticationExtensionsClientOutputs, in
              10.5
       * FAR, in 10.9
       * flags, in 6.1
       * FRR, in 10.9
       * getClientExtensionResults(), in 5.1
       * Hash of the serialized client data, in 5.10.1
       * Human Palatability, in 4
       * icon, in 5.4.1
       * id
            + dfn for public key credential source, in 4
            + dict-member for PublicKeyCredentialDescriptor, in 5.10.3
            + dict-member for PublicKeyCredentialRpEntity, in 5.4.2
            + dict-member for PublicKeyCredentialUserEntity, in 5.4.3
            + dict-member for TokenBinding, in 5.10.1
       * [[identifier]], in 5.1
       * identifier of the ECDAA-Issuer public key, in 8.2
       * indirect, in 5.4.6
       * internal, in 5.10.4
       * isUserVerifyingPlatformAuthenticatorAvailable(), in 5.1.7
       * JSON-serialized client data, in 5.10.1
```

```
       * clientExtensionResults
            + dfn for assertionCreationData, in 5.1.4.1
            + dfn for credentialCreationData, in 5.1.3
       * [[clientExtensionsResults]], in 5.1
       * Client-Side, in 4
       * client-side credential private key storage, in 4
       * Client-side-resident Credential Private Key, in 4
       * CollectedClientData, in 5.10.1
       * [[CollectFromCredentialStore]](origin, options,
         sameOriginWithAncestors), in 5.1.4
       * Conforming User Agent, in 4
       * content, in 10.3
       * contentType, in 10.3
       * COSEAlgorithmIdentifier
            + (typedef), in 5.10.5
            + definition of, in 5.10.5
       * [[Create]](origin, options, sameOriginWithAncestors), in 5.1.3
       * Credential ID, in 4
       * credentialId, in 6.3.1
       * credentialIdLength, in 6.3.1
       * credentialIdResult, in 5.1.4.1
       * credential key pair, in 4
       * credential private key, in 4
       * Credential Public Key, in 4
       * credentialPublicKey, in 6.3.1
       * credentials map, in 6
       * "cross-platform", in 5.4.5
       * cross-platform attached, in 5.4.5
       * cross-platform attachment, in 5.4.5
       * DAA, in 6.3.3
       * direct, in 5.4.6
       * "discouraged", in 5.10.6
       * [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors), in 5.1.4.1
       * [[discovery]], in 5.1
       * displayName, in 5.4.3
       * ECDAA, in 6.3.3
       * ECDAA-Issuer public key, in 8.2
       * effective user verification requirement for assertion, in 5.1.4.1
       * effective user verification requirement for credential creation, in
         5.1.3
       * Elliptic Curve based Direct Anonymous Attestation, in 6.3.3
       * excludeCredentials, in 5.4
       * extension identifier, in 9.1
       * extensions
            + dict-member for PublicKeyCredentialCreationOptions, in 5.4
            + dict-member for PublicKeyCredentialRequestOptions, in 5.5
       * exts
            + dict-member for AuthenticationExtensionsClientInputs, in 10.5
            + dict-member for AuthenticationExtensionsClientOutputs, in
              10.5
       * FAR, in 10.9
       * flags, in 6.1
       * FRR, in 10.9
       * getClientExtensionResults(), in 5.1
       * Hash of the serialized client data, in 5.10.1
       * Human Palatability, in 4
       * icon, in 5.4.1
       * id
            + dfn for public key credential source, in 4
            + dict-member for PublicKeyCredentialDescriptor, in 5.10.3
            + dict-member for PublicKeyCredentialRpEntity, in 5.4.2
            + dict-member for PublicKeyCredentialUserEntity, in 5.4.3
            + dict-member for TokenBinding, in 5.10.1
       * [[identifier]], in 5.1
       * identifier of the ECDAA-Issuer public key, in 8.2
       * indirect, in 5.4.6

       * isUserVerifyingPlatformAuthenticatorAvailable(), in 5.1.7
       * JSON-serialized client data, in 5.10.1
```

* loc
    + dict-member for AuthenticationExtensionsClientInputs, in 10.7
    + dict-member for AuthenticationExtensionsClientOutputs, in
      10.7
* looking up, in 6.2.1
* managing authenticator, in 4
* name, in 5.4.1
* nfc, in 5.10.4
* No attestation statement, in 6.3.3
* None, in 6.3.3
* none, in 5.4.6
* none attestation statement format, in 8.7
* "not-supported", in 5.10.1
* origin, in 5.10.1
* otherUI, in 4
* "platform", in 5.4.5
* platform attachment, in 5.4.5
* platform authenticators, in 5.4.5
* platform credential, in 5.4.5
* "preferred", in 5.10.6
* "present", in 5.10.1
* [[preventSilentAccess]](credential, sameOriginWithAncestors), in
  5.1.6
* privateKey, in 4
* pubKeyCredParams, in 5.4
* publicKey
    + dict-member for CredentialCreationOptions, in 5.1.1
    + dict-member for CredentialRequestOptions, in 5.1.2
* public-key, in 5.10.2
* Public Key Credential, in 4
* PublicKeyCredential, in 5.1
* PublicKeyCredentialCreationOptions, in 5.4
* PublicKeyCredentialDescriptor, in 5.10.3
* PublicKeyCredentialEntity, in 5.4.1
* PublicKeyCredentialParameters, in 5.3
* PublicKeyCredentialRequestOptions, in 5.5
* PublicKeyCredentialRpEntity, in 5.4.2
* Public Key Credential Source, in 4
* PublicKeyCredentialType, in 5.10.2
* PublicKeyCredentialUserEntity, in 5.4.3
* Rate Limiting, in 4
* rawId, in 5.1
* Registration, in 4
* registration extension, in 9
* Relying Party, in 4
* Relying Party Identifier, in 4
* "required", in 5.10.6
* requireResidentKey, in 5.4.4
* response, in 5.1
* roaming authenticators, in 5.4.5
* roaming credential, in 5.4.5
* rp, in 5.4
* rpId
    + dfn for public key credential source, in 4
    + dict-member for PublicKeyCredentialRequestOptions, in 5.5
* RP ID, in 4
* rpIdHash, in 6.1
* Self, in 6.3.3
* 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
* status, in 5.10.1
* [[Store]](credential, sameOriginWithAncestors), in 5.1.5
* "supported", in 5.10.1
* Test of User Presence, in 4
* timeout
    + dict-member for PublicKeyCredentialCreationOptions, in 5.4

* loc
    + dict-member for AuthenticationExtensionsClientInputs, in 10.7
    + dict-member for AuthenticationExtensionsClientOutputs, in
      10.7
* looking up, in 6.2.1
* managing authenticator, in 4
* name, in 5.4.1
* nfc, in 5.10.4
* No attestation statement, in 6.3.3
* None, in 6.3.3
* none, in 5.4.6
* none attestation statement format, in 8.7
* "not-supported", in 5.10.1
* origin, in 5.10.1
* otherUI, in 4
* "platform", in 5.4.5
* platform attachment, in 5.4.5
* platform authenticators, in 5.4.5
* platform credential, in 5.4.5
* "preferred", in 5.10.6
* "present", in 5.10.1
* [[preventSilentAccess]](credential, sameOriginWithAncestors), in
  5.1.6
* privateKey, in 4
* pubKeyCredParams, in 5.4
* publicKey
    + dict-member for CredentialCreationOptions, in 5.1.1
    + dict-member for CredentialRequestOptions, in 5.1.2
* public-key, in 5.10.2
* Public Key Credential, in 4
* PublicKeyCredential, in 5.1
* PublicKeyCredentialCreationOptions, in 5.4
* PublicKeyCredentialDescriptor, in 5.10.3
* PublicKeyCredentialEntity, in 5.4.1
* PublicKeyCredentialParameters, in 5.3
* PublicKeyCredentialRequestOptions, in 5.5
* PublicKeyCredentialRpEntity, in 5.4.2
* Public Key Credential Source, in 4
* PublicKeyCredentialType, in 5.10.2
* PublicKeyCredentialUserEntity, in 5.4.3
* Rate Limiting, in 4
* rawId, in 5.1
* Registration, in 4
* registration extension, in 9
* Relying Party, in 4
* Relying Party Identifier, in 4
* "required", in 5.10.6
* requireResidentKey, in 5.4.4
* response, in 5.1
* roaming authenticators, in 5.4.5
* roaming credential, in 5.4.5
* rp, in 5.4
* rpId
    + dfn for public key credential source, in 4
    + dict-member for PublicKeyCredentialRequestOptions, in 5.5
* RP ID, in 4
* rpIdHash, in 6.1
* Self, in 6.3.3
* 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
* status, in 5.10.1
* [[Store]](credential, sameOriginWithAncestors), in 5.1.5
* "supported", in 5.10.1
* Test of User Presence, in 4
* timeout
    + dict-member for PublicKeyCredentialCreationOptions, in 5.4

```
           + dict-member for PublicKeyCredentialRequestOptions, in 5.5
       * tokenBinding, in 5.10.1
       * TokenBinding, in 5.10.1
       * TokenBindingStatus, in 5.10.1
       * transports, in 5.10.3
       * txAuthGeneric
           + dict-member for AuthenticationExtensionsClientInputs, in 10.3
           + dict-member for AuthenticationExtensionsClientOutputs, in
             10.3
       * txAuthGenericArg, in 10.3
       * txAuthSimple
           + dict-member for AuthenticationExtensionsClientInputs, in 10.2
           + dict-member for AuthenticationExtensionsClientOutputs, in
             10.2
       * [[type]], in 5.1
       * type
           + dfn for public key credential source, in 4
           + dict-member for CollectedClientData, in 5.10.1
           + dict-member for PublicKeyCredentialDescriptor, in 5.10.3
           + dict-member for PublicKeyCredentialParameters, in 5.3
       * UP, in 4
       * usb, in 5.10.4
       * user, in 5.4
       * User Consent, in 4
       * userHandle
           + attribute for AuthenticatorAssertionResponse, in 5.2.2
           + dfn for public key credential source, in 4
       * User Handle, in 4
       * userHandleResult, in 5.1.4.1
       * User Present, in 4
       * User Public Key, in 4
       * userVerification
           + dict-member for AuthenticatorSelectionCriteria, in 5.4.4
           + dict-member for PublicKeyCredentialRequestOptions, in 5.5
       * User Verification, in 4
       * UserVerificationRequirement, in 5.10.6
       * User Verified, in 4
       * UV, in 4
       * uvi
           + dict-member for AuthenticationExtensionsClientInputs, in 10.6
           + dict-member for AuthenticationExtensionsClientOutputs, in
             10.6
       * uvm
           + dict-member for AuthenticationExtensionsClientInputs, in 10.8
           + dict-member for AuthenticationExtensionsClientOutputs, in
             10.8
       * UvmEntries, in 10.8
       * UvmEntry, in 10.8
       * Verification procedure, in 6.3.2
       * verification procedure inputs, in 6.3.2
       * Web Authentication API, in 5
       * WebAuthn Client, in 4
```

```
     https://w3c.github.io/webappsec-credential-management/#credentialRefere
     nced in:
       * 3. Dependencies
       * 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6)

     https://w3c.github.io/webappsec-credential-management/#dictdef-credenti
     alcreationoptionsReferenced in:
       * 5.1.1. CredentialCreationOptions Dictionary Extension (2)
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method

     https://w3c.github.io/webappsec-credential-management/#dictdef-credenti
     alrequestoptionsReferenced in:
       * 5.1.2. CredentialRequestOptions Dictionary Extension (2)
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
```

```
           + dict-member for PublicKeyCredentialRequestOptions, in 5.5
       * tokenBinding, in 5.10.1
       * TokenBinding, in 5.10.1
       * TokenBindingStatus, in 5.10.1
       * transports, in 5.10.3
       * txAuthGeneric
           + dict-member for AuthenticationExtensionsClientInputs, in 10.3
           + dict-member for AuthenticationExtensionsClientOutputs, in
             10.3
       * txAuthGenericArg, in 10.3
       * txAuthSimple
           + dict-member for AuthenticationExtensionsClientInputs, in 10.2
           + dict-member for AuthenticationExtensionsClientOutputs, in
             10.2
       * [[type]], in 5.1
       * type
           + dfn for public key credential source, in 4
           + dict-member for CollectedClientData, in 5.10.1
           + dict-member for PublicKeyCredentialDescriptor, in 5.10.3
           + dict-member for PublicKeyCredentialParameters, in 5.3
       * UP, in 4
       * usb, in 5.10.4
       * user, in 5.4
       * User Consent, in 4
       * userHandle
           + attribute for AuthenticatorAssertionResponse, in 5.2.2
           + dfn for public key credential source, in 4
       * User Handle, in 4
       * userHandleResult, in 5.1.4.1
       * User Present, in 4
       * User Public Key, in 4
       * userVerification
           + dict-member for AuthenticatorSelectionCriteria, in 5.4.4
           + dict-member for PublicKeyCredentialRequestOptions, in 5.5
       * User Verification, in 4
       * UserVerificationRequirement, in 5.10.6
       * User Verified, in 4
       * UV, in 4
       * uvi
           + dict-member for AuthenticationExtensionsClientInputs, in 10.6
           + dict-member for AuthenticationExtensionsClientOutputs, in
             10.6
       * uvm
           + dict-member for AuthenticationExtensionsClientInputs, in 10.8
           + dict-member for AuthenticationExtensionsClientOutputs, in
             10.8
       * UvmEntries, in 10.8
       * UvmEntry, in 10.8
       * Verification procedure, in 6.3.2
       * verification procedure inputs, in 6.3.2
       * Web Authentication API, in 5
       * WebAuthn Client, in 4

     https://w3c.github.io/webappsec-credential-management/#credentialRefere
     nced in:
       * 3. Dependencies
       * 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6)

     https://w3c.github.io/webappsec-credential-management/#dictdef-credenti
     alcreationoptionsReferenced in:
       * 5.1.1. CredentialCreationOptions Dictionary Extension (2)
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method

     https://w3c.github.io/webappsec-credential-management/#dictdef-credenti
     alrequestoptionsReferenced in:
       * 5.1.2. CredentialRequestOptions Dictionary Extension (2)
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method

       https://w3c.github.io/webappsec-credential-management/#credentialsconta
       inerReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method
         * 5.4. Options for Credential Creation (dictionary
           PublicKeyCredentialCreationOptions)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)

       https://w3c.github.io/webappsec-credential-management/#abstract-opdef-r
       equest-a-credentialReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method

       https://w3c.github.io/webappsec-credential-management/#collectfromcrede
       ntialstore-origin-options-sameoriginwithancestorsReferenced in:
         * 5.1. PublicKeyCredential Interface
         * 5.1.4. Use an existing credential to make an assertion -
           PublicKeyCredential's [[Get]](options) method

       https://w3c.github.io/webappsec-credential-management/#create-origin-op
       tions-sameoriginwithancestorsReferenced in:
         * 5.6. Abort operations with AbortSignal

       https://w3c.github.io/webappsec-credential-management/#store-credential
       -sameoriginwithancestorsReferenced in:
         * 5.1. PublicKeyCredential Interface

       https://w3c.github.io/webappsec-credential-management/#dom-credential-d
       iscovery-slotReferenced in:
         * 5.1. PublicKeyCredential Interface

       https://w3c.github.io/webappsec-credential-management/#dom-credential-t
       ype-slotReferenced in:
         * 5.1. PublicKeyCredential Interface

       https://w3c.github.io/webappsec-credential-management/#dom-credentialsc
       ontainer-createReferenced in:
         * 1. Introduction
         * 5.1. PublicKeyCredential Interface (2)
         * 5.1.1. CredentialCreationOptions Dictionary Extension
         * 5.1.3. Create a new credential - PublicKeyCredential's
           [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
           (4) (5)
         * 5.2. Authenticator Responses (interface AuthenticatorResponse)
         * 5.4. Options for Credential Creation (dictionary
           PublicKeyCredentialCreationOptions)
         * 5.4.4. Authenticator Selection Criteria (dictionary
           AuthenticatorSelectionCriteria)
         * 5.10.3. Credential Descriptor (dictionary
           PublicKeyCredentialDescriptor)
         * 7. Relying Party Operations
         * 7.1. Registering a new credential (2) (3)
         * 9. WebAuthn Extensions (2) (3)
         * 9.2. Defining extensions
         * 9.3. Extending request parameters (2)
         * 10.1. FIDO AppID Extension (appid)
         * 10.4. Authenticator Selection Extension (authnSel)
         * 10.9. Biometric Authenticator Performance Bounds Extension
           (biometricPerfBounds)

       https://w3c.github.io/webappsec-credential-management/#concept-credenti
       alReferenced in:
         * 4. Terminology
         * 5.1.4. Use an existing credential to make an assertion -
           PublicKeyCredential's [[Get]](options) method (2) (3)

```

```

       https://w3c.github.io/webappsec-credential-management/#credentialsconta
       inerReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method
         * 5.4. Options for Credential Creation (dictionary
           PublicKeyCredentialCreationOptions)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)

       https://w3c.github.io/webappsec-credential-management/#abstract-opdef-r
       equest-a-credentialReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method

       https://w3c.github.io/webappsec-credential-management/#collectfromcrede
       ntialstore-origin-options-sameoriginwithancestorsReferenced in:
         * 5.1. PublicKeyCredential Interface
         * 5.1.4. Use an existing credential to make an assertion -
           PublicKeyCredential's [[Get]](options) method

       https://w3c.github.io/webappsec-credential-management/#create-origin-op
       tions-sameoriginwithancestorsReferenced in:
         * 5.6. Abort operations with AbortSignal

       https://w3c.github.io/webappsec-credential-management/#store-credential
       -sameoriginwithancestorsReferenced in:
         * 5.1. PublicKeyCredential Interface

       https://w3c.github.io/webappsec-credential-management/#dom-credential-d
       iscovery-slotReferenced in:
         * 5.1. PublicKeyCredential Interface

       https://w3c.github.io/webappsec-credential-management/#dom-credential-t
       ype-slotReferenced in:
         * 5.1. PublicKeyCredential Interface

       https://w3c.github.io/webappsec-credential-management/#dom-credentialsc
       ontainer-createReferenced in:
         * 1. Introduction
         * 5.1. PublicKeyCredential Interface (2)
         * 5.1.1. CredentialCreationOptions Dictionary Extension
         * 5.1.3. Create a new credential - PublicKeyCredential's
           [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
           (4) (5)
         * 5.2. Authenticator Responses (interface AuthenticatorResponse)
         * 5.4. Options for Credential Creation (dictionary
           PublicKeyCredentialCreationOptions)
         * 5.4.4. Authenticator Selection Criteria (dictionary
           AuthenticatorSelectionCriteria)
         * 5.10.3. Credential Descriptor (dictionary
           PublicKeyCredentialDescriptor)
         * 7. Relying Party Operations
         * 7.1. Registering a new credential (2) (3)
         * 9. WebAuthn Extensions (2) (3)
         * 9.2. Defining extensions
         * 9.3. Extending request parameters (2)
         * 10.1. FIDO AppID Extension (appid)
         * 10.4. Authenticator Selection Extension (authnSel)
         * 10.9. Biometric Authenticator Performance Bounds Extension
           (biometricPerfBounds)

       https://w3c.github.io/webappsec-credential-management/#concept-credenti
       alReferenced in:
         * 4. Terminology
         * 5.1.4. Use an existing credential to make an assertion -
           PublicKeyCredential's [[Get]](options) method (2) (3)

```

```
    https://w3c.github.io/webappsec-credential-management/#credential-sourc
    eReferenced in:
      * 4. Terminology
      * 5.1.4. Use an existing credential to make an assertion -
        PublicKeyCredential's [[Get]](options) method (2) (3) (4) (5)

    https://w3c.github.io/webappsec-credential-management/#dom-credentialsc
    ontainer-getReferenced in:
      * 1. Introduction
      * 4. Terminology
      * 5.1. PublicKeyCredential Interface (2)
      * 5.1.2. CredentialRequestOptions Dictionary Extension
      * 5.1.4. Use an existing credential to make an assertion -
        PublicKeyCredential's [[Get]](options) method (2)
      * 5.1.4.1. PublicKeyCredential's
        [[DiscoverFromExternalSource]](origin, options,
        sameOriginWithAncestors) method (2) (3)
      * 5.2. Authenticator Responses (interface AuthenticatorResponse)
      * 5.5. Options for Assertion Generation (dictionary
        PublicKeyCredentialRequestOptions) (2)
      * 5.10.3. Credential Descriptor (dictionary
        PublicKeyCredentialDescriptor)
      * 7. Relying Party Operations
      * 7.2. Verifying an authentication assertion (2)
      * 9. WebAuthn Extensions (2) (3)
      * 9.2. Defining extensions
      * 9.3. Extending request parameters (2)

    https://w3c.github.io/webappsec-credential-management/#dom-credential-i
    dReferenced in:
      * 5.1. PublicKeyCredential Interface
      * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)

    https://w3c.github.io/webappsec-credential-management/#dom-credential-d
    iscovery-remoteReferenced in:
      * 5.1. PublicKeyCredential Interface

    https://w3c.github.io/webappsec-credential-management/#same-origin-with
    -its-ancestorsReferenced 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 (2)
      * 5.1.5. Store an existing credential - PublicKeyCredential's
        [[Store]](credential, sameOriginWithAncestors) method

    https://w3c.github.io/webappsec-credential-management/#dom-credentialre
    questoptions-signalReferenced in:
      * 5.1.4.1. PublicKeyCredential's
        [[DiscoverFromExternalSource]](origin, options,
        sameOriginWithAncestors) method (2)

    https://w3c.github.io/webappsec-credential-management/#dom-credentialsc
    ontainer-storeReferenced in:
      * 5.1.5. Store an existing credential - PublicKeyCredential's
        [[Store]](credential, sameOriginWithAncestors) method

    https://w3c.github.io/webappsec-credential-management/#dom-credential-t
    ypeReferenced in:
      * 5.1. PublicKeyCredential Interface

    https://w3c.github.io/webappsec-credential-management/#user-mediatedRef
    erenced in:
      * 5.1.4. Use an existing credential to make an assertion -
        PublicKeyCredential's [[Get]](options) method

    https://dom.spec.whatwg.org/#abortcontrollerReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method
```

    https://w3c.github.io/webappsec-credential-management/#credential-sourc
    eReferenced in:
      * 4. Terminology
      * 5.1.4. Use an existing credential to make an assertion -
        PublicKeyCredential's [[Get]](options) method (2) (3) (4) (5)

    https://w3c.github.io/webappsec-credential-management/#dom-credentialsc
    ontainer-getReferenced in:
      * 1. Introduction
      * 4. Terminology
      * 5.1. PublicKeyCredential Interface (2)
      * 5.1.2. CredentialRequestOptions Dictionary Extension
      * 5.1.4. Use an existing credential to make an assertion -
        PublicKeyCredential's [[Get]](options) method (2)
      * 5.1.4.1. PublicKeyCredential's
        [[DiscoverFromExternalSource]](origin, options,
        sameOriginWithAncestors) method (2) (3)
      * 5.2. Authenticator Responses (interface AuthenticatorResponse)
      * 5.5. Options for Assertion Generation (dictionary
        PublicKeyCredentialRequestOptions) (2)
      * 5.10.3. Credential Descriptor (dictionary
        PublicKeyCredentialDescriptor)
      * 7. Relying Party Operations
      * 7.2. Verifying an authentication assertion (2)
      * 9. WebAuthn Extensions (2) (3)
      * 9.2. Defining extensions
      * 9.3. Extending request parameters (2)

    https://w3c.github.io/webappsec-credential-management/#dom-credential-i
    dReferenced in:
      * 5.1. PublicKeyCredential Interface
      * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)

    https://w3c.github.io/webappsec-credential-management/#dom-credential-d
    iscovery-remoteReferenced in:
      * 5.1. PublicKeyCredential Interface

    https://w3c.github.io/webappsec-credential-management/#same-origin-with
    -its-ancestorsReferenced 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 (2)
      * 5.1.5. Store an existing credential - PublicKeyCredential's
        [[Store]](credential, sameOriginWithAncestors) method

    https://w3c.github.io/webappsec-credential-management/#dom-credentialre
    questoptions-signalReferenced in:
      * 5.1.4.1. PublicKeyCredential's
        [[DiscoverFromExternalSource]](origin, options,
        sameOriginWithAncestors) method (2)

    https://w3c.github.io/webappsec-credential-management/#dom-credentialsc
    ontainer-storeReferenced in:
      * 5.1.5. Store an existing credential - PublicKeyCredential's
        [[Store]](credential, sameOriginWithAncestors) method

    https://w3c.github.io/webappsec-credential-management/#dom-credential-t
    ypeReferenced in:
      * 5.1. PublicKeyCredential Interface

    https://w3c.github.io/webappsec-credential-management/#user-mediatedRef
    erenced in:
      * 5.1.4. Use an existing credential to make an assertion -
        PublicKeyCredential's [[Get]](options) method

    https://dom.spec.whatwg.org/#abortcontrollerReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method

```
     * 5.6. Abort operations with AbortSignal (2)

   https://dom.spec.whatwg.org/#abortsignal-aborted-flagReferenced 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 (2)
     * 5.6. Abort operations with AbortSignal (2)

   https://dom.spec.whatwg.org/#concept-documentReferenced in:
     * 5.6. Abort operations with AbortSignal

   https://tc39.github.io/ecma262/#sec-arraybuffer-constructorReferenced
   in:
     * 3. Dependencies
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2) (3) (4) (5) (6)
     * 9. WebAuthn Extensions (2)

   https://tc39.github.io/ecma262/#sec-object-internal-methods-and-interna
   l-slotsReferenced in:
     * 4. Terminology
     * 5.1. PublicKeyCredential Interface (2) (3) (4) (5)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 5.1.4. Use an existing credential to make an assertion -
       PublicKeyCredential's [[Get]](options) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.1.5. Store an existing credential - PublicKeyCredential's
       [[Store]](credential, sameOriginWithAncestors) method
     * 5.1.6. Preventing silent access to an existing credential -
       PublicKeyCredential's [[preventSilentAccess]](credential,
       sameOriginWithAncestors) method

   https://tc39.github.io/ecma262/#sec-object-internal-methods-and-interna
   l-slotsReferenced in:
     * 4. Terminology
     * 5.1. PublicKeyCredential Interface (2) (3) (4) (5)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 5.1.4. Use an existing credential to make an assertion -
       PublicKeyCredential's [[Get]](options) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.1.5. Store an existing credential - PublicKeyCredential's
       [[Store]](credential, sameOriginWithAncestors) method
     * 5.1.6. Preventing silent access to an existing credential -
       PublicKeyCredential's [[preventSilentAccess]](credential,
       sameOriginWithAncestors) method

   https://tc39.github.io/ecma262/#sec-json.stringifyReferenced in:
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)

   https://encoding.spec.whatwg.org/#utf-8-decodeReferenced in:
     * 7.1. Registering a new credential (2) (3)
     * 7.2. Verifying an authentication assertion (2) (3)

   https://encoding.spec.whatwg.org/#utf-8-encodeReferenced in:
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)
     * 8.5. Android SafetyNet Attestation Statement Format
     * 10.5. Supported Extensions Extension (exts)
```

```
     * 5.6. Abort operations with AbortSignal (2)

   https://dom.spec.whatwg.org/#abortsignal-aborted-flagReferenced 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 (2)
     * 5.6. Abort operations with AbortSignal (2)

   https://dom.spec.whatwg.org/#concept-documentReferenced in:
     * 5.6. Abort operations with AbortSignal

   https://tc39.github.io/ecma262/#sec-arraybuffer-constructorReferenced
   in:
     * 3. Dependencies
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2) (3) (4) (5) (6)
     * 9. WebAuthn Extensions (2)

   https://tc39.github.io/ecma262/#sec-object-internal-methods-and-interna
   l-slotsReferenced in:
     * 4. Terminology
     * 5.1. PublicKeyCredential Interface (2) (3) (4) (5)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 5.1.4. Use an existing credential to make an assertion -
       PublicKeyCredential's [[Get]](options) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.1.5. Store an existing credential - PublicKeyCredential's
       [[Store]](credential, sameOriginWithAncestors) method
     * 5.1.6. Preventing silent access to an existing credential -
       PublicKeyCredential's [[preventSilentAccess]](credential,
       sameOriginWithAncestors) method

   https://tc39.github.io/ecma262/#sec-object-internal-methods-and-interna
   l-slotsReferenced in:
     * 4. Terminology
     * 5.1. PublicKeyCredential Interface (2) (3) (4) (5)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 5.1.4. Use an existing credential to make an assertion -
       PublicKeyCredential's [[Get]](options) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.1.5. Store an existing credential - PublicKeyCredential's
       [[Store]](credential, sameOriginWithAncestors) method
     * 5.1.6. Preventing silent access to an existing credential -
       PublicKeyCredential's [[preventSilentAccess]](credential,
       sameOriginWithAncestors) method

   https://tc39.github.io/ecma262/#sec-json.stringifyReferenced in:
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)

   https://encoding.spec.whatwg.org/#utf-8-decodeReferenced in:
     * 7.1. Registering a new credential (2) (3)
     * 7.2. Verifying an authentication assertion (2) (3)

   https://encoding.spec.whatwg.org/#utf-8-encodeReferenced in:
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)
     * 8.5. Android SafetyNet Attestation Statement Format
     * 10.5. Supported Extensions Extension (exts)

   https://fetch.spec.whatwg.org/#concept-request-windowReferenced in:
     * 5.6. Abort operations with AbortSignal (2)

   https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-appid-and
   -facets-v1.2-ps-20170411.html#determining-if-a-caller-s-facetid-is-auth
   orized-for-an-appidReferenced in:
     * 3. Dependencies
     * 10.1. FIDO AppID Extension (appid)

   https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-appid-and
   -facets-v1.2-ps-20170411.html#determining-the-facetid-of-a-calling-appl
   icationReferenced in:
     * 3. Dependencies
     * 10.1. FIDO AppID Extension (appid)

   https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client-to-aut
   henticator-protocol-v2.0-ps-20170927.html#ctap2-canonical-cbor-encoding
   -formReferenced in:
     * 2.4. All Conformance Classes (2)
     * 3. Dependencies
     * 6.3.1. Attested credential data
     * 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE_Key
       format
     * 9. WebAuthn Extensions

   https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0
   -ps-20170927.html#user-verification-methodsReferenced in:
     * 10.8. User Verification Method Extension (uvm)

   https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0
   -ps-20170927.html#key-protection-typesReferenced in:
     * 10.8. User Verification Method Extension (uvm)

   https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0
   -ps-20170927.html#matcher-protection-typesReferenced in:
     * 10.8. User Verification Method Extension (uvm)

   https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0
   -ps-20170927.html#public-key-representation-formatsReferenced in:
     * 8.6. FIDO U2F Attestation Statement Format

   https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m
   essage-formats-v1.1-id-20160915.html#authentication-request-message---u
   2f_authenticateReferenced in:
     * 6.1.2. FIDO U2F signature format compatibility

   https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m
   essage-formats-v1.1-id-20160915.html#registration-response-message-succ
   essReferenced in:
     * 8.6. FIDO U2F Attestation Statement Format (2)

   https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m
   essage-formats-v1.1-id-20160915.html#authentication-response-message-su
   ccessReferenced in:
     * 6.1.2. FIDO U2F signature format compatibility

   https://dev.w3.org/geo/api/spec-source.html#coordinates_interfaceRefere
   nced in:
     * 10.7. Location Extension (loc) (2) (3)

   https://html.spec.whatwg.org/multipage/origin.html#ascii-serialisation-
   of-an-originReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://html.spec.whatwg.org/multipage/origin.html#concept-origin-effec
```

/Users/jehodges/Documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.txt, Top line: 6334

```

   https://fetch.spec.whatwg.org/#concept-request-windowReferenced in:
     * 5.6. Abort operations with AbortSignal (2)

   https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-appid-and
   -facets-v1.2-ps-20170411.html#determining-if-a-caller-s-facetid-is-auth
   orized-for-an-appidReferenced in:
     * 3. Dependencies
     * 10.1. FIDO AppID Extension (appid)

   https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-appid-and
   -facets-v1.2-ps-20170411.html#determining-the-facetid-of-a-calling-appl
   icationReferenced in:
     * 3. Dependencies
     * 10.1. FIDO AppID Extension (appid)

   https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client-to-aut
   henticator-protocol-v2.0-ps-20170927.html#ctap2-canonical-cbor-encoding
   -formReferenced in:
     * 2.4. All Conformance Classes (2)
     * 3. Dependencies
     * 6.3.1. Attested credential data
     * 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE_Key
       format
     * 9. WebAuthn Extensions

   https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0
   -ps-20170927.html#user-verification-methodsReferenced in:
     * 10.8. User Verification Method Extension (uvm)

   https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0
   -ps-20170927.html#key-protection-typesReferenced in:
     * 10.8. User Verification Method Extension (uvm)

   https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0
   -ps-20170927.html#matcher-protection-typesReferenced in:
     * 10.8. User Verification Method Extension (uvm)

   https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0
   -ps-20170927.html#public-key-representation-formatsReferenced in:
     * 8.6. FIDO U2F Attestation Statement Format

   https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m
   essage-formats-v1.1-id-20160915.html#authentication-request-message---u
   2f_authenticateReferenced in:
     * 6.1.2. FIDO U2F signature format compatibility

   https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m
   essage-formats-v1.1-id-20160915.html#registration-response-message-succ
   essReferenced in:
     * 8.6. FIDO U2F Attestation Statement Format (2)

   https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m
   essage-formats-v1.1-id-20160915.html#authentication-response-message-su
   ccessReferenced in:
     * 6.1.2. FIDO U2F signature format compatibility

   https://dev.w3.org/geo/api/spec-source.html#coordinates_interfaceRefere
   nced in:
     * 10.7. Location Extension (loc) (2) (3)

   https://html.spec.whatwg.org/multipage/origin.html#ascii-serialisation-
   of-an-originReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://html.spec.whatwg.org/multipage/origin.html#concept-origin-effec
   tive-domainReferenced in:
     * 4. Terminology (2) (3) (4)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
       (4)
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2) (3) (4)
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions)
     * 6.1. Authenticator data (2)

   https://html.spec.whatwg.org/multipage/webappapis.html#environment-sett
   ings-objectReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.1.5. Store an existing credential - PublicKeyCredential's
       [[Store]](credential, sameOriginWithAncestors) method

   https://html.spec.whatwg.org/multipage/webappapis.html#concept-settings
   -object-globalReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://html.spec.whatwg.org/multipage/origin.html#is-a-registrable-dom
   ain-suffix-of-or-is-equal-toReferenced in:
     * 3. Dependencies
     * 4. Terminology
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 6.1. Authenticator data

   https://html.spec.whatwg.org/multipage/origin.html#is-a-registrable-dom
   ain-suffix-of-or-is-equal-toReferenced in:
     * 3. Dependencies
     * 4. Terminology
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 6.1. Authenticator data

   https://html.spec.whatwg.org/multipage/webappapis.html#concept-settings
   -object-originReferenced in:
     * 4. Terminology (2)
     * 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 (2)
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions)

   https://html.spec.whatwg.org/multipage/webappapis.html#relevant-setting
   s-objectReferenced in:
     * 3. Dependencies
```

```
   tive-domainReferenced in:
     * 4. Terminology (2) (3) (4)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
       (4)
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2) (3) (4)
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions)
     * 6.1. Authenticator data (2)

   https://html.spec.whatwg.org/multipage/webappapis.html#environment-sett
   ings-objectReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.1.5. Store an existing credential - PublicKeyCredential's
       [[Store]](credential, sameOriginWithAncestors) method

   https://html.spec.whatwg.org/multipage/webappapis.html#concept-settings
   -object-globalReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://html.spec.whatwg.org/multipage/origin.html#is-a-registrable-dom
   ain-suffix-of-or-is-equal-toReferenced in:
     * 3. Dependencies
     * 4. Terminology
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 6.1. Authenticator data

   https://html.spec.whatwg.org/multipage/origin.html#is-a-registrable-dom
   ain-suffix-of-or-is-equal-toReferenced in:
     * 3. Dependencies
     * 4. Terminology
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 6.1. Authenticator data

   https://html.spec.whatwg.org/multipage/webappapis.html#concept-settings
   -object-originReferenced in:
     * 4. Terminology (2)
     * 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 (2)
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions)

   https://html.spec.whatwg.org/multipage/webappapis.html#relevant-setting
   s-objectReferenced in:
     * 3. Dependencies
```

```
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method
  * 5.1.4.1. PublicKeyCredential's
    [[DiscoverFromExternalSource]](origin, options,
    sameOriginWithAncestors) method
  * 5.4. Options for Credential Creation (dictionary
    PublicKeyCredentialCreationOptions)
  * 5.5. Options for Assertion Generation (dictionary
    PublicKeyCredentialRequestOptions)

https://w3c.github.io/html/browsers.html#dom-document-domainReferenced
in:
  * 4. Terminology

https://w3c.github.io/html/browsers.html#opaque-originReferenced in:
  * 3. Dependencies
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method
  * 5.1.4.1. PublicKeyCredential's
    [[DiscoverFromExternalSource]](origin, options,
    sameOriginWithAncestors) method

https://w3c.github.io/html/browsers.html#concept-cross-originReferenced
in:
  * 1. Introduction
  * 3. Dependencies
  * 4. Terminology (2) (3)
  * 5. Web Authentication API (2) (3) (4) (5) (6) (7)
  * 5.10.1. Client data used in WebAuthn signatures (dictionary
    CollectedClientData)
  * 6.1. Authenticator data
  * 7.1. Registering a new credential
  * 7.2. Verifying an authentication assertion
  * 10.1. FIDO AppID Extension (appid)
  * 12.3. Authentication
  * 14.1. Attestation Privacy (2)

https://infra.spec.whatwg.org/#set-appendReferenced in:
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method
  * 5.1.4.1. PublicKeyCredential's
    [[DiscoverFromExternalSource]](origin, options,
    sameOriginWithAncestors) method (2)
  * 6.2.3. The authenticatorGetAssertion operation (2)

https://infra.spec.whatwg.org/#byte-sequenceReferenced in:
  * 4. Terminology (2)

https://infra.spec.whatwg.org/#iteration-continueReferenced in:
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
    (4) (5) (6) (7) (8)
  * 5.1.4.1. PublicKeyCredential's
    [[DiscoverFromExternalSource]](origin, options,
    sameOriginWithAncestors) method (2) (3) (4) (5)

https://infra.spec.whatwg.org/#map-iterateReferenced in:
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method
  * 5.1.4.1. PublicKeyCredential's
    [[DiscoverFromExternalSource]](origin, options,
    sameOriginWithAncestors) method
  * 6.2.1. Lookup Credential Source by Credential ID algorithm
  * 6.2.2. The authenticatorMakeCredential operation
  * 6.2.3. The authenticatorGetAssertion operation (2)

https://infra.spec.whatwg.org/#list-is-emptyReferenced in:
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
  * 5.1.4.1. PublicKeyCredential's
```

```
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method
  * 5.1.4.1. PublicKeyCredential's
    [[DiscoverFromExternalSource]](origin, options,
    sameOriginWithAncestors) method
  * 5.4. Options for Credential Creation (dictionary
    PublicKeyCredentialCreationOptions)
  * 5.5. Options for Assertion Generation (dictionary
    PublicKeyCredentialRequestOptions)

https://w3c.github.io/html/browsers.html#dom-document-domainReferenced
in:
  * 4. Terminology

https://w3c.github.io/html/browsers.html#opaque-originReferenced in:
  * 3. Dependencies
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method
  * 5.1.4.1. PublicKeyCredential's
    [[DiscoverFromExternalSource]](origin, options,
    sameOriginWithAncestors) method

https://w3c.github.io/html/browsers.html#concept-cross-originReferenced
in:
  * 1. Introduction
  * 3. Dependencies
  * 4. Terminology (2) (3)
  * 5. Web Authentication API (2) (3) (4) (5) (6) (7)
  * 5.10.1. Client data used in WebAuthn signatures (dictionary
    CollectedClientData)
  * 6.1. Authenticator data
  * 7.1. Registering a new credential
  * 7.2. Verifying an authentication assertion
  * 10.1. FIDO AppID Extension (appid)
  * 12.3. Authentication
  * 14.1. Attestation Privacy (2)

https://infra.spec.whatwg.org/#set-appendReferenced in:
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method
  * 5.1.4.1. PublicKeyCredential's
    [[DiscoverFromExternalSource]](origin, options,
    sameOriginWithAncestors) method (2)
  * 6.2.3. The authenticatorGetAssertion operation (2)

https://infra.spec.whatwg.org/#byte-sequenceReferenced in:
  * 4. Terminology (2)

https://infra.spec.whatwg.org/#iteration-continueReferenced in:
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
    (4) (5) (6) (7) (8)
  * 5.1.4.1. PublicKeyCredential's
    [[DiscoverFromExternalSource]](origin, options,
    sameOriginWithAncestors) method (2) (3) (4) (5)

https://infra.spec.whatwg.org/#map-iterateReferenced in:
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method
  * 5.1.4.1. PublicKeyCredential's
    [[DiscoverFromExternalSource]](origin, options,
    sameOriginWithAncestors) method
  * 6.2.1. Lookup Credential Source by Credential ID algorithm
  * 6.2.2. The authenticatorMakeCredential operation
  * 6.2.3. The authenticatorGetAssertion operation (2)

https://infra.spec.whatwg.org/#list-is-emptyReferenced in:
  * 5.1.3. Create a new credential - PublicKeyCredential's
    [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
  * 5.1.4.1. PublicKeyCredential's
```

```
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method (2) (3) (4) (5)

   https://infra.spec.whatwg.org/#list-is-emptyReferenced in:
    * 5.1.3. Create a new credential - PublicKeyCredential's
      [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method (2) (3) (4) (5)

   https://infra.spec.whatwg.org/#struct-itemReferenced in:
    * 4. Terminology (2)
    * 5.1.3. Create a new credential - PublicKeyCredential's
      [[Create]](origin, options, sameOriginWithAncestors) method
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method

   https://infra.spec.whatwg.org/#listReferenced 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
    * 6.2.3. The authenticatorGetAssertion operation

   https://infra.spec.whatwg.org/#ordered-mapReferenced in:
    * 5.1. PublicKeyCredential Interface
    * 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 (2)
    * 6. WebAuthn Authenticator Model
    * 6.2.2. The authenticatorMakeCredential operation
    * 6.2.3. The authenticatorGetAssertion operation
    * 9.4. Client extension processing

   https://infra.spec.whatwg.org/#ordered-setReferenced in:
    * 5.1.3. Create a new credential - PublicKeyCredential's
      [[Create]](origin, options, sameOriginWithAncestors) method
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method (2) (3)
    * 6.2.3. The authenticatorGetAssertion operation

   https://infra.spec.whatwg.org/#list-removeReferenced in:
    * 5.1.3. Create a new credential - PublicKeyCredential's
      [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
      (4) (5) (6) (7) (8) (9)
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7)
    * 6.2.3. The authenticatorGetAssertion operation

   https://infra.spec.whatwg.org/#map-setReferenced 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 (2)
    * 6.2.2. The authenticatorMakeCredential operation

   https://infra.spec.whatwg.org/#structReferenced in:
    * 4. Terminology
    * 5.1.3. Create a new credential - PublicKeyCredential's
      [[Create]](origin, options, sameOriginWithAncestors) method
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method
```

```
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method (2) (3) (4) (5)

   https://infra.spec.whatwg.org/#list-is-emptyReferenced in:
    * 5.1.3. Create a new credential - PublicKeyCredential's
      [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method (2) (3) (4) (5)

   https://infra.spec.whatwg.org/#struct-itemReferenced in:
    * 4. Terminology (2)
    * 5.1.3. Create a new credential - PublicKeyCredential's
      [[Create]](origin, options, sameOriginWithAncestors) method
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method

   https://infra.spec.whatwg.org/#listReferenced 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
    * 6.2.3. The authenticatorGetAssertion operation

   https://infra.spec.whatwg.org/#ordered-mapReferenced in:
    * 5.1. PublicKeyCredential Interface
    * 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 (2) (3)
    * 6. WebAuthn Authenticator Model
    * 6.2.2. The authenticatorMakeCredential operation
    * 6.2.3. The authenticatorGetAssertion operation
    * 9.4. Client extension processing

   https://infra.spec.whatwg.org/#ordered-setReferenced 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 (2) (3) (4)
    * 6.2.3. The authenticatorGetAssertion operation

   https://infra.spec.whatwg.org/#list-removeReferenced in:
    * 5.1.3. Create a new credential - PublicKeyCredential's
      [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
      (4) (5) (6) (7) (8) (9) (10)
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7) (8) (9)
    * 6.2.3. The authenticatorGetAssertion operation

   https://infra.spec.whatwg.org/#map-setReferenced 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 (2)
    * 6.2.2. The authenticatorMakeCredential operation

   https://infra.spec.whatwg.org/#structReferenced in:
    * 4. Terminology
    * 5.1.3. Create a new credential - PublicKeyCredential's
      [[Create]](origin, options, sameOriginWithAncestors) method
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method

   https://infra.spec.whatwg.org/#iteration-whileReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://infra.spec.whatwg.org/#willful-violationReferenced in:
     * 4. Terminology

   https://w3c.github.io/webappsec-mixed-content/#a-priori-authenticated-u
   rlReferenced in:
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)

   https://www.w3.org/TR/page-visibility/#visibility-statesReferenced in:
     * 5.6. Abort operations with AbortSignal

   https://tools.ietf.org/html/rfc4949#page-182Referenced in:
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation

   https://tools.ietf.org/html/rfc4949#page-186Referenced in:
     * 13.3. Security Benefits for Relying Parties
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation (2) (3) (4)

   https://tools.ietf.org/html/rfc8152#section-7Referenced in:
     * 6.3.1. Attested credential data
     * 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE_Key
       format
     * 8.6. FIDO U2F Attestation Statement Format

   https://w3c.github.io/webappsec-secure-contexts/#secure-contextsReferen
   ced in:
     * 5. Web Authentication API
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://tools.ietf.org/html/draft-ietf-tokbind-protocol#token-bindingRe
   ferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)
     * 7.1. Registering a new credential (2)
     * 7.2. Verifying an authentication assertion (2)

   https://tools.ietf.org/html/draft-ietf-tokbind-protocol#section-3.2Refe
   renced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion

   https://url.spec.whatwg.org/#concept-domainReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
```

```

   https://infra.spec.whatwg.org/#iteration-whileReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://infra.spec.whatwg.org/#willful-violationReferenced in:
     * 4. Terminology

   https://w3c.github.io/webappsec-mixed-content/#a-priori-authenticated-u
   rlReferenced in:
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)

   https://www.w3.org/TR/page-visibility/#visibility-statesReferenced in:
     * 5.6. Abort operations with AbortSignal

   https://tools.ietf.org/html/rfc4949#page-182Referenced in:
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation

   https://tools.ietf.org/html/rfc4949#page-186Referenced in:
     * 13.3. Security Benefits for Relying Parties
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation (2) (3) (4)

   https://tools.ietf.org/html/rfc8152#section-7Referenced in:
     * 6.3.1. Attested credential data
     * 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE_Key
       format
     * 8.6. FIDO U2F Attestation Statement Format

   https://w3c.github.io/webappsec-secure-contexts/#secure-contextsReferen
   ced in:
     * 5. Web Authentication API
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://tools.ietf.org/html/draft-ietf-tokbind-protocol#token-bindingRe
   ferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)
     * 7.1. Registering a new credential (2)
     * 7.2. Verifying an authentication assertion (2)

   https://tools.ietf.org/html/draft-ietf-tokbind-protocol#section-3.2Refe
   renced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion

   https://url.spec.whatwg.org/#concept-domainReferenced 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 (2)

   https://url.spec.whatwg.org/#empty-hostReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://url.spec.whatwg.org/#concept-url-hostReferenced 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 (2)

   https://url.spec.whatwg.org/#concept-ipv4Referenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://url.spec.whatwg.org/#concept-ipv6Referenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://url.spec.whatwg.org/#opaque-hostReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://url.spec.whatwg.org/#concept-url-serializerReferenced in:
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)

   https://url.spec.whatwg.org/#valid-domainReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://url.spec.whatwg.org/#valid-domain-stringReferenced in:
     * 4. Terminology

   https://heycam.github.io/webidl/#aborterrorReferenced 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 (2)

   https://heycam.github.io/webidl/#idl-ArrayBufferReferenced in:
     * 5.1. PublicKeyCredential Interface (2)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2) (3) (4) (5) (6)
     * 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse) (2)
```

```
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2)

   https://url.spec.whatwg.org/#empty-hostReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://url.spec.whatwg.org/#concept-url-hostReferenced 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 (2)

   https://url.spec.whatwg.org/#concept-ipv4Referenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://url.spec.whatwg.org/#concept-ipv6Referenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://url.spec.whatwg.org/#opaque-hostReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://url.spec.whatwg.org/#concept-url-serializerReferenced in:
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)

   https://url.spec.whatwg.org/#valid-domainReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   https://url.spec.whatwg.org/#valid-domain-stringReferenced in:
     * 4. Terminology

   https://heycam.github.io/webidl/#aborterrorReferenced 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 (2)

   https://heycam.github.io/webidl/#idl-ArrayBufferReferenced in:
     * 5.1. PublicKeyCredential Interface (2)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2) (3) (4) (5) (6)
     * 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse) (2)
```

```
      * 5.2.2. Web Authentication Assertion (interface
        AuthenticatorAssertionResponse) (2) (3) (4) (5) (6)
      * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
        (2)
      * 10.6. User Verification Index Extension (uvi)

   https://heycam.github.io/webidl/#BufferSourceReferenced in:
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions) (2)
      * 5.4.3. User Account Parameters for Credential Generation
        (dictionary PublicKeyCredentialUserEntity) (2)
      * 5.5. Options for Assertion Generation (dictionary
        PublicKeyCredentialRequestOptions) (2)
      * 5.10.3. Credential Descriptor (dictionary
        PublicKeyCredentialDescriptor)
      * 10.4. Authenticator Selection Extension (authnSel)

   https://heycam.github.io/webidl/#constrainterrorReferenced in:
      * 6.2.2. The authenticatorMakeCredential operation (2)

   https://heycam.github.io/webidl/#idl-DOMExceptionReferenced in:
      * 3. Dependencies
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
        (4) (5) (6) (7) (8) (9)
      * 5.1.4.1. PublicKeyCredential's
        [[DiscoverFromExternalSource]](origin, options,
        sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7)
      * 5.1.5. Store an existing credential - PublicKeyCredential's
        [[Store]](credential, sameOriginWithAncestors) method
      * 10.1. FIDO AppID Extension (appid) (2)

   https://heycam.github.io/webidl/#idl-DOMStringReferenced in:
      * 5.4.1. Public Key Entity Description (dictionary
        PublicKeyCredentialEntity) (2)
      * 5.4.2. RP Parameters for Credential Generation (dictionary
        PublicKeyCredentialRpEntity) (2)
      * 5.4.3. User Account Parameters for Credential Generation
        (dictionary PublicKeyCredentialUserEntity) (2)
      * 5.9. Authentication Extensions Authenticator Inputs (typedef
        AuthenticationExtensionsAuthenticatorInputs) (2)
      * 5.10.1. Client data used in WebAuthn signatures (dictionary
        CollectedClientData) (2) (3) (4)

   https://heycam.github.io/webidl/#ExposedReferenced in:
      * 5.1. PublicKeyCredential Interface
      * 5.2. Authenticator Responses (interface AuthenticatorResponse)
      * 5.2.1. Information about Public Key Credential (interface
        AuthenticatorAttestationResponse)
      * 5.2.2. Web Authentication Assertion (interface
        AuthenticatorAssertionResponse)

   https://heycam.github.io/webidl/#invalidstateerrorReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
      * 6.2.2. The authenticatorMakeCredential operation

   https://heycam.github.io/webidl/#notallowederrorReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
      * 5.1.4.1. PublicKeyCredential's
        [[DiscoverFromExternalSource]](origin, options,
        sameOriginWithAncestors) method (2) (3)
      * 6.2.2. The authenticatorMakeCredential operation (2)
      * 6.2.3. The authenticatorGetAssertion operation (2)

   https://heycam.github.io/webidl/#notsupportederrorReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method
      * 5.1.5. Store an existing credential - PublicKeyCredential's
```

```
      * 5.2.2. Web Authentication Assertion (interface
        AuthenticatorAssertionResponse) (2) (3) (4) (5) (6)
      * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
        (2)
      * 10.6. User Verification Index Extension (uvi)

   https://heycam.github.io/webidl/#BufferSourceReferenced in:
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions) (2)
      * 5.4.3. User Account Parameters for Credential Generation
        (dictionary PublicKeyCredentialUserEntity) (2)
      * 5.5. Options for Assertion Generation (dictionary
        PublicKeyCredentialRequestOptions) (2)
      * 5.10.3. Credential Descriptor (dictionary
        PublicKeyCredentialDescriptor)
      * 10.4. Authenticator Selection Extension (authnSel)

   https://heycam.github.io/webidl/#constrainterrorReferenced in:
      * 6.2.2. The authenticatorMakeCredential operation (2)

   https://heycam.github.io/webidl/#idl-DOMExceptionReferenced in:
      * 3. Dependencies
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
        (4) (5) (6) (7) (8) (9)
      * 5.1.4.1. PublicKeyCredential's
        [[DiscoverFromExternalSource]](origin, options,
        sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7) (8) (9)
      * 5.1.5. Store an existing credential - PublicKeyCredential's
        [[Store]](credential, sameOriginWithAncestors) method
      * 10.1. FIDO AppID Extension (appid) (2)

   https://heycam.github.io/webidl/#idl-DOMStringReferenced in:
      * 5.4.1. Public Key Entity Description (dictionary
        PublicKeyCredentialEntity) (2)
      * 5.4.2. RP Parameters for Credential Generation (dictionary
        PublicKeyCredentialRpEntity) (2)
      * 5.4.3. User Account Parameters for Credential Generation
        (dictionary PublicKeyCredentialUserEntity) (2)
      * 5.9. Authentication Extensions Authenticator Inputs (typedef
        AuthenticationExtensionsAuthenticatorInputs) (2)
      * 5.10.1. Client data used in WebAuthn signatures (dictionary
        CollectedClientData) (2) (3) (4)

   https://heycam.github.io/webidl/#ExposedReferenced in:
      * 5.1. PublicKeyCredential Interface
      * 5.2. Authenticator Responses (interface AuthenticatorResponse)
      * 5.2.1. Information about Public Key Credential (interface
        AuthenticatorAttestationResponse)
      * 5.2.2. Web Authentication Assertion (interface
        AuthenticatorAssertionResponse)

   https://heycam.github.io/webidl/#invalidstateerrorReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
      * 6.2.2. The authenticatorMakeCredential operation

   https://heycam.github.io/webidl/#notallowederrorReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
      * 5.1.4.1. PublicKeyCredential's
        [[DiscoverFromExternalSource]](origin, options,
        sameOriginWithAncestors) method (2) (3) (4) (5)
      * 6.2.2. The authenticatorMakeCredential operation (2)
      * 6.2.3. The authenticatorGetAssertion operation (2)

   https://heycam.github.io/webidl/#notsupportederrorReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method
      * 5.1.5. Store an existing credential - PublicKeyCredential's
```

```
       [[Store]](credential, sameOriginWithAncestors) method
     * 6.2.2. The authenticatorMakeCredential operation
     * 10.1. FIDO AppID Extension (appid)

   https://heycam.github.io/webidl/#idl-promiseReferenced in:
     * 3. Dependencies
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.1.5. Store an existing credential - PublicKeyCredential's
       [[Store]](credential, sameOriginWithAncestors) method

   https://heycam.github.io/webidl/#SameObjectReferenced in:
     * 5.1. PublicKeyCredential Interface (2)
     * 5.2. Authenticator Responses (interface AuthenticatorResponse)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse) (2) (3)

   https://heycam.github.io/webidl/#SecureContextReferenced in:
     * 5.1. PublicKeyCredential Interface
     * 5.2. Authenticator Responses (interface AuthenticatorResponse)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse)

   https://heycam.github.io/webidl/#securityerrorReferenced 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 (2)
     * 10.1. FIDO AppID Extension (appid)

   https://heycam.github.io/webidl/#idl-USVStringReferenced in:
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity) (2)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions) (2)
     * 10.1. FIDO AppID Extension (appid)
     * 10.2. Simple Transaction Authorization Extension (txAuthSimple) (2)
     * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
     * 10.5. Supported Extensions Extension (exts)

   https://heycam.github.io/webidl/#unknownerrorReferenced in:
     * 6.2.2. The authenticatorMakeCredential operation (2)
     * 6.2.3. The authenticatorGetAssertion operation (2)

   https://heycam.github.io/webidl/#idl-booleanReferenced in:
     * 5.1.7. Availability of User-Verifying Platform Authenticator -
       PublicKeyCredential's
       isUserVerifyingPlatformAuthenticatorAvailable() method
     * 5.4.4. Authenticator Selection Criteria (dictionary
       AuthenticatorSelectionCriteria) (2)
     * 10.1. FIDO AppID Extension (appid)
     * 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)

   https://heycam.github.io/webidl/#idl-floatReferenced in:
     * 10.9. Biometric Authenticator Performance Bounds Extension
       (biometricPerfBounds) (2)

   https://heycam.github.io/webidl/#dfn-interface-objectReferenced in:
```

```
       [[Store]](credential, sameOriginWithAncestors) method
     * 6.2.2. The authenticatorMakeCredential operation
     * 10.1. FIDO AppID Extension (appid)

   https://heycam.github.io/webidl/#idl-promiseReferenced in:
     * 3. Dependencies
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.1.5. Store an existing credential - PublicKeyCredential's
       [[Store]](credential, sameOriginWithAncestors) method

   https://heycam.github.io/webidl/#SameObjectReferenced in:
     * 5.1. PublicKeyCredential Interface (2)
     * 5.2. Authenticator Responses (interface AuthenticatorResponse)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse) (2) (3)

   https://heycam.github.io/webidl/#SecureContextReferenced in:
     * 5.1. PublicKeyCredential Interface
     * 5.2. Authenticator Responses (interface AuthenticatorResponse)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse)

   https://heycam.github.io/webidl/#securityerrorReferenced 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 (2)
     * 10.1. FIDO AppID Extension (appid)

   https://heycam.github.io/webidl/#idl-USVStringReferenced in:
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity) (2)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions) (2)
     * 10.1. FIDO AppID Extension (appid)
     * 10.2. Simple Transaction Authorization Extension (txAuthSimple) (2)
     * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
     * 10.5. Supported Extensions Extension (exts)

   https://heycam.github.io/webidl/#unknownerrorReferenced in:
     * 6.2.2. The authenticatorMakeCredential operation (2)
     * 6.2.3. The authenticatorGetAssertion operation (2)

   https://heycam.github.io/webidl/#idl-booleanReferenced in:
     * 5.1.7. Availability of User-Verifying Platform Authenticator -
       PublicKeyCredential's
       isUserVerifyingPlatformAuthenticatorAvailable() method
     * 5.4.4. Authenticator Selection Criteria (dictionary
       AuthenticatorSelectionCriteria) (2)
     * 10.1. FIDO AppID Extension (appid)
     * 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)

   https://heycam.github.io/webidl/#idl-floatReferenced in:
     * 10.9. Biometric Authenticator Performance Bounds Extension
       (biometricPerfBounds) (2)

   https://heycam.github.io/webidl/#dfn-interface-objectReferenced in:
        * 5.1. PublicKeyCredential Interface (2) (3)
        * 5.1.3. Create a new credential - PublicKeyCredential's
          [[Create]](origin, options, sameOriginWithAncestors) method

      https://heycam.github.io/webidl/#idl-longReferenced in:
        * 5.10.5. Cryptographic Algorithm Identifier (typedef
          COSEAlgorithmIdentifier)

      https://heycam.github.io/webidl/#dfn-presentReferenced in:
        * 5.1.3. Create a new credential - PublicKeyCredential's
          [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
          (4) (5) (6) (7) (8) (9) (10)
        * 5.1.4.1. PublicKeyCredential's
          [[DiscoverFromExternalSource]](origin, options,
          sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7)
        * 5.4.4. Authenticator Selection Criteria (dictionary
          AuthenticatorSelectionCriteria)

      https://heycam.github.io/webidl/#idl-unsigned-longReferenced in:
        * 5.4. Options for Credential Creation (dictionary
          PublicKeyCredentialCreationOptions) (2)
        * 5.5. Options for Assertion Generation (dictionary
          PublicKeyCredentialRequestOptions) (2)
        * 10.8. User Verification Method Extension (uvm)

      https://html.spec.whatwg.org/#focusReferenced in:
        * 5.6. Abort operations with AbortSignal

      https://html.spec.whatwg.org/#attr-fe-autocomplete-usernameReferenced
      in:
        * 5.4.1. Public Key Entity Description (dictionary
          PublicKeyCredentialEntity)

     Terms defined by reference

        * [CREDENTIAL-MANAGEMENT-1] defines the following terms:
             + Credential
             + CredentialCreationOptions
             + CredentialRequestOptions
             + CredentialsContainer
             + Request a Credential
             + [[CollectFromCredentialStore]](origin, options,
               sameOriginWithAncestors)
             + [[Create]](origin, options, sameOriginWithAncestors)
             + [[Store]](credential, sameOriginWithAncestors)
             + [[discovery]]
             + [[type]]
             + create()
             + credential
             + credential source
             + get()
             + id
             + remote
             + same-origin with its ancestors
             + signal (for CredentialRequestOptions)
             + store()
             + type
             + user mediation
        * [DOM4] defines the following terms:
             + AbortController
             + aborted flag
             + document
        * [ECMAScript] defines the following terms:
             + %arraybuffer%
             + internal method
             + internal slot
             + stringify
        * [ENCODING] defines the following terms:
             + utf-8 decode
             + utf-8 encode
```

```
        * 5.1. PublicKeyCredential Interface (2) (3)
        * 5.1.3. Create a new credential - PublicKeyCredential's
          [[Create]](origin, options, sameOriginWithAncestors) method

      https://heycam.github.io/webidl/#idl-longReferenced in:
        * 5.10.5. Cryptographic Algorithm Identifier (typedef
          COSEAlgorithmIdentifier)

      https://heycam.github.io/webidl/#dfn-presentReferenced in:
        * 5.1.3. Create a new credential - PublicKeyCredential's
          [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
          (4) (5) (6) (7) (8) (9) (10)
        * 5.1.4.1. PublicKeyCredential's
          [[DiscoverFromExternalSource]](origin, options,
          sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7)
        * 5.4.4. Authenticator Selection Criteria (dictionary
          AuthenticatorSelectionCriteria)

      https://heycam.github.io/webidl/#idl-unsigned-longReferenced in:
        * 5.4. Options for Credential Creation (dictionary
          PublicKeyCredentialCreationOptions) (2)
        * 5.5. Options for Assertion Generation (dictionary
          PublicKeyCredentialRequestOptions) (2)
        * 10.8. User Verification Method Extension (uvm)

      https://html.spec.whatwg.org/#focusReferenced in:
        * 5.6. Abort operations with AbortSignal

      https://html.spec.whatwg.org/#attr-fe-autocomplete-usernameReferenced
      in:
        * 5.4.1. Public Key Entity Description (dictionary
          PublicKeyCredentialEntity)

     Terms defined by reference

        * [CREDENTIAL-MANAGEMENT-1] defines the following terms:
             + Credential
             + CredentialCreationOptions
             + CredentialRequestOptions
             + CredentialsContainer
             + Request a Credential
             + [[CollectFromCredentialStore]](origin, options,
               sameOriginWithAncestors)
             + [[Create]](origin, options, sameOriginWithAncestors)
             + [[Store]](credential, sameOriginWithAncestors)
             + [[discovery]]
             + [[type]]
             + create()
             + credential
             + credential source
             + get()
             + id
             + remote
             + same-origin with its ancestors
             + signal (for CredentialRequestOptions)
             + store()
             + type
             + user mediation
        * [DOM4] defines the following terms:
             + AbortController
             + aborted flag
             + document
        * [ECMAScript] defines the following terms:
             + %arraybuffer%
             + internal method
             + internal slot
             + stringify
        * [ENCODING] defines the following terms:
             + utf-8 decode
             + utf-8 encode
     * [FETCH] defines the following terms:
          + window
     * [FIDO-APPID] defines the following terms:
          + determining if a caller's facetid is authorized for an appid
          + determining the facetid of a calling application
     * [FIDO-CTAP] defines the following terms:
          + ctap2 canonical cbor encoding form
     * [FIDO-Registry] defines the following terms:
          + section 3.1 user verification methods
          + section 3.2 key protection types
          + section 3.3 matcher protection types
          + section 3.6.2 public key representation formats
     * [FIDO-U2F-Message-Formats] defines the following terms:
          + application parameter
          + section 4.3
          + section 5.4
     * [Geolocation-API] defines the following terms:
          + Coordinates
     * [HTML] defines the following terms:
          + ascii serialization of an origin
          + 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
     * [HTML52] defines the following terms:
          + document.domain
          + opaque origin
          + origin
     * [INFRA] defines the following terms:
          + append (for set)
          + byte sequence
          + continue
          + for each (for map)
          + is empty
          + is not empty
          + item (for struct)
          + list
          + map
          + ordered set
          + remove
          + set (for map)
          + struct
          + while
          + willful violation
     * [mixed-content] defines the following terms:
          + a priori authenticated url
     * [page-visibility] defines the following terms:
          + visibility states
     * [RFC4949] defines the following terms:
          + leap of faith
          + man-in-the-middle attack
     * [RFC8152] defines the following terms:
          + section 7
     * [secure-contexts] defines the following terms:
          + secure contexts
     * [TokenBinding] defines the following terms:
          + token binding
          + token binding id
     * [URL] defines the following terms:
          + domain
          + empty host
          + host
          + ipv4 address
          + ipv6 address
          + opaque host
          + url serializer
          + valid domain
```

```
     * [FETCH] defines the following terms:
          + window
     * [FIDO-APPID] defines the following terms:
          + determining if a caller's facetid is authorized for an appid
          + determining the facetid of a calling application
     * [FIDO-CTAP] defines the following terms:
          + ctap2 canonical cbor encoding form
     * [FIDO-Registry] defines the following terms:
          + section 3.1 user verification methods
          + section 3.2 key protection types
          + section 3.3 matcher protection types
          + section 3.6.2 public key representation formats
     * [FIDO-U2F-Message-Formats] defines the following terms:
          + application parameter
          + section 4.3
          + section 5.4
     * [Geolocation-API] defines the following terms:
          + Coordinates
     * [HTML] defines the following terms:
          + ascii serialization of an origin
          + 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
     * [HTML52] defines the following terms:
          + document.domain
          + opaque origin
          + origin
     * [INFRA] defines the following terms:
          + append (for set)
          + byte sequence
          + continue
          + for each (for map)
          + is empty
          + is not empty
          + item (for struct)
          + list
          + map
          + ordered set
          + remove
          + set (for map)
          + struct
          + while
          + willful violation
     * [mixed-content] defines the following terms:
          + a priori authenticated url
     * [page-visibility] defines the following terms:
          + visibility states
     * [RFC4949] defines the following terms:
          + leap of faith
          + man-in-the-middle attack
     * [RFC8152] defines the following terms:
          + section 7
     * [secure-contexts] defines the following terms:
          + secure contexts
     * [TokenBinding] defines the following terms:
          + token binding
          + token binding id
     * [URL] defines the following terms:
          + domain
          + empty host
          + host
          + ipv4 address
          + ipv6 address
          + opaque host
          + url serializer
          + valid domain
```

```
            + valid domain string
        * [WebIDL] defines the following terms:
            + AbortError
            + ArrayBuffer
            + BufferSource
            + ConstraintError
            + DOMException
            + DOMString
            + Exposed
            + InvalidStateError
            + NotAllowedError
            + NotSupportedError
            + Promise
            + SameObject
            + SecureContext
            + SecurityError
            + USVString
            + UnknownError
            + boolean
            + float
            + interface object
            + long
            + present
            + unsigned long
        * [whatwg html] defines the following terms:
            + focus
            + username

References

  Normative References

   [CDDL]
          C. Vigano; H. Birkholz. CBOR data definition language (CDDL): a
          notational convention to express CBOR data structures. 21
          September 2016. Internet Draft (work in progress). URL:
          https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl

   [CREDENTIAL-MANAGEMENT-1]
          Mike West. Credential Management Level 1. 4 August 2017. WD.
          URL: https://www.w3.org/TR/credential-management-1/

   [DOM4]
          Anne van Kesteren. DOM Standard. Living Standard. URL:
          https://dom.spec.whatwg.org/

   [ECMAScript]
          ECMAScript Language Specification. URL:
          https://tc39.github.io/ecma262/

   [ENCODING]
          Anne van Kesteren. Encoding Standard. Living Standard. URL:
          https://encoding.spec.whatwg.org/

   [FETCH]
          Anne van Kesteren. Fetch Standard. Living Standard. URL:
          https://fetch.spec.whatwg.org/

   [FIDO-APPID]
          D. Balfanz; et al. FIDO AppID and Facets. FIDO Alliance Proposed
          Standard. URL:
          https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-appid-
          and-facets-v2.0-ps-20170927.html

   [FIDO-CTAP]
          R. Lindemann; et al. FIDO 2.0: Client to Authenticator Protocol.
          FIDO Alliance Proposed Standard. URL:
          https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client
          -to-authenticator-protocol-v2.0-ps-20170927.html
```

```
            + valid domain string
        * [WebIDL] defines the following terms:
            + AbortError
            + ArrayBuffer
            + BufferSource
            + ConstraintError
            + DOMException
            + DOMString
            + Exposed
            + InvalidStateError
            + NotAllowedError
            + NotSupportedError
            + Promise
            + SameObject
            + SecureContext
            + SecurityError
            + USVString
            + UnknownError
            + boolean
            + float
            + interface object
            + long
            + present
            + unsigned long
        * [whatwg html] defines the following terms:
            + focus
            + username

References

  Normative References

   [CDDL]
          C. Vigano; H. Birkholz. CBOR data definition language (CDDL): a
          notational convention to express CBOR data structures. 21
          September 2016. Internet Draft (work in progress). URL:
          https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl

   [CREDENTIAL-MANAGEMENT-1]
          Mike West. Credential Management Level 1. 4 August 2017. WD.
          URL: https://www.w3.org/TR/credential-management-1/

   [DOM4]
          Anne van Kesteren. DOM Standard. Living Standard. URL:
          https://dom.spec.whatwg.org/

   [ECMAScript]
          ECMAScript Language Specification. URL:
          https://tc39.github.io/ecma262/

   [ENCODING]
          Anne van Kesteren. Encoding Standard. Living Standard. URL:
          https://encoding.spec.whatwg.org/

   [FETCH]
          Anne van Kesteren. Fetch Standard. Living Standard. URL:
          https://fetch.spec.whatwg.org/

   [FIDO-APPID]
          D. Balfanz; et al. FIDO AppID and Facets. FIDO Alliance Proposed
          Standard. URL:
          https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-appid-
          and-facets-v2.0-ps-20170927.html

   [FIDO-CTAP]
          R. Lindemann; et al. FIDO 2.0: Client to Authenticator Protocol.
          FIDO Alliance Proposed Standard. URL:
          https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client
          -to-authenticator-protocol-v2.0-ps-20170927.html
```

[FIDO-Privacy-Principles]
FIDO Alliance. FIDO Privacy Principles. FIDO Alliance Whitepaper. URL: https://fidoalliance.org/wp-content/uploads/2014/12/FIDO_Alliance_Whitepaper_Privacy_Principles.pdf

[FIDO-Registry]
R. Lindemann. FIDO Registry of Predefined Values. 27 September 2017. FIDO Alliance Proposed Standard. URL: https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0-ps-20170927.html

[FIDO-U2F-Message-Formats]
D. Balfanz; J. Ehrensvard; J. Lang. FIDO U2F Raw Message Formats. FIDO Alliance Implementation Draft. URL: https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-message-formats-v1.1-id-20160915.html

[FIDOEcdaaAlgorithm]
R. Lindemann; et al. FIDO ECDAA Algorithm. FIDO Alliance Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ecdaa-algorithm-v1.1-id-20170202.html

[Geolocation-API]
Andrei Popescu. Geolocation API Specification 2nd Edition. 8 November 2016. REC. URL: https://www.w3.org/TR/geolocation-API/

[HTML]
Anne van Kesteren; et al. HTML Standard. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[HTML52]
Steve Faulkner; et al. HTML 5.2. 14 December 2017. REC. URL: https://www.w3.org/TR/html52/

[IANA-COSE-ALGS-REG]
IANA CBOR Object Signing and Encryption (COSE) Algorithms Registry. URL: https://www.iana.org/assignments/cose/cose.xhtml#algorithms

[INFRA]
Anne van Kesteren; Domenic Denicola. Infra Standard. Living Standard. URL: https://infra.spec.whatwg.org/

[MIXED-CONTENT]
Mike West. Mixed Content. 2 August 2016. CR. URL: https://www.w3.org/TR/mixed-content/

[PAGE-VISIBILITY]
Jatinder Mann; Arvind Jain. Page Visibility (Second Edition). 29 October 2013. REC. URL: https://www.w3.org/TR/page-visibility/

[RFC2119]
S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC4648]
S. Josefsson. The Base16, Base32, and Base64 Data Encodings. October 2006. Proposed Standard. URL: https://tools.ietf.org/html/rfc4648

[RFC4949]
R. Shirey. Internet Security Glossary, Version 2. August 2007. Informational. URL: https://tools.ietf.org/html/rfc4949

[RFC5234]
D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax Specifications: ABNF. January 2008. Internet Standard. URL:

[FIDO-Privacy-Principles]
FIDO Alliance. FIDO Privacy Principles. FIDO Alliance Whitepaper. URL: https://fidoalliance.org/wp-content/uploads/2014/12/FIDO_Alliance_Whitepaper_Privacy_Principles.pdf

[FIDO-Registry]
R. Lindemann. FIDO Registry of Predefined Values. 27 September 2017. FIDO Alliance Proposed Standard. URL: https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0-ps-20170927.html

[FIDO-U2F-Message-Formats]
D. Balfanz; J. Ehrensvard; J. Lang. FIDO U2F Raw Message Formats. FIDO Alliance Implementation Draft. URL: https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-message-formats-v1.1-id-20160915.html

[FIDOEcdaaAlgorithm]
R. Lindemann; et al. FIDO ECDAA Algorithm. FIDO Alliance Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ecdaa-algorithm-v1.1-id-20170202.html

[Geolocation-API]
Andrei Popescu. Geolocation API Specification 2nd Edition. 8 November 2016. REC. URL: https://www.w3.org/TR/geolocation-API/

[HTML]
Anne van Kesteren; et al. HTML Standard. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[HTML52]
Steve Faulkner; et al. HTML 5.2. 14 December 2017. REC. URL: https://www.w3.org/TR/html52/

[IANA-COSE-ALGS-REG]
IANA CBOR Object Signing and Encryption (COSE) Algorithms Registry. URL: https://www.iana.org/assignments/cose/cose.xhtml#algorithms

[INFRA]
Anne van Kesteren; Domenic Denicola. Infra Standard. Living Standard. URL: https://infra.spec.whatwg.org/

[MIXED-CONTENT]
Mike West. Mixed Content. 2 August 2016. CR. URL: https://www.w3.org/TR/mixed-content/

[PAGE-VISIBILITY]
Jatinder Mann; Arvind Jain. Page Visibility (Second Edition). 29 October 2013. REC. URL: https://www.w3.org/TR/page-visibility/

[RFC2119]
S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC4648]
S. Josefsson. The Base16, Base32, and Base64 Data Encodings. October 2006. Proposed Standard. URL: https://tools.ietf.org/html/rfc4648

[RFC4949]
R. Shirey. Internet Security Glossary, Version 2. August 2007. Informational. URL: https://tools.ietf.org/html/rfc4949

[RFC5234]
D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax Specifications: ABNF. January 2008. Internet Standard. URL:

https://tools.ietf.org/html/rfc5234

[RFC5890]
J. Klensin. Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework. August 2010. Proposed Standard. URL: https://tools.ietf.org/html/rfc5890

[RFC7049]
C. Bormann; P. Hoffman. Concise Binary Object Representation (CBOR). October 2013. Proposed Standard. URL: https://tools.ietf.org/html/rfc7049

[RFC8152]
J. Schaad. CBOR Object Signing and Encryption (COSE). July 2017. Proposed Standard. URL: https://tools.ietf.org/html/rfc8152

[RFC8230]
M. Jones. Using RSA Algorithms with CBOR Object Signing and Encryption (COSE) Messages. September 2017. Proposed Standard. URL: https://tools.ietf.org/html/rfc8230

[SEC1]
SEC1: Elliptic Curve Cryptography, Version 2.0. URL: http://www.secg.org/sec1-v2.pdf

[SECURE-CONTEXTS]
Mike West. Secure Contexts. 15 September 2016. CR. URL: https://www.w3.org/TR/secure-contexts/

[TCG-CMCProfile-AIKCertEnroll]
Scott Kelly; et al. TCG Infrastructure Working Group: A CMC Profile for AIK Certificate Enrollment. 24 March 2011. Published. URL: https://trustedcomputinggroup.org/wp-content/uploads/IWG_CMC_Profile_Cert_Enrollment_v1_r7.pdf

[TokenBinding]
A. Popov; et al. The Token Binding Protocol Version 1.0. February 16, 2017. Internet-Draft. URL: https://tools.ietf.org/html/draft-ietf-tokbind-protocol

[URL]
Anne van Kesteren. URL Standard. Living Standard. URL: https://url.spec.whatwg.org/

[WebAuthn-COSE-Algs]
Michael B. Jones. COSE Algorithms for Web Authentication (WebAuthn). May 2018. Active Internet-Draft. URL: https://tools.ietf.org/html/draft-jones-webauthn-cose-algorithms

[WebAuthn-Registries]
Jeff Hodges; Giridhar Mandyam; Michael B. Jones. Registries for Web Authentication (WebAuthn). February 2018. Active Internet-Draft. URL: https://tools.ietf.org/html/draft-hodges-webauthn-registries

[WebIDL]
Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15 December 2016. ED. URL: https://heycam.github.io/webidl/

[WebIDL-1]
Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL: https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/

Informative References

[Ceremony]
Carl Ellison. Ceremony Design and Analysis. 2007. URL: https://eprint.iacr.org/2007/399.pdf

https://tools.ietf.org/html/rfc5234

[RFC5890]
J. Klensin. Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework. August 2010. Proposed Standard. URL: https://tools.ietf.org/html/rfc5890

[RFC7049]
C. Bormann; P. Hoffman. Concise Binary Object Representation (CBOR). October 2013. Proposed Standard. URL: https://tools.ietf.org/html/rfc7049

[RFC8152]
J. Schaad. CBOR Object Signing and Encryption (COSE). July 2017. Proposed Standard. URL: https://tools.ietf.org/html/rfc8152

[RFC8230]
M. Jones. Using RSA Algorithms with CBOR Object Signing and Encryption (COSE) Messages. September 2017. Proposed Standard. URL: https://tools.ietf.org/html/rfc8230

[SEC1]
SEC1: Elliptic Curve Cryptography, Version 2.0. URL: http://www.secg.org/sec1-v2.pdf

[SECURE-CONTEXTS]
Mike West. Secure Contexts. 15 September 2016. CR. URL: https://www.w3.org/TR/secure-contexts/

[TCG-CMCProfile-AIKCertEnroll]
Scott Kelly; et al. TCG Infrastructure Working Group: A CMC Profile for AIK Certificate Enrollment. 24 March 2011. Published. URL: https://trustedcomputinggroup.org/wp-content/uploads/IWG_CMC_Profile_Cert_Enrollment_v1_r7.pdf

[TokenBinding]
A. Popov; et al. The Token Binding Protocol Version 1.0. February 16, 2017. Internet-Draft. URL: https://tools.ietf.org/html/draft-ietf-tokbind-protocol

[URL]
Anne van Kesteren. URL Standard. Living Standard. URL: https://url.spec.whatwg.org/

[WebAuthn-COSE-Algs]
Michael B. Jones. COSE Algorithms for Web Authentication (WebAuthn). May 2018. Active Internet-Draft. URL: https://tools.ietf.org/html/draft-jones-webauthn-cose-algorithms

[WebAuthn-Registries]
Jeff Hodges; Giridhar Mandyam; Michael B. Jones. Registries for Web Authentication (WebAuthn). February 2018. Active Internet-Draft. URL: https://tools.ietf.org/html/draft-hodges-webauthn-registries

[WebIDL]
Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15 December 2016. ED. URL: https://heycam.github.io/webidl/

[WebIDL-1]
Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL: https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/

Informative References

[Ceremony]
Carl Ellison. Ceremony Design and Analysis. 2007. URL: https://eprint.iacr.org/2007/399.pdf

[EduPersonObjectClassSpec]
EduPerson Object Class Specification (200604a). May 15, 2007. URL: https://www.internet2.edu/media/medialibrary/2013/09/04/internet2-mace-dir-eduperson-200604.html

[Feature-Policy]
Feature Policy. Draft Community Group Report. URL: https://wicg.github.io/feature-policy/

[FIDO-UAF-AUTHNR-CMDS]
R. Lindemann; J. Kemp. FIDO UAF Authenticator Commands. FIDO Alliance Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-authnr-cmds-v1.1-id-20170202.html

[FIDOAuthnrSecReqs]
D. Biggs; et al. FIDO Authenticator Security Requirements. FIDO Alliance Final Documents. URL: https://fidoalliance.org/specs/fido-security-requirements-v1.0-fd-20170524/

[FIDOMetadataService]
R. Lindemann; B. Hill; D. Baghdasaryan. FIDO Metadata Service v1.0. FIDO Alliance Proposed Standard. URL: https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-metadata-service-v1.0-ps-20141208.html

[FIDOSecRef]
R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference. FIDO Alliance Proposed Standard. URL: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-security-ref-v1.2-ps-20170411.html

[FIDOUAFAuthenticatorMetadataStatements]
B. Hill; D. Baghdasaryan; J. Kemp. FIDO UAF Authenticator Metadata Statements v1.0. FIDO Alliance Proposed Standard. URL: https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-authnr-metadata-v1.0-ps-20141208.html

[ISOBiometricVocabulary]
ISO/IEC JTC1/SC37. Information technology -- Vocabulary -- Biometrics. 15 December 2012. International Standard: ISO/IEC 2382-37:2012(E) First Edition. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194_ISOIEC_2382-37_2012.zip

[RFC3279]
L. Bassham; W. Polk; R. Housley. Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. April 2002. Proposed Standard. URL: https://tools.ietf.org/html/rfc3279

[RFC5280]
D. Cooper; et al. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. May 2008. Proposed Standard. URL: https://tools.ietf.org/html/rfc5280

[RFC6265]
A. Barth. HTTP State Management Mechanism. April 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6265

[RFC6454]
A. Barth. The Web Origin Concept. December 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6454

[RFC7515]
M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May 2015. Proposed Standard. URL:

[EduPersonObjectClassSpec]
EduPerson Object Class Specification (200604a). May 15, 2007. URL: https://www.internet2.edu/media/medialibrary/2013/09/04/internet2-mace-dir-eduperson-200604.html

[Feature-Policy]
Feature Policy. Draft Community Group Report. URL: https://wicg.github.io/feature-policy/

[FIDO-UAF-AUTHNR-CMDS]
R. Lindemann; J. Kemp. FIDO UAF Authenticator Commands. FIDO Alliance Implementation Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-authnr-cmds-v1.1-id-20170202.html

[FIDOAuthnrSecReqs]
D. Biggs; et al. FIDO Authenticator Security Requirements. FIDO Alliance Final Documents. URL: https://fidoalliance.org/specs/fido-security-requirements-v1.0-fd-20170524/

[FIDOMetadataService]
R. Lindemann; B. Hill; D. Baghdasaryan. FIDO Metadata Service v1.0. FIDO Alliance Proposed Standard. URL: https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-metadata-service-v1.0-ps-20141208.html

[FIDOSecRef]
R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference. FIDO Alliance Proposed Standard. URL: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-security-ref-v1.2-ps-20170411.html

[FIDOUAFAuthenticatorMetadataStatements]
B. Hill; D. Baghdasaryan; J. Kemp. FIDO UAF Authenticator Metadata Statements v1.0. FIDO Alliance Proposed Standard. URL: https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-authnr-metadata-v1.0-ps-20141208.html

[ISOBiometricVocabulary]
ISO/IEC JTC1/SC37. Information technology -- Vocabulary -- Biometrics. 15 December 2012. International Standard: ISO/IEC 2382-37:2012(E) First Edition. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194_ISOIEC_2382-37_2012.zip

[RFC3279]
L. Bassham; W. Polk; R. Housley. Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. April 2002. Proposed Standard. URL: https://tools.ietf.org/html/rfc3279

[RFC5280]
D. Cooper; et al. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. May 2008. Proposed Standard. URL: https://tools.ietf.org/html/rfc5280

[RFC6265]
A. Barth. HTTP State Management Mechanism. April 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6265

[RFC6454]
A. Barth. The Web Origin Concept. December 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6454

[RFC7515]
M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May 2015. Proposed Standard. URL:

```
                https://tools.ietf.org/html/rfc7515

          [RFC8017]
                K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography
                Specifications Version 2.2. November 2016. Informational. URL:
                https://tools.ietf.org/html/rfc8017

          [TPMv2-EK-Profile]
                TCG EK Credential Profile for TPM Family 2.0. URL:
                http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti
                al_Profile_EK_V2.0_R14_published.pdf

          [TPMv2-Part1]
                Trusted Platform Module Library, Part 1: Architecture. URL:
                http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
                2.0-Part-1-Architecture-01.38.pdf

          [TPMv2-Part2]
                Trusted Platform Module Library, Part 2: Structures. URL:
                http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
                2.0-Part-2-Structures-01.38.pdf

          [TPMv2-Part3]
                Trusted Platform Module Library, Part 3: Commands. URL:
                http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
                2.0-Part-3-Commands-01.38.pdf

          [UAFProtocol]
                R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO
                Alliance Proposed Standard. URL:
                https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua
                f-protocol-v1.0-ps-20141208.html

        IDL Index

        [SecureContext, Exposed=Window]
        interface PublicKeyCredential : Credential {
            [SameObject] readonly attribute ArrayBuffer              rawId;
            [SameObject] readonly attribute AuthenticatorResponse    response;
            AuthenticationExtensionsClientOutputs getClientExtensionResults();
        };

        partial dictionary CredentialCreationOptions {
            PublicKeyCredentialCreationOptions      publicKey;
        };

        partial dictionary CredentialRequestOptions {
            PublicKeyCredentialRequestOptions      publicKey;
        };

        partial interface PublicKeyCredential {
            static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();
        };

        [SecureContext, Exposed=Window]
        interface AuthenticatorResponse {
            [SameObject] readonly attribute ArrayBuffer      clientDataJSON;
        };

        [SecureContext, Exposed=Window]
        interface AuthenticatorAttestationResponse : AuthenticatorResponse {
            [SameObject] readonly attribute ArrayBuffer      attestationObject;
        };

        [SecureContext, Exposed=Window]
        interface AuthenticatorAssertionResponse : AuthenticatorResponse {
            [SameObject] readonly attribute ArrayBuffer      authenticatorData;
            [SameObject] readonly attribute ArrayBuffer      signature;
            [SameObject] readonly attribute ArrayBuffer?     userHandle;
        };
```

```
                https://tools.ietf.org/html/rfc7515

          [RFC8017]
                K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography
                Specifications Version 2.2. November 2016. Informational. URL:
                https://tools.ietf.org/html/rfc8017

          [TPMv2-EK-Profile]
                TCG EK Credential Profile for TPM Family 2.0. URL:
                http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti
                al_Profile_EK_V2.0_R14_published.pdf

          [TPMv2-Part1]
                Trusted Platform Module Library, Part 1: Architecture. URL:
                http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
                2.0-Part-1-Architecture-01.38.pdf

          [TPMv2-Part2]
                Trusted Platform Module Library, Part 2: Structures. URL:
                http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
                2.0-Part-2-Structures-01.38.pdf

          [TPMv2-Part3]
                Trusted Platform Module Library, Part 3: Commands. URL:
                http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
                2.0-Part-3-Commands-01.38.pdf

          [UAFProtocol]
                R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO
                Alliance Proposed Standard. URL:
                https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua
                f-protocol-v1.0-ps-20141208.html

        IDL Index

        [SecureContext, Exposed=Window]
        interface PublicKeyCredential : Credential {
            [SameObject] readonly attribute ArrayBuffer              rawId;
            [SameObject] readonly attribute AuthenticatorResponse    response;
            AuthenticationExtensionsClientOutputs getClientExtensionResults();
        };

        partial dictionary CredentialCreationOptions {
            PublicKeyCredentialCreationOptions      publicKey;
        };

        partial dictionary CredentialRequestOptions {
            PublicKeyCredentialRequestOptions      publicKey;
        };

        partial interface PublicKeyCredential {
            static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();
        };

        [SecureContext, Exposed=Window]
        interface AuthenticatorResponse {
            [SameObject] readonly attribute ArrayBuffer      clientDataJSON;
        };

        [SecureContext, Exposed=Window]
        interface AuthenticatorAttestationResponse : AuthenticatorResponse {
            [SameObject] readonly attribute ArrayBuffer      attestationObject;
        };

        [SecureContext, Exposed=Window]
        interface AuthenticatorAssertionResponse : AuthenticatorResponse {
            [SameObject] readonly attribute ArrayBuffer      authenticatorData;
            [SameObject] readonly attribute ArrayBuffer      signature;
            [SameObject] readonly attribute ArrayBuffer?     userHandle;
        };
```

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

    dictionary PublicKeyCredentialCreationOptions {
        required PublicKeyCredentialRpEntity         rp;
        required PublicKeyCredentialUserEntity       user;

        required BufferSource                             challenge;
        required sequence<PublicKeyCredentialParameters>  pubKeyCredParams;

        unsigned long                                timeout;
        sequence<PublicKeyCredentialDescriptor>      excludeCredentials = [];
        AuthenticatorSelectionCriteria               authenticatorSelection;
        AttestationConveyancePreference              attestation = "none";
        AuthenticationExtensionsClientInputs         extensions;
    };

    dictionary PublicKeyCredentialEntity {
        required DOMString    name;
        USVString             icon;
    };

    dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
        DOMString      id;
    };

    dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
        required BufferSource   id;
        required DOMString      displayName;
    };

    dictionary AuthenticatorSelectionCriteria {
        AuthenticatorAttachment      authenticatorAttachment;
        boolean                      requireResidentKey = false;
        UserVerificationRequirement  userVerification = "preferred";
    };

    enum AuthenticatorAttachment {
        "platform",       // Platform attachment
        "cross-platform"  // Cross-platform attachment
    };

    enum AttestationConveyancePreference {
        "none",
        "indirect",
        "direct"
    };

    dictionary PublicKeyCredentialRequestOptions {
        required BufferSource                challenge;
        unsigned long                        timeout;
        USVString                            rpId;
        sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
        UserVerificationRequirement          userVerification = "preferred";
        AuthenticationExtensionsClientInputs extensions;
    };

    dictionary AuthenticationExtensionsClientInputs {
    };

    dictionary AuthenticationExtensionsClientOutputs {
    };

    typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs
    ;

    dictionary CollectedClientData {
```

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

    dictionary PublicKeyCredentialCreationOptions {
        required PublicKeyCredentialRpEntity         rp;
        required PublicKeyCredentialUserEntity       user;

        required BufferSource                             challenge;
        required sequence<PublicKeyCredentialParameters>  pubKeyCredParams;

        unsigned long                                timeout;
        sequence<PublicKeyCredentialDescriptor>      excludeCredentials = [];
        AuthenticatorSelectionCriteria               authenticatorSelection;
        AttestationConveyancePreference              attestation = "none";
        AuthenticationExtensionsClientInputs         extensions;
    };

    dictionary PublicKeyCredentialEntity {
        required DOMString    name;
        USVString             icon;
    };

    dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
        DOMString      id;
    };

    dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
        required BufferSource   id;
        required DOMString      displayName;
    };

    dictionary AuthenticatorSelectionCriteria {
        AuthenticatorAttachment      authenticatorAttachment;
        boolean                      requireResidentKey = false;
        UserVerificationRequirement  userVerification = "preferred";
    };

    enum AuthenticatorAttachment {
        "platform",       // Platform attachment
        "cross-platform"  // Cross-platform attachment
    };

    enum AttestationConveyancePreference {
        "none",
        "indirect",
        "direct"
    };

    dictionary PublicKeyCredentialRequestOptions {
        required BufferSource                challenge;
        unsigned long                        timeout;
        USVString                            rpId;
        sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
        UserVerificationRequirement          userVerification = "preferred";
        AuthenticationExtensionsClientInputs extensions;
    };

    dictionary AuthenticationExtensionsClientInputs {
    };

    dictionary AuthenticationExtensionsClientOutputs {
    };

    typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs
    ;

    dictionary CollectedClientData {
```

/Users/jehodges/Documents/work/standards/W3C/webauthn/index-master-3c5e383.txt, Top line: 7413

```
        required DOMString          type;
        required DOMString          challenge;
        required DOMString          origin;
        TokenBinding               tokenBinding;
    };

    dictionary TokenBinding {
        required TokenBindingStatus status;
        DOMString id;
    };

    enum TokenBindingStatus { "present", "supported", "not-supported" };

    enum PublicKeyCredentialType {
        "public-key"
    };

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

    enum AuthenticatorTransport {
        "usb",
        "nfc",
        "ble",
        "internal"
    };

    typedef long COSEAlgorithmIdentifier;

    enum UserVerificationRequirement {
        "required",
        "preferred",
        "discouraged"
    };

    partial dictionary AuthenticationExtensionsClientInputs {
      USVString appid;
    };

    partial dictionary AuthenticationExtensionsClientOutputs {
      boolean appid;
    };

    partial dictionary AuthenticationExtensionsClientInputs {
      USVString txAuthSimple;
    };

    partial dictionary AuthenticationExtensionsClientOutputs {
      USVString txAuthSimple;
    };

    dictionary txAuthGenericArg {
        required USVString contentType;    // MIME-Type of the content, e.g., "image
/png"
        required ArrayBuffer content;
    };

    partial dictionary AuthenticationExtensionsClientInputs {
      txAuthGenericArg txAuthGeneric;
    };

    partial dictionary AuthenticationExtensionsClientOutputs {
      ArrayBuffer txAuthGeneric;
    };

    typedef sequence<AAGUID> AuthenticatorSelectionList;

```

/Users/jehodges/Documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.txt, Top line: 7454

```
        required DOMString          type;
        required DOMString          challenge;
        required DOMString          origin;
        TokenBinding               tokenBinding;
    };

    dictionary TokenBinding {
        required TokenBindingStatus status;
        DOMString id;
    };

    enum TokenBindingStatus { "present", "supported", "not-supported" };

    enum PublicKeyCredentialType {
        "public-key"
    };

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

    enum AuthenticatorTransport {
        "usb",
        "nfc",
        "ble"
    };

    typedef long COSEAlgorithmIdentifier;

    enum UserVerificationRequirement {
        "required",
        "preferred",
        "discouraged"
    };

    partial dictionary AuthenticationExtensionsClientInputs {
      USVString appid;
    };

    partial dictionary AuthenticationExtensionsClientOutputs {
      boolean appid;
    };

    partial dictionary AuthenticationExtensionsClientInputs {
      USVString txAuthSimple;
    };

    partial dictionary AuthenticationExtensionsClientOutputs {
      USVString txAuthSimple;
    };

    dictionary txAuthGenericArg {
        required USVString contentType;    // MIME-Type of the content, e.g., "image
/png"
        required ArrayBuffer content;
    };

    partial dictionary AuthenticationExtensionsClientInputs {
      txAuthGenericArg txAuthGeneric;
    };

    partial dictionary AuthenticationExtensionsClientOutputs {
      ArrayBuffer txAuthGeneric;
    };

    typedef sequence<AAGUID> AuthenticatorSelectionList;

```

```
partial dictionary AuthenticationExtensionsClientInputs {
  AuthenticatorSelectionList authnSel;
};

typedef BufferSource      AAGUID;

partial dictionary AuthenticationExtensionsClientOutputs {
  boolean authnSel;
};

partial dictionary AuthenticationExtensionsClientInputs {
  boolean exts;
};

typedef sequence<USVString> AuthenticationExtensionsSupported;

partial dictionary AuthenticationExtensionsClientOutputs {
  AuthenticationExtensionsSupported exts;
};

partial dictionary AuthenticationExtensionsClientInputs {
  boolean uvi;
};

partial dictionary AuthenticationExtensionsClientOutputs {
  ArrayBuffer uvi;
};

partial dictionary AuthenticationExtensionsClientInputs {
  boolean loc;
};

partial dictionary AuthenticationExtensionsClientOutputs {
  Coordinates loc;
};

partial dictionary AuthenticationExtensionsClientInputs {
  boolean uvm;
};

typedef sequence<unsigned long> UvmEntry;
typedef sequence<UvmEntry> UvmEntries;

partial dictionary AuthenticationExtensionsClientOutputs {
  UvmEntries uvm;
};

dictionary authenticatorBiometricPerfBounds{
    float FAR;
    float FRR;
    };
```

Issues Index

The definitions of "lifetime of" and "becomes available" are intended to represent how devices are hot-plugged into (USB) or discovered by (NFC) browsers, and are underspecified. Resolving this with good definitions or some other means will be addressed by resolving Issue #613. RET

@balfanz 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 an "AttestationNotPrivateError". RET

The definitions of "lifetime of" and "becomes available" are intended to represent how devices are hot-plugged into (USB) or discovered by (NFC) browsers, and are underspecified. 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

```
partial dictionary AuthenticationExtensionsClientInputs {
  AuthenticatorSelectionList authnSel;
};

typedef BufferSource      AAGUID;

partial dictionary AuthenticationExtensionsClientOutputs {
  boolean authnSel;
};

partial dictionary AuthenticationExtensionsClientInputs {
  boolean exts;
};

typedef sequence<USVString> AuthenticationExtensionsSupported;

partial dictionary AuthenticationExtensionsClientOutputs {
  AuthenticationExtensionsSupported exts;
};

partial dictionary AuthenticationExtensionsClientInputs {
  boolean uvi;
};

partial dictionary AuthenticationExtensionsClientOutputs {
  ArrayBuffer uvi;
};

partial dictionary AuthenticationExtensionsClientInputs {
  boolean loc;
};

partial dictionary AuthenticationExtensionsClientOutputs {
  Coordinates loc;
};

partial dictionary AuthenticationExtensionsClientInputs {
  boolean uvm;
};

typedef sequence<unsigned long> UvmEntry;
typedef sequence<UvmEntry> UvmEntries;

partial dictionary AuthenticationExtensionsClientOutputs {
  UvmEntries uvm;
};

dictionary authenticatorBiometricPerfBounds{
    float FAR;
    float FRR;
    };
```

Issues Index

@balfanz 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 an "AttestationNotPrivateError". RET

```
    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
    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

    #base64url-encodingReferenced in:
      * 5.1. PublicKeyCredential Interface
      * 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 (2)
      * 5.10.1. Client data used in WebAuthn signatures (dictionary
        CollectedClientData)
      * 7.1. Registering a new credential
      * 7.2. Verifying an authentication assertion (2)

    #cborReferenced in:
      * 2.4. All Conformance Classes
      * 3. Dependencies
      * 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. Authenticator data (2)
      * 6.2.2. The authenticatorMakeCredential operation
      * 6.2.3. The authenticatorGetAssertion operation
      * 9. WebAuthn Extensions (2) (3) (4) (5) (6) (7)
      * 9.2. Defining extensions (2)
      * 9.3. Extending request parameters
      * 9.4. Client extension processing (2)
      * 9.5. Authenticator extension processing (2)

    #assertionReferenced in:
      * 7.1. Registering a new credential
      * 10.1. FIDO AppID Extension (appid)
      * 13.4. credentialId Unsigned

    #attestationReferenced in:
      * 4. Terminology (2)
      * 5.4.6. Attestation Conveyance Preference enumeration (enum
        AttestationConveyancePreference) (2)
      * 6. WebAuthn Authenticator Model (2)
      * 6.3. Attestation (2) (3) (4)
      * 8.2. Packed Attestation Statement Format
      * 11.1. WebAuthn Attestation Statement Format Identifier
        Registrations
      * 13. Security Considerations
      * 13.3.1. Considerations for Self and None Attestation Types and
        Ignoring Attestation

    #attestation-certificateReferenced in:
      * 4. Terminology (2)
      * 6.3.3. Attestation Types
      * 8.3.1. TPM attestation statement certificate requirements

    #attestation-key-pairReferenced in:
      * 4. Terminology (2)
      * 6.3. Attestation
      * 6.3.3. Attestation Types

    #attestation-private-keyReferenced in:
      * 6. WebAuthn Authenticator Model
      * 6.3. Attestation
      * 8.2. Packed Attestation Statement Format
```

```
    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

    #base64url-encodingReferenced in:
      * 5.1. PublicKeyCredential Interface
      * 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 (2)
      * 5.10.1. Client data used in WebAuthn signatures (dictionary
        CollectedClientData)
      * 7.1. Registering a new credential
      * 7.2. Verifying an authentication assertion (2)

    #cborReferenced in:
      * 2.4. All Conformance Classes
      * 3. Dependencies
      * 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. Authenticator data (2)
      * 6.2.2. The authenticatorMakeCredential operation
      * 6.2.3. The authenticatorGetAssertion operation
      * 9. WebAuthn Extensions (2) (3) (4) (5) (6) (7)
      * 9.2. Defining extensions (2)
      * 9.3. Extending request parameters
      * 9.4. Client extension processing (2)
      * 9.5. Authenticator extension processing (2)

    #assertionReferenced in:
      * 7.1. Registering a new credential
      * 10.1. FIDO AppID Extension (appid)
      * 13.4. credentialId Unsigned

    #attestationReferenced in:
      * 4. Terminology (2)
      * 5.4.6. Attestation Conveyance Preference enumeration (enum
        AttestationConveyancePreference) (2)
      * 6. WebAuthn Authenticator Model (2)
      * 6.3. Attestation (2) (3) (4)
      * 8.2. Packed Attestation Statement Format
      * 11.1. WebAuthn Attestation Statement Format Identifier
        Registrations
      * 13. Security Considerations
      * 13.3.1. Considerations for Self and None Attestation Types and
        Ignoring Attestation

    #attestation-certificateReferenced in:
      * 4. Terminology (2)
      * 6.3.3. Attestation Types
      * 8.3.1. TPM attestation statement certificate requirements

    #attestation-key-pairReferenced in:
      * 4. Terminology (2)
      * 6.3. Attestation
      * 6.3.3. Attestation Types

    #attestation-private-keyReferenced in:
      * 6. WebAuthn Authenticator Model
      * 6.3. Attestation
      * 8.2. Packed Attestation Statement Format

      #attestation-public-keyReferenced in:
        * 6.3. Attestation
        * 8.2. Packed Attestation Statement Format

      #authenticationReferenced in:
        * 1. Introduction (2)
        * 4. Terminology (2) (3) (4) (5) (6) (7)
        * 7.2. Verifying an authentication assertion (2) (3) (4)
        * 13. Security Considerations
        * 13.3. Security Benefits for Relying Parties
        * 13.3.1. Considerations for Self and None Attestation Types and
          Ignoring Attestation (2)
        * 14.3. Authentication Ceremony Privacy

      #authentication-assertionReferenced in:
        * 1. Introduction
        * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
        * 5.1. PublicKeyCredential Interface
        * 5.2.2. Web Authentication Assertion (interface
          AuthenticatorAssertionResponse)
        * 5.5. Options for Assertion Generation (dictionary
          PublicKeyCredentialRequestOptions)
        * 9. WebAuthn Extensions
        * 13.3.1. Considerations for Self and None Attestation Types and
          Ignoring Attestation

      #authenticatorReferenced in:
        * 1. Introduction (2) (3) (4)
        * 1.1. Use Cases
        * 2.2. Authenticators
        * 2.2.1. Backwards Compatibility with FIDO U2F
        * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
          (14) (15) (16) (17) (18) (19) (20) (21)
        * 5. Web Authentication API (2) (3)
        * 5.1. PublicKeyCredential Interface
        * 5.1.3. Create a new credential - PublicKeyCredential's
          [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
          (4)
        * 5.1.4.1. PublicKeyCredential's
          [[DiscoverFromExternalSource]](origin, options,
          sameOriginWithAncestors) method (2) (3) (4) (5) (6)
        * 5.2. Authenticator Responses (interface AuthenticatorResponse)
        * 5.2.1. Information about Public Key Credential (interface
          AuthenticatorAttestationResponse) (2)
        * 5.2.2. Web Authentication Assertion (interface
          AuthenticatorAssertionResponse)
        * 5.4.1. Public Key Entity Description (dictionary
          PublicKeyCredentialEntity) (2)
        * 5.4.3. User Account Parameters for Credential Generation
          (dictionary PublicKeyCredentialUserEntity)
        * 5.4.5. Authenticator Attachment enumeration (enum
          AuthenticatorAttachment)
        * 5.4.6. Attestation Conveyance Preference enumeration (enum
          AttestationConveyancePreference) (2)
        * 5.5. Options for Assertion Generation (dictionary
          PublicKeyCredentialRequestOptions)
        * 5.10.4. Authenticator Transport enumeration (enum
          AuthenticatorTransport)
        * 6. WebAuthn Authenticator Model (2) (3) (4) (5) (6)
        * 6.1. Authenticator data
        * 6.2.1. Lookup Credential Source by Credential ID algorithm
        * 6.2.2. The authenticatorMakeCredential operation (2)
        * 6.2.3. The authenticatorGetAssertion operation (2) (3) (4)
        * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)
        * 6.3.2. Attestation Statement Formats
        * 6.3.3. Attestation Types (2) (3) (4)
        * 6.3.4. Generating an Attestation Object
        * 7.1. Registering a new credential (2)
        * 7.2. Verifying an authentication assertion
```

```

      #attestation-public-keyReferenced in:
        * 6.3. Attestation
        * 8.2. Packed Attestation Statement Format

      #authenticationReferenced in:
        * 1. Introduction (2)
        * 4. Terminology (2) (3) (4) (5) (6) (7)
        * 7.2. Verifying an authentication assertion (2) (3) (4)
        * 13. Security Considerations
        * 13.3. Security Benefits for Relying Parties
        * 13.3.1. Considerations for Self and None Attestation Types and
          Ignoring Attestation (2)
        * 14.3. Authentication Ceremony Privacy

      #authentication-assertionReferenced in:
        * 1. Introduction
        * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
        * 5.1. PublicKeyCredential Interface
        * 5.2.2. Web Authentication Assertion (interface
          AuthenticatorAssertionResponse)
        * 5.5. Options for Assertion Generation (dictionary
          PublicKeyCredentialRequestOptions)
        * 9. WebAuthn Extensions
        * 13.3.1. Considerations for Self and None Attestation Types and
          Ignoring Attestation

      #authenticatorReferenced in:
        * 1. Introduction (2) (3) (4)
        * 1.1. Use Cases
        * 2.2. Authenticators
        * 2.2.1. Backwards Compatibility with FIDO U2F
        * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
          (14) (15) (16) (17) (18) (19) (20) (21)
        * 5. Web Authentication API (2) (3)
        * 5.1. PublicKeyCredential Interface
        * 5.1.3. Create a new credential - PublicKeyCredential's
          [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
          (4) (5) (6) (7)
        * 5.1.4.1. PublicKeyCredential's
          [[DiscoverFromExternalSource]](origin, options,
          sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7) (8)
        * 5.2. Authenticator Responses (interface AuthenticatorResponse)
        * 5.2.1. Information about Public Key Credential (interface
          AuthenticatorAttestationResponse) (2)
        * 5.2.2. Web Authentication Assertion (interface
          AuthenticatorAssertionResponse)
        * 5.4.1. Public Key Entity Description (dictionary
          PublicKeyCredentialEntity) (2)
        * 5.4.3. User Account Parameters for Credential Generation
          (dictionary PublicKeyCredentialUserEntity)
        * 5.4.5. Authenticator Attachment enumeration (enum
          AuthenticatorAttachment)
        * 5.4.6. Attestation Conveyance Preference enumeration (enum
          AttestationConveyancePreference) (2)
        * 5.5. Options for Assertion Generation (dictionary
          PublicKeyCredentialRequestOptions)


        * 6. WebAuthn Authenticator Model (2) (3) (4) (5) (6)
        * 6.1. Authenticator data
        * 6.2.1. Lookup Credential Source by Credential ID algorithm
        * 6.2.2. The authenticatorMakeCredential operation (2)
        * 6.2.3. The authenticatorGetAssertion operation (2) (3) (4)
        * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)
        * 6.3.2. Attestation Statement Formats
        * 6.3.3. Attestation Types (2) (3) (4)
        * 6.3.4. Generating an Attestation Object
        * 7.1. Registering a new credential (2)
        * 7.2. Verifying an authentication assertion
```

/Users/jehodges/Documents/work/standards/W3C/webauthn/index-master-3c5e383.txt, Top line: 7693

```
     * 8.2. Packed Attestation Statement Format
     * 8.4. Android Key Attestation Statement Format
     * 8.5. Android SafetyNet Attestation Statement Format
     * 8.7. None Attestation Statement Format
     * 10.5. Supported Extensions Extension (exts)
     * 10.6. User Verification Index Extension (uvi)
     * 10.8. User Verification Method Extension (uvm)
     * 12. Sample scenarios
     * 13. Security Considerations (2) (3) (4) (5)
     * 13.2.2. Attestation Certificate and Attestation Certificate CA
       Compromise
     * 13.3. Security Benefits for Relying Parties (2) (3) (4) (5) (6)
     * 13.4. credentialId Unsigned
     * 14.1. Attestation Privacy (2) (3)
     * 14.2. Registration Ceremony Privacy (2) (3) (4) (5) (6)

   #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)
     * 5.1.6. Preventing silent access to an existing credential -
       PublicKeyCredential's [[preventSilentAccess]](credential,
       sameOriginWithAncestors) method

   #biometric-recognitionReferenced in:
     * 4. Terminology (2) (3)

   #biometric-authenticatorReferenced in:
     * 10.9. Biometric Authenticator Performance Bounds Extension
       (biometricPerfBounds)

   #ceremonyReferenced in:
     * 1. Introduction
     * 4. Terminology (2) (3) (4) (5) (6) (7)
     * 7.1. Registering a new credential (2)
     * 7.2. Verifying an authentication assertion (2)
     * 13. Security Considerations
     * 13.3. Security Benefits for Relying Parties
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation (2)
     * 14.2. Registration Ceremony Privacy
     * 14.3. Authentication Ceremony Privacy (2)

   #clientReferenced in:
     * 4. Terminology





     * 5.1.7. Availability of User-Verifying Platform Authenticator -
       PublicKeyCredential's
       isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
     * 5.4.5. Authenticator Attachment enumeration (enum
       AuthenticatorAttachment) (2) (3)
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 5.10.4. Authenticator Transport enumeration (enum
       AuthenticatorTransport)
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation

   #client-side-resident-credential-private-keyReferenced in:
     * 4. Terminology (2)
     * 5.1.3. Create a new credential - PublicKeyCredential's
```

/Users/jehodges/Documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.txt, Top line: 7716

```
     * 8.2. Packed Attestation Statement Format
     * 8.4. Android Key Attestation Statement Format
     * 8.5. Android SafetyNet Attestation Statement Format
     * 8.7. None Attestation Statement Format
     * 10.5. Supported Extensions Extension (exts)
     * 10.6. User Verification Index Extension (uvi)
     * 10.8. User Verification Method Extension (uvm)
     * 12. Sample scenarios
     * 13. Security Considerations (2) (3) (4) (5)
     * 13.2.2. Attestation Certificate and Attestation Certificate CA
       Compromise
     * 13.3. Security Benefits for Relying Parties (2) (3) (4) (5) (6)
     * 13.4. credentialId Unsigned
     * 14.1. Attestation Privacy (2) (3)
     * 14.2. Registration Ceremony Privacy (2) (3) (4) (5) (6)

   #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)
     * 5.1.6. Preventing silent access to an existing credential -
       PublicKeyCredential's [[preventSilentAccess]](credential,
       sameOriginWithAncestors) method

   #biometric-recognitionReferenced in:
     * 4. Terminology (2) (3)

   #biometric-authenticatorReferenced in:
     * 10.9. Biometric Authenticator Performance Bounds Extension
       (biometricPerfBounds)

   #ceremonyReferenced in:
     * 1. Introduction
     * 4. Terminology (2) (3) (4) (5) (6) (7)
     * 7.1. Registering a new credential (2)
     * 7.2. Verifying an authentication assertion (2)
     * 13. Security Considerations
     * 13.3. Security Benefits for Relying Parties
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation (2)
     * 14.2. Registration Ceremony Privacy
     * 14.3. Authentication Ceremony Privacy (2)

   #clientReferenced in:
     * 4. Terminology
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.1.7. Availability of User-Verifying Platform Authenticator -
       PublicKeyCredential's
       isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
     * 5.4.5. Authenticator Attachment enumeration (enum
       AuthenticatorAttachment) (2) (3)
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)


     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation

   #client-side-resident-credential-private-keyReferenced in:
     * 4. Terminology (2)
     * 5.1.3. Create a new credential - PublicKeyCredential's
```

```
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.4.4. Authenticator Selection Criteria (dictionary
       AuthenticatorSelectionCriteria) (2)
     * 6.2.2. The authenticatorMakeCredential operation (2)

   #conforming-user-agentReferenced in:
     * 1. Introduction
     * 2.1. User Agents
     * 2.2. Authenticators
     * 4. Terminology (2)

   #credential-idReferenced in:
     * 4. Terminology (2) (3) (4)
     * 5.1. PublicKeyCredential Interface (2)
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 6.2.1. Lookup Credential Source by Credential ID algorithm
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation
     * 6.3.1. Attested credential data
     * 7.1. Registering a new credential
     * 8.6. FIDO U2F Attestation Statement Format
     * 12.1. Registration
     * 12.3. Authentication (2) (3)
     * 13.4. credentialId Unsigned (2) (3)

   #credential-public-keyReferenced in:
     * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 6. WebAuthn Authenticator Model
     * 6.3. Attestation (2) (3)
     * 6.3.1. Attested credential data (2) (3)
     * 12.1. Registration (2)
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation
     * 13.4. credentialId Unsigned

   #user-public-keyReferenced in:
     * 4. Terminology
     * 8.6. FIDO U2F Attestation Statement Format

   #credential-key-pairReferenced in:
     * 4. Terminology (2) (3)

   #credential-private-keyReferenced in:
     * 4. Terminology (2) (3) (4) (5) (6)
     * 5.1. PublicKeyCredential Interface
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse)
     * 6. WebAuthn Authenticator Model
     * 6.3. Attestation (2)
     * 7.2. Verifying an authentication assertion

   #human-palatabilityReferenced in:
     * 4. Terminology
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity) (2)

   #public-key-credential-sourceReferenced in:
     * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 6. WebAuthn Authenticator Model
     * 6.2.1. Lookup Credential Source by Credential ID algorithm (2)
```

```
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.4.4. Authenticator Selection Criteria (dictionary
       AuthenticatorSelectionCriteria) (2)
     * 6.2.2. The authenticatorMakeCredential operation (2)

   #conforming-user-agentReferenced in:
     * 1. Introduction
     * 2.1. User Agents
     * 2.2. Authenticators
     * 4. Terminology (2)

   #credential-idReferenced in:
     * 4. Terminology (2) (3) (4)
     * 5.1. PublicKeyCredential Interface (2)
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 6.2.1. Lookup Credential Source by Credential ID algorithm
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation
     * 6.3.1. Attested credential data
     * 7.1. Registering a new credential
     * 8.6. FIDO U2F Attestation Statement Format
     * 12.1. Registration
     * 12.3. Authentication (2) (3)
     * 13.4. credentialId Unsigned (2) (3)

   #credential-public-keyReferenced in:
     * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 6. WebAuthn Authenticator Model
     * 6.3. Attestation (2) (3)
     * 6.3.1. Attested credential data (2) (3)
     * 12.1. Registration (2)
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation
     * 13.4. credentialId Unsigned

   #user-public-keyReferenced in:
     * 4. Terminology
     * 8.6. FIDO U2F Attestation Statement Format

   #credential-key-pairReferenced in:
     * 4. Terminology (2) (3)

   #credential-private-keyReferenced in:
     * 4. Terminology (2) (3) (4) (5) (6)
     * 5.1. PublicKeyCredential Interface
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse)
     * 6. WebAuthn Authenticator Model
     * 6.3. Attestation (2)
     * 7.2. Verifying an authentication assertion

   #human-palatabilityReferenced in:
     * 4. Terminology
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity) (2)

   #public-key-credential-sourceReferenced in:
     * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 6. WebAuthn Authenticator Model
     * 6.2.1. Lookup Credential Source by Credential ID algorithm (2)
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation (2)

   #public-key-credential-source-typeReferenced in:
     * 6.2.2. The authenticatorMakeCredential operation (2)

   #public-key-credential-source-idReferenced in:
     * 6.2.1. Lookup Credential Source by Credential ID algorithm (2)
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation

   #public-key-credential-source-privatekeyReferenced in:
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation

   #public-key-credential-source-rpidReferenced in:
     * 6. WebAuthn Authenticator Model
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation

   #public-key-credential-source-userhandleReferenced in:
     * 6. WebAuthn Authenticator Model
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation (2)

   #public-key-credential-source-otheruiReferenced in:
     * 6.2.2. The authenticatorMakeCredential operation

   #public-key-credential-source-managing-authenticatorReferenced in:
     * 4. Terminology
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)

   #public-key-credentialReferenced in:
     * 1. Introduction (2) (3) (4) (5)
     * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
     * 5. Web Authentication API (2) (3) (4)
     * 5.1. PublicKeyCredential Interface
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4. Use an existing credential to make an assertion -
       PublicKeyCredential's [[Get]](options) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)
     * 5.4.4. Authenticator Selection Criteria (dictionary
       AuthenticatorSelectionCriteria)
     * 5.4.5. Authenticator Attachment enumeration (enum
       AuthenticatorAttachment) (2) (3)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions)
     * 5.10. Supporting Data Structures
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor) (2) (3) (4)
     * 6. WebAuthn Authenticator Model (2)
     * 6.2.3. The authenticatorGetAssertion operation
     * 6.3. Attestation (2)
     * 6.3.2. Attestation Statement Formats
     * 6.3.3. Attestation Types
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion (2)
     * 9. WebAuthn Extensions (2)
     * 12. Sample scenarios
     * 13.2.2. Attestation Certificate and Attestation Certificate CA
       Compromise (2)
     * 14.2. Registration Ceremony Privacy (2) (3)
```

```
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation (2)

   #public-key-credential-source-typeReferenced in:
     * 6.2.2. The authenticatorMakeCredential operation (2)

   #public-key-credential-source-idReferenced in:
     * 6.2.1. Lookup Credential Source by Credential ID algorithm (2)
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation

   #public-key-credential-source-privatekeyReferenced in:
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation

   #public-key-credential-source-rpidReferenced in:
     * 6. WebAuthn Authenticator Model
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation

   #public-key-credential-source-userhandleReferenced in:
     * 6. WebAuthn Authenticator Model
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation (2)

   #public-key-credential-source-otheruiReferenced in:
     * 6.2.2. The authenticatorMakeCredential operation

   #public-key-credential-source-managing-authenticatorReferenced in:
     * 4. Terminology
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)

   #public-key-credentialReferenced in:
     * 1. Introduction (2) (3) (4) (5)
     * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
     * 5. Web Authentication API (2) (3) (4)
     * 5.1. PublicKeyCredential Interface
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4. Use an existing credential to make an assertion -
       PublicKeyCredential's [[Get]](options) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2) (3)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)
     * 5.4.4. Authenticator Selection Criteria (dictionary
       AuthenticatorSelectionCriteria)
     * 5.4.5. Authenticator Attachment enumeration (enum
       AuthenticatorAttachment) (2) (3)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions)
     * 5.10. Supporting Data Structures
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor) (2) (3) (4)
     * 6. WebAuthn Authenticator Model (2)
     * 6.2.3. The authenticatorGetAssertion operation
     * 6.3. Attestation (2)
     * 6.3.2. Attestation Statement Formats
     * 6.3.3. Attestation Types
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion (2)
     * 9. WebAuthn Extensions (2)
     * 12. Sample scenarios
     * 13.2.2. Attestation Certificate and Attestation Certificate CA
       Compromise (2)
     * 14.2. Registration Ceremony Privacy (2) (3)
```

```
       * 14.3. Authentication Ceremony Privacy (2) (3) (4) (5)

     #registrationReferenced in:
       * 1. Introduction (2)
       * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
       * 7.1. Registering a new credential (2) (3)
       * 10.9. Biometric Authenticator Performance Bounds Extension
         (biometricPerfBounds)
       * 13. Security Considerations
       * 13.3. Security Benefits for Relying Parties
       * 13.3.1. Considerations for Self and None Attestation Types and
         Ignoring Attestation

     #relying-partyReferenced in:
       * 1. Introduction (2) (3) (4) (5) (6) (7)
       * 1.1.3. Other use cases and configurations
       * 2.3. Relying Parties
       * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
         (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26)
         (27) (28) (29) (30)
       * 5. Web Authentication API (2) (3) (4) (5) (6) (7)
       * 5.1. PublicKeyCredential Interface (2)
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
         (4) (5)
       * 5.1.4. Use an existing credential to make an assertion -
         PublicKeyCredential's [[Get]](options) method (2)
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method (2) (3) (4)
       * 5.1.7. Availability of User-Verifying Platform Authenticator -
         PublicKeyCredential's
         isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3)
       * 5.2. Authenticator Responses (interface AuthenticatorResponse)
       * 5.2.1. Information about Public Key Credential (interface
         AuthenticatorAttestationResponse) (2)
       * 5.2.2. Web Authentication Assertion (interface
         AuthenticatorAssertionResponse)
       * 5.4. Options for Credential Creation (dictionary
         PublicKeyCredentialCreationOptions) (2) (3) (4) (5)
       * 5.4.1. Public Key Entity Description (dictionary
         PublicKeyCredentialEntity) (2) (3) (4) (5)
       * 5.4.2. RP Parameters for Credential Generation (dictionary
         PublicKeyCredentialRpEntity) (2)
       * 5.4.3. User Account Parameters for Credential Generation
         (dictionary PublicKeyCredentialUserEntity)
       * 5.4.4. Authenticator Selection Criteria (dictionary
         AuthenticatorSelectionCriteria) (2) (3)
       * 5.4.5. Authenticator Attachment enumeration (enum
         AuthenticatorAttachment) (2) (3) (4) (5) (6)
       * 5.4.6. Attestation Conveyance Preference enumeration (enum
         AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)
       * 5.5. Options for Assertion Generation (dictionary
         PublicKeyCredentialRequestOptions)
       * 5.10.1. Client data used in WebAuthn signatures (dictionary
         CollectedClientData) (2) (3) (4)
       * 5.10.4. Authenticator Transport enumeration (enum
         AuthenticatorTransport) (2)
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement) (2) (3) (4)
       * 6. WebAuthn Authenticator Model (2) (3)
       * 6.1. Authenticator data (2)
       * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6)
       * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)
         (6)
       * 6.2.3. The authenticatorGetAssertion operation (2) (3)
       * 6.3. Attestation (2) (3) (4) (5) (6)
       * 6.3.3. Attestation Types
       * 7. Relying Party Operations (2) (3) (4)
       * 7.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)
```

```
       * 14.3. Authentication Ceremony Privacy (2) (3) (4) (5)

     #registrationReferenced in:
       * 1. Introduction (2)
       * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
       * 7.1. Registering a new credential (2) (3)
       * 10.9. Biometric Authenticator Performance Bounds Extension
         (biometricPerfBounds)
       * 13. Security Considerations
       * 13.3. Security Benefits for Relying Parties
       * 13.3.1. Considerations for Self and None Attestation Types and
         Ignoring Attestation

     #relying-partyReferenced in:
       * 1. Introduction (2) (3) (4) (5) (6) (7)
       * 1.1.3. Other use cases and configurations
       * 2.3. Relying Parties
       * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
         (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26)
         (27) (28) (29) (30)
       * 5. Web Authentication API (2) (3) (4) (5) (6) (7)
       * 5.1. PublicKeyCredential Interface (2)
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
         (4) (5)
       * 5.1.4. Use an existing credential to make an assertion -
         PublicKeyCredential's [[Get]](options) method (2)
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method (2) (3) (4)
       * 5.1.7. Availability of User-Verifying Platform Authenticator -
         PublicKeyCredential's
         isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3)
       * 5.2. Authenticator Responses (interface AuthenticatorResponse)
       * 5.2.1. Information about Public Key Credential (interface
         AuthenticatorAttestationResponse) (2)
       * 5.2.2. Web Authentication Assertion (interface
         AuthenticatorAssertionResponse)
       * 5.4. Options for Credential Creation (dictionary
         PublicKeyCredentialCreationOptions) (2) (3) (4) (5)
       * 5.4.1. Public Key Entity Description (dictionary
         PublicKeyCredentialEntity) (2) (3) (4) (5)
       * 5.4.2. RP Parameters for Credential Generation (dictionary
         PublicKeyCredentialRpEntity) (2)
       * 5.4.3. User Account Parameters for Credential Generation
         (dictionary PublicKeyCredentialUserEntity)
       * 5.4.4. Authenticator Selection Criteria (dictionary
         AuthenticatorSelectionCriteria) (2) (3)
       * 5.4.5. Authenticator Attachment enumeration (enum
         AuthenticatorAttachment) (2) (3) (4) (5) (6)
       * 5.4.6. Attestation Conveyance Preference enumeration (enum
         AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)
       * 5.5. Options for Assertion Generation (dictionary
         PublicKeyCredentialRequestOptions)
       * 5.10.1. Client data used in WebAuthn signatures (dictionary
         CollectedClientData) (2) (3) (4)
       * 5.10.4. Authenticator Transport enumeration (enum
         AuthenticatorTransport) (2)
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement) (2) (3) (4)
       * 6. WebAuthn Authenticator Model (2) (3)
       * 6.1. Authenticator data (2)
       * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6)
       * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)
         (6)
       * 6.2.3. The authenticatorGetAssertion operation (2) (3)
       * 6.3. Attestation (2) (3) (4) (5) (6)
       * 6.3.3. Attestation Types
       * 7. Relying Party Operations (2) (3) (4)
       * 7.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)
```

```
          (10) (11) (12) (13)
        * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)
          (8)
        * 8.4. Android Key Attestation Statement Format
        * 8.7. None Attestation Statement Format
        * 9. WebAuthn Extensions (2) (3) (4) (5)
        * 9.2. Defining extensions (2)
        * 9.3. Extending request parameters (2) (3) (4)
        * 10.1. FIDO AppID Extension (appid) (2)
        * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
        * 10.4. Authenticator Selection Extension (authnSel) (2) (3)
        * 10.5. Supported Extensions Extension (exts) (2)
        * 10.6. User Verification Index Extension (uvi)
        * 10.7. Location Extension (loc) (2)
        * 10.8. User Verification Method Extension (uvm)
        * 10.9. Biometric Authenticator Performance Bounds Extension
          (biometricPerfBounds) (2) (3)
        * 11.2. WebAuthn Extension Identifier Registrations (2)
        * 12.1. Registration (2) (3) (4) (5)
        * 12.2. Registration Specifically with User Verifying Platform
          Authenticator (2) (3)
        * 12.3. Authentication (2) (3) (4) (5)
        * 12.5. Decommissioning (2)
        * 13. Security Considerations (2) (3) (4)
        * 13.1. Cryptographic Challenges
        * 13.2.2. Attestation Certificate and Attestation Certificate CA
          Compromise (2) (3) (4) (5) (6)
        * 13.3. Security Benefits for Relying Parties (2) (3) (4) (5) (6) (7)
        * 13.3.1. Considerations for Self and None Attestation Types and
          Ignoring Attestation (2) (3) (4) (5) (6)
        * 13.4. credentialId Unsigned
        * 14.1. Attestation Privacy
        * 14.2. Registration Ceremony Privacy (2) (3) (4)
        * 14.3. Authentication Ceremony Privacy (2) (3) (4)

      #relying-party-identifierReferenced in:
        * 4. Terminology
        * 5. Web Authentication API
        * 5.4. Options for Credential Creation (dictionary
          PublicKeyCredentialCreationOptions)
        * 5.5. Options for Assertion Generation (dictionary
          PublicKeyCredentialRequestOptions)

      #rp-idReferenced in:
        * 4. Terminology (2) (3) (4) (5)
        * 5. Web Authentication API (2) (3) (4) (5)
        * 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 (2)
        * 5.4.2. RP Parameters for Credential Generation (dictionary
          PublicKeyCredentialRpEntity)
        * 6.1. Authenticator data (2) (3) (4) (5) (6)
        * 6.1.1. Signature Counter Considerations
        * 6.2.2. The authenticatorMakeCredential operation (2) (3)
        * 6.2.3. The authenticatorGetAssertion operation (2)
        * 7.1. Registering a new credential (2)
        * 7.2. Verifying an authentication assertion
        * 8.4. Android Key Attestation Statement Format
        * 8.6. FIDO U2F Attestation Statement Format
        * 10.1. FIDO AppID Extension (appid)

      #test-of-user-presenceReferenced in:
        * 4. Terminology (2) (3) (4) (5) (6)
        * 6.2.2. The authenticatorMakeCredential operation (2)
        * 6.2.3. The authenticatorGetAssertion operation
        * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
        * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)

```

```
          (10) (11) (12) (13)
        * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)
          (8)
        * 8.4. Android Key Attestation Statement Format
        * 8.7. None Attestation Statement Format
        * 9. WebAuthn Extensions (2) (3) (4) (5)
        * 9.2. Defining extensions (2)
        * 9.3. Extending request parameters (2) (3) (4)
        * 10.1. FIDO AppID Extension (appid) (2)
        * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
        * 10.4. Authenticator Selection Extension (authnSel) (2) (3)
        * 10.5. Supported Extensions Extension (exts) (2)
        * 10.6. User Verification Index Extension (uvi)
        * 10.7. Location Extension (loc) (2)
        * 10.8. User Verification Method Extension (uvm)
        * 10.9. Biometric Authenticator Performance Bounds Extension
          (biometricPerfBounds) (2) (3)
        * 11.2. WebAuthn Extension Identifier Registrations (2)
        * 12.1. Registration (2) (3) (4) (5)
        * 12.2. Registration Specifically with User Verifying Platform
          Authenticator (2) (3)
        * 12.3. Authentication (2) (3) (4) (5)
        * 12.5. Decommissioning (2)
        * 13. Security Considerations (2) (3) (4)
        * 13.1. Cryptographic Challenges
        * 13.2.2. Attestation Certificate and Attestation Certificate CA
          Compromise (2) (3) (4) (5) (6)
        * 13.3. Security Benefits for Relying Parties (2) (3) (4) (5) (6) (7)
        * 13.3.1. Considerations for Self and None Attestation Types and
          Ignoring Attestation (2) (3) (4) (5) (6)
        * 13.4. credentialId Unsigned
        * 14.1. Attestation Privacy
        * 14.2. Registration Ceremony Privacy (2) (3) (4)
        * 14.3. Authentication Ceremony Privacy (2) (3) (4)

      #relying-party-identifierReferenced in:
        * 4. Terminology
        * 5. Web Authentication API
        * 5.4. Options for Credential Creation (dictionary
          PublicKeyCredentialCreationOptions)
        * 5.5. Options for Assertion Generation (dictionary
          PublicKeyCredentialRequestOptions)

      #rp-idReferenced in:
        * 4. Terminology (2) (3) (4) (5)
        * 5. Web Authentication API (2) (3) (4) (5)
        * 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 (2)
        * 5.4.2. RP Parameters for Credential Generation (dictionary
          PublicKeyCredentialRpEntity)
        * 6.1. Authenticator data (2) (3) (4) (5) (6)
        * 6.1.1. Signature Counter Considerations
        * 6.2.2. The authenticatorMakeCredential operation (2) (3)
        * 6.2.3. The authenticatorGetAssertion operation (2)
        * 7.1. Registering a new credential (2)
        * 7.2. Verifying an authentication assertion
        * 8.4. Android Key Attestation Statement Format
        * 8.6. FIDO U2F Attestation Statement Format
        * 10.1. FIDO AppID Extension (appid)

      #test-of-user-presenceReferenced in:
        * 4. Terminology (2) (3) (4) (5) (6)
        * 6.2.2. The authenticatorMakeCredential operation (2)
        * 6.2.3. The authenticatorGetAssertion operation
        * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
        * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)

        #user-consentReferenced in:
          * 1. Introduction (2)
          * 4. Terminology (2)
          * 5. Web Authentication API
          * 5.1.3. Create a new credential - PublicKeyCredential's
            [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
          * 5.1.4. Use an existing credential to make an assertion -
            PublicKeyCredential's [[Get]](options) method
          * 5.1.4.1. PublicKeyCredential's
            [[DiscoverFromExternalSource]](origin, options,
            sameOriginWithAncestors) method
          * 5.2.2. Web Authentication Assertion (interface
            AuthenticatorAssertionResponse)
          * 5.4.6. Attestation Conveyance Preference enumeration (enum
            AttestationConveyancePreference)
          * 6. WebAuthn Authenticator Model (2) (3)
          * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)
            (6) (7) (8)
          * 6.2.3. The authenticatorGetAssertion operation (2) (3) (4) (5)
          * 11.2. WebAuthn Extension Identifier Registrations
          * 14.2. Registration Ceremony Privacy (2)
          * 14.3. Authentication Ceremony Privacy (2) (3)

        #user-handleReferenced in:
          * 2.2.1. Backwards Compatibility with FIDO U2F
          * 4. Terminology
          * 5.1.4.1. PublicKeyCredential's
            [[DiscoverFromExternalSource]](origin, options,
            sameOriginWithAncestors) method (2)
          * 5.2.2. Web Authentication Assertion (interface
            AuthenticatorAssertionResponse) (2)
          * 5.4.3. User Account Parameters for Credential Generation
            (dictionary PublicKeyCredentialUserEntity)
          * 6.2.2. The authenticatorMakeCredential operation

        #user-verificationReferenced in:
          * 1. Introduction
          * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
          * 5.1.3. Create a new credential - PublicKeyCredential's
            [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
          * 5.1.4.1. PublicKeyCredential's
            [[DiscoverFromExternalSource]](origin, options,
            sameOriginWithAncestors) method (2) (3)
          * 5.1.7. Availability of User-Verifying Platform Authenticator -
            PublicKeyCredential's
            isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
            (5)
          * 5.4.4. Authenticator Selection Criteria (dictionary
            AuthenticatorSelectionCriteria)
          * 5.5. Options for Assertion Generation (dictionary
            PublicKeyCredentialRequestOptions)
          * 5.10.6. User Verification Requirement enumeration (enum
            UserVerificationRequirement) (2) (3) (4)
          * 6.2.2. The authenticatorMakeCredential operation (2) (3)
          * 6.2.3. The authenticatorGetAssertion operation
          * 7.1. Registering a new credential
          * 7.2. Verifying an authentication assertion
          * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
          * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
          * 12.2. Registration Specifically with User Verifying Platform
            Authenticator
          * 13.3. Security Benefits for Relying Parties

        #concept-user-presentReferenced in:
          * 4. Terminology
          * 6.1. Authenticator data (2) (3)
          * 7.1. Registering a new credential
          * 7.2. Verifying an authentication assertion

        #upReferenced in:
```

```
        #user-consentReferenced in:
          * 1. Introduction (2)
          * 4. Terminology (2)
          * 5. Web Authentication API
          * 5.1.3. Create a new credential - PublicKeyCredential's
            [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
          * 5.1.4. Use an existing credential to make an assertion -
            PublicKeyCredential's [[Get]](options) method
          * 5.1.4.1. PublicKeyCredential's
            [[DiscoverFromExternalSource]](origin, options,
            sameOriginWithAncestors) method
          * 5.2.2. Web Authentication Assertion (interface
            AuthenticatorAssertionResponse)
          * 5.4.6. Attestation Conveyance Preference enumeration (enum
            AttestationConveyancePreference)
          * 6. WebAuthn Authenticator Model (2) (3)
          * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)
            (6) (7) (8)
          * 6.2.3. The authenticatorGetAssertion operation (2) (3) (4) (5)
          * 11.2. WebAuthn Extension Identifier Registrations
          * 14.2. Registration Ceremony Privacy (2)
          * 14.3. Authentication Ceremony Privacy (2) (3)

        #user-handleReferenced in:
          * 2.2.1. Backwards Compatibility with FIDO U2F
          * 4. Terminology
          * 5.1.4.1. PublicKeyCredential's
            [[DiscoverFromExternalSource]](origin, options,
            sameOriginWithAncestors) method (2)
          * 5.2.2. Web Authentication Assertion (interface
            AuthenticatorAssertionResponse) (2)
          * 5.4.3. User Account Parameters for Credential Generation
            (dictionary PublicKeyCredentialUserEntity)
          * 6.2.2. The authenticatorMakeCredential operation

        #user-verificationReferenced in:
          * 1. Introduction
          * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
          * 5.1.3. Create a new credential - PublicKeyCredential's
            [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
          * 5.1.4.1. PublicKeyCredential's
            [[DiscoverFromExternalSource]](origin, options,
            sameOriginWithAncestors) method (2) (3)
          * 5.1.7. Availability of User-Verifying Platform Authenticator -
            PublicKeyCredential's
            isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
            (5)
          * 5.4.4. Authenticator Selection Criteria (dictionary
            AuthenticatorSelectionCriteria)
          * 5.5. Options for Assertion Generation (dictionary
            PublicKeyCredentialRequestOptions)
          * 5.10.6. User Verification Requirement enumeration (enum
            UserVerificationRequirement) (2) (3) (4)
          * 6.2.2. The authenticatorMakeCredential operation (2) (3)
          * 6.2.3. The authenticatorGetAssertion operation
          * 7.1. Registering a new credential
          * 7.2. Verifying an authentication assertion
          * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
          * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
          * 12.2. Registration Specifically with User Verifying Platform
            Authenticator
          * 13.3. Security Benefits for Relying Parties

        #concept-user-presentReferenced in:
          * 4. Terminology
          * 6.1. Authenticator data (2) (3)
          * 7.1. Registering a new credential
          * 7.2. Verifying an authentication assertion

        #upReferenced in:
```

```
     * 6.1. Authenticator data
     * 6.1.2. FIDO U2F signature format compatibility

   #concept-user-verifiedReferenced in:
     * 4. Terminology
     * 6.1. Authenticator data (2) (3)
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion

   #uvReferenced in:
     * 5.10.6. User Verification Requirement enumeration (enum
       UserVerificationRequirement) (2)
     * 6.1. Authenticator data

   #webauthn-clientReferenced in:
     * 4. Terminology (2) (3) (4)
     * 6.2. Authenticator operations
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation
     * 13. Security Considerations

   #web-authentication-apiReferenced in:
     * 1. Introduction (2) (3)
     * 4. Terminology (2) (3) (4)
     * 13. Security Considerations

   #publickeycredentialReferenced in:
     * 1. Introduction
     * 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
     * 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
     * 5.1.5. Store an existing credential - PublicKeyCredential's
       [[Store]](credential, sameOriginWithAncestors) method (2)
     * 5.1.7. Availability of User-Verifying Platform Authenticator -
       PublicKeyCredential's
       isUserVerifyingPlatformAuthenticatorAvailable() method
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 7. Relying Party Operations
     * 7.2. Verifying an authentication assertion

   #dom-publickeycredential-rawidReferenced in:
     * 5.1. PublicKeyCredential Interface
     * 7.2. Verifying an authentication assertion

   #dom-publickeycredential-getclientextensionresultsReferenced in:
     * 5.1. PublicKeyCredential Interface
     * 9.4. Client extension processing

   #dom-publickeycredential-responseReferenced in:
     * 5.1. PublicKeyCredential Interface
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 7.2. Verifying an authentication assertion (2)

   #dom-publickeycredential-identifier-slotReferenced in:
     * 5.1. PublicKeyCredential Interface (2)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   #dom-publickeycredential-clientextensionsresults-slotReferenced in:
```

```
     * 6.1. Authenticator data
     * 6.1.2. FIDO U2F signature format compatibility

   #concept-user-verifiedReferenced in:
     * 4. Terminology
     * 6.1. Authenticator data (2) (3)
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion

   #uvReferenced in:
     * 5.10.6. User Verification Requirement enumeration (enum
       UserVerificationRequirement) (2)
     * 6.1. Authenticator data

   #webauthn-clientReferenced in:
     * 4. Terminology (2) (3) (4)
     * 6.2. Authenticator operations
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation
     * 13. Security Considerations

   #web-authentication-apiReferenced in:
     * 1. Introduction (2) (3)
     * 4. Terminology (2) (3) (4)
     * 13. Security Considerations

   #publickeycredentialReferenced in:
     * 1. Introduction
     * 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
     * 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
     * 5.1.5. Store an existing credential - PublicKeyCredential's
       [[Store]](credential, sameOriginWithAncestors) method (2)
     * 5.1.7. Availability of User-Verifying Platform Authenticator -
       PublicKeyCredential's
       isUserVerifyingPlatformAuthenticatorAvailable() method
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 7. Relying Party Operations
     * 7.2. Verifying an authentication assertion

   #dom-publickeycredential-rawidReferenced in:
     * 5.1. PublicKeyCredential Interface
     * 7.2. Verifying an authentication assertion

   #dom-publickeycredential-getclientextensionresultsReferenced in:
     * 5.1. PublicKeyCredential Interface
     * 9.4. Client extension processing

   #dom-publickeycredential-responseReferenced in:
     * 5.1. PublicKeyCredential Interface
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 7.2. Verifying an authentication assertion (2)

   #dom-publickeycredential-identifier-slotReferenced in:
     * 5.1. PublicKeyCredential Interface (2)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   #dom-publickeycredential-clientextensionsresults-slotReferenced in:
     * 5.1. PublicKeyCredential Interface
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   #dom-credentialcreationoptions-publickeyReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)

   #dom-credentialrequestoptions-publickeyReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2) (3)

   #dom-publickeycredential-create-slotReferenced in:
     * 4. Terminology
     * 5.1. PublicKeyCredential Interface
     * 5.4.5. Authenticator Attachment enumeration (enum
       AuthenticatorAttachment)
     * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
     * 6.2.2. The authenticatorMakeCredential operation
     * 14.2. Registration Ceremony Privacy

   #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
   originReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method

   #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
   optionsReferenced in:
     * 7.1. Registering a new credential

   #effective-user-verification-requirement-for-credential-creationReferen
   ced in:
     * 6.2.2. The authenticatorMakeCredential operation

   #credentialcreationdata-attestationobjectresultReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
       (4) (5)

   #credentialcreationdata-clientdatajsonresultReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method

   #credentialcreationdata-attestationconveyancepreferenceoptionReferenced
   in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method

   #credentialcreationdata-clientextensionresultsReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method

   #dom-publickeycredential-collectfromcredentialstore-slotReferenced in:
     * 5.1.4. Use an existing credential to make an assertion -
       PublicKeyCredential's [[Get]](options) method

   #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
     * 4. Terminology
     * 5.1. PublicKeyCredential Interface
     * 5.1.4. Use an existing credential to make an assertion -
       PublicKeyCredential's [[Get]](options) method
     * 5.4.5. Authenticator Attachment enumeration (enum
       AuthenticatorAttachment)
     * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
     * 6.2.3. The authenticatorGetAssertion operation
     * 14.3. Authentication Ceremony Privacy
```

```
     * 5.1. PublicKeyCredential Interface
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method

   #dom-credentialcreationoptions-publickeyReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)

   #dom-credentialrequestoptions-publickeyReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2) (3)

   #dom-publickeycredential-create-slotReferenced in:
     * 4. Terminology
     * 5.1. PublicKeyCredential Interface
     * 5.4.5. Authenticator Attachment enumeration (enum
       AuthenticatorAttachment)
     * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
     * 6.2.2. The authenticatorMakeCredential operation
     * 14.2. Registration Ceremony Privacy

   #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
   originReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method

   #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
   optionsReferenced in:
     * 7.1. Registering a new credential

   #effective-user-verification-requirement-for-credential-creationReferen
   ced in:
     * 6.2.2. The authenticatorMakeCredential operation

   #credentialcreationdata-attestationobjectresultReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
       (4) (5)

   #credentialcreationdata-clientdatajsonresultReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method

   #credentialcreationdata-attestationconveyancepreferenceoptionReferenced
   in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method

   #credentialcreationdata-clientextensionresultsReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method

   #dom-publickeycredential-collectfromcredentialstore-slotReferenced in:
     * 5.1.4. Use an existing credential to make an assertion -
       PublicKeyCredential's [[Get]](options) method

   #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
     * 4. Terminology
     * 5.1. PublicKeyCredential Interface
     * 5.1.4. Use an existing credential to make an assertion -
       PublicKeyCredential's [[Get]](options) method
     * 5.4.5. Authenticator Attachment enumeration (enum
       AuthenticatorAttachment)
     * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
     * 6.2.3. The authenticatorGetAssertion operation
     * 14.3. Authentication Ceremony Privacy

   #dom-publickeycredential-discoverfromexternalsource-origin-options-same
   originwithancestors-originReferenced in:
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method

   #effective-user-verification-requirement-for-assertionReferenced in:
    * 6.2.3. The authenticatorGetAssertion operation

   #assertioncreationdata-credentialidresultReferenced in:
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method (2) (3)

   #assertioncreationdata-clientdatajsonresultReferenced in:
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method

   #assertioncreationdata-authenticatordataresultReferenced 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 (2) (3) (4)
    * 6.2.3. The authenticatorGetAssertion operation

   #assertioncreationdata-clientextensionresultsReferenced in:
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method

   #authenticatorresponseReferenced in:
    * 5.1. PublicKeyCredential Interface (2)
    * 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)
    * 5.2.1. Information about Public Key Credential (interface
      AuthenticatorAttestationResponse) (2)
    * 5.2.2. Web Authentication Assertion (interface
      AuthenticatorAssertionResponse) (2)

   #dom-authenticatorresponse-clientdatajsonReferenced 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 (2)
    * 5.2. Authenticator Responses (interface AuthenticatorResponse)
    * 5.2.1. Information about Public Key Credential (interface
      AuthenticatorAttestationResponse)
    * 5.2.2. Web Authentication Assertion (interface
      AuthenticatorAssertionResponse)
    * 7.1. Registering a new credential (2)
    * 7.2. Verifying an authentication assertion

   #authenticatorattestationresponseReferenced in:
    * 5.1. PublicKeyCredential Interface
    * 5.1.3. Create a new credential - PublicKeyCredential's
      [[Create]](origin, options, sameOriginWithAncestors) method
    * 5.2.1. Information about Public Key Credential (interface
      AuthenticatorAttestationResponse) (2)
    * 7. Relying Party Operations
```

```

   #dom-publickeycredential-discoverfromexternalsource-origin-options-same
   originwithancestors-originReferenced in:
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method

   #effective-user-verification-requirement-for-assertionReferenced in:
    * 6.2.3. The authenticatorGetAssertion operation

   #assertioncreationdata-credentialidresultReferenced in:
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method (2) (3)

   #assertioncreationdata-clientdatajsonresultReferenced in:
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method

   #assertioncreationdata-authenticatordataresultReferenced 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 (2) (3) (4)
    * 6.2.3. The authenticatorGetAssertion operation

   #assertioncreationdata-clientextensionresultsReferenced in:
    * 5.1.4.1. PublicKeyCredential's
      [[DiscoverFromExternalSource]](origin, options,
      sameOriginWithAncestors) method

   #authenticatorresponseReferenced in:
    * 5.1. PublicKeyCredential Interface (2)
    * 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)
    * 5.2.1. Information about Public Key Credential (interface
      AuthenticatorAttestationResponse) (2)
    * 5.2.2. Web Authentication Assertion (interface
      AuthenticatorAssertionResponse) (2)

   #dom-authenticatorresponse-clientdatajsonReferenced 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 (2)
    * 5.2. Authenticator Responses (interface AuthenticatorResponse)
    * 5.2.1. Information about Public Key Credential (interface
      AuthenticatorAttestationResponse)
    * 5.2.2. Web Authentication Assertion (interface
      AuthenticatorAssertionResponse)
    * 7.1. Registering a new credential (2)
    * 7.2. Verifying an authentication assertion

   #authenticatorattestationresponseReferenced in:
    * 5.1. PublicKeyCredential Interface
    * 5.1.3. Create a new credential - PublicKeyCredential's
      [[Create]](origin, options, sameOriginWithAncestors) method
    * 5.2.1. Information about Public Key Credential (interface
      AuthenticatorAttestationResponse) (2)
    * 7. Relying Party Operations
```

```
     * 7.1. Registering a new credential (2)

   #dom-authenticatorattestationresponse-attestationobjectReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 7.1. Registering a new credential

   #authenticatorassertionresponseReferenced in:
     * 4. Terminology
     * 5.1. PublicKeyCredential Interface
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse) (2)
     * 7. Relying Party Operations

   #dom-authenticatorassertionresponse-authenticatordataReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse)
     * 7.2. Verifying an authentication assertion

   #dom-authenticatorassertionresponse-signatureReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse)
     * 7.2. Verifying an authentication assertion

   #dom-authenticatorassertionresponse-userhandleReferenced in:
     * 2.2.1. Backwards Compatibility with FIDO U2F
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse)
     * 7.2. Verifying an authentication assertion

   #dictdef-publickeycredentialparametersReferenced in:
     * 5.3. Parameters for Credential Generation (dictionary
       PublicKeyCredentialParameters)
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)

   #dom-publickeycredentialparameters-typeReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 5.3. Parameters for Credential Generation (dictionary
       PublicKeyCredentialParameters)

   #dom-publickeycredentialparameters-algReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.3. Parameters for Credential Generation (dictionary
       PublicKeyCredentialParameters)

   #dictdef-publickeycredentialcreationoptionsReferenced in:
     * 5.1.1. CredentialCreationOptions Dictionary Extension
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)

   #dom-publickeycredentialcreationoptions-rpReferenced in:
```

```
     * 7.1. Registering a new credential (2)

   #dom-authenticatorattestationresponse-attestationobjectReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 7.1. Registering a new credential

   #authenticatorassertionresponseReferenced in:
     * 4. Terminology
     * 5.1. PublicKeyCredential Interface
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse) (2)
     * 7. Relying Party Operations

   #dom-authenticatorassertionresponse-authenticatordataReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse)
     * 7.2. Verifying an authentication assertion

   #dom-authenticatorassertionresponse-signatureReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse)
     * 7.2. Verifying an authentication assertion

   #dom-authenticatorassertionresponse-userhandleReferenced in:
     * 2.2.1. Backwards Compatibility with FIDO U2F
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.2.2. Web Authentication Assertion (interface
       AuthenticatorAssertionResponse)
     * 7.2. Verifying an authentication assertion

   #dictdef-publickeycredentialparametersReferenced in:
     * 5.3. Parameters for Credential Generation (dictionary
       PublicKeyCredentialParameters)
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)

   #dom-publickeycredentialparameters-typeReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 5.3. Parameters for Credential Generation (dictionary
       PublicKeyCredentialParameters)

   #dom-publickeycredentialparameters-algReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.3. Parameters for Credential Generation (dictionary
       PublicKeyCredentialParameters)

   #dictdef-publickeycredentialcreationoptionsReferenced in:
     * 5.1.1. CredentialCreationOptions Dictionary Extension
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)

   #dom-publickeycredentialcreationoptions-rpReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
        (4) (5) (6)
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)

    #dom-publickeycredentialcreationoptions-userReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 7.1. Registering a new credential

    #dom-publickeycredentialcreationoptions-challengeReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 13.1. Cryptographic Challenges

    #dom-publickeycredentialcreationoptions-pubkeycredparamsReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2)
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)

    #dom-publickeycredentialcreationoptions-timeoutReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2)
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)

    #dom-publickeycredentialcreationoptions-excludecredentialsReferenced
    in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 14.2. Registration Ceremony Privacy (2)

    #dom-publickeycredentialcreationoptions-authenticatorselectionReference
    d in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
        (4) (5) (6)
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 6.2.2. The authenticatorMakeCredential operation

    #dom-publickeycredentialcreationoptions-attestationReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)

    #dom-publickeycredentialcreationoptions-extensionsReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2)
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 7.1. Registering a new credential (2)
      * 7.2. Verifying an authentication assertion
      * 9.3. Extending request parameters

    #dictdef-publickeycredentialentityReferenced in:
      * 5.4.1. Public Key Entity Description (dictionary
        PublicKeyCredentialEntity) (2) (3)
      * 5.4.2. RP Parameters for Credential Generation (dictionary
        PublicKeyCredentialRpEntity)
      * 5.4.3. User Account Parameters for Credential Generation
```

```
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
        (4) (5) (6)
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)

    #dom-publickeycredentialcreationoptions-userReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 7.1. Registering a new credential

    #dom-publickeycredentialcreationoptions-challengeReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 13.1. Cryptographic Challenges

    #dom-publickeycredentialcreationoptions-pubkeycredparamsReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2)
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)

    #dom-publickeycredentialcreationoptions-timeoutReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2)
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)

    #dom-publickeycredentialcreationoptions-excludecredentialsReferenced
    in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 14.2. Registration Ceremony Privacy (2)

    #dom-publickeycredentialcreationoptions-authenticatorselectionReference
    d in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
        (4) (5) (6)
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 6.2.2. The authenticatorMakeCredential operation

    #dom-publickeycredentialcreationoptions-attestationReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)

    #dom-publickeycredentialcreationoptions-extensionsReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2)
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 7.1. Registering a new credential (2)
      * 7.2. Verifying an authentication assertion
      * 9.3. Extending request parameters

    #dictdef-publickeycredentialentityReferenced in:
      * 5.4.1. Public Key Entity Description (dictionary
        PublicKeyCredentialEntity) (2) (3)
      * 5.4.2. RP Parameters for Credential Generation (dictionary
        PublicKeyCredentialRpEntity)
      * 5.4.3. User Account Parameters for Credential Generation
```

```
     (dictionary PublicKeyCredentialUserEntity)

   #dom-publickeycredentialentity-nameReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity) (2) (3) (4)
     * 6.2.2. The authenticatorMakeCredential operation (2)

   #dom-publickeycredentialentity-iconReferenced in:
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)

   #dictdef-publickeycredentialrpentityReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)
     * 5.4.2. RP Parameters for Credential Generation (dictionary
       PublicKeyCredentialRpEntity) (2)
     * 6.2.2. The authenticatorMakeCredential operation

   #dom-publickeycredentialrpentity-idReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
       (4) (5)
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)
     * 5.4.2. RP Parameters for Credential Generation (dictionary
       PublicKeyCredentialRpEntity)
     * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)

   #dictdef-publickeycredentialuserentityReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity) (2)
     * 5.4.3. User Account Parameters for Credential Generation
       (dictionary PublicKeyCredentialUserEntity) (2)
     * 6.2.2. The authenticatorMakeCredential operation

   #dom-publickeycredentialuserentity-idReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)
     * 5.4.3. User Account Parameters for Credential Generation
       (dictionary PublicKeyCredentialUserEntity)
     * 6.2.2. The authenticatorMakeCredential operation

   #dom-publickeycredentialuserentity-displaynameReferenced in:
     * 4. Terminology
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)
     * 5.4.3. User Account Parameters for Credential Generation
       (dictionary PublicKeyCredentialUserEntity) (2) (3)
     * 6.2.2. The authenticatorMakeCredential operation

   #dictdef-authenticatorselectioncriteriaReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.4.4. Authenticator Selection Criteria (dictionary
       AuthenticatorSelectionCriteria) (2)

   #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced
   in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.4.4. Authenticator Selection Criteria (dictionary
       AuthenticatorSelectionCriteria)
```

```
     (dictionary PublicKeyCredentialUserEntity)

   #dom-publickeycredentialentity-nameReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity) (2) (3) (4)
     * 6.2.2. The authenticatorMakeCredential operation (2)

   #dom-publickeycredentialentity-iconReferenced in:
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)

   #dictdef-publickeycredentialrpentityReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)
     * 5.4.2. RP Parameters for Credential Generation (dictionary
       PublicKeyCredentialRpEntity) (2)
     * 6.2.2. The authenticatorMakeCredential operation

   #dom-publickeycredentialrpentity-idReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
       (4) (5)
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)
     * 5.4.2. RP Parameters for Credential Generation (dictionary
       PublicKeyCredentialRpEntity)
     * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)

   #dictdef-publickeycredentialuserentityReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity) (2)
     * 5.4.3. User Account Parameters for Credential Generation
       (dictionary PublicKeyCredentialUserEntity) (2)
     * 6.2.2. The authenticatorMakeCredential operation

   #dom-publickeycredentialuserentity-idReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)
     * 5.4.3. User Account Parameters for Credential Generation
       (dictionary PublicKeyCredentialUserEntity)
     * 6.2.2. The authenticatorMakeCredential operation

   #dom-publickeycredentialuserentity-displaynameReferenced in:
     * 4. Terminology
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions)
     * 5.4.1. Public Key Entity Description (dictionary
       PublicKeyCredentialEntity)
     * 5.4.3. User Account Parameters for Credential Generation
       (dictionary PublicKeyCredentialUserEntity) (2) (3)
     * 6.2.2. The authenticatorMakeCredential operation

   #dictdef-authenticatorselectioncriteriaReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.4.4. Authenticator Selection Criteria (dictionary
       AuthenticatorSelectionCriteria) (2)

   #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced
   in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.4.4. Authenticator Selection Criteria (dictionary
       AuthenticatorSelectionCriteria)
```

```

    #dom-authenticatorselectioncriteria-requireresidentkeyReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2)
      * 5.4.4. Authenticator Selection Criteria (dictionary
        AuthenticatorSelectionCriteria)
      * 6.2.2. The authenticatorMakeCredential operation

    #dom-authenticatorselectioncriteria-userverificationReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2)
      * 5.4.4. Authenticator Selection Criteria (dictionary
        AuthenticatorSelectionCriteria)

    #enumdef-authenticatorattachmentReferenced in:
      * 5.4.4. Authenticator Selection Criteria (dictionary
        AuthenticatorSelectionCriteria) (2)
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)

    #attachment-modalityReferenced in:
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)

    #platform-authenticatorsReferenced in:
      * 5.1.7. Availability of User-Verifying Platform Authenticator -
        PublicKeyCredential's
        isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
        (5)
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)
      * 12.1. Registration
      * 12.2. Registration Specifically with User Verifying Platform
        Authenticator (2)
      * 14.2. Registration Ceremony Privacy

    #roaming-authenticatorsReferenced in:
      * 1.1.3. Other use cases and configurations
      * 4. Terminology
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)
      * 12.1. Registration

    #platform-attachmentReferenced in:
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment)

    #platform-credentialReferenced in:
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)

    #cross-platform-attachedReferenced in:
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)

    #roaming-credentialReferenced in:
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)

    #attestation-conveyanceReferenced in:
      * 4. Terminology
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 5.4.6. Attestation Conveyance Preference enumeration (enum
        AttestationConveyancePreference)

    #enumdef-attestationconveyancepreferenceReferenced in:
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions) (2)
      * 5.4.6. Attestation Conveyance Preference enumeration (enum
```

```

    #dom-authenticatorselectioncriteria-requireresidentkeyReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2)
      * 5.4.4. Authenticator Selection Criteria (dictionary
        AuthenticatorSelectionCriteria)
      * 6.2.2. The authenticatorMakeCredential operation

    #dom-authenticatorselectioncriteria-userverificationReferenced in:
      * 5.1.3. Create a new credential - PublicKeyCredential's
        [[Create]](origin, options, sameOriginWithAncestors) method (2)
      * 5.4.4. Authenticator Selection Criteria (dictionary
        AuthenticatorSelectionCriteria)

    #enumdef-authenticatorattachmentReferenced in:
      * 5.4.4. Authenticator Selection Criteria (dictionary
        AuthenticatorSelectionCriteria) (2)
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)

    #attachment-modalityReferenced in:
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)

    #platform-authenticatorsReferenced in:
      * 5.1.7. Availability of User-Verifying Platform Authenticator -
        PublicKeyCredential's
        isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
        (5)
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)
      * 12.1. Registration
      * 12.2. Registration Specifically with User Verifying Platform
        Authenticator (2)
      * 14.2. Registration Ceremony Privacy

    #roaming-authenticatorsReferenced in:
      * 1.1.3. Other use cases and configurations
      * 4. Terminology
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)
      * 12.1. Registration

    #platform-attachmentReferenced in:
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment)

    #platform-credentialReferenced in:
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)

    #cross-platform-attachedReferenced in:
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)

    #roaming-credentialReferenced in:
      * 5.4.5. Authenticator Attachment enumeration (enum
        AuthenticatorAttachment) (2)

    #attestation-conveyanceReferenced in:
      * 4. Terminology
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions)
      * 5.4.6. Attestation Conveyance Preference enumeration (enum
        AttestationConveyancePreference)

    #enumdef-attestationconveyancepreferenceReferenced in:
      * 5.4. Options for Credential Creation (dictionary
        PublicKeyCredentialCreationOptions) (2)
      * 5.4.6. Attestation Conveyance Preference enumeration (enum
         AttestationConveyancePreference) (2)

       #dom-attestationconveyancepreference-noneReferenced in:
         * 5.4. Options for Credential Creation (dictionary
           PublicKeyCredentialCreationOptions)
         * 5.4.6. Attestation Conveyance Preference enumeration (enum
           AttestationConveyancePreference)

       #dom-attestationconveyancepreference-indirectReferenced in:
         * 5.4.6. Attestation Conveyance Preference enumeration (enum
           AttestationConveyancePreference)

       #dom-attestationconveyancepreference-directReferenced in:
         * 5.4.6. Attestation Conveyance Preference enumeration (enum
           AttestationConveyancePreference)

       #dictdef-publickeycredentialrequestoptionsReferenced in:
         * 5.1.2. CredentialRequestOptions Dictionary Extension
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions) (2)
         * 7.2. Verifying an authentication assertion

       #dom-publickeycredentialrequestoptions-challengeReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions) (2)
         * 13.1. Cryptographic Challenges

       #dom-publickeycredentialrequestoptions-timeoutReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method (2)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)

       #dom-publickeycredentialrequestoptions-rpidReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method (2) (3) (4)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)

       #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method (2) (3) (4)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)
         * 7.2. Verifying an authentication assertion (2)
         * 14.3. Authentication Ceremony Privacy (2)

       #dom-publickeycredentialrequestoptions-userverificationReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method (2)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)

       #dom-publickeycredentialrequestoptions-extensionsReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method (2)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)
         * 7.2. Verifying an authentication assertion
```

```
         AttestationConveyancePreference) (2)

       #dom-attestationconveyancepreference-noneReferenced in:
         * 5.4. Options for Credential Creation (dictionary
           PublicKeyCredentialCreationOptions)
         * 5.4.6. Attestation Conveyance Preference enumeration (enum
           AttestationConveyancePreference)

       #dom-attestationconveyancepreference-indirectReferenced in:
         * 5.4.6. Attestation Conveyance Preference enumeration (enum
           AttestationConveyancePreference)

       #dom-attestationconveyancepreference-directReferenced in:
         * 5.4.6. Attestation Conveyance Preference enumeration (enum
           AttestationConveyancePreference)

       #dictdef-publickeycredentialrequestoptionsReferenced in:
         * 5.1.2. CredentialRequestOptions Dictionary Extension
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions) (2)
         * 7.2. Verifying an authentication assertion

       #dom-publickeycredentialrequestoptions-challengeReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions) (2)
         * 13.1. Cryptographic Challenges

       #dom-publickeycredentialrequestoptions-timeoutReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method (2)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)

       #dom-publickeycredentialrequestoptions-rpidReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method (2) (3) (4)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)

       #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method (2) (3) (4) (5) (6)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)
         * 7.2. Verifying an authentication assertion (2)
         * 14.3. Authentication Ceremony Privacy (2)

       #dom-publickeycredentialrequestoptions-userverificationReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method (2)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)

       #dom-publickeycredentialrequestoptions-extensionsReferenced in:
         * 5.1.4.1. PublicKeyCredential's
           [[DiscoverFromExternalSource]](origin, options,
           sameOriginWithAncestors) method (2)
         * 5.5. Options for Assertion Generation (dictionary
           PublicKeyCredentialRequestOptions)
         * 7.2. Verifying an authentication assertion
```

```

   #dictdef-authenticationextensionsclientinputsReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions) (2)
     * 10.1. FIDO AppID Extension (appid)
     * 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)

   #dictdef-authenticationextensionsclientoutputsReferenced in:
     * 5.1. PublicKeyCredential Interface
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion
     * 10.1. FIDO AppID Extension (appid)
     * 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)

   #dictdef-collectedclientdataReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData) (2) (3)

   #client-dataReferenced in:
     * 5.2. Authenticator Responses (interface AuthenticatorResponse)
     * 6. WebAuthn Authenticator Model (2) (3) (4)
     * 6.1. Authenticator data (2)
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion
     * 9. WebAuthn Extensions

   #dictdef-tokenbindingReferenced in:
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)

   #dom-tokenbinding-statusReferenced in:
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion

   #dom-tokenbinding-idReferenced in:
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion

   #enumdef-tokenbindingstatusReferenced in:
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)

   #dom-collectedclientdata-typeReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
```

```

   #dictdef-authenticationextensionsclientinputsReferenced in:
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions) (2)
     * 10.1. FIDO AppID Extension (appid)
     * 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)

   #dictdef-authenticationextensionsclientoutputsReferenced in:
     * 5.1. PublicKeyCredential Interface
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion
     * 10.1. FIDO AppID Extension (appid)
     * 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)

   #dictdef-collectedclientdataReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData) (2) (3)

   #client-dataReferenced in:
     * 5.2. Authenticator Responses (interface AuthenticatorResponse)
     * 6. WebAuthn Authenticator Model (2) (3) (4)
     * 6.1. Authenticator data (2)
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion
     * 9. WebAuthn Extensions

   #dictdef-tokenbindingReferenced in:
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)

   #dom-tokenbinding-statusReferenced in:
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion

   #dom-tokenbinding-idReferenced in:
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion

   #enumdef-tokenbindingstatusReferenced in:
     * 5.10.1. Client data used in WebAuthn signatures (dictionary
       CollectedClientData)

   #dom-collectedclientdata-typeReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
```

```
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.1. Client data used in WebAuthn signatures (dictionary
         CollectedClientData)
       * 7.1. Registering a new credential
       * 7.2. Verifying an authentication assertion

     #dom-collectedclientdata-challengeReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.1. 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]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.1. Client data used in WebAuthn signatures (dictionary
         CollectedClientData)
       * 7.1. Registering a new credential
       * 7.2. Verifying an authentication assertion

     #dom-collectedclientdata-tokenbindingReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.1. Client data used in WebAuthn signatures (dictionary
         CollectedClientData)
       * 7.1. Registering a new credential (2)
       * 7.2. Verifying an authentication assertion (2)

     #collectedclientdata-json-serialized-client-dataReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.2. Authenticator Responses (interface AuthenticatorResponse)
       * 5.2.1. Information about Public Key Credential (interface
         AuthenticatorAttestationResponse) (2)
       * 5.2.2. Web Authentication Assertion (interface
         AuthenticatorAssertionResponse)
       * 5.10.1. Client data used in WebAuthn signatures (dictionary
         CollectedClientData)

     #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.2.1. Information about Public Key Credential (interface
         AuthenticatorAttestationResponse)
       * 5.2.2. Web Authentication Assertion (interface
         AuthenticatorAssertionResponse)
       * 6. WebAuthn Authenticator Model
       * 6.1.2. FIDO U2F signature format compatibility (2)
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.2.3. The authenticatorGetAssertion operation (2)
       * 6.3.2. Attestation Statement Formats (2)
       * 6.3.4. Generating an Attestation Object
```

```
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.1. Client data used in WebAuthn signatures (dictionary
         CollectedClientData)
       * 7.1. Registering a new credential
       * 7.2. Verifying an authentication assertion

     #dom-collectedclientdata-challengeReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.1. 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]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.1. Client data used in WebAuthn signatures (dictionary
         CollectedClientData)
       * 7.1. Registering a new credential
       * 7.2. Verifying an authentication assertion

     #dom-collectedclientdata-tokenbindingReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.1. Client data used in WebAuthn signatures (dictionary
         CollectedClientData)
       * 7.1. Registering a new credential (2)
       * 7.2. Verifying an authentication assertion (2)

     #collectedclientdata-json-serialized-client-dataReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.2. Authenticator Responses (interface AuthenticatorResponse)
       * 5.2.1. Information about Public Key Credential (interface
         AuthenticatorAttestationResponse) (2)
       * 5.2.2. Web Authentication Assertion (interface
         AuthenticatorAssertionResponse)
       * 5.10.1. Client data used in WebAuthn signatures (dictionary
         CollectedClientData)

     #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.2.1. Information about Public Key Credential (interface
         AuthenticatorAttestationResponse)
       * 5.2.2. Web Authentication Assertion (interface
         AuthenticatorAssertionResponse)
       * 6. WebAuthn Authenticator Model
       * 6.1.2. FIDO U2F signature format compatibility (2)
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.2.3. The authenticatorGetAssertion operation (2)
       * 6.3.2. Attestation Statement Formats (2)
       * 6.3.4. Generating an Attestation Object
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format
     * 8.4. Android Key Attestation Statement Format
     * 8.5. Android SafetyNet Attestation Statement Format
     * 8.6. FIDO U2F Attestation Statement Format

   #enumdef-publickeycredentialtypeReferenced in:
     * 4. Terminology
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 5.3. Parameters for Credential Generation (dictionary
       PublicKeyCredentialParameters)
     * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 6.2.2. The authenticatorMakeCredential operation (2) (3)

   #dom-publickeycredentialtype-public-keyReferenced in:
     * 4. Terminology
     * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
     * 6.2.2. The authenticatorMakeCredential operation

   #dictdef-publickeycredentialdescriptorReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions) (2) (3)
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation

   #dom-publickeycredentialdescriptor-typeReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 6.2.2. The authenticatorMakeCredential operation

   #dom-publickeycredentialdescriptor-idReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method (2)
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation

   #dom-publickeycredentialdescriptor-transportsReferenced 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 (2)
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)

   #enumdef-authenticatortransportReferenced in:
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 5.10.4. Authenticator Transport enumeration (enum
       AuthenticatorTransport)

   #dom-authenticatortransport-usbReferenced in:
     * 5.10.4. Authenticator Transport enumeration (enum
```

```
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format
     * 8.4. Android Key Attestation Statement Format
     * 8.5. Android SafetyNet Attestation Statement Format
     * 8.6. FIDO U2F Attestation Statement Format

   #enumdef-publickeycredentialtypeReferenced in:
     * 4. Terminology
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 5.3. Parameters for Credential Generation (dictionary
       PublicKeyCredentialParameters)
     * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 6.2.2. The authenticatorMakeCredential operation (2) (3)

   #dom-publickeycredentialtype-public-keyReferenced in:
     * 4. Terminology
     * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
     * 6.2.2. The authenticatorMakeCredential operation

   #dictdef-publickeycredentialdescriptorReferenced in:



     * 5.4. Options for Credential Creation (dictionary
       PublicKeyCredentialCreationOptions) (2)
     * 5.5. Options for Assertion Generation (dictionary
       PublicKeyCredentialRequestOptions) (2) (3)
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation

   #dom-publickeycredentialdescriptor-typeReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 6.2.2. The authenticatorMakeCredential operation

   #dom-publickeycredentialdescriptor-idReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation

   #dom-publickeycredentialdescriptor-transportsReferenced 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 (2) (3)
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)

   #enumdef-authenticatortransportReferenced in:
     * 5.10.3. Credential Descriptor (dictionary
       PublicKeyCredentialDescriptor)
     * 5.10.4. Authenticator Transport enumeration (enum
       AuthenticatorTransport)

   #dom-authenticatortransport-usbReferenced in:
     * 5.10.4. Authenticator Transport enumeration (enum
```

/Users/jehodges/Documents/work/standards/W3C/webauthn/index-master-3c5e383.txt, Top line: 8878

```
       AuthenticatorTransport)

     #dom-authenticatortransport-nfcReferenced in:
       * 5.10.4. Authenticator Transport enumeration (enum
         AuthenticatorTransport)

     #dom-authenticatortransport-bleReferenced in:
       * 5.10.4. Authenticator Transport enumeration (enum
         AuthenticatorTransport)

     #dom-authenticatortransport-internalReferenced in:
       * 5.10.4. Authenticator Transport enumeration (enum
         AuthenticatorTransport)

     #typedefdef-cosealgorithmidentifierReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.3. Parameters for Credential Generation (dictionary
         PublicKeyCredentialParameters)
       * 5.10.5. Cryptographic Algorithm Identifier (typedef
         COSEAlgorithmIdentifier)
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.3.1. Attested credential data
       * 8.2. Packed Attestation Statement Format
       * 8.3. TPM Attestation Statement Format

     #enumdef-userverificationrequirementReferenced in:
       * 5.4.4. Authenticator Selection Criteria (dictionary
         AuthenticatorSelectionCriteria) (2)
       * 5.5. Options for Assertion Generation (dictionary
         PublicKeyCredentialRequestOptions) (2)
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement)

     #dom-userverificationrequirement-requiredReferenced 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 (2)
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement)

     #dom-userverificationrequirement-preferredReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement)

     #dom-userverificationrequirement-discouragedReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement)

     #authenticator-modelReferenced in:
       * 6. WebAuthn Authenticator Model

     #authenticator-credentials-mapReferenced in:
       * 6.2.1. Lookup Credential Source by Credential ID algorithm
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.2.3. The authenticatorGetAssertion operation

     #attestation-signatureReferenced in:
```

/Users/jehodges/Documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.txt, Top line: 8901

```
       AuthenticatorTransport)

     #dom-authenticatortransport-nfcReferenced in:
       * 5.10.4. Authenticator Transport enumeration (enum
         AuthenticatorTransport)

     #dom-authenticatortransport-bleReferenced in:
       * 5.10.4. Authenticator Transport enumeration (enum
         AuthenticatorTransport)

     #typedefdef-cosealgorithmidentifierReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.3. Parameters for Credential Generation (dictionary
         PublicKeyCredentialParameters)
       * 5.10.5. Cryptographic Algorithm Identifier (typedef
         COSEAlgorithmIdentifier)
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.3.1. Attested credential data
       * 8.2. Packed Attestation Statement Format
       * 8.3. TPM Attestation Statement Format

     #enumdef-userverificationrequirementReferenced in:
       * 5.4.4. Authenticator Selection Criteria (dictionary
         AuthenticatorSelectionCriteria) (2)
       * 5.5. Options for Assertion Generation (dictionary
         PublicKeyCredentialRequestOptions) (2)
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement)

     #dom-userverificationrequirement-requiredReferenced 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 (2)
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement)

     #dom-userverificationrequirement-preferredReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement)

     #dom-userverificationrequirement-discouragedReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement)

     #authenticator-modelReferenced in:
       * 6. WebAuthn Authenticator Model

     #authenticator-credentials-mapReferenced in:
       * 6.2.1. Lookup Credential Source by Credential ID algorithm
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.2.3. The authenticatorGetAssertion operation

     #attestation-signatureReferenced in:
       * 4. Terminology
       * 6. WebAuthn Authenticator Model (2) (3)
       * 6.3. Attestation
       * 7.1. Registering a new credential
       * 8.6. FIDO U2F Attestation Statement Format

     #assertion-signatureReferenced in:
       * 6. WebAuthn Authenticator Model (2)
       * 6.1.2. FIDO U2F signature format compatibility (2)
       * 6.2.3. The authenticatorGetAssertion operation (2) (3)

     #authenticator-dataReferenced in:
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.2.1. Information about Public Key Credential (interface
         AuthenticatorAttestationResponse) (2)
       * 5.2.2. Web Authentication Assertion (interface
         AuthenticatorAssertionResponse)
       * 6. WebAuthn Authenticator Model (2)
       * 6.1. Authenticator data (2) (3) (4) (5) (6) (7) (8) (9)
       * 6.1.1. Signature Counter Considerations (2)
       * 6.1.2. FIDO U2F signature format compatibility (2) (3)
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.2.3. The authenticatorGetAssertion operation
       * 6.3. Attestation (2)
       * 6.3.1. Attested credential data
       * 6.3.2. Attestation Statement Formats (2)
       * 6.3.4. Generating an Attestation Object
       * 7.1. Registering a new credential
       * 8.5. Android SafetyNet Attestation Statement Format
       * 9.5. Authenticator extension processing (2)
       * 10.6. User Verification Index Extension (uvi)
       * 10.8. User Verification Method Extension (uvm)
       * 13.2.1. Attestation Certificate Hierarchy
       * 13.4. credentialId Unsigned

     #rpidhashReferenced in:
       * 6.1.2. FIDO U2F signature format compatibility
       * 7.2. Verifying an authentication assertion

     #flagsReferenced in:
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement) (2)
       * 6.1. Authenticator data
       * 6.1.2. FIDO U2F signature format compatibility
       * 7.1. Registering a new credential (2)
       * 7.2. Verifying an authentication assertion (2)

     #signcountReferenced in:
       * 6.1.1. Signature Counter Considerations (2)
       * 7.2. Verifying an authentication assertion (2) (3)

     #attestedcredentialdataReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 6.1. Authenticator data (2)
       * 6.1.2. FIDO U2F signature format compatibility
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.2.3. 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

     #authdataextensionsReferenced in:
       * 6.1. Authenticator data
       * 6.1.2. FIDO U2F signature format compatibility
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.2.3. The authenticatorGetAssertion operation
```

       * 4. Terminology
       * 6. WebAuthn Authenticator Model (2) (3)
       * 6.3. Attestation
       * 7.1. Registering a new credential
       * 8.6. FIDO U2F Attestation Statement Format

     #assertion-signatureReferenced in:
       * 6. WebAuthn Authenticator Model (2)
       * 6.1.2. FIDO U2F signature format compatibility (2)
       * 6.2.3. The authenticatorGetAssertion operation (2) (3)

     #authenticator-dataReferenced in:
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 5.2.1. Information about Public Key Credential (interface
         AuthenticatorAttestationResponse) (2)
       * 5.2.2. Web Authentication Assertion (interface
         AuthenticatorAssertionResponse)
       * 6. WebAuthn Authenticator Model (2)
       * 6.1. Authenticator data (2) (3) (4) (5) (6) (7) (8) (9)
       * 6.1.1. Signature Counter Considerations (2)
       * 6.1.2. FIDO U2F signature format compatibility (2) (3)
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.2.3. The authenticatorGetAssertion operation
       * 6.3. Attestation (2)
       * 6.3.1. Attested credential data
       * 6.3.2. Attestation Statement Formats (2)
       * 6.3.4. Generating an Attestation Object
       * 7.1. Registering a new credential
       * 8.5. Android SafetyNet Attestation Statement Format
       * 9.5. Authenticator extension processing (2)
       * 10.6. User Verification Index Extension (uvi)
       * 10.8. User Verification Method Extension (uvm)
       * 13.2.1. Attestation Certificate Hierarchy
       * 13.4. credentialId Unsigned

     #rpidhashReferenced in:
       * 6.1.2. FIDO U2F signature format compatibility
       * 7.2. Verifying an authentication assertion

     #flagsReferenced in:
       * 5.10.6. User Verification Requirement enumeration (enum
         UserVerificationRequirement) (2)
       * 6.1. Authenticator data
       * 6.1.2. FIDO U2F signature format compatibility
       * 7.1. Registering a new credential (2)
       * 7.2. Verifying an authentication assertion (2)

     #signcountReferenced in:
       * 6.1.1. Signature Counter Considerations (2)
       * 7.2. Verifying an authentication assertion (2) (3)

     #attestedcredentialdataReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 6.1. Authenticator data (2)
       * 6.1.2. FIDO U2F signature format compatibility
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.2.3. 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

     #authdataextensionsReferenced in:
       * 6.1. Authenticator data
       * 6.1.2. FIDO U2F signature format compatibility
       * 6.2.2. The authenticatorMakeCredential operation
       * 6.2.3. The authenticatorGetAssertion operation

/Users/jehodges/Documents/work/standards/W3C/webauthn/index-master-3c5e383.txt, Top line: 9018

```
9018      * 7.1. Registering a new credential (2)
9019      * 7.2. Verifying an authentication assertion (2)
9020      * 9.5. Authenticator extension processing (2)
9021
9022    #signature-counterReferenced in:
9023      * 6.1. Authenticator data
9024      * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6) (7) (8)
9025        (9) (10)
9026      * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)
9027      * 6.2.3. The authenticatorGetAssertion operation (2)
9028      * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6)
9029
9030    #authenticator-operationsReferenced in:
9031      * 4. Terminology
9032
9033    #authenticator-sessionReferenced in:
9034      * 5.6. Abort operations with AbortSignal (2)
9035      * 6.2.2. The authenticatorMakeCredential operation
9036      * 6.2.3. The authenticatorGetAssertion operation
9037      * 6.2.4. The authenticatorCancel operation (2)
9038
9039    #credential-id-looking-upReferenced in:
9040      * 6.2.2. The authenticatorMakeCredential operation
9041      * 6.2.3. The authenticatorGetAssertion operation
9042
9043    #authenticatormakecredentialReferenced in:
9044      * 4. Terminology (2) (3) (4)
9045      * 5.1.3. Create a new credential - PublicKeyCredential's
9046        [[Create]](origin, options, sameOriginWithAncestors) method (2)
9047      * 6. WebAuthn Authenticator Model
9048      * 6.1.2. FIDO U2F signature format compatibility
9049      * 6.2.4. The authenticatorCancel operation (2)
9050      * 9. WebAuthn Extensions
9051      * 9.2. Defining extensions
9052      * 9.5. Authenticator extension processing
9053
9054    #authenticatorgetassertionReferenced in:
9055      * 4. Terminology (2) (3)
9056      * 5.1.4.1. PublicKeyCredential's
9057        [[DiscoverFromExternalSource]](origin, options,
9058        sameOriginWithAncestors) method (2) (3) (4)
9059      * 6. WebAuthn Authenticator Model
9060      * 6.1. Authenticator data
9061      * 6.1.1. Signature Counter Considerations (2) (3)
9062      * 6.2.4. The authenticatorCancel operation (2)
9063      * 9. WebAuthn Extensions
9064      * 9.2. Defining extensions
9065      * 9.5. Authenticator extension processing
9066      * 10.1. FIDO AppID Extension (appid)
9067
9068    #authenticatorcancelReferenced in:
9069      * 5.1.3. Create a new credential - PublicKeyCredential's
9070        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
9071        (4) (5)
9072      * 5.1.4.1. PublicKeyCredential's
9073        [[DiscoverFromExternalSource]](origin, options,
9074 🟩     sameOriginWithAncestors) method (2) (3) (4)
9075      * 6.2.2. The authenticatorMakeCredential operation
9076      * 6.2.3. The authenticatorGetAssertion operation
9077
9078    #attestation-objectReferenced in:
9079      * 4. Terminology (2) (3)
9080      * 5. Web Authentication API
9081      * 5.2.1. Information about Public Key Credential (interface
9082        AuthenticatorAttestationResponse) (2)
9083      * 5.4. Options for Credential Creation (dictionary
9084        PublicKeyCredentialCreationOptions) (2)
9085      * 6.2.2. The authenticatorMakeCredential operation (2)
9086      * 6.3. Attestation (2) (3)
9087      * 6.3.1. Attested credential data
```

/Users/jehodges/Documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.txt, Top line: 9037

```
9037      * 7.1. Registering a new credential (2)
9038      * 7.2. Verifying an authentication assertion (2)
9039      * 9.5. Authenticator extension processing (2)
9040
9041    #signature-counterReferenced in:
9042      * 6.1. Authenticator data
9043      * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6) (7) (8)
9044        (9) (10)
9045      * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)
9046      * 6.2.3. The authenticatorGetAssertion operation (2)
9047      * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6)
9048
9049    #authenticator-operationsReferenced in:
9050      * 4. Terminology
9051
9052    #authenticator-sessionReferenced in:
9053      * 5.6. Abort operations with AbortSignal (2)
9054      * 6.2.2. The authenticatorMakeCredential operation
9055      * 6.2.3. The authenticatorGetAssertion operation
9056      * 6.2.4. The authenticatorCancel operation (2)
9057
9058    #credential-id-looking-upReferenced in:
9059      * 6.2.2. The authenticatorMakeCredential operation
9060      * 6.2.3. The authenticatorGetAssertion operation
9061
9062    #authenticatormakecredentialReferenced in:
9063      * 4. Terminology (2) (3) (4)
9064      * 5.1.3. Create a new credential - PublicKeyCredential's
9065        [[Create]](origin, options, sameOriginWithAncestors) method (2)
9066      * 6. WebAuthn Authenticator Model
9067      * 6.1.2. FIDO U2F signature format compatibility
9068      * 6.2.4. The authenticatorCancel operation (2)
9069      * 9. WebAuthn Extensions
9070      * 9.2. Defining extensions
9071      * 9.5. Authenticator extension processing
9072
9073    #authenticatorgetassertionReferenced in:
9074      * 4. Terminology (2) (3)
9075      * 5.1.4.1. PublicKeyCredential's
9076        [[DiscoverFromExternalSource]](origin, options,
9077        sameOriginWithAncestors) method (2) (3) (4)
9078      * 6. WebAuthn Authenticator Model
9079      * 6.1. Authenticator data
9080      * 6.1.1. Signature Counter Considerations (2) (3)
9081      * 6.2.4. The authenticatorCancel operation (2)
9082      * 9. WebAuthn Extensions
9083      * 9.2. Defining extensions
9084      * 9.5. Authenticator extension processing
9085      * 10.1. FIDO AppID Extension (appid)
9086
9087    #authenticatorcancelReferenced in:
9088      * 5.1.3. Create a new credential - PublicKeyCredential's
9089        [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
9090        (4) (5)
9091      * 5.1.4.1. PublicKeyCredential's
9092        [[DiscoverFromExternalSource]](origin, options,
9093 🟦     sameOriginWithAncestors) method (2) (3) (4) (5)
9094      * 6.2.2. The authenticatorMakeCredential operation
9095      * 6.2.3. The authenticatorGetAssertion operation
9096
9097    #attestation-objectReferenced in:
9098      * 4. Terminology (2) (3)
9099      * 5. Web Authentication API
9100      * 5.2.1. Information about Public Key Credential (interface
9101        AuthenticatorAttestationResponse) (2)
9102      * 5.4. Options for Credential Creation (dictionary
9103        PublicKeyCredentialCreationOptions) (2)
9104      * 6.2.2. The authenticatorMakeCredential operation (2)
9105      * 6.3. Attestation (2) (3)
9106      * 6.3.1. Attested credential data
```

/Users/jehodges/Documents/work/standards/W3C/webauthn/index-master-3c5e383.txt, Top line: 9088

```
     * 6.3.4. Generating an Attestation Object (2)
     * 7.1. Registering a new credential

   #attestation-statementReferenced in:
     * 4. Terminology (2)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse) (2) (3)
     * 5.4.6. Attestation Conveyance Preference enumeration (enum
       AttestationConveyancePreference) (2) (3) (4) (5) (6)
     * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8)
     * 6.3.2. Attestation Statement Formats (2) (3) (4)
     * 7.1. Registering a new credential
     * 8.7. None Attestation Statement Format
     * 13.3. Security Benefits for Relying Parties
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation (2) (3)

   #attestation-statement-formatReferenced in:
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 5.10.4. Authenticator Transport enumeration (enum
       AuthenticatorTransport)
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.3. Attestation (2) (3) (4) (5) (6) (7)
     * 6.3.2. Attestation Statement Formats (2) (3) (4)
     * 6.3.4. Generating an Attestation Object
     * 7.1. Registering a new credential (2)

   #attestation-typeReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 6.3. Attestation (2) (3) (4) (5) (6)
     * 6.3.2. Attestation Statement Formats (2)

   #attested-credential-dataReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 6.1. Authenticator data (2) (3) (4) (5)
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.3. Attestation (2)
     * 6.3.1. Attested credential data
     * 6.3.3. Attestation Types

   #aaguidReferenced in:
     * 4. Terminology
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
       (4)
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format

   #credentialidlengthReferenced in:
     * 6.1. Authenticator data

   #credentialidReferenced 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 (2)

   #credentialpublickeyReferenced in:
     * 6.1. Authenticator data
     * 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE_Key
       format
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format
```

/Users/jehodges/Documents/work/standards/W3C/agl/webauthn/index-agl-issue905-0244f7c.txt, Top line: 9107

```
     * 6.3.4. Generating an Attestation Object (2)
     * 7.1. Registering a new credential

   #attestation-statementReferenced in:
     * 4. Terminology (2)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse) (2) (3)
     * 5.4.6. Attestation Conveyance Preference enumeration (enum
       AttestationConveyancePreference) (2) (3) (4) (5) (6)
     * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8)
     * 6.3.2. Attestation Statement Formats (2) (3) (4)
     * 7.1. Registering a new credential
     * 8.7. None Attestation Statement Format
     * 13.3. Security Benefits for Relying Parties
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation (2) (3)

   #attestation-statement-formatReferenced in:
     * 5.2.1. Information about Public Key Credential (interface
       AuthenticatorAttestationResponse)
     * 5.10.4. Authenticator Transport enumeration (enum
       AuthenticatorTransport)
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.3. Attestation (2) (3) (4) (5) (6) (7)
     * 6.3.2. Attestation Statement Formats (2) (3) (4)
     * 6.3.4. Generating an Attestation Object
     * 7.1. Registering a new credential (2)

   #attestation-typeReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 6.3. Attestation (2) (3) (4) (5) (6)
     * 6.3.2. Attestation Statement Formats (2)

   #attested-credential-dataReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2)
     * 6.1. Authenticator data (2) (3) (4) (5)
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.3. Attestation (2)
     * 6.3.1. Attested credential data
     * 6.3.3. Attestation Types

   #aaguidReferenced in:
     * 4. Terminology
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
       (4)
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format

   #credentialidlengthReferenced in:
     * 6.1. Authenticator data

   #credentialidReferenced 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 (2)

   #credentialpublickeyReferenced in:
     * 6.1. Authenticator data
     * 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE_Key
       format
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format
     * 8.4. Android Key Attestation Statement Format

   #signing-procedureReferenced in:
     * 6.3.2. Attestation Statement Formats
     * 6.3.4. Generating an Attestation Object

   #authenticator-data-for-the-attestationReferenced in:
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format
     * 8.4. Android Key Attestation Statement Format (2)
     * 8.5. Android SafetyNet Attestation Statement Format
     * 8.6. FIDO U2F Attestation Statement Format

   #verification-procedure-inputsReferenced in:
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format
     * 8.4. Android Key Attestation Statement Format
     * 8.5. Android SafetyNet Attestation Statement Format
     * 8.6. FIDO U2F Attestation Statement Format

   #authenticator-data-claimed-to-have-been-used-for-the-attestationRefere
   nced in:
     * 8.4. Android Key Attestation Statement Format

   #attestation-trust-pathReferenced in:
     * 6.3.2. Attestation Statement Formats
     * 8.2. Packed Attestation Statement Format (2) (3)
     * 8.3. TPM Attestation Statement Format
     * 8.4. Android Key Attestation Statement Format
     * 8.5. Android SafetyNet Attestation Statement Format
     * 8.6. FIDO U2F Attestation Statement Format

   #basic-attestationReferenced in:
     * 14.1. Attestation Privacy

   #basicReferenced in:
     * 8.2. Packed Attestation Statement Format (2)
     * 8.4. Android Key Attestation Statement Format (2)
     * 8.5. Android SafetyNet Attestation Statement Format (2)
     * 8.6. FIDO U2F Attestation Statement Format (2)

   #self-attestationReferenced in:
     * 4. Terminology (2) (3) (4)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.4.6. Attestation Conveyance Preference enumeration (enum
       AttestationConveyancePreference)
     * 6.3. Attestation (2)
     * 6.3.2. Attestation Statement Formats
     * 6.3.3. Attestation Types
     * 7.1. Registering a new credential (2) (3)
     * 8.2. Packed Attestation Statement Format (2)
     * 13.2.2. Attestation Certificate and Attestation Certificate CA
       Compromise
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation

   #selfReferenced in:
     * 8.2. Packed Attestation Statement Format

   #attestation-caReferenced in:
     * 5.4.6. Attestation Conveyance Preference enumeration (enum
       AttestationConveyancePreference)
     * 6.3.3. Attestation Types (2)
     * 14.1. Attestation Privacy (2)

   #attcaReferenced in:
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format (2)
     * 8.6. FIDO U2F Attestation Statement Format
```

```
     * 8.4. Android Key Attestation Statement Format

   #signing-procedureReferenced in:
     * 6.3.2. Attestation Statement Formats
     * 6.3.4. Generating an Attestation Object

   #authenticator-data-for-the-attestationReferenced in:
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format
     * 8.4. Android Key Attestation Statement Format (2)
     * 8.5. Android SafetyNet Attestation Statement Format
     * 8.6. FIDO U2F Attestation Statement Format

   #verification-procedure-inputsReferenced in:
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format
     * 8.4. Android Key Attestation Statement Format
     * 8.5. Android SafetyNet Attestation Statement Format
     * 8.6. FIDO U2F Attestation Statement Format

   #authenticator-data-claimed-to-have-been-used-for-the-attestationRefere
   nced in:
     * 8.4. Android Key Attestation Statement Format

   #attestation-trust-pathReferenced in:
     * 6.3.2. Attestation Statement Formats
     * 8.2. Packed Attestation Statement Format (2) (3)
     * 8.3. TPM Attestation Statement Format
     * 8.4. Android Key Attestation Statement Format
     * 8.5. Android SafetyNet Attestation Statement Format
     * 8.6. FIDO U2F Attestation Statement Format

   #basic-attestationReferenced in:
     * 14.1. Attestation Privacy

   #basicReferenced in:
     * 8.2. Packed Attestation Statement Format (2)
     * 8.4. Android Key Attestation Statement Format (2)
     * 8.5. Android SafetyNet Attestation Statement Format (2)
     * 8.6. FIDO U2F Attestation Statement Format (2)

   #self-attestationReferenced in:
     * 4. Terminology (2) (3) (4)
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.4.6. Attestation Conveyance Preference enumeration (enum
       AttestationConveyancePreference)
     * 6.3. Attestation (2)
     * 6.3.2. Attestation Statement Formats
     * 6.3.3. Attestation Types
     * 7.1. Registering a new credential (2) (3)
     * 8.2. Packed Attestation Statement Format (2)
     * 13.2.2. Attestation Certificate and Attestation Certificate CA
       Compromise
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation

   #selfReferenced in:
     * 8.2. Packed Attestation Statement Format

   #attestation-caReferenced in:
     * 5.4.6. Attestation Conveyance Preference enumeration (enum
       AttestationConveyancePreference)
     * 6.3.3. Attestation Types (2)
     * 14.1. Attestation Privacy (2)

   #attcaReferenced in:
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format (2)
     * 8.6. FIDO U2F Attestation Statement Format

   #elliptic-curve-based-direct-anonymous-attestationReferenced in:
     * 14.1. Attestation Privacy

   #ecdaaReferenced in:
     * 6.3.2. Attestation Statement Formats
     * 6.3.3. Attestation Types
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format (2) (3) (4)
     * 8.3. TPM Attestation Statement Format (2) (3) (4)
     * 13.2.2. Attestation Certificate and Attestation Certificate CA
       Compromise

   #no-attestation-statementReferenced in:
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation

   #noneReferenced in:
     * 8.7. None Attestation Statement Format (2)
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation

   #attestation-statement-format-identifierReferenced in:
     * 6.3.2. Attestation Statement Formats
     * 6.3.4. Generating an Attestation Object

   #identifier-of-the-ecdaa-issuer-public-keyReferenced in:
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format (2)

   #ecdaa-issuer-public-keyReferenced in:
     * 6.3.2. Attestation Statement Formats
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format (2) (3)
     * 14.1. Attestation Privacy

   #registration-extensionReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
     * 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)
     * 10.9. Biometric Authenticator Performance Bounds Extension
       (biometricPerfBounds)
     * 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
       (6) (7)

   #authentication-extensionReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
     * 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)
       (6)

   #client-extensionReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
```

```

   #elliptic-curve-based-direct-anonymous-attestationReferenced in:
     * 14.1. Attestation Privacy

   #ecdaaReferenced in:
     * 6.3.2. Attestation Statement Formats
     * 6.3.3. Attestation Types
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format (2) (3) (4)
     * 8.3. TPM Attestation Statement Format (2) (3) (4)
     * 13.2.2. Attestation Certificate and Attestation Certificate CA
       Compromise

   #no-attestation-statementReferenced in:
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation

   #noneReferenced in:
     * 8.7. None Attestation Statement Format (2)
     * 13.3.1. Considerations for Self and None Attestation Types and
       Ignoring Attestation

   #attestation-statement-format-identifierReferenced in:
     * 6.3.2. Attestation Statement Formats
     * 6.3.4. Generating an Attestation Object

   #identifier-of-the-ecdaa-issuer-public-keyReferenced in:
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format
     * 8.3. TPM Attestation Statement Format (2)

   #ecdaa-issuer-public-keyReferenced in:
     * 6.3.2. Attestation Statement Formats
     * 7.1. Registering a new credential
     * 8.2. Packed Attestation Statement Format (2) (3)
     * 14.1. Attestation Privacy

   #registration-extensionReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
     * 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)
     * 10.9. Biometric Authenticator Performance Bounds Extension
       (biometricPerfBounds)
     * 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
       (6) (7)

   #authentication-extensionReferenced in:
     * 5.1.4.1. PublicKeyCredential's
       [[DiscoverFromExternalSource]](origin, options,
       sameOriginWithAncestors) method
     * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
     * 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)
       (6)

   #client-extensionReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.1.4.1. PublicKeyCredential's
```

```
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 9. WebAuthn Extensions
       * 9.2. Defining extensions
       * 9.4. Client extension processing
       * 10.1. FIDO AppID Extension (appid)

     #authenticator-extensionReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 9. WebAuthn Extensions (2) (3)
       * 9.2. Defining extensions (2)
       * 9.3. Extending request parameters
       * 9.5. Authenticator extension processing

     #extension-identifierReferenced in:
       * 5.1. PublicKeyCredential Interface
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 6.1. Authenticator data
       * 6.2.2. The authenticatorMakeCredential operation (2)
       * 6.2.3. The authenticatorGetAssertion operation (2)
       * 7.1. Registering a new credential (2)
       * 7.2. Verifying an authentication assertion (2)
       * 9. WebAuthn Extensions (2)
       * 9.2. Defining extensions
       * 9.3. Extending request parameters
       * 9.4. Client extension processing (2)
       * 9.5. Authenticator extension processing (2)
       * 10.5. Supported Extensions Extension (exts) (2)
       * 11.2. WebAuthn Extension Identifier Registrations

     #client-extension-inputReferenced in:
       * 5.7. Authentication Extensions Client Inputs (typedef
         AuthenticationExtensionsClientInputs)
       * 7.1. Registering a new credential
       * 7.2. Verifying an authentication assertion
       * 9. WebAuthn Extensions (2) (3) (4)
       * 9.2. Defining extensions
       * 9.3. Extending request parameters (2) (3) (4) (5) (6)
       * 9.4. Client extension processing (2) (3) (4)

     #authenticator-extension-inputReferenced in:
       * 5.9. Authentication Extensions Authenticator Inputs (typedef
         AuthenticationExtensionsAuthenticatorInputs)
       * 6.2.2. The authenticatorMakeCredential operation (2)
       * 6.2.3. The authenticatorGetAssertion operation (2)
       * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
       * 9.2. Defining extensions
       * 9.3. Extending request parameters (2) (3)
       * 9.4. Client extension processing
       * 9.5. Authenticator extension processing (2) (3)

     #client-extension-processingReferenced in:
       * 5.1. PublicKeyCredential Interface
       * 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 (2)
       * 9. WebAuthn Extensions (2) (3) (4)
       * 9.2. Defining extensions

     #client-extension-outputReferenced in:
```

```
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 9. WebAuthn Extensions
       * 9.2. Defining extensions
       * 9.4. Client extension processing
       * 10.1. FIDO AppID Extension (appid)

     #authenticator-extensionReferenced in:
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 9. WebAuthn Extensions (2) (3)
       * 9.2. Defining extensions (2)
       * 9.3. Extending request parameters
       * 9.5. Authenticator extension processing

     #extension-identifierReferenced in:
       * 5.1. PublicKeyCredential Interface
       * 5.1.3. Create a new credential - PublicKeyCredential's
         [[Create]](origin, options, sameOriginWithAncestors) method
       * 5.1.4.1. PublicKeyCredential's
         [[DiscoverFromExternalSource]](origin, options,
         sameOriginWithAncestors) method
       * 6.1. Authenticator data
       * 6.2.2. The authenticatorMakeCredential operation (2)
       * 6.2.3. The authenticatorGetAssertion operation (2)
       * 7.1. Registering a new credential (2)
       * 7.2. Verifying an authentication assertion (2)
       * 9. WebAuthn Extensions (2)
       * 9.2. Defining extensions
       * 9.3. Extending request parameters
       * 9.4. Client extension processing (2)
       * 9.5. Authenticator extension processing (2)
       * 10.5. Supported Extensions Extension (exts) (2)
       * 11.2. WebAuthn Extension Identifier Registrations

     #client-extension-inputReferenced in:
       * 5.7. Authentication Extensions Client Inputs (typedef
         AuthenticationExtensionsClientInputs)
       * 7.1. Registering a new credential
       * 7.2. Verifying an authentication assertion
       * 9. WebAuthn Extensions (2) (3) (4)
       * 9.2. Defining extensions
       * 9.3. Extending request parameters (2) (3) (4) (5) (6)
       * 9.4. Client extension processing (2) (3) (4)

     #authenticator-extension-inputReferenced in:
       * 5.9. Authentication Extensions Authenticator Inputs (typedef
         AuthenticationExtensionsAuthenticatorInputs)
       * 6.2.2. The authenticatorMakeCredential operation (2)
       * 6.2.3. The authenticatorGetAssertion operation (2)
       * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
       * 9.2. Defining extensions
       * 9.3. Extending request parameters (2) (3)
       * 9.4. Client extension processing
       * 9.5. Authenticator extension processing (2) (3)

     #client-extension-processingReferenced in:
       * 5.1. PublicKeyCredential Interface
       * 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 (2)
       * 9. WebAuthn Extensions (2) (3) (4)
       * 9.2. Defining extensions

     #client-extension-outputReferenced in:
```

```
     * 5.1. PublicKeyCredential Interface
     * 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 (2)
     * 5.8. Authentication Extensions Client Outputs (typedef
       AuthenticationExtensionsClientOutputs)
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion
     * 9. WebAuthn Extensions (2) (3) (4)
     * 9.2. Defining extensions (2) (3)
     * 9.4. Client extension processing (2) (3)

   #authenticator-extension-processingReferenced in:
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation
     * 9. WebAuthn Extensions
     * 9.2. Defining extensions
     * 9.5. Authenticator extension processing

   #authenticator-extension-outputReferenced in:
     * 6.1. Authenticator data
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion
     * 9. WebAuthn Extensions (2) (3) (4)
     * 9.2. Defining extensions (2) (3)
     * 9.4. Client extension processing
     * 9.5. Authenticator extension processing
     * 10.5. Supported Extensions Extension (exts)
     * 10.6. User Verification Index Extension (uvi)
     * 10.8. User Verification Method Extension (uvm)

   #appidReferenced in:
     * 3. Dependencies

   #dictdef-txauthgenericargReferenced in:
     * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)

   #typedefdef-authenticatorselectionlistReferenced in:
     * 10.4. Authenticator Selection Extension (authnSel) (2)

   #typedefdef-aaguidReferenced in:
     * 10.4. Authenticator Selection Extension (authnSel)

   #typedefdef-authenticationextensionssupportedReferenced in:
     * 10.5. Supported Extensions Extension (exts)

   #typedefdef-uvmentryReferenced in:
     * 10.8. User Verification Method Extension (uvm)

   #typedefdef-uvmentriesReferenced in:
     * 10.8. User Verification Method Extension (uvm)

   #anonymization-caReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.4.6. Attestation Conveyance Preference enumeration (enum
       AttestationConveyancePreference)
     * 14.1. Attestation Privacy (2) (3)
```

```
     * 5.1. PublicKeyCredential Interface
     * 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 (2)
     * 5.8. Authentication Extensions Client Outputs (typedef
       AuthenticationExtensionsClientOutputs)
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion
     * 9. WebAuthn Extensions (2) (3) (4)
     * 9.2. Defining extensions (2) (3)
     * 9.4. Client extension processing (2) (3)

   #authenticator-extension-processingReferenced in:
     * 6.2.2. The authenticatorMakeCredential operation
     * 6.2.3. The authenticatorGetAssertion operation
     * 9. WebAuthn Extensions
     * 9.2. Defining extensions
     * 9.5. Authenticator extension processing

   #authenticator-extension-outputReferenced in:
     * 6.1. Authenticator data
     * 7.1. Registering a new credential
     * 7.2. Verifying an authentication assertion
     * 9. WebAuthn Extensions (2) (3) (4)
     * 9.2. Defining extensions (2) (3)
     * 9.4. Client extension processing
     * 9.5. Authenticator extension processing
     * 10.5. Supported Extensions Extension (exts)
     * 10.6. User Verification Index Extension (uvi)
     * 10.8. User Verification Method Extension (uvm)

   #appidReferenced in:
     * 3. Dependencies

   #dictdef-txauthgenericargReferenced in:
     * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)

   #typedefdef-authenticatorselectionlistReferenced in:
     * 10.4. Authenticator Selection Extension (authnSel) (2)

   #typedefdef-aaguidReferenced in:
     * 10.4. Authenticator Selection Extension (authnSel)

   #typedefdef-authenticationextensionssupportedReferenced in:
     * 10.5. Supported Extensions Extension (exts)

   #typedefdef-uvmentryReferenced in:
     * 10.8. User Verification Method Extension (uvm)

   #typedefdef-uvmentriesReferenced in:
     * 10.8. User Verification Method Extension (uvm)

   #anonymization-caReferenced in:
     * 5.1.3. Create a new credential - PublicKeyCredential's
       [[Create]](origin, options, sameOriginWithAncestors) method
     * 5.4.6. Attestation Conveyance Preference enumeration (enum
       AttestationConveyancePreference)
     * 14.1. Attestation Privacy (2) (3)
```