

0001 THE_URL:file://localhost/Users/jehodges/documents/work/standards/W3C/WebAuthn/index-master-3ee8ed5.html

0002 THE_TITLE:Web Authentication: An API for accessing Public Key Credentials - Level 1
0003 W3C

0004 Web Authentication:
0005 An API for accessing Public Key Credentials
0006 Level 1
0007
0008

0009 Editor's Draft, 28 September 2017

0010 This version:
0011 <https://w3c.github.io/webauthn/>

0012 Latest published version:
0013 <https://www.w3.org/TR/webauthn/>

0014 Previous Versions:
0015 <https://www.w3.org/TR/2017/WD-webauthn-20170811/>
0016 <https://www.w3.org/TR/2017/WD-webauthn-20170505/>
0017 <https://www.w3.org/TR/2017/WD-webauthn-20170216/>
0018 <https://www.w3.org/TR/2016/WD-webauthn-20161207/>
0019 <https://www.w3.org/TR/2016/WD-webauthn-20160928/>
0020 <https://www.w3.org/TR/2016/WD-webauthn-20160902/>
0021 <https://www.w3.org/TR/2016/WD-webauthn-20160531/>

0022 Issue Tracking:
0023 Github

0024 Editors:
0025 Vijay Bharadwaj (Microsoft)
0026 Hubert Le Van Gong (PayPal)
0027 Dirk Balfanz (Google)
0028 Alexei Czeskis (Google)
0029 Arnar Birgisson (Google)
0030 Jeff Hodges (PayPal)
0031 Michael B. Jones (Microsoft)
0032 Rolf Lindemann (Nok Nok Labs)
0033 J.C. Jones (Mozilla)

0034 Tests:
0035 web-platform-tests webauthn/ (ongoing work)

0036 Copyright 2017 W3C[^] (MIT, ERCIM, Keio, Beihang). W3C liability,
0037 trademark and document use rules apply.

0038 **Abstract**

0039 This specification defines an API enabling the creation and use of
0040 strong, attested, scoped, public key-based credentials by web
0041 applications, for the purpose of strongly authenticating users.
0042 Conceptually, one or more public key credentials, each scoped to a
0043 given Relying Party, are created and stored on an authenticator by the
0044 user agent in conjunction with the web application. The user agent
0045 mediates access to public key credentials in order to preserve user
0046 privacy. Authenticators are responsible for ensuring that no operation
0047 is performed without user consent. Authenticators provide cryptographic
0048 proof of their properties to relying parties via attestation. This
0049 specification also describes the functional model for WebAuthn
0050 conformant authenticators, including their signature and attestation
0051 functionality.

0052 **Status of this document**

0053 This section describes the status of this document at the time of its
0054 publication. Other documents may supersede this document. A list of
0055 current W3C publications and the latest revision of this technical
0056 report can be found in the W3C technical reports index at
0057 <http://www.w3.org/TR/>.

0001 THE_URL:file://localhost/Users/jehodges/documents/work/standards/W3C/WebAuthn/index-jeffh-fixup-algs-contd-3-a1b998f.html

0002 THE_TITLE:Web Authentication: An API for accessing Public Key Credentials - Level 1
0003 W3C

0004 Web Authentication:
0005 An API for accessing Public Key Credentials
0006 Level 1
0007
0008

0009 Editor's Draft, 2 October 2017

0010 This version:
0011 <https://w3c.github.io/webauthn/>

0012 Latest published version:
0013 <https://www.w3.org/TR/webauthn/>

0014 Previous Versions:
0015 <https://www.w3.org/TR/2017/WD-webauthn-20170811/>
0016 <https://www.w3.org/TR/2017/WD-webauthn-20170505/>
0017 <https://www.w3.org/TR/2017/WD-webauthn-20170216/>
0018 <https://www.w3.org/TR/2016/WD-webauthn-20161207/>
0019 <https://www.w3.org/TR/2016/WD-webauthn-20160928/>
0020 <https://www.w3.org/TR/2016/WD-webauthn-20160902/>
0021 <https://www.w3.org/TR/2016/WD-webauthn-20160531/>

0022 Issue Tracking:
0023 Github

0024 Editors:
0025 Vijay Bharadwaj (Microsoft)
0026 Hubert Le Van Gong (PayPal)
0027 Dirk Balfanz (Google)
0028 Alexei Czeskis (Google)
0029 Arnar Birgisson (Google)
0030 Jeff Hodges (PayPal)
0031 Michael B. Jones (Microsoft)
0032 Rolf Lindemann (Nok Nok Labs)
0033 J.C. Jones (Mozilla)

0034 Tests:
0035 web-platform-tests webauthn/ (ongoing work)

0036 Copyright 2017 W3C[^] (MIT, ERCIM, Keio, Beihang). W3C liability,
0037 trademark and document use rules apply.

0038 **Abstract**

0039 This specification defines an API enabling the creation and use of
0040 strong, attested, scoped, public key-based credentials by web
0041 applications, for the purpose of strongly authenticating users.
0042 Conceptually, one or more public key credentials, each scoped to a
0043 given Relying Party, are created and stored on an authenticator by the
0044 user agent in conjunction with the web application. The user agent
0045 mediates access to public key credentials in order to preserve user
0046 privacy. Authenticators are responsible for ensuring that no operation
0047 is performed without user consent. Authenticators provide cryptographic
0048 proof of their properties to relying parties via attestation. This
0049 specification also describes the functional model for WebAuthn
0050 conformant authenticators, including their signature and attestation
0051 functionality.

0052 **Status of this document**

0053 This section describes the status of this document at the time of its
0054 publication. Other documents may supersede this document. A list of
0055 current W3C publications and the latest revision of this technical
0056 report can be found in the W3C technical reports index at
0057 <http://www.w3.org/TR/>.

0070 This document was published by the Web Authentication Working Group as
0071 an Editors' Draft. This document is intended to become a W3C
0072 Recommendation. Feedback and comments on this specification are
0073 welcome. Please use Github issues. Discussions may also be found in the
0074 public-webauthn@w3.org archives.
0075

0076 Publication as an Editors' Draft does not imply endorsement by the W3C
0077 Membership. This is a draft document and may be updated, replaced or
0078 obsoleted by other documents at any time. It is inappropriate to cite
0079 this document as other than work in progress.
0080

0081 This document was produced by a group operating under the 5 February
0082 2004 W3C Patent Policy. W3C maintains a public list of any patent
0083 disclosures made in connection with the deliverables of the group; that
0084 page also includes instructions for disclosing a patent. An individual
0085 who has actual knowledge of a patent which the individual believes
0086 contains Essential Claim(s) must disclose the information in accordance
0087 with section 6 of the W3C Patent Policy.
0088

0089 This document is governed by the 1 March 2017 W3C Process Document.
0090

0091 Table of Contents
0092

0093
0094 1. 1 Introduction
0095 1. 1.1 Use Cases
0096 1. 1.1.1 Registration
0097 2. 1.1.2 Authentication
0098 3. 1.1.3 Other use cases and configurations
0099 2. 2 Conformance
0100 1. 2.1 Dependencies
0101 3. 3 Terminology
0102 4. 4 Web Authentication API
0103 1. 4.1 PublicKeyCredential Interface
0104 1. 4.1.1 CredentialCreationOptions Extension
0105 2. 4.1.2 CredentialRequestOptions Extension
0106 3. 4.1.3 Create a new credential - PublicKeyCredential's
0107 [[Create]](options) method

0108 4. 4.1.4 Use an existing credential to make an assertion -
0109 PublicKeyCredential's
0110 [[DiscoverFromExternalSource]](options) method
0111 5. 4.1.5 Platform Authenticator Availability -
0112 PublicKeyCredential's isPlatformAuthenticatorAvailable()
0113 method
0114 2. 4.2 Authenticator Responses (interface AuthenticatorResponse)
0115 1. 4.2.1 Information about Public Key Credential (interface
0116 AuthenticatorAttestationResponse)
0117 2. 4.2.2 Web Authentication Assertion (interface
0118 AuthenticatorAssertionResponse)
0119 3. 4.3 Parameters for Credential Generation (dictionary
0120 PublicKeyCredentialParameters)
0121 4. 4.4 Options for Credential Creation (dictionary
0122 MakePublicKeyCredentialOptions)
0123 1. 4.4.1 Public Key Entity Description (dictionary
0124 PublicKeyCredentialEntity)
0125 2. 4.4.2 RP Parameters for Credential Generation (dictionary
0126 PublicKeyCredentialRpEntity)
0127 3. 4.4.3 User Account Parameters for Credential Generation
0128 (dictionary PublicKeyCredentialUserEntity)
0129 4. 4.4.4 Authenticator Selection Criteria (dictionary
0130 AuthenticatorSelectionCriteria)
0131 5. 4.4.5 Authenticator Attachment enumeration (enum
0132 AuthenticatorAttachment)
0133 5. 4.5 Options for Assertion Generation (dictionary
0134 PublicKeyCredentialRequestOptions)
0135 6. 4.6 Authentication Extensions (typedef
0136 AuthenticationExtensions)
0137 7. 4.7 Supporting Data Structures

0070 This document was published by the Web Authentication Working Group as
0071 an Editors' Draft. This document is intended to become a W3C
0072 Recommendation. Feedback and comments on this specification are
0073 welcome. Please use Github issues. Discussions may also be found in the
0074 public-webauthn@w3.org archives.
0075

0076 Publication as an Editors' Draft does not imply endorsement by the W3C
0077 Membership. This is a draft document and may be updated, replaced or
0078 obsoleted by other documents at any time. It is inappropriate to cite
0079 this document as other than work in progress.
0080

0081 This document was produced by a group operating under the 5 February
0082 2004 W3C Patent Policy. W3C maintains a public list of any patent
0083 disclosures made in connection with the deliverables of the group; that
0084 page also includes instructions for disclosing a patent. An individual
0085 who has actual knowledge of a patent which the individual believes
0086 contains Essential Claim(s) must disclose the information in accordance
0087 with section 6 of the W3C Patent Policy.
0088

0089 This document is governed by the 1 March 2017 W3C Process Document.
0090

0091 Table of Contents
0092

0093
0094 1. 1 Introduction
0095 1. 1.1 Use Cases
0096 1. 1.1.1 Registration
0097 2. 1.1.2 Authentication
0098 3. 1.1.3 Other use cases and configurations
0099 2. 2 Conformance
0100 1. 2.1 Dependencies
0101 3. 3 Terminology
0102 4. 4 Web Authentication API
0103 1. 4.1 PublicKeyCredential Interface
0104 1. 4.1.1 CredentialCreationOptions Extension
0105 2. 4.1.2 CredentialRequestOptions Extension
0106 3. 4.1.3 Create a new credential - PublicKeyCredential's
0107 [[Create]](options) method
0108 1. 4.1.3.1 Construct the credential -
0109 constructCredentialCallback method
0110 4. 4.1.4 Use an existing credential to make an assertion -
0111 PublicKeyCredential's
0112 [[DiscoverFromExternalSource]](options) method
0113 5. 4.1.5 Platform Authenticator Availability -
0114 PublicKeyCredential's isPlatformAuthenticatorAvailable()
0115 method
0116 2. 4.2 Authenticator Responses (interface AuthenticatorResponse)
0117 1. 4.2.1 Information about Public Key Credential (interface
0118 AuthenticatorAttestationResponse)
0119 2. 4.2.2 Web Authentication Assertion (interface
0120 AuthenticatorAssertionResponse)
0121 3. 4.3 Parameters for Credential Generation (dictionary
0122 PublicKeyCredentialParameters)
0123 4. 4.4 Options for Credential Creation (dictionary
0124 MakePublicKeyCredentialOptions)
0125 1. 4.4.1 Public Key Entity Description (dictionary
0126 PublicKeyCredentialEntity)
0127 2. 4.4.2 RP Parameters for Credential Generation (dictionary
0128 PublicKeyCredentialRpEntity)
0129 3. 4.4.3 User Account Parameters for Credential Generation
0130 (dictionary PublicKeyCredentialUserEntity)
0131 4. 4.4.4 Authenticator Selection Criteria (dictionary
0132 AuthenticatorSelectionCriteria)
0133 5. 4.4.5 Authenticator Attachment enumeration (enum
0134 AuthenticatorAttachment)
0135 5. 4.5 Options for Assertion Generation (dictionary
0136 PublicKeyCredentialRequestOptions)
0137 6. 4.6 Authentication Extensions (typedef
0138 AuthenticationExtensions)
0139 7. 4.7 Supporting Data Structures

- 013E 1. 4.7.1 Client data used in WebAuthn signatures (dictionary
CollectedClientData)
- 0140 2. 4.7.2 Credential Type enumeration (enum
PublicKeyCredentialType)
- 0142 3. 4.7.3 Credential Descriptor (dictionary
PublicKeyCredentialDescriptor)
- 0144 4. 4.7.4 Authenticator Transport enumeration (enum
AuthenticatorTransport)
- 0146 5. 4.7.5 Cryptographic Algorithm Identifier (typedef
COSEAlgorithmIdentifier)
- 0147 5. 5 WebAuthn Authenticator model
- 0148 1. 5.1 Authenticator data
- 0149 2. 5.2 Authenticator operations
- 0150 1. 5.2.1 The authenticatorMakeCredential operation
- 0151 2. 5.2.2 The authenticatorGetAssertion operation
- 0152 3. 5.2.3 The authenticatorCancel operation
- 0153 3. 5.3 Attestation
- 0154 1. 5.3.1 Attestation data
- 0155 2. 5.3.2 Attestation Statement Formats
- 0156 3. 5.3.3 Attestation Types
- 0157 4. 5.3.4 Generating an Attestation Object
- 0158 5. 5.3.5 Security Considerations
- 0159 1. 5.3.5.1 Privacy
- 0160 2. 5.3.5.2 Attestation Certificate and Attestation
Certificate CA Compromise
- 0161 3. 5.3.5.3 Attestation Certificate Hierarchy
- 0162 6. 6 Relying Party Operations
- 0163 1. 6.1 Registering a new credential
- 0164 2. 6.2 Verifying an authentication assertion
- 0165 7. 7 Defined Attestation Statement Formats
- 0166 1. 7.1 Attestation Statement Format Identifiers
- 0167 2. 7.2 Packed Attestation Statement Format
- 0168 1. 7.2.1 Packed attestation statement certificate
requirements
- 0169 3. 7.3 TPM Attestation Statement Format
- 0170 1. 7.3.1 TPM attestation statement certificate requirements
- 0171 4. 7.4 Android Key Attestation Statement Format
- 0172 5. 7.5 Android SafetyNet Attestation Statement Format
- 0173 6. 7.6 FIDO U2F Attestation Statement Format
- 0174 8. 8 WebAuthn Extensions
- 0175 1. 8.1 Extension Identifiers
- 0176 2. 8.2 Defining extensions
- 0177 3. 8.3 Extending request parameters
- 0178 4. 8.4 Client extension processing
- 0179 5. 8.5 Authenticator extension processing
- 0180 6. 8.6 Example Extension
- 0181 9. 9 Defined Extensions
- 0182 1. 9.1 FIDO AppId Extension (appid)
- 0183 2. 9.2 Simple Transaction Authorization Extension (txAuthSimple)
- 0184 3. 9.3 Generic Transaction Authorization Extension
(txAuthGeneric)
- 0185 4. 9.4 Authenticator Selection Extension (authnSel)
- 0186 5. 9.5 Supported Extensions Extension (exts)
- 0187 6. 9.6 User Verification Index Extension (uvi)
- 0188 7. 9.7 Location Extension (loc)
- 0189 8. 9.8 User Verification Method Extension (uvm)
- 0190 10. 10 IANA Considerations
- 0191 1. 10.1 WebAuthn Attestation Statement Format Identifier
Registrations
- 0192 2. 10.2 WebAuthn Extension Identifier Registrations
- 0193 3. 10.3 COSE Algorithm Registrations
- 0194 11. 11 Sample scenarios
- 0195 1. 11.1 Registration
- 0196 2. 11.2 Registration Specifically with Platform Authenticator
- 0197 3. 11.3 Authentication
- 0198 4. 11.4 Decommissioning
- 0199 12. 12 Acknowledgements
- 0200 13. Index
- 0201 1. Terms defined by this specification
- 0202 2. Terms defined by reference

- 014C 1. 4.7.1 Client data used in WebAuthn signatures (dictionary
CollectedClientData)
- 014D 2. 4.7.2 Credential Type enumeration (enum
PublicKeyCredentialType)
- 014E 3. 4.7.3 Credential Descriptor (dictionary
PublicKeyCredentialDescriptor)
- 014F 4. 4.7.4 Authenticator Transport enumeration (enum
AuthenticatorTransport)
- 0150 5. 4.7.5 Cryptographic Algorithm Identifier (typedef
COSEAlgorithmIdentifier)
- 0151 5. 5 WebAuthn Authenticator model
- 0152 1. 5.1 Authenticator data
- 0153 2. 5.2 Authenticator operations
- 0154 1. 5.2.1 The authenticatorMakeCredential operation
- 0155 2. 5.2.2 The authenticatorGetAssertion operation
- 0156 3. 5.2.3 The authenticatorCancel operation
- 0157 3. 5.3 Attestation
- 0158 1. 5.3.1 Attestation data
- 0159 2. 5.3.2 Attestation Statement Formats
- 0160 3. 5.3.3 Attestation Types
- 0161 4. 5.3.4 Generating an Attestation Object
- 0162 5. 5.3.5 Security Considerations
- 0163 1. 5.3.5.1 Privacy
- 0164 2. 5.3.5.2 Attestation Certificate and Attestation
Certificate CA Compromise
- 0165 3. 5.3.5.3 Attestation Certificate Hierarchy
- 0166 6. 6 Relying Party Operations
- 0167 1. 6.1 Registering a new credential
- 0168 2. 6.2 Verifying an authentication assertion
- 0169 7. 7 Defined Attestation Statement Formats
- 0170 1. 7.1 Attestation Statement Format Identifiers
- 0171 2. 7.2 Packed Attestation Statement Format
- 0172 1. 7.2.1 Packed attestation statement certificate
requirements
- 0173 3. 7.3 TPM Attestation Statement Format
- 0174 1. 7.3.1 TPM attestation statement certificate requirements
- 0175 4. 7.4 Android Key Attestation Statement Format
- 0176 5. 7.5 Android SafetyNet Attestation Statement Format
- 0177 6. 7.6 FIDO U2F Attestation Statement Format
- 0178 8. 8 WebAuthn Extensions
- 0179 1. 8.1 Extension Identifiers
- 0180 2. 8.2 Defining extensions
- 0181 3. 8.3 Extending request parameters
- 0182 4. 8.4 Client extension processing
- 0183 5. 8.5 Authenticator extension processing
- 0184 6. 8.6 Example Extension
- 0185 9. 9 Defined Extensions
- 0186 1. 9.1 FIDO AppId Extension (appid)
- 0187 2. 9.2 Simple Transaction Authorization Extension (txAuthSimple)
- 0188 3. 9.3 Generic Transaction Authorization Extension
(txAuthGeneric)
- 0189 4. 9.4 Authenticator Selection Extension (authnSel)
- 0190 5. 9.5 Supported Extensions Extension (exts)
- 0191 6. 9.6 User Verification Index Extension (uvi)
- 0192 7. 9.7 Location Extension (loc)
- 0193 8. 9.8 User Verification Method Extension (uvm)
- 0194 10. 10 IANA Considerations
- 0195 1. 10.1 WebAuthn Attestation Statement Format Identifier
Registrations
- 0196 2. 10.2 WebAuthn Extension Identifier Registrations
- 0197 3. 10.3 COSE Algorithm Registrations
- 0198 11. 11 Sample scenarios
- 0199 1. 11.1 Registration
- 0200 2. 11.2 Registration Specifically with Platform Authenticator
- 0201 3. 11.3 Authentication
- 0202 4. 11.4 Decommissioning
- 0203 12. 12 Acknowledgements
- 0204 13. Index
- 0205 1. Terms defined by this specification
- 0206 2. Terms defined by reference

020E 14. References
 0209 1. Normative References
 021C 2. Informative References
 0211 15. IDL Index

0212
 0213 1. Introduction
 0214
 0215 This section is not normative.
 0216
 0217 This specification defines an API enabling the creation and use of
 0218 strong, attested, scoped, public key-based credentials by web
 0219 applications, for the purpose of strongly authenticating users. A
 0220 public key credential is created and stored by an authenticator at the
 0221 behest of a Relying Party, subject to user consent. Subsequently, the
 0222 public key credential can only be accessed by origins belonging to that
 0223 Relying Party. This scoping is enforced jointly by conforming User
 0224 Agents and authenticators. Additionally, privacy across Relying Parties
 0225 is maintained; Relying Parties are not able to detect any properties,
 0226 or even the existence, of credentials scoped to other Relying Parties.
 0227
 0228 Relying Parties employ the Web Authentication API during two distinct,
 0229 but related, ceremonies involving a user. The first is Registration,
 0230 where a public key credential is created on an authenticator, and
 0231 associated by a Relying Party with the present user's account (the
 0232 account may already exist or may be created at this time). The second
 0233 is Authentication, where the Relying Party is presented with an
 0234 Authentication Assertion proving the presence and consent of the user
 0235 who registered the public key credential. Functionally, the Web
 0236 Authentication API comprises a PublicKeyCredential which extends the
 0237 Credential Management API [CREDENTIAL-MANAGEMENT-1], and infrastructure
 0238 which allows those credentials to be used with
 0239 navigator.credentials.create() and navigator.credentials.get(). The
 0240 former is used during Registration, and the latter during
 0241 Authentication.
 0242
 0243 Broadly, compliant authenticators protect public key credentials, and
 0244 interact with user agents to implement the Web Authentication API. Some
 0245 authenticators may run on the same computing device (e.g., smart phone,
 0246 tablet, desktop PC) as the user agent is running on. For instance, such
 0247 an authenticator might consist of a Trusted Execution Environment (TEE)
 0248 applet, a Trusted Platform Module (TPM), or a Secure Element (SE)
 0249 integrated into the computing device in conjunction with some means for
 0250 user verification, along with appropriate platform software to mediate
 0251 access to these components' functionality. Other authenticators may
 0252 operate autonomously from the computing device running the user agent,
 0253 and be accessed over a transport such as Universal Serial Bus (USB),
 0254 Bluetooth Low Energy (BLE) or Near Field Communications (NFC).
 0255
 0256 1.1. Use Cases
 0257
 0258 The below use case scenarios illustrate use of two very different types
 0259 of authenticators, as well as outline further scenarios. Additional
 0260 scenarios, including sample code, are given later in 11 Sample
 0261 scenarios.
 0262
 0263 1.1.1. Registration
 0264
 0265 * On a phone:
 0266 + User navigates to example.com in a browser and signs in to an
 0267 existing account using whatever method they have been using
 0268 (possibly a legacy method such as a password), or creates a
 0269 new account.
 0270 + The phone prompts, "Do you want to register this device with
 0271 example.com?"
 0272 + User agrees.
 0273 + The phone prompts the user for a previously configured
 0274 authorization gesture (PIN, biometric, etc.); the user
 0275 provides this.
 0276 + Website shows message, "Registration complete."

021C 14. References
 0211 1. Normative References
 0212 2. Informative References
 0213 15. IDL Index
 0214 16. Issues Index
 0215
 0216 1. Introduction
 0217
 0218 This section is not normative.
 0219
 0220 This specification defines an API enabling the creation and use of
 0221 strong, attested, scoped, public key-based credentials by web
 0222 applications, for the purpose of strongly authenticating users. A
 0223 public key credential is created and stored by an authenticator at the
 0224 behest of a Relying Party, subject to user consent. Subsequently, the
 0225 public key credential can only be accessed by origins belonging to that
 0226 Relying Party. This scoping is enforced jointly by conforming User
 0227 Agents and authenticators. Additionally, privacy across Relying Parties
 0228 is maintained; Relying Parties are not able to detect any properties,
 0229 or even the existence, of credentials scoped to other Relying Parties.
 0230
 0231 Relying Parties employ the Web Authentication API during two distinct,
 0232 but related, ceremonies involving a user. The first is Registration,
 0233 where a public key credential is created on an authenticator, and
 0234 associated by a Relying Party with the present user's account (the
 0235 account may already exist or may be created at this time). The second
 0236 is Authentication, where the Relying Party is presented with an
 0237 Authentication Assertion proving the presence and consent of the user
 0238 who registered the public key credential. Functionally, the Web
 0239 Authentication API comprises a PublicKeyCredential which extends the
 0240 Credential Management API [CREDENTIAL-MANAGEMENT-1], and infrastructure
 0241 which allows those credentials to be used with
 0242 navigator.credentials.create() and navigator.credentials.get(). The
 0243 former is used during Registration, and the latter during
 0244 Authentication.
 0245
 0246 Broadly, compliant authenticators protect public key credentials, and
 0247 interact with user agents to implement the Web Authentication API. Some
 0248 authenticators may run on the same computing device (e.g., smart phone,
 0249 tablet, desktop PC) as the user agent is running on. For instance, such
 0250 an authenticator might consist of a Trusted Execution Environment (TEE)
 0251 applet, a Trusted Platform Module (TPM), or a Secure Element (SE)
 0252 integrated into the computing device in conjunction with some means for
 0253 user verification, along with appropriate platform software to mediate
 0254 access to these components' functionality. Other authenticators may
 0255 operate autonomously from the computing device running the user agent,
 0256 and be accessed over a transport such as Universal Serial Bus (USB),
 0257 Bluetooth Low Energy (BLE) or Near Field Communications (NFC).
 0258
 0259 1.1. Use Cases
 0260
 0261 The below use case scenarios illustrate use of two very different types
 0262 of authenticators, as well as outline further scenarios. Additional
 0263 scenarios, including sample code, are given later in 11 Sample
 0264 scenarios.
 0265
 0266 1.1.1. Registration
 0267
 0268 * On a phone:
 0269 + User navigates to example.com in a browser and signs in to an
 0270 existing account using whatever method they have been using
 0271 (possibly a legacy method such as a password), or creates a
 0272 new account.
 0273 + The phone prompts, "Do you want to register this device with
 0274 example.com?"
 0275 + User agrees.
 0276 + The phone prompts the user for a previously configured
 0277 authorization gesture (PIN, biometric, etc.); the user
 0278 provides this.
 0279 + Website shows message, "Registration complete."

0277
0278
0279
0280
0281
0282
0283
0284
0285
0286
0287
0288
0289
0290
0291
0292
0293
0294
0295
0296
0297
0298
0299
0300
0301
0302
0303
0304
0305
0306
0307
0308
0309
0310
0311
0312
0313
0314
0315
0316
0317
0318
0319
0320
0321
0322
0323
0324
0325
0326
0327
0328
0329
0330
0331
0332
0333
0334
0335
0336
0337
0338
0339
0340
0341
0342
0343
0344
0345
0346

1.1.2. Authentication

- * On a laptop or desktop:
 - + User navigates to example.com in a browser, sees an option to "Sign in with your phone."
 - + User chooses this option and gets a message from the browser, "Please complete this action on your phone."
- * Next, on their phone:
 - + User sees a discrete prompt or notification, "Sign in to example.com."
 - + User 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 an 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 through 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 criteria for a Conforming User Agent: A User Agent MUST behave as described in this specification 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]

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

2.1. Dependencies

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

Base64url encoding

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

CBOR

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

CDDL

0280
0281
0282
0283
0284
0285
0286
0287
0288
0289
0290
0291
0292
0293
0294
0295
0296
0297
0298
0299
0300
0301
0302
0303
0304
0305
0306
0307
0308
0309
0310
0311
0312
0313
0314
0315
0316
0317
0318
0319
0320
0321
0322
0323
0324
0325
0326
0327
0328
0329
0330
0331
0332
0333
0334
0335
0336
0337
0338
0339
0340
0341
0342
0343
0344
0345
0346
0347
0348
0349

1.1.2. Authentication

- * On a laptop or desktop:
 - + User navigates to example.com in a browser, sees an option to "Sign in with your phone."
 - + User chooses this option and gets a message from the browser, "Please complete this action on your phone."
- * Next, on their phone:
 - + User sees a discrete prompt or notification, "Sign in to example.com."
 - + User 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 an 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 through 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 criteria for a Conforming User Agent: A User Agent MUST behave as described in this specification 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]

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

2.1. Dependencies

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

Base64url encoding

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

CBOR

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

CDDL

0347 This specification describes the syntax of all CBOR-encoded data
 0348 using the CBOR Data Definition Language (CDDL) [CDDL].
 0349
 0350 **COSE**
 0351 CBOR Object Signing and Encryption (COSE) [RFC8152]. The IANA
 0352 COSE Algorithms registry established by this specification is
 0353 also used.
 0354
 0355 **Credential Management**
 0356 The API described in this document is an extension of the
 0357 Credential concept defined in [CREDENTIAL-MANAGEMENT-1].
 0358
 0359 **DOM**
 0360 DOMException and the DOMException values used in this
 0361 specification are defined in [DOM4].
 0362
 0363 **ECMAScript**
 0364 %ArrayBuffer% is defined in [ECMAScript].
 0365
 0366 **HTML**
 0367 The concepts of relevant settings object, origin, opaque origin,
 0368 and is a registrable domain suffix of or is equal to are defined
 0369 in [HTML52].
 0370
 0371 **Web IDL**
 0372 Many of the interface definitions and all of the IDL in this
 0373 specification depend on [WebIDL-1]. This updated version of the
 0374 Web IDL standard adds support for Promises, which are now the
 0375 preferred mechanism for asynchronous interaction in all new web
 0376 APIs.
 0377
 0378 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
 0379 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
 0380 document are to be interpreted as described in [RFC2119].
 0381
 0382 **3. Terminology**
 0383
 0384 **Assertion**
 0385 See Authentication Assertion.
 0386
 0387 **Attestation**
 0388 Generally, attestation is a statement serving to bear witness,
 0389 confirm, or authenticate. In the WebAuthn context, attestation
 0390 is employed to attest to the provenance of an authenticator and
 0391 the data it emits; including, for example: credential IDs,
 0392 credential key pairs, signature counters, etc. An attestation
 0393 statement is conveyed in an attestation object during
 0394 registration. See also 5.3 Attestation and Figure 3.
 0395
 0396 **Attestation Certificate**
 0397 A X.509 Certificate for the attestation key pair used by an
 0398 authenticator to attest to its manufacture and capabilities. At
 0399 registration time, the authenticator uses the attestation
 0400 private key to sign the Relying Party-specific credential public
 0401 key (and additional data) that it generates and returns via the
 0402 authenticatorMakeCredential operation. Relying Parties use the
 0403 attestation public key conveyed in the attestation certificate
 0404 to verify the attestation signature. Note that in the case of
 0405 self attestation, the authenticator has no distinct attestation
 0406 key pair nor attestation certificate, see self attestation for
 0407 details.
 0408
 0409 **Authentication**
 0410 The ceremony where a user, and the user's computing device(s)
 0411 (containing at least one authenticator) work in concert to
 0412 cryptographically prove to an Relying Party that the user
 0413 controls the credential private key associated with a
 0414 previously-registered public key credential (see Registration).
 0415 Note that this typically includes employing a test of user
 0416 presence or user verification.

0350 This specification describes the syntax of all CBOR-encoded data
 0351 using the CBOR Data Definition Language (CDDL) [CDDL].
 0352
 0353 **COSE**
 0354 CBOR Object Signing and Encryption (COSE) [RFC8152]. The IANA
 0355 COSE Algorithms registry established by this specification is
 0356 also used.
 0357
 0358 **Credential Management**
 0359 The API described in this document is an extension of the
 0360 Credential concept defined in [CREDENTIAL-MANAGEMENT-1].
 0361
 0362 **DOM**
 0363 DOMException and the DOMException values used in this
 0364 specification are defined in [DOM4].
 0365
 0366 **ECMAScript**
 0367 %ArrayBuffer% is defined in [ECMAScript].
 0368
 0369 **HTML**
 0370 The concepts of relevant settings object, origin, opaque origin,
 0371 and is a registrable domain suffix of or is equal to are defined
 0372 in [HTML52].
 0373
 0374 **Web IDL**
 0375 Many of the interface definitions and all of the IDL in this
 0376 specification depend on [WebIDL-1]. This updated version of the
 0377 Web IDL standard adds support for Promises, which are now the
 0378 preferred mechanism for asynchronous interaction in all new web
 0379 APIs.
 0380
 0381 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
 0382 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
 0383 document are to be interpreted as described in [RFC2119].
 0384
 0385 **3. Terminology**
 0386
 0387 **Assertion**
 0388 See Authentication Assertion.
 0389
 0390 **Attestation**
 0391 Generally, attestation is a statement serving to bear witness,
 0392 confirm, or authenticate. In the WebAuthn context, attestation
 0393 is employed to attest to the provenance of an authenticator and
 0394 the data it emits; including, for example: credential IDs,
 0395 credential key pairs, signature counters, etc. An attestation
 0396 statement is conveyed in an attestation object during
 0397 registration. See also 5.3 Attestation and Figure 3.
 0398
 0399 **Attestation Certificate**
 0400 A X.509 Certificate for the attestation key pair used by an
 0401 authenticator to attest to its manufacture and capabilities. At
 0402 registration time, the authenticator uses the attestation
 0403 private key to sign the Relying Party-specific credential public
 0404 key (and additional data) that it generates and returns via the
 0405 authenticatorMakeCredential operation. Relying Parties use the
 0406 attestation public key conveyed in the attestation certificate
 0407 to verify the attestation signature. Note that in the case of
 0408 self attestation, the authenticator has no distinct attestation
 0409 key pair nor attestation certificate, see self attestation for
 0410 details.
 0411
 0412 **Authentication**
 0413 The ceremony where a user, and the user's computing device(s)
 0414 (containing at least one authenticator) work in concert to
 0415 cryptographically prove to an Relying Party that the user
 0416 controls the credential private key associated with a
 0417 previously-registered public key credential (see Registration).
 0418 Note that this typically includes employing a test of user
 0419 presence or user verification.

0417
 0418
 0419
 0420
 0421
 0422
 0423
 0424
 0425
 0426
 0427
 0428
 0429
 0430
 0431
 0432
 0433
 0434
 0435
 0436
 0437
 0438
 0439
 0440
 0441
 0442
 0443
 0444
 0445
 0446
 0447
 0448
 0449
 0450
 0451
 0452
 0453
 0454
 0455
 0456
 0457
 0458
 0459
 0460
 0461
 0462
 0463
 0464
 0465
 0466
 0467
 0468
 0469
 0470
 0471
 0472
 0473
 0474
 0475
 0476
 0477
 0478
 0479
 0480
 0481
 0482
 0483
 0484
 0485
 0486

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

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

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

Biometric Recognition
 The automated recognition of individuals based on their biological and behavioral characteristics [ISOBiometricVocabulary].

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

Client
 See Conforming User Agent.

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

Client-side-resident Credential Private Key
 A Client-side-resident Credential Private Key is stored either on the client platform, or in some cases on the authenticator itself, e.g., in the case of a discrete first-factor roaming authenticator. Such client-side credential private key storage has the property that the authenticator is able to select the 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 Public Key
 The public key portion of an Relying Party-specific credential

0420
 0421
 0422
 0423
 0424
 0425
 0426
 0427
 0428
 0429
 0430
 0431
 0432
 0433
 0434
 0435
 0436
 0437
 0438
 0439
 0440
 0441
 0442
 0443
 0444
 0445
 0446
 0447
 0448
 0449
 0450
 0451
 0452
 0453
 0454
 0455
 0456
 0457
 0458
 0459
 0460
 0461
 0462
 0463
 0464
 0465
 0466
 0467
 0468
 0469
 0470
 0471
 0472
 0473
 0474
 0475
 0476
 0477
 0478
 0479
 0480
 0481
 0482
 0483
 0484
 0485
 0486
 0487
 0488
 0489

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

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

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

Biometric Recognition
 The automated recognition of individuals based on their biological and behavioral characteristics [ISOBiometricVocabulary].

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

Client
 See Conforming User Agent.

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

Client-side-resident Credential Private Key
 A Client-side-resident Credential Private Key is stored either on the client platform, or in some cases on the authenticator itself, e.g., in the case of a discrete first-factor roaming authenticator. Such client-side credential private key storage has the property that the authenticator is able to select the 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 Public Key
 The public key portion of an Relying Party-specific credential

0487 key pair, generated by an authenticator and returned to an
 0488 Relying Party at registration time (see also public key
 0489 credential). The private key portion of the credential key pair
 0490 is known as the credential private key. Note that in the case of
 0491 self attestation, the credential key pair is also used as the
 0492 attestation key pair, see self attestation for details.

0493 **Rate Limiting**

0494 The process (also known as throttling) by which an authenticator
 0495 implements controls against brute force attacks by limiting the
 0496 number of consecutive failed authentication attempts within a
 0497 given period of time. If the limit is reached, the authenticator
 0498 should impose a delay that increases exponentially with each
 0499 successive attempt, or disable the current authentication
 0500 modality and offer a different authentication factor if
 0501 available. Rate limiting is often implemented as an aspect of
 0502 user verification.

0503 **Registration**

0504 The ceremony where a user, a Relying Party, and the user's
 0505 computing device(s) (containing at least one authenticator) work
 0506 in concert to create a public key credential and associate it
 0507 with the user's Relying Party account. Note that this typically
 0508 includes employing a test of user presence or user verification.

0509 **Relying Party**

0510 The entity whose web application utilizes the Web Authentication
 0511 API to register and authenticate users. See Registration and
 0512 Authentication, respectively.

0513 Note: While the term Relying Party is used in other contexts
 0514 (e.g., X.509 and OAuth), an entity acting as a Relying Party in
 0515 one context is not necessarily a Relying Party in other
 0516 contexts.

0517 **Relying Party Identifier**

0518 **RP ID**

0519 A valid domain string that identifies the Relying Party on whose
 0520 behalf a given registration or authentication ceremony is being
 0521 performed. A public key credential can only be used for
 0522 authentication with the same entity (as identified by RP ID) it
 0523 was registered with. By default, the RP ID for a WebAuthn
 0524 operation is set to the caller's origin's effective domain. This
 0525 default MAY be overridden by the caller, as long as the
 0526 caller-specified RP ID value is a registrable domain suffix of
 0527 or is equal to the caller's origin's effective domain. See also
 0528 4.1.3 Create a new credential - PublicKeyCredential's
 0529 [[Create]](options) method and 4.1.4 Use an existing credential
 0530 to make an assertion - PublicKeyCredential's
 0531 [[DiscoverFromExternalSource]](options) method.

0532 Note: A Public key credential's scope is for a Relying Party's
 0533 origin, with the following restrictions and relaxations:

- 0534 + The scheme is always https (i.e., a restriction), and,
- 0535 + the host may be equal to the Relying Party's origin's
 0536 effective domain, or it may be equal to a registrable domain
 0537 suffix of the Relying Party's origin's effective domain (i.e.,
 0538 an available relaxation), and,
- 0539 + all (TCP) ports on that host (i.e., a relaxation).

0540 This is done in order to match the behavior of pervasively
 0541 deployed ambient credentials (e.g., cookies, [RFC6265]). Please
 0542 note that this is a greater relaxation of "same-origin"
 0543 restrictions than what document.domain's setter provides.

0544 **Public Key Credential**

0545 Generically, a credential is data one entity presents to another
 0546 in order to authenticate the former to the latter [RFC4949]. A
 0547 WebAuthn public key credential is a { identifier, type } pair

0490 key pair, generated by an authenticator and returned to an
 0491 Relying Party at registration time (see also public key
 0492 credential). The private key portion of the credential key pair
 0493 is known as the credential private key. Note that in the case of
 0494 self attestation, the credential key pair is also used as the
 0495 attestation key pair, see self attestation for details.

0496 **Rate Limiting**

0497 The process (also known as throttling) by which an authenticator
 0498 implements controls against brute force attacks by limiting the
 0499 number of consecutive failed authentication attempts within a
 0500 given period of time. If the limit is reached, the authenticator
 0501 should impose a delay that increases exponentially with each
 0502 successive attempt, or disable the current authentication
 0503 modality and offer a different authentication factor if
 0504 available. Rate limiting is often implemented as an aspect of
 0505 user verification.

0506 **Registration**

0507 The ceremony where a user, a Relying Party, and the user's
 0508 computing device(s) (containing at least one authenticator) work
 0509 in concert to create a public key credential and associate it
 0510 with the user's Relying Party account. Note that this typically
 0511 includes employing a test of user presence or user verification.

0512 **Relying Party**

0513 The entity whose web application utilizes the Web Authentication
 0514 API to register and authenticate users. See Registration and
 0515 Authentication, respectively.

0516 Note: While the term Relying Party is used in other contexts
 0517 (e.g., X.509 and OAuth), an entity acting as a Relying Party in
 0518 one context is not necessarily a Relying Party in other
 0519 contexts.

0520 **Relying Party Identifier**

0521 **RP ID**

0522 A valid domain string that identifies the Relying Party on whose
 0523 behalf a given registration or authentication ceremony is being
 0524 performed. A public key credential can only be used for
 0525 authentication with the same entity (as identified by RP ID) it
 0526 was registered with. By default, the RP ID for a WebAuthn
 0527 operation is set to the caller's origin's effective domain. This
 0528 default MAY be overridden by the caller, as long as the
 0529 caller-specified RP ID value is a registrable domain suffix of
 0530 or is equal to the caller's origin's effective domain. See also
 0531 4.1.3 Create a new credential - PublicKeyCredential's
 0532 [[Create]](options) method and 4.1.4 Use an existing credential
 0533 to make an assertion - PublicKeyCredential's
 0534 [[DiscoverFromExternalSource]](options) method.

0535 Note: A Public key credential's scope is for a Relying Party's
 0536 origin, with the following restrictions and relaxations:

- 0537 + The scheme is always https (i.e., a restriction), and,
- 0538 + the host may be equal to the Relying Party's origin's
 0539 effective domain, or it may be equal to a registrable domain
 0540 suffix of the Relying Party's origin's effective domain (i.e.,
 0541 an available relaxation), and,
- 0542 + all (TCP) ports on that host (i.e., a relaxation).

0543 This is done in order to match the behavior of pervasively
 0544 deployed ambient credentials (e.g., cookies, [RFC6265]). Please
 0545 note that this is a greater relaxation of "same-origin"
 0546 restrictions than what document.domain's setter provides.

0547 **Public Key Credential**

0548 Generically, a credential is data one entity presents to another
 0549 in order to authenticate the former to the latter [RFC4949]. A
 0550 WebAuthn public key credential is a { identifier, type } pair

0557 identifying authentication information established by the
0558 authenticator and the Relying Party, together, at registration
0559 time. The authentication information consists of an asymmetric
0560 key pair, where the public key portion is returned to the
0561 Relying Party, who then stores it in conjunction with the
0562 present user's account. The authenticator maps the private key
0563 portion to the Relying Party's RP ID and stores it.
0564 Subsequently, only that Relying Party, as identified by its RP
0565 ID, is able to employ the public key credential in
0566 authentication ceremonies, via the get() method. The Relying
0567 Party uses its stored copy of the credential public key to
0568 verify the resultant authentication assertion.

0570 **Test of User Presence**
0571 A test of user presence is a simple form of authorization
0572 gesture and technical process where a user interacts with an
0573 authenticator by (typically) simply touching it (other
0574 modalities may also exist), yielding a boolean result. Note that
0575 this does not constitute user verification because a user
0576 presence test, by definition, is not capable of biometric
0577 recognition, nor does it involve the presentation of a shared
0578 secret such as a password or PIN.

0580 **User Consent**
0581 User consent means the user agrees with what they are being
0582 asked, i.e., it encompasses reading and understanding prompts.
0583 An authorization gesture is a ceremony component often employed
0584 to indicate user consent.

0586 **User Verification**
0587 The technical process by which an authenticator locally
0588 authorizes the invocation of the authenticatorMakeCredential and
0589 authenticatorGetAssertion operations. User verification may be
0590 instigated through various authorization gesture modalities; for
0591 example, through a touch plus pin code, password entry, or
0592 biometric recognition (e.g., presenting a fingerprint)
0593 [ISOBiometricVocabulary]. The intent is to be able to
0594 distinguish individual users. Note that invocation of the
0595 authenticatorMakeCredential and authenticatorGetAssertion
0596 operations implies use of key material managed by the
0597 authenticator. Note that for security, user verification and use
0598 of credential private keys must occur within a single logical
0599 security boundary defining the authenticator.

0601 **User Present**
0602 **UP**
0603 Upon successful completion of a user presence test, the user is
0604 said to be "present".

0606 **User Verified**
0607 **UV**
0608 Upon successful completion of a user verification process, the
0609 user is said to be "verified".

0611 **WebAuthn Client**
0612 Also referred to herein as simply a client. See also Conforming
0613 User Agent.

0615 **4. Web Authentication API**
0616
0617 This section normatively specifies the API for creating and using
0618 public key credentials. The basic idea is that the credentials belong
0619 to the user and are managed by an authenticator, with which the Relying
0620 Party interacts through the client (consisting of the browser and
0621 underlying OS platform). Scripts can (with the user's consent) request
0622 the browser to create a new credential for future use by the Relying
0623 Party. Scripts can also request the user's permission to perform
0624 authentication operations with an existing credential. All such
0625 operations are performed in the authenticator and are mediated by the
0626 browser and/or platform on the user's behalf. At no point does the

0560 identifying authentication information established by the
0561 authenticator and the Relying Party, together, at registration
0562 time. The authentication information consists of an asymmetric
0563 key pair, where the public key portion is returned to the
0564 Relying Party, who then stores it in conjunction with the
0565 present user's account. The authenticator maps the private key
0566 portion to the Relying Party's RP ID and stores it.
0567 Subsequently, only that Relying Party, as identified by its RP
0568 ID, is able to employ the public key credential in
0569 authentication ceremonies, via the get() method. The Relying
0570 Party uses its stored copy of the credential public key to
0571 verify the resultant authentication assertion.

0573 **Test of User Presence**
0574 A test of user presence is a simple form of authorization
0575 gesture and technical process where a user interacts with an
0576 authenticator by (typically) simply touching it (other
0577 modalities may also exist), yielding a boolean result. Note that
0578 this does not constitute user verification because a user
0579 presence test, by definition, is not capable of biometric
0580 recognition, nor does it involve the presentation of a shared
0581 secret such as a password or PIN.

0583 **User Consent**
0584 User consent means the user agrees with what they are being
0585 asked, i.e., it encompasses reading and understanding prompts.
0586 An authorization gesture is a ceremony component often employed
0587 to indicate user consent.

0589 **User Verification**
0590 The technical process by which an authenticator locally
0591 authorizes the invocation of the authenticatorMakeCredential and
0592 authenticatorGetAssertion operations. User verification may be
0593 instigated through various authorization gesture modalities; for
0594 example, through a touch plus pin code, password entry, or
0595 biometric recognition (e.g., presenting a fingerprint)
0596 [ISOBiometricVocabulary]. The intent is to be able to
0597 distinguish individual users. Note that invocation of the
0598 authenticatorMakeCredential and authenticatorGetAssertion
0599 operations implies use of key material managed by the
0600 authenticator. Note that for security, user verification and use
0601 of credential private keys must occur within a single logical
0602 security boundary defining the authenticator.

0604 **User Present**
0605 **UP**
0606 Upon successful completion of a user presence test, the user is
0607 said to be "present".

0609 **User Verified**
0610 **UV**
0611 Upon successful completion of a user verification process, the
0612 user is said to be "verified".

0614 **WebAuthn Client**
0615 Also referred to herein as simply a client. See also Conforming
0616 User Agent.

0618 **4. Web Authentication API**
0619
0620 This section normatively specifies the API for creating and using
0621 public key credentials. The basic idea is that the credentials belong
0622 to the user and are managed by an authenticator, with which the Relying
0623 Party interacts through the client (consisting of the browser and
0624 underlying OS platform). Scripts can (with the user's consent) request
0625 the browser to create a new credential for future use by the Relying
0626 Party. Scripts can also request the user's permission to perform
0627 authentication operations with an existing credential. All such
0628 operations are performed in the authenticator and are mediated by the
0629 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 5.2 Authenticator operations, the full origin of the requester is included, and signed over, in the attestation object produced when a new credential is created as well as in all assertions produced by WebAuthn credentials.

Additionally, to maintain user privacy and prevent malicious Relying Parties from probing for the presence of public key credentials belonging to other Relying Parties, each credential is also associated with a Relying Party Identifier, or RP ID. This RP ID is provided 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.

4.1. PublicKeyCredential Interface

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

```
[SecureContext]
interface PublicKeyCredential : Credential {
  [SameObject] readonly attribute ArrayBuffer rawId;
  [SameObject] readonly attribute AuthenticatorResponse response;
  [SameObject] readonly attribute AuthenticationExtensions clientExtensionResu
};

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
```

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 5.2 Authenticator operations, the full origin of the requester is included, and signed over, in the attestation object produced when a new credential is created as well as in all assertions produced by WebAuthn credentials.

Additionally, to maintain user privacy and prevent malicious Relying Parties from probing for the presence of public key credentials belonging to other Relying Parties, each credential is also associated with a Relying Party Identifier, or RP ID. This RP ID is provided 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.

4.1. PublicKeyCredential Interface

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

```
[SecureContext]
interface PublicKeyCredential : Credential {
  [SameObject] readonly attribute ArrayBuffer rawId;
  [SameObject] readonly attribute AuthenticatorResponse response;
  [SameObject] readonly attribute AuthenticationExtensions clientExtensionResu
};

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
```

0697 generate an authentication assertion. If the `PublicKeyCredential`
0698 is created in response to `create()`, this attribute's value will
0699 be an `AuthenticatorAttestationResponse`, otherwise, the
0700 `PublicKeyCredential` was created in response to `get()`, and this
0701 attribute's value will be an `AuthenticatorAssertionResponse`.
0702

0703 `clientExtensionResults`, of type `AuthenticationExtensions`, readonly
0704 This attribute contains a map containing extension identifier ->
0705 client extension output entries produced by the extension's
0706 client extension processing.
0707

0708 **[[type]]**
0709 The `PublicKeyCredential` interface object's **[[type]]** internal
0710 slot's value is the string "public-key".
0711

0712 Note: This is reflected via the type attribute getter inherited
0713 from `Credential`.
0714

0715 **[[discovery]]**
0716 The `PublicKeyCredential` interface object's **[[discovery]]**
0717 internal slot's value is "remote".
0718

0719 **[[identifier]]**
0720 This internal slot contains an identifier for the credential,
0721 chosen by the platform with help from the authenticator. This
0722 identifier is used to look up credentials for use, and is
0723 therefore expected to be globally unique with high probability
0724 across all credentials of the same type, across all
0725 authenticators. This API does not constrain the format or length
0726 of this identifier, except that it must be sufficient for the
0727 platform to uniquely select a key. For example, an authenticator
0728 without on-board storage may create identifiers containing a
0729 credential private key wrapped with a symmetric key that is
0730 burned into the authenticator.
0731

0732 `PublicKeyCredential`'s interface object inherits `Credential`'s
0733 implementation of **[[CollectFromCredentialStore]](options)** and
0734 **[[Store]](credential)**, and defines its own implementation of
0735 **[[DiscoverFromExternalSource]](options)** and **[[Create]](options)**.
0736

0737 4.1.1. `CredentialCreationOptions` Extension
0738

0739 To support registration via `navigator.credentials.create()`, this
0740 document extends the `CredentialCreationOptions` dictionary as follows:
0741 partial dictionary `CredentialCreationOptions` {
0742 `MakePublicKeyCredentialOptions` `publicKey`;
0743 };
0744

0745 4.1.2. `CredentialRequestOptions` Extension
0746

0747 To support obtaining assertions via `navigator.credentials.get()`, this
0748 document extends the `CredentialRequestOptions` dictionary as follows:
0749 partial dictionary `CredentialRequestOptions` {
0750 `PublicKeyCredentialRequestOptions` `publicKey`;
0751 };
0752

0753 4.1.3. Create a new credential - `PublicKeyCredential`'s **[[Create]](options)**
0754 method
0755

0756 `PublicKeyCredential`'s interface object's implementation of the
0757 **[[Create]](options)** method allows scripts to call
0758 `navigator.credentials.create()` to request the creation of a new
0759 credential key pair and `PublicKeyCredential`, managed by an
0760 authenticator. On success, the returned promise will be resolved with a
0761 `PublicKeyCredential` containing an `AuthenticatorAttestationResponse`
0762 object.
0763

0764 Note: This algorithm is synchronous; the Promise resolution/rejection
0765 is handled by `navigator.credentials.create()`.
0766

0700 generate an authentication assertion. If the `PublicKeyCredential`
0701 is created in response to `create()`, this attribute's value will
0702 be an `AuthenticatorAttestationResponse`, otherwise, the
0703 `PublicKeyCredential` was created in response to `get()`, and this
0704 attribute's value will be an `AuthenticatorAssertionResponse`.
0705

0706 `clientExtensionResults`, of type `AuthenticationExtensions`, readonly
0707 This attribute contains a map containing extension identifier ->
0708 client extension output entries produced by the extension's
0709 client extension processing.
0710

0711 **[[type]]**
0712 The `PublicKeyCredential` interface object's **[[type]]** internal
0713 slot's value is the string "public-key".
0714

0715 Note: This is reflected via the type attribute getter inherited
0716 from `Credential`.
0717

0718 **[[discovery]]**
0719 The `PublicKeyCredential` interface object's **[[discovery]]**
0720 internal slot's value is "remote".
0721

0722 **[[identifier]]**
0723 This internal slot contains an identifier for the credential,
0724 chosen by the platform with help from the authenticator. This
0725 identifier is used to look up credentials for use, and is
0726 therefore expected to be globally unique with high probability
0727 across all credentials of the same type, across all
0728 authenticators. This API does not constrain the format or length
0729 of this identifier, except that it must be sufficient for the
0730 platform to uniquely select a key. For example, an authenticator
0731 without on-board storage may create identifiers containing a
0732 credential private key wrapped with a symmetric key that is
0733 burned into the authenticator.
0734

0735 `PublicKeyCredential`'s interface object inherits `Credential`'s
0736 implementation of **[[CollectFromCredentialStore]](options)** and
0737 **[[Store]](credential)**, and defines its own implementation of
0738 **[[DiscoverFromExternalSource]](options)** and **[[Create]](options)**.
0739

0740 4.1.1. `CredentialCreationOptions` Extension
0741

0742 To support registration via `navigator.credentials.create()`, this
0743 document extends the `CredentialCreationOptions` dictionary as follows:
0744 partial dictionary `CredentialCreationOptions` {
0745 `MakePublicKeyCredentialOptions` `publicKey`;
0746 };
0747

0748 4.1.2. `CredentialRequestOptions` Extension
0749

0750 To support obtaining assertions via `navigator.credentials.get()`, this
0751 document extends the `CredentialRequestOptions` dictionary as follows:
0752 partial dictionary `CredentialRequestOptions` {
0753 `PublicKeyCredentialRequestOptions` `publicKey`;
0754 };
0755

0756 4.1.3. Create a new credential - `PublicKeyCredential`'s **[[Create]](options)**
0757 method
0758

0759 `PublicKeyCredential`'s interface object's implementation of the
0760 **[[Create]](options)** method allows scripts to call
0761 `navigator.credentials.create()` to request the creation of a new
0762 credential key pair and `PublicKeyCredential`, managed by an
0763 authenticator. On success, the returned promise will be resolved with a
0764 `PublicKeyCredential` containing an `AuthenticatorAttestationResponse`
0765 object.
0766

0767 Note: This algorithm is synchronous; the Promise resolution/rejection
0768 is handled by `navigator.credentials.create()`.
0769

This method accepts a single argument:

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

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

1. Assert: options.publicKey is present.
2. Let options be the value of options.publicKey.
3. If any of the name member of options.rp, the name member of options.user, the displayName member of options.user, or the id member of options.user are not present, return a TypeError simple exception.
4. If the timeout member of options is present, check if its value lies within a reasonable range as defined by the platform and if not, correct it to the closest value lying within that range. Set adjustedTimeout to this adjusted value. If the timeout member of options is not present, then set adjustedTimeout to a platform-specific default.
5. Let global be the PublicKeyCredential's interface object's environment settings object's global object.
6. Let callerOrigin be the origin specified by this PublicKeyCredential interface object's relevant settings object. If callerOrigin is an opaque origin, return a DOMException whose name is "NotAllowedError", and terminate this algorithm.
7. Let effectiveDomain be the callerOrigin's effective domain. If effective domain is not a valid domain, then return a DOMException whose name is "SecurityError" and terminate this algorithm.
 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.
8. Let rpId be effectiveDomain.
9. If options.rp.id is present:
 1. If options.rp.id is not a registrable domain suffix of and is not equal to effectiveDomain, return a DOMException whose name is "SecurityError", and terminate this algorithm.
 2. Set rpId to options.rp.id.
 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.rp.id when calling create().
10. Let credTypesAndPubKeyAlgs be a new list whose items are pairs of PublicKeyCredentialType and a COSEAlgorithmIdentifier.
11. 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.
12. If credTypesAndPubKeyAlgs is empty and options.pubKeyCredParams is not empty, cancel the timer started in step 2, return a DOMException whose name is "NotSupportedError", and terminate this algorithm.
13. Let clientExtensions be a new map and let authenticatorExtensions be a new map.
14. If the extensions member of options is present, then for each extensionId -> clientExtensionInput of options.extensions:
 1. If extensionId is not supported by this client platform or is not a registration extension, then continue.
 2. Set clientExtensions[extensionId] to clientExtensionInput.
 3. If extensionId is not an authenticator extension, then continue.

This method accepts a single argument:

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

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

1. Assert: options.publicKey is present.
2. Let options be the value of options.publicKey.
3. If any of the name member of options.rp, the name member of options.user, the displayName member of options.user, or the id member of options.user are not present, return a TypeError simple exception.
4. If the timeout member of options is present, check if its value lies within a reasonable range as defined by the platform and if not, correct it to the closest value lying within that range. Set adjustedTimeout to this adjusted value. If the timeout member of options is not present, then set adjustedTimeout to a platform-specific default.
5. Let global be the PublicKeyCredential's interface object's environment settings object's global object.
6. Let callerOrigin be the origin specified by this PublicKeyCredential interface object's relevant settings object. If callerOrigin is an opaque origin, return a DOMException whose name is "NotAllowedError", and terminate this algorithm.
7. Let effectiveDomain be the callerOrigin's effective domain. If effective domain is not a valid domain, then return a DOMException whose name is "SecurityError" and terminate this algorithm.
 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.
8. 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().
9. Let credTypesAndPubKeyAlgs be a new list whose items are pairs of PublicKeyCredentialType and a COSEAlgorithmIdentifier.
10. 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.
11. If credTypesAndPubKeyAlgs is empty and options.pubKeyCredParams is not empty, return a DOMException whose name is "NotSupportedError", and terminate this algorithm.
12. Let clientExtensions be a new map and let authenticatorExtensions be a new map.
13. 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.

0832 4. Let authenticatorExtensionInput be the (CBOR) result of
0833 running extensionId's client extension processing algorithm on
0834 clientExtensionInput. If the algorithm returned an error,
0835 continue.
0836 5. Set authenticatorExtensions[extensionId] to the base64url
0837 encoding of authenticatorExtensionInput.
0838 15. Let collectedClientData be a new CollectedClientData instance whose
0839 fields are:
0840
0841 challenge
0842 The base64url encoding of options.challenge.
0843
0844 origin
0845 The serialization of callerOrigin.
0846
0847 hashAlgorithm
0848 The recognized algorithm name of the hash algorithm
0849 selected by the client for generating the hash of the
0850 serialized client data.
0851
0852 tokenBindingId
0853 The Token Binding ID associated with callerOrigin, if one
0854 is available.
0855
0856 clientExtensions
0857 clientExtensions
0858
0859 authenticatorExtensions
0860 authenticatorExtensions
0861
0862 16. Let clientDataJSON be the JSON-serialized client data constructed
0863 from collectedClientData.
0864 17. Let clientDataHash be the hash of the serialized client data
0865 represented by clientDataJSON.
0866 18. Let currentlyAvailableAuthenticators be a new ordered set
0867 consisting of all authenticators currently available on this
0868 platform.
0869 19. Let selectedAuthenticators be a new ordered set.
0870 20. If currentlyAvailableAuthenticators is empty, return a DOMException
0871 whose name is "NotFoundError", and terminate this algorithm.
0872 21. If options.authenticatorSelection is present, iterate through
0873 currentlyAvailableAuthenticators and do the following for each
0874 authenticator:
0875 1. If authenticatorAttachment is present and its value is not
0876 equal to authenticator's attachment modality, continue.
0877 2. If requireResidentKey is set to true and the authenticator is
0878 not capable of storing a Client-Side-Resident Credential
0879 Private Key, continue.
0880 3. If requireUserVerification is set to true and the
0881 authenticator is not capable of performing user verification,
0882 continue.
0883 4. Append authenticator to selectedAuthenticators.
0884 22. If selectedAuthenticators is empty, return a DOMException whose
0885 name is "ConstraintError", and terminate this algorithm.
0886 23. Let issuedRequests be a new ordered set.
0887 24. For each authenticator in currentlyAvailableAuthenticators:
0888 1. Let excludeCredentialDescriptorList be a new list.
0889 2. For each credential descriptor C in
0890 options.excludeCredentials:
0891 1. If C.transports is not empty, and authenticator is
0892 connected over a transport not mentioned in C.transports,
0893 the client MAY continue.
0894 2. Otherwise, Append C to excludeCredentialDescriptorList.
0895 3. In parallel, invoke the authenticatorMakeCredential operation
0896 on authenticator with rpId, clientDataHash, options.rp,
0897 options.user,
0898 options.authenticatorSelection.requireResidentKey,
0899 credTypesAndPubKeyAlgs, excludeCredentialDescriptorList, and
0900 authenticatorExtensions as parameters.
0901 4. Append authenticator to issuedRequests.

0839 4. Let authenticatorExtensionInput be the (CBOR) result of
0840 running extensionId's client extension processing algorithm on
0841 clientExtensionInput. If the algorithm returned an error,
0842 continue.
0843 5. Set authenticatorExtensions[extensionId] to the base64url
0844 encoding of authenticatorExtensionInput.
0845 14. Let collectedClientData be a new CollectedClientData instance whose
0846 fields are:
0847
0848 challenge
0849 The base64url encoding of options.challenge.
0850
0851 origin
0852 The serialization of callerOrigin.
0853
0854 hashAlgorithm
0855 The recognized algorithm name of the hash algorithm
0856 selected by the client for generating the hash of the
0857 serialized client data.
0858
0859 tokenBindingId
0860 The Token Binding ID associated with callerOrigin, if one
0861 is available.
0862
0863 clientExtensions
0864 clientExtensions
0865
0866 authenticatorExtensions
0867 authenticatorExtensions
0868
0869 15. Let clientDataJSON be the JSON-serialized client data constructed
0870 from collectedClientData.
0871 16. Let clientDataHash be the hash of the serialized client data
0872 represented by clientDataJSON.
0873 17. Let currentlyAvailableAuthenticators be a new ordered set
0874 consisting of all authenticators currently available on this
0875 platform.
0876 18. Let selectedAuthenticators be a new ordered set.
0877 19. If currentlyAvailableAuthenticators is empty, return a DOMException
0878 whose name is "NotFoundError", and terminate this algorithm.
0879 20. If options.authenticatorSelection is present, iterate through
0880 currentlyAvailableAuthenticators and do the following for each
0881 authenticator:
0882 1. If authenticatorAttachment is present and its value is not
0883 equal to authenticator's attachment modality, continue.
0884 2. If requireResidentKey is set to true and the authenticator is
0885 not capable of storing a Client-Side-Resident Credential
0886 Private Key, continue.
0887 3. If requireUserVerification is set to true and the
0888 authenticator is not capable of performing user verification,
0889 continue.
0890 4. Append authenticator to selectedAuthenticators.
0891 21. If selectedAuthenticators is empty, return a DOMException whose
0892 name is "ConstraintError", and terminate this algorithm.
0893 22. Let issuedRequests be a new ordered set.
0894 23. For each authenticator in currentlyAvailableAuthenticators:
0895 1. Let excludeCredentialDescriptorList be a new list.
0896 2. For each credential descriptor C in
0897 options.excludeCredentials:
0898 1. If C.transports is not empty, and authenticator is
0899 connected over a transport not mentioned in C.transports,
0900 the client MAY continue.
0901 2. Otherwise, Append C to excludeCredentialDescriptorList.
0902 3. Invoke the authenticatorMakeCredential operation on
0903 authenticator with clientDataHash, options.rp, options.user,
0904 options.authenticatorSelection.rk, credTypesAndPubKeyAlgs,
0905 excludeCredentialDescriptorList, and authenticatorExtensions
0906 as parameters.
0907 4. Append authenticator to issuedRequests.

0902 25. Start a timer for adjustedTimeout milliseconds. Then execute the
 0903 following steps in parallel. The task source for these tasks is the
 0904 dom manipulation task source.
 0905 26. While issuedRequests is not empty, perform the following actions
 0906 depending upon the adjustedTimeout timer and responses from the
 0907 authenticators:

0908 If the adjustedTimeout timer expires,
 0909 For each authenticator in issuedRequests invoke the
 0910 authenticatorCancel operation on authenticator and remove
 0911 authenticator from issuedRequests.
 0912
 0913 If any authenticator returns a status indicating that the user
 0914 cancelled the operation,
 0915
 0916 1. Remove authenticator from issuedRequests.
 0917 2. For each remaining authenticator in issuedRequests invoke
 0918 the authenticatorCancel operation on authenticator and
 0919 remove it from issuedRequests.
 0920
 0921 If any authenticator returns an error status,
 0922 Remove authenticator from issuedRequests.
 0923
 0924 If any authenticator indicates success,
 0925
 0926 1. Remove authenticator from issuedRequests.
 0927 2. Let attestationObject be a new ArrayBuffer, created using
 0928 global's %ArrayBuffer%, containing the bytes of the value
 0929 returned from the successful authenticatorMakeCredential
 0930 operation (which is attObj, as defined in 5.3.4
 0931 Generating an Attestation Object).
 0932 3. Let id be attestationObject.authData.attestation
 0933 data.credential ID (see 5.3.1 Attestation data and 5.1
 0934 Authenticator data).
 0935 4. Let value be a new PublicKeyCredential object associated
 0936 with global whose fields are:
 0937

0938 [[identifier]]
 0939 id
 0940
 0941 response
 0942 A new AuthenticatorAttestationResponse object
 0943 associated with global whose fields are:
 0944

0945 clientDataJSON
 0946 A new ArrayBuffer, created using
 0947 global's %ArrayBuffer%, containing the
 0948 bytes of clientDataJSON.
 0949
 0950 attestationObject
 0951 attestationObject
 0952
 0953 clientExtensionResults
 0954 A new AuthenticationExtensions object
 0955 containing the extension identifier -> client
 0956 extension output entries created by running
 0957 each extension's client extension processing
 0958
 0959 algorithm to create the client extension
 0960 outputs, for each client extension in
 0961 clientDataJSON.clientExtensions.
 0962
 0963 5. For each remaining authenticator in issuedRequests invoke

0908 24. Start a timer for adjustedTimeout milliseconds.
 0909 25. While issuedRequests is not empty, perform the following actions

0910 depending upon the adjustedTimeout timer and responses from the
 0911 authenticators:
 0912
 0913 If the adjustedTimeout timer expires,
 0914 For each authenticator in issuedRequests invoke the
 0915 authenticatorCancel operation on authenticator and remove
 0916 authenticator from issuedRequests.
 0917
 0918 If any authenticator returns a status indicating that the user
 0919 cancelled the operation,
 0920
 0921 1. Remove authenticator from issuedRequests.
 0922 2. For each remaining authenticator in issuedRequests invoke
 0923 the authenticatorCancel operation on authenticator and
 0924 remove it from issuedRequests.
 0925
 0926 If any authenticator returns an error status,
 0927 Remove authenticator from issuedRequests.
 0928
 0929 If any authenticator indicates success,
 0930
 0931 1. Remove authenticator from issuedRequests.
 0932 2. Let credentialCreationData be a struct whose items are:

0933
 0934 attestationObjectResult
 0935 whose value is the bytes returned from the
 0936 successful authenticatorMakeCredential
 0937 operation.
 0938
 0939 Note: this value is attObj, as defined in
 0940 5.3.4 Generating an Attestation Object.

0941
 0942 clientDataJSONResult
 0943 whose value is the bytes of clientDataJSON.
 0944
 0945 extensionOutputsMap
 0946 whose value is an ordered map with keys of
 0947 type extension identifier and values of type
 0948 client extension output. extensionOutputsMap's
 0949 entries are created by running each
 0950 extension's client extension processing
 0951 algorithm to create the client extension
 0952 outputs, for each client extension in
 0953 clientDataJSON.clientExtensions.
 0954
 0955 is there any special way to declare credentialCreationData
 0956 such that it properly exists when
 0957 constructCredentialCallback (defined just below) is
 0958 invoked?
 0959
 0960 3. Let value be a struct whose item is:

0964 the authenticatorCancel operation on authenticator and
0965 remove it from issuedRequests.
0966 6. Return value and terminate this algorithm.

0967 27. Return a DOMException whose name is "NotAllowedError".

0968 During the above process, the user agent SHOULD show some UI to the
0970 user to guide them in the process of selecting and authorizing an
0971 authenticator.
0972
0973

0974 4.1.4. Use an existing credential to make an assertion -
0975 PublicKeyCredential's [[DiscoverFromExternalSource]](options) method

0962 constructCredentialCallback
0963 whose value is a Web IDL Function callback
0964 function type value implementing the steps
0965 defined in 4.1.3.1 Construct the credential -
0966 constructCredentialCallback method.
0967

- 0968 4. For each remaining authenticator in issuedRequests invoke
0969 the authenticatorCancel operation on authenticator and
0970 remove it from issuedRequests.
0971 5. Return value and terminate this algorithm.

0972 26. Return a DOMException whose name is "NotAllowedError".

0973 During the above process, the user agent SHOULD show some UI to the
0974 user to guide them in the process of selecting and authorizing an
0975 authenticator.
0976
0977

0978 4.1.3.1. Construct the credential - constructCredentialCallback method

0979 This method's purpose is to map credentialCreationData's items into a
0980 new PublicKeyCredential object associated with the caller's global
0981 object, while running on the main event loop. See also
0982 [CREDENTIAL-MANAGEMENT-1].
0983
0984

0985 This algorithm accepts one argument:

0986 global
0987 Which must be its caller's current settings object's global
0988 object.
0989
0990

0991 When this method is invoked, the user agent MUST execute the following
0992 algorithm:

- 0993 1. Let attestationObject be a new ArrayBuffer, created using global's
0994 %ArrayBuffer%, containing the bytes of
0995 credentialCreationData.attestationObjectResult's value.
0996 2. Let id be attestationObject.authData.attestation data.credential ID
0997 (see 5.3.1 Attestation data and 5.1 Authenticator data).
0998 3. Let pubKeyCred be a new PublicKeyCredential object associated with
0999 global whose fields are:

1000 [[identifier]]
1001 id

1002 response
1003 A new AuthenticatorAttestationResponse object associated
1004 with global whose fields are:

1005 clientDataJSON
1006 A new ArrayBuffer, created using global's
1007 %ArrayBuffer%, containing the bytes of
1008 credentialCreationData.clientDataJSONResult.
1009

1010 attestationObject
1011 attestationObject

1012 clientExtensionResults
1013 A new AuthenticationExtensions object associated with
1014 global containing the bytes of
1015 credentialCreationData.extensionOutputsMap's value.
1016

1017 Note: credentialCreationData.extensionOutputsMap is an
1018 ordered map whose keys are all of type extension
1019 identifiers and values are all of type client extension
1020 output. Thus the latter is implicitly a record type, which
1021 is the AuthenticationExtensions's type.
1022

1023 4. Return pubKeyCred.
1024

1025 4.1.4. Use an existing credential to make an assertion -
1026 PublicKeyCredential's [[DiscoverFromExternalSource]](options) method
1027
1028
1029
1030
1031

0976 The `[[DiscoverFromExternalSource]](options)` method is used to discover
 0977 and use an existing public key credential, with the user's consent. The
 0978 script optionally specifies some criteria to indicate what credentials
 0979 are acceptable to it. The user agent and/or platform locates
 0980 credentials matching the specified criteria, and guides the user to
 0981 pick one that the script will be allowed to use. The user may choose
 0982 not to provide a credential even if one is present, for example to
 0983 maintain privacy.
 0984
 0985 Note: This algorithm is synchronous; the Promise resolution/rejection
 0986 is handled by `navigator.credentials.get()`.
 0987
 0988 This method accepts a single argument:
 0989
 0990 options
 0991 This argument is a `CredentialRequestOptions` object whose
 0992 `options.publicKey` member contains a challenge and additional
 0993 options as described in 4.5 Options for Assertion Generation
 0994 (dictionary `PublicKeyCredentialRequestOptions`). The selected
 0995 authenticator signs the challenge along with other collected
 0996 data in order to produce an assertion. See 5.2.2 The
 0997 `authenticatorGetAssertion` operation.
 0998
 1000 When this method is invoked, the user agent MUST execute the following
 1001 algorithm:
 1002 1. Assert: `options.publicKey` is present.
 1003 2. Let `options` be the value of `options.publicKey`.
 1004 3. If the `timeout` member of `options` is present, check if its value
 1005 lies within a reasonable range as defined by the platform and if
 1006 not, correct it to the closest value lying within that range. Set
 1007 `adjustedTimeout` to this adjusted value. If the `timeout` member of
 1008 `options` is not present, then set `adjustedTimeout` to a
 1009 platform-specific default.
 1010 4. Let `global` be the `PublicKeyCredential`'s interface object's
 1011 `environment settings object`'s global object.
 1012 5. Let `callerOrigin` be the origin specified by this
 1013 `PublicKeyCredential` interface object's relevant settings object. If
 1014 `callerOrigin` is an opaque origin, return a `DOMException` whose name
 1015 is "NotAllowedError", and terminate this algorithm.
 1016 6. Let `effectiveDomain` be the `callerOrigin`'s effective domain. If
 1017 `effectiveDomain` is not a valid domain, then return a `DOMException`
 1018 whose name is "SecurityError" and terminate this algorithm.
 1019 Note: An effective domain may resolve to a host, which can be
 1020 represented in various manners, such as domain, ipv4 address, ipv6
 1021 address, opaque host, or empty host. Only the domain format of host
 1022 is allowed here.
 1023 7. If `options.rpld` is not present, then set `rpld` to `effectiveDomain`.
 1024 Otherwise:
 1025 1. If `options.rpld` is not a registrable domain suffix of and is
 1026 not equal to `effectiveDomain`, return a `DOMException` whose name
 1027 is "SecurityError", and terminate this algorithm.
 1028 2. Set `rpld` to `options.rpld`.
 1029 Note: `rpld` represents the caller's RP ID. The RP ID defaults
 1030 to being the caller's origin's effective domain unless the
 1031 caller has explicitly set `options.rpld` when calling `get()`.
 1032 8. Let `clientExtensions` be a new map and let `authenticatorExtensions`
 1033 be a new map.
 1034 9. If the extensions member of `options` is present, then for each
 1035 `extensionId` -> `clientExtensionInput` of `options.extensions`:
 1036 1. If `extensionId` is not supported by this client platform or is
 1037 not an authentication extension, then continue.
 1038 2. Set `clientExtensions[extensionId]` to `clientExtensionInput`.
 1039 3. If `extensionId` is not an authenticator extension, then
 1040 continue.
 1041 4. Let `authenticatorExtensionInput` be the (CBOR) result of
 1042 running `extensionId`'s client extension processing algorithm on
 1043 `clientExtensionInput`. If the algorithm returned an error,
 1044 continue.
 1045 5. Set `authenticatorExtensions[extensionId]` to the base64url

1032 The `[[DiscoverFromExternalSource]](options)` method is used to discover
 1033 and use an existing public key credential, with the user's consent. The
 1034 script optionally specifies some criteria to indicate what credentials
 1035 are acceptable to it. The user agent and/or platform locates
 1036 credentials matching the specified criteria, and guides the user to
 1037 pick one that the script will be allowed to use. The user may choose
 1038 not to provide a credential even if one is present, for example to
 1039 maintain privacy.
 1040
 1041 Note: This algorithm is synchronous; the Promise resolution/rejection
 1042 is handled by `navigator.credentials.get()`.
 1043
 1044 This method accepts a single argument:
 1045
 1046 options
 1047 This argument is a `CredentialRequestOptions` object whose
 1048 `options.publicKey` member contains a challenge and additional
 1049 options as described in 4.5 Options for Assertion Generation
 1050 (dictionary `PublicKeyCredentialRequestOptions`). The selected
 1051 authenticator signs the challenge along with other collected
 1052 data in order to produce an assertion. See 5.2.2 The
 1053 `authenticatorGetAssertion` operation.
 1054
 1055 When this method is invoked, the user agent MUST execute the following
 1056 algorithm:
 1057 1. Assert: `options.publicKey` is present.
 1058 2. Let `options` be the value of `options.publicKey`.
 1059 3. If the `timeout` member of `options` is present, check if its value
 1060 lies within a reasonable range as defined by the platform and if
 1061 not, correct it to the closest value lying within that range. Set
 1062 `adjustedTimeout` to this adjusted value. If the `timeout` member of
 1063 `options` is not present, then set `adjustedTimeout` to a
 1064 platform-specific default.
 1065 4. Let `global` be the `PublicKeyCredential`'s interface object's **relevant**
 1066 `global object`.
 1067 5. Let `callerOrigin` be the origin specified by this
 1068 `PublicKeyCredential` interface object's relevant settings object. If
 1069 `callerOrigin` is an opaque origin, return a `DOMException` whose name
 1070 is "NotAllowedError", and terminate this algorithm.
 1071 6. Let `effectiveDomain` be the `callerOrigin`'s effective domain. If
 1072 `effectiveDomain` is not a valid domain, then return a `DOMException`
 1073 whose name is "SecurityError" and terminate this algorithm.
 1074 Note: An effective domain may resolve to a host, which can be
 1075 represented in various manners, such as domain, ipv4 address, ipv6
 1076 address, opaque host, or empty host. Only the domain format of host
 1077 is allowed here.
 1078 7. If `options.rpld` is not present, then set `rpld` to `effectiveDomain`.
 1079 Otherwise:
 1080 1. If `options.rpld` is not a registrable domain suffix of and is
 1081 not equal to `effectiveDomain`, return a `DOMException` whose name
 1082 is "SecurityError", and terminate this algorithm.
 1083 2. Set `rpld` to `options.rpld`.
 1084 Note: `rpld` represents the caller's RP ID. The RP ID defaults
 1085 to being the caller's origin's effective domain unless the
 1086 caller has explicitly set `options.rpld` when calling `get()`.
 1087 8. Let `clientExtensions` be a new map and let `authenticatorExtensions`
 1088 be a new map.
 1089 9. If the extensions member of `options` is present, then for each
 1090 `extensionId` -> `clientExtensionInput` of `options.extensions`:
 1091 1. If `extensionId` is not supported by this client platform or is
 1092 not an authentication extension, then continue.
 1093 2. Set `clientExtensions[extensionId]` to `clientExtensionInput`.
 1094 3. If `extensionId` is not an authenticator extension, then
 1095 continue.
 1096 4. Let `authenticatorExtensionInput` be the (CBOR) result of
 1097 running `extensionId`'s client extension processing algorithm on
 1098 `clientExtensionInput`. If the algorithm returned an error,
 1099 continue.
 1100 5. Set `authenticatorExtensions[extensionId]` to the base64url

1046 encoding of authenticatorExtensionInput.
1047 10. Let collectedClientData be a new CollectedClientData instance whose
1048 fields are:
1049
1050 challenge
1051 The base64url encoding of options.challenge
1052
1053 origin
1054 The serialization of callerOrigin.
1055
1056 hashAlgorithm
1057 The recognized algorithm name of the hash algorithm
1058 selected by the client for generating the hash of the
1059 serialized client data
1060
1061 tokenBindingId
1062 The Token Binding ID associated with callerOrigin, if one
1063 is available.
1064
1065 clientExtensions
1066 clientExtensions
1067
1068 authenticatorExtensions
1069 authenticatorExtensions
1070
1071 11. Let clientDataJSON be the JSON-serialized client data constructed
1072 from collectedClientData.
1073 12. Let clientDataHash be the hash of the serialized client data
1074 represented by clientDataJSON.
1075 13. Let issuedRequests be a new ordered set.
1076 14. If there are no authenticators currently available on this
1077 platform, return a DOMException whose name is "NotFoundError", and
1078 terminate this algorithm.
1079 15. Let authenticator be a platform-specific handle whose value
1080 identifies an authenticator.
1081 16. For each authenticator currently available on this platform,
1082 perform the following steps:
1083 1. Let allowCredentialDescriptorList be a new list.
1084 2. If options.allowCredentials is not empty, execute a
1085 platform-specific procedure to determine which, if any, public
1086 key credentials described by options.allowCredentials are
1087 bound to this authenticator, by matching with rpId,
1088 options.allowCredentials.id, and
1089 options.allowCredentials.type. Set
1090 allowCredentialDescriptorList to this filtered list.
1091 3. If allowCredentialDescriptorList
1092
1093 is not empty
1094
1095 1. Let distinctTransports be a new ordered set.
1096 2. For each credential descriptor C in
1097
1098 allowCredentialDescriptorList, append each value, if
1099 any, of C.transports to distinctTransports.
1100 Note: This will aggregate only distinct values of
1101 transports (for this authenticator) in
1102 distinctTransports due to the properties of ordered
1103 sets.
1104 3. If distinctTransports
1105
1106 is not empty
1107 The client selects one transport value
1108 from distinctTransports, possibly
1109 incorporating local configuration
1110 knowledge of the appropriate transport
1111 to use with authenticator in making its
selection.

1102 encoding of authenticatorExtensionInput.
1103 10. Let collectedClientData be a new CollectedClientData instance whose
1104 fields are:
1105
1106 challenge
1107 The base64url encoding of options.challenge
1108
1109 origin
1110 The serialization of callerOrigin.
1111
1112 hashAlgorithm
1113 The recognized algorithm name of the hash algorithm
1114 selected by the client for generating the hash of the
1115 serialized client data
1116
1117 tokenBindingId
1118 The Token Binding ID associated with callerOrigin, if one
1119 is available.
1120
1121 clientExtensions
1122 clientExtensions
1123
1124 authenticatorExtensions
1125 authenticatorExtensions
1126
1127 11. Let clientDataJSON be the JSON-serialized client data constructed
1128 from collectedClientData.
1129 12. Let clientDataHash be the hash of the serialized client data
1130 represented by clientDataJSON.
1131 13. Let issuedRequests be a new ordered set.
1132 14. If there are no authenticators currently available on this
1133 platform, return a DOMException whose name is "NotFoundError", and
1134 terminate this algorithm.
1135 15. Let authenticator be a platform-specific handle whose value
1136 identifies an authenticator.
1137 16. For each authenticator currently available on this platform,
1138 perform the following steps:
1139 1. Let allowCredentialDescriptorList be a new list.
1140 2. If options.allowCredentials is not empty, execute a
1141 platform-specific procedure to determine which, if any, public
1142 key credentials described by options.allowCredentials are
1143 bound to this authenticator, by matching with rpId,
1144 options.allowCredentials.id, and
1145 options.allowCredentials.type. Set
1146 allowCredentialDescriptorList to this filtered list.
1147 3. If allowCredentialDescriptorList
1148
1149 is not empty
1150
1151 1. Let distinctTransports be a new ordered set.
1152 2. If allowCredentialDescriptorList has exactly one
1153 value, let savedCredentialId be a new ArrayBuffer,
1154 created using global's %ArrayBuffer%, and containing
1155 the bytes of allowCredentialDescriptorList[0].id.
1156 3. For each credential descriptor C in
1157 allowCredentialDescriptorList, append each value, if
1158 any, of C.transports to distinctTransports.
1159 Note: This will aggregate only distinct values of
1160 transports (for this authenticator) in
1161 distinctTransports due to the properties of ordered
1162 sets.
1163 4. If distinctTransports
1164
1165 is not empty
1166 The client selects one transport value
1167 from distinctTransports, possibly
1168 incorporating local configuration
1169 knowledge of the appropriate transport
1170 to use with authenticator in making its
1171 selection.

1112 Then, using transport, invoke in
 1113 parallel the authenticatorGetAssertion
 1114 operation on authenticator, with rpId,
 1115 clientDataHash,
 1116 allowCredentialDescriptorList, and
 1117 authenticatorExtensions as parameters.
 1118
 1119 is empty
 1120 Using local configuration knowledge of
 1121 the appropriate transport to use with
 1122 authenticator, invoke in parallel the
 1123 authenticatorGetAssertion operation on
 1124 authenticator with rpId, clientDataHash,
 1125 allowCredentialDescriptorList, and
 1126 clientExtensions as parameters.
 1127
 1128 is empty
 1129 Using local configuration knowledge of the
 1130 appropriate transport to use with authenticator,
 1131 invoke in parallel the authenticatorGetAssertion
 1132 operation on authenticator with rpId,
 1133 clientDataHash, and clientExtensions as parameters.
 1134
 1135 Note: In this case, the Relying Party did not supply
 1136 a list of acceptable credential descriptors. Thus
 1137 the authenticator is being asked to exercise any
 1138 credential it may possess that is bound to the
 1139 Relying Party, as identified by rpId.
 1140
 1141 4. Append authenticator to issuedRequests.
 1142 17. Start a timer for adjustedTimeout milliseconds. Then execute the
 1143 following steps in parallel. The task source for these tasks is the
 1144 dom manipulation task source.
 1145 18. While issuedRequests is not empty, perform the following actions
 1146 depending upon the adjustedTimeout timer and responses from the
 1147 authenticators:
 1148
 1149 If the adjustedTimeout timer expires,
 1150 For each authenticator in issuedRequests invoke the
 1151 authenticatorCancel operation on authenticator and remove
 1152 authenticator from issuedRequests.
 1153
 1154 If any authenticator returns a status indicating that the user
 1155 cancelled the operation,
 1156
 1157 1. Remove authenticator from issuedRequests.
 1158 2. For each remaining authenticator in issuedRequests invoke
 1159 the authenticatorCancel operation on authenticator and
 1160 remove it from issuedRequests.
 1161
 1162 If any authenticator returns an error status,
 1163 Remove authenticator from issuedRequests.
 1164
 1165 If any authenticator indicates success,
 1166
 1167 1. Remove authenticator from issuedRequests.
 1168 2. Let value be a new PublicKeyCredential associated with
 1169 global whose fields are:
 1170
 1171 [[identifier]]
 1172 A new ArrayBuffer, created using global's
 1173 %ArrayBuffer%, containing the bytes of the
 1174 credential ID returned from the successful
 1175 authenticatorGetAssertion operation, as
 1176 defined in 5.2.2 The
 1177

1172 Then, using transport, invoke in
 1173 parallel the authenticatorGetAssertion
 1174 operation on authenticator, with rpId,
 1175 clientDataHash,
 1176 allowCredentialDescriptorList, and
 1177 authenticatorExtensions as parameters.
 1178
 1179 is empty
 1180 Using local configuration knowledge of
 1181 the appropriate transport to use with
 1182 authenticator, invoke in parallel the
 1183 authenticatorGetAssertion operation on
 1184 authenticator with rpId, clientDataHash,
 1185 allowCredentialDescriptorList, and
 1186 clientExtensions as parameters.
 1187
 1188 is empty
 1189 Using local configuration knowledge of the
 1190 appropriate transport to use with authenticator,
 1191 invoke in parallel the authenticatorGetAssertion
 1192 operation on authenticator with rpId,
 1193 clientDataHash, and clientExtensions as parameters.
 1194
 1195 Note: In this case, the Relying Party did not supply
 1196 a list of acceptable credential descriptors. Thus
 1197 the authenticator is being asked to exercise any
 1198 credential it may possess that is bound to the
 1199 Relying Party, as identified by rpId.
 1200
 1201 4. Append authenticator to issuedRequests.
 1202 17. Start a timer for adjustedTimeout milliseconds. Then execute the
 1203 following steps in parallel. The task source for these tasks is the
 1204 dom manipulation task source.
 1205 1. While issuedRequests is not empty, perform the following
 1206 actions depending upon the adjustedTimeout timer and responses
 1207 from the authenticators:
 1208
 1209 If the adjustedTimeout timer expires,
 1210 For each authenticator in issuedRequests invoke the
 1211 authenticatorCancel operation on authenticator and
 1212 remove authenticator from issuedRequests.
 1213
 1214 If any authenticator returns a status indicating that the
 1215 user cancelled the operation,
 1216
 1217 1. Remove authenticator from issuedRequests.
 1218 2. For each remaining authenticator in issuedRequests
 1219 invoke the authenticatorCancel operation on
 1220 authenticator and remove it from issuedRequests.
 1221
 1222 If any authenticator returns an error status,
 1223 Remove authenticator from issuedRequests.
 1224
 1225 If any authenticator indicates success,
 1226
 1227 1. Remove authenticator from issuedRequests.
 1228 2. Let value be a new PublicKeyCredential associated
 1229 with global whose fields are:
 1230
 1231 [[identifier]]
 1232 Create a new ArrayBuffer, using global's
 1233 %ArrayBuffer%. If savedCredentialId
 1234 exists, set the value of the new
 1235 ArrayBuffer to be the bytes of
 1236 savedCredentialId. Otherwise, set the
 1237 value of the new ArrayBuffer to be the
 1238 bytes of the credential ID returned from
 1239 the successful authenticatorGetAssertion
 1240 operation, as defined in 5.2.2 The
 1241

1178 authenticatorGetAssertion operation.
 1179
 1180 response
 1181 A new AuthenticatorAssertionResponse object
 1182 associated with global whose fields are:

1183
 1184 clientDataJSON
 1185 A new ArrayBuffer, created using
 1186 global's %ArrayBuffer%, containing the
 1187 bytes of clientDataJSON
 1188
 1189 authenticatorData
 1190 A new ArrayBuffer, created using
 1191 global's %ArrayBuffer%, containing the
 1192 bytes of the returned authenticatorData

1193
 1194 signature
 1195 A new ArrayBuffer, created using
 1196 global's %ArrayBuffer%, containing the
 1197 bytes of the returned signature

1198
 1199 clientExtensionResults
 1200 A new AuthenticationExtensions object
 1201 containing the extension identifier -> client
 1202 extension output entries created by running
 1203 each extension's client extension processing
 1204 algorithm to create the client extension
 1205 outputs, for each client extension in

1206
 1207 clientDataJSON.clientExtensions.
 1208
 1209 3. For each remaining authenticator in issuedRequests invoke
 1210 the authenticatorCancel operation on authenticator and
 1211 remove it from issuedRequests.
 1212 4. Return value and terminate this algorithm.

1213 19. Return a DOMException whose name is "NotAllowedError".

1214
 1215 During the above process, the user agent SHOULD show some UI to the
 1216 user to guide them in the process of selecting and authorizing an
 1217 authenticator with which to complete the operation.

1218
 1219 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
 1220 isPlatformAuthenticatorAvailable() method
 1221
 1222 Relying Parties use this method to determine whether they can create a
 1223 new credential using a platform authenticator. Upon invocation, the
 1224 client employs a platform-specific procedure to discover available
 1225 platform authenticators. If successful, the client then assesses
 1226 whether the user is willing to create a credential using one of the
 1227 available platform authenticators. This assessment may include various
 1228 factors, such as:
 1229 * Whether the user is running in private or incognito mode.
 1230 * Whether the user has configured the client to not create such
 1231 credentials.
 1232 * Whether the user has previously expressed an unwillingness to
 1233 create a new credential for this Relying Party, either through
 1234 configuration or by declining a user interface prompt.
 1235 * The user's explicitly stated intentions, determined through user
 1236 interaction.

1237
 1238 If this assessment is affirmative, the promise is resolved with the
 1239 value of True. Otherwise, the promise is resolved with the value of
 1240 False. Based on the result, the Relying Party can take further actions
 1241 to guide the user to create a credential.
 1242
 1243 This method has no arguments and returns a boolean value.

1242 authenticatorGetAssertion operation.
 1243
 1244 response
 1245 A new AuthenticatorAssertionResponse
 1246 object associated with global whose
 1247 fields are:

1248
 1249 clientDataJSON
 1250 A new ArrayBuffer, created using
 1251 global's %ArrayBuffer%, containing
 1252 the bytes of clientDataJSON.
 1253
 1254 authenticatorData
 1255 A new ArrayBuffer, created using
 1256 global's %ArrayBuffer%, containing
 1257 the bytes of the returned
 1258 authenticatorData.
 1259
 1260 signature
 1261 A new ArrayBuffer, created using
 1262 global's %ArrayBuffer%, containing
 1263 the bytes of the returned
 1264 signature.
 1265
 1266 clientExtensionResults
 1267 A new AuthenticationExtensions object
 1268 containing the extension identifier ->
 1269 client extension output entries created
 1270 by running each extension's client
 1271 extension processing algorithm to create
 1272 the client extension outputs, for each
 1273 client extension in
 1274 clientDataJSON.clientExtensions.
 1275
 1276 3. For each remaining authenticator in issuedRequests
 1277 invoke the authenticatorCancel operation on
 1278 authenticator and remove it from issuedRequests.
 1279 4. Return value and terminate this algorithm.

1280
 1281 18. Return a DOMException whose name is "NotAllowedError".

1282
 1283 During the above process, the user agent SHOULD show some UI to the
 1284 user to guide them in the process of selecting and authorizing an
 1285 authenticator with which to complete the operation.

1286
 1287 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
 1288 isPlatformAuthenticatorAvailable() method
 1289
 1290 Relying Parties use this method to determine whether they can create a
 1291 new credential using a platform authenticator. Upon invocation, the
 1292 client employs a platform-specific procedure to discover available
 1293 platform authenticators. If successful, the client then assesses
 1294 whether the user is willing to create a credential using one of the
 1295 available platform authenticators. This assessment may include various
 1296 factors, such as:
 1297 * Whether the user is running in private or incognito mode.
 1298 * Whether the user has configured the client to not create such
 1299 credentials.
 1300 * Whether the user has previously expressed an unwillingness to
 1301 create a new credential for this Relying Party, either through
 1302 configuration or by declining a user interface prompt.
 1303 * The user's explicitly stated intentions, determined through user
 1304 interaction.

1305
 1306 If this assessment is affirmative, the promise is resolved with the
 1307 value of True. Otherwise, the promise is resolved with the value of
 1308 False. Based on the result, the Relying Party can take further actions
 1309 to guide the user to create a credential.
 1310
 1311 This method has no arguments and returns a boolean value.

1244
 1245 If the promise will return False, the client SHOULD wait a fixed period
 1246 of time from the invocation of the method before returning False. This
 1247 is done so that callers can not distinguish between the case where the
 1248 user was unwilling to create a credential using one of the available
 1249 platform authenticators and the case where no platform authenticator
 1250 exists. Trying to make these cases indistinguishable is done in an
 1251 attempt to not provide additional information that could be used for
 1252 fingerprinting. A timeout value on the order of 10 minutes is
 1253 recommended; this is enough time for successful user interactions to be
 1254 performed but short enough that the dangling promise will still be
 1255 resolved in a reasonably timely fashion.
 1256 [SecureContext]
 1257 partial interface PublicKeyCredential {
 1258 [Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();
 1259 };
 1260
 1261 4.2. Authenticator Responses (interface AuthenticatorResponse)
 1262
 1263 Authenticators respond to Relying Party requests by returning an object
 1264 derived from the AuthenticatorResponse interface:
 1265 [SecureContext]
 1266 interface AuthenticatorResponse {
 1267 [SameObject] readonly attribute ArrayBuffer clientDataJSON;
 1268 };
 1269 clientDataJSON, of type ArrayBuffer, readonly
 1270 This attribute contains a JSON serialization of the client data
 1271 passed to the authenticator by the client in its call to either
 1272 create() or get().
 1273
 1274 4.2.1. Information about Public Key Credential (interface
 1275 AuthenticatorAttestationResponse)
 1276
 1277 The AuthenticatorAttestationResponse interface represents the
 1278 authenticator's response to a client's request for the creation of a
 1279 new public key credential. It contains information about the new
 1280 credential that can be used to identify it for later use, and metadata
 1281 that can be used by the Relying Party to assess the characteristics of
 1282 the credential during registration.
 1283 [SecureContext]
 1284 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
 1285 [SameObject] readonly attribute ArrayBuffer attestationObject;
 1286 };
 1287 clientDataJSON
 1288 This attribute, inherited from AuthenticatorResponse, contains
 1289 the JSON-serialized client data (see 5.3 Attestation) passed to
 1290 the authenticator by the client in order to generate this
 1291 credential. The exact JSON serialization must be preserved, as
 1292 the hash of the serialized client data has been computed over
 1293 it.
 1294 attestationObject, of type ArrayBuffer, readonly
 1295 This attribute contains an attestation object, which is opaque
 1296 to, and cryptographically protected against tampering by, the
 1297 client. The attestation object contains both authenticator data
 1298 and an attestation statement. The former contains the AAGUID, a
 1299 unique credential ID, and the credential public key. The
 1300 contents of the attestation statement are determined by the
 1301 attestation statement format used by the authenticator. It also
 1302 contains any additional information that the Relying Party's
 1303 server requires to validate the attestation statement, as well
 1304 as to decode and validate the authenticator data along with the
 1305 JSON-serialized client data. For more details, see 5.3
 1306 Attestation, 5.3.4 Generating an Attestation Object, and Figure
 1307 3.
 1308
 1309 4.2.2. Web Authentication Assertion (interface
 1310 AuthenticatorAssertionResponse)
 1311
 1312
 1313

1312
 1313 If the promise will return False, the client SHOULD wait a fixed period
 1314 of time from the invocation of the method before returning False. This
 1315 is done so that callers can not distinguish between the case where the
 1316 user was unwilling to create a credential using one of the available
 1317 platform authenticators and the case where no platform authenticator
 1318 exists. Trying to make these cases indistinguishable is done in an
 1319 attempt to not provide additional information that could be used for
 1320 fingerprinting. A timeout value on the order of 10 minutes is
 1321 recommended; this is enough time for successful user interactions to be
 1322 performed but short enough that the dangling promise will still be
 1323 resolved in a reasonably timely fashion.
 1324 [SecureContext]
 1325 partial interface PublicKeyCredential {
 1326 [Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();
 1327 };
 1328
 1329 4.2. Authenticator Responses (interface AuthenticatorResponse)
 1330
 1331 Authenticators respond to Relying Party requests by returning an object
 1332 derived from the AuthenticatorResponse interface:
 1333 [SecureContext]
 1334 interface AuthenticatorResponse {
 1335 [SameObject] readonly attribute ArrayBuffer clientDataJSON;
 1336 };
 1337 clientDataJSON, of type ArrayBuffer, readonly
 1338 This attribute contains a JSON serialization of the client data
 1339 passed to the authenticator by the client in its call to either
 1340 create() or get().
 1341
 1342 4.2.1. Information about Public Key Credential (interface
 1343 AuthenticatorAttestationResponse)
 1344
 1345 The AuthenticatorAttestationResponse interface represents the
 1346 authenticator's response to a client's request for the creation of a
 1347 new public key credential. It contains information about the new
 1348 credential that can be used to identify it for later use, and metadata
 1349 that can be used by the Relying Party to assess the characteristics of
 1350 the credential during registration.
 1351 [SecureContext]
 1352 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
 1353 [SameObject] readonly attribute ArrayBuffer attestationObject;
 1354 };
 1355 clientDataJSON
 1356 This attribute, inherited from AuthenticatorResponse, contains
 1357 the JSON-serialized client data (see 5.3 Attestation) passed to
 1358 the authenticator by the client in order to generate this
 1359 credential. The exact JSON serialization must be preserved, as
 1360 the hash of the serialized client data has been computed over
 1361 it.
 1362 attestationObject, of type ArrayBuffer, readonly
 1363 This attribute contains an attestation object, which is opaque
 1364 to, and cryptographically protected against tampering by, the
 1365 client. The attestation object contains both authenticator data
 1366 and an attestation statement. The former contains the AAGUID, a
 1367 unique credential ID, and the credential public key. The
 1368 contents of the attestation statement are determined by the
 1369 attestation statement format used by the authenticator. It also
 1370 contains any additional information that the Relying Party's
 1371 server requires to validate the attestation statement, as well
 1372 as to decode and validate the authenticator data along with the
 1373 JSON-serialized client data. For more details, see 5.3
 1374 Attestation, 5.3.4 Generating an Attestation Object, and Figure
 1375 3.
 1376
 1377 4.2.2. Web Authentication Assertion (interface
 1378 AuthenticatorAssertionResponse)
 1379
 1380
 1381

1314 The AuthenticatorAssertionResponse interface represents an
 1315 authenticator's response to a client's request for generation of a new
 1316 authentication assertion given the Relying Party's challenge and
 1317 optional list of credentials it is aware of. This response contains a
 1318 cryptographic signature proving possession of the credential private
 1319 key, and optionally evidence of user consent to a specific transaction.
 1320 [SecureContext]
 1321 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
 1322 [SameObject] readonly attribute ArrayBuffer authenticatorData;
 1323 [SameObject] readonly attribute ArrayBuffer signature;
 1324 };
 1325
 1326 clientDataJSON
 1327 This attribute, inherited from AuthenticatorResponse, contains
 1328 the JSON-serialized client data (see 4.7.1 Client data used in
 1329 WebAuthn signatures (dictionary CollectedClientData)) passed to
 1330 the authenticator by the client in order to generate this
 1331 assertion. The exact JSON serialization must be preserved, as
 1332 the hash of the serialized client data has been computed over
 1333 it.
 1334
 1335 authenticatorData, of type ArrayBuffer, readonly
 1336 This attribute contains the authenticator data returned by the
 1337 authenticator. See 5.1 Authenticator data.
 1338
 1339 signature, of type ArrayBuffer, readonly
 1340 This attribute contains the raw signature returned from the
 1341 authenticator. See 5.2.2 The authenticatorGetAssertion
 1342 operation.
 1343
 1344 4.3. Parameters for Credential Generation (dictionary
 1345 PublicKeyCredentialParameters)
 1346 dictionary PublicKeyCredentialParameters {
 1347 required PublicKeyCredentialType type;
 1348 required COSEAlgorithmIdentifier alg;
 1349 };
 1350
 1351 This dictionary is used to supply additional parameters when creating a
 1352 new credential.
 1353
 1354 The type member specifies the type of credential to be created.
 1355
 1356 The alg member specifies the cryptographic signature algorithm with
 1357 which the newly generated credential will be used, and thus also the
 1358 type of asymmetric key pair to be generated, e.g., RSA or Elliptic
 1359 Curve.
 1360
 1361 Note: we use "alg" as the latter member name, rather than spelling-out
 1362 "algorithm", because it will be serialized into a message to the
 1363 authenticator, which may be sent over a low-bandwidth link.
 1364
 1365 4.4. Options for Credential Creation (dictionary
 1366 MakePublicKeyCredentialOptions)
 1367 dictionary MakePublicKeyCredentialOptions {
 1368 required PublicKeyCredentialRpEntity rp;
 1369 required PublicKeyCredentialUserEntity user;
 1370
 1371 required BufferSource challenge;
 1372 required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
 1373
 1374 unsigned long timeout;
 1375 sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
 1376 AuthenticatorSelectionCriteria authenticatorSelection;
 1377 AuthenticationExtensions extensions;
 1378 };
 1379 rp, of type PublicKeyCredentialRpEntity

1382 The AuthenticatorAssertionResponse interface represents an
 1383 authenticator's response to a client's request for generation of a new
 1384 authentication assertion given the Relying Party's challenge and
 1385 optional list of credentials it is aware of. This response contains a
 1386 cryptographic signature proving possession of the credential private
 1387 key, and optionally evidence of user consent to a specific transaction.
 1388 [SecureContext]
 1389 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
 1390 [SameObject] readonly attribute ArrayBuffer authenticatorData;
 1391 [SameObject] readonly attribute ArrayBuffer signature;
 1392 };
 1393
 1394 clientDataJSON
 1395 This attribute, inherited from AuthenticatorResponse, contains
 1396 the JSON-serialized client data (see 4.7.1 Client data used in
 1397 WebAuthn signatures (dictionary CollectedClientData)) passed to
 1398 the authenticator by the client in order to generate this
 1399 assertion. The exact JSON serialization must be preserved, as
 1400 the hash of the serialized client data has been computed over
 1401 it.
 1402
 1403 authenticatorData, of type ArrayBuffer, readonly
 1404 This attribute contains the authenticator data returned by the
 1405 authenticator. See 5.1 Authenticator data.
 1406
 1407 signature, of type ArrayBuffer, readonly
 1408 This attribute contains the raw signature returned from the
 1409 authenticator. See 5.2.2 The authenticatorGetAssertion
 1410 operation.
 1411
 1412 4.3. Parameters for Credential Generation (dictionary
 1413 PublicKeyCredentialParameters)
 1414 dictionary PublicKeyCredentialParameters {
 1415 required PublicKeyCredentialType type;
 1416 required COSEAlgorithmIdentifier alg;
 1417 };
 1418
 1419 This dictionary is used to supply additional parameters when creating a
 1420 new credential.
 1421
 1422 The type member specifies the type of credential to be created.
 1423
 1424 The alg member specifies the cryptographic signature algorithm with
 1425 which the newly generated credential will be used, and thus also the
 1426 type of asymmetric key pair to be generated, e.g., RSA or Elliptic
 1427 Curve.
 1428
 1429 Note: we use "alg" as the latter member name, rather than spelling-out
 1430 "algorithm", because it will be serialized into a message to the
 1431 authenticator, which may be sent over a low-bandwidth link.
 1432
 1433 4.4. Options for Credential Creation (dictionary
 1434 MakePublicKeyCredentialOptions)
 1435 dictionary MakePublicKeyCredentialOptions {
 1436 required PublicKeyCredentialRpEntity rp;
 1437 required PublicKeyCredentialUserEntity user;
 1438
 1439 required BufferSource challenge;
 1440 required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
 1441
 1442 unsigned long timeout;
 1443 sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
 1444 AuthenticatorSelectionCriteria authenticatorSelection;
 1445 AuthenticationExtensions extensions;
 1446 };
 1447 rp, of type PublicKeyCredentialRpEntity

1384 This member contains data about the Relying Party responsible
1385 for the request.
1386
1387 Its value's name member is required, and contains the friendly
1388 name of the Relying Party (e.g. "Acme Corporation", "Widgets,
1389 Inc.", or "Awesome Site").
1390
1391 Its value's id member specifies the relying party identifier
1392 with which the credential should be associated. If omitted, its
1393 value will be the CredentialsContainer object's relevant
1394 settings object's origin's effective domain.
1395
1396 user, of type PublicKeyCredentialUserEntity
1397 This member contains data about the user account for which the
1398 Relying Party is requesting attestation.
1399
1400 Its value's name member is required, and contains a name for the
1401 user account (e.g., "john.p.smith@example.com" or
1402 "+14255551234").
1403
1404 Its value's displayName member is required, and contains a
1405 friendly name for the user account (e.g., "John P. Smith").
1406
1407 Its value's id member is required, and contains an identifier
1408 for the account, specified by the Relying Party. This is not
1409 meant to be displayed to the user, but is used by the Relying
1410 Party to control the number of credentials - an authenticator
1411 will never contain more than one credential for a given Relying
1412 Party under the same id.
1413
1414 challenge, of type BufferSource
1415 This member contains a challenge intended to be used for
1416 generating the newly created credential's attestation object.
1417
1418 pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>
1419 This member contains information about the desired properties of
1420 the credential to be created. The sequence is ordered from most
1421 preferred to least preferred. The platform makes a best-effort
1422 to create the most preferred credential that it can.
1423
1424 timeout, of type unsigned long
1425 This member specifies a time, in milliseconds, that the caller
1426 is willing to wait for the call to complete. This is treated as
1427 a hint, and may be overridden by the platform.
1428
1429 excludeCredentials, of type sequence<PublicKeyCredentialDescriptor>,
1430 defaulting to None
1431 This member is intended for use by Relying Parties that wish to
1432 limit the creation of multiple credentials for the same account
1433 on a single authenticator. The platform is requested to return
1434 an error if the new credential would be created on an
1435 authenticator that also contains one of the credentials
1436 enumerated in this parameter.
1437
1438 authenticatorSelection, of type AuthenticatorSelectionCriteria
1439 This member is intended for use by Relying Parties that wish to
1440 select the appropriate authenticators to participate in the
1441 create() or get() operation.
1442
1443 extensions, of type AuthenticationExtensions
1444 This member contains additional parameters requesting additional
1445 processing by the client and authenticator. For example, the
1446 caller may request that only authenticators with certain
1447 capabilities be used to create the credential, or that particular
1448 information be returned in the attestation object. Some
1449 extensions are defined in 8 WebAuthn Extensions; consult the
1450 IANA "WebAuthn Extension Identifier" registry established by
1451 [WebAuthn-Registries] for an up-to-date list of registered
1452 WebAuthn Extensions.
1453

1452 This member contains data about the Relying Party responsible
1453 for the request.
1454
1455 Its value's name member is required, and contains the friendly
1456 name of the Relying Party (e.g. "Acme Corporation", "Widgets,
1457 Inc.", or "Awesome Site").
1458
1459 Its value's id member specifies the relying party identifier
1460 with which the credential should be associated. If omitted, its
1461 value will be the CredentialsContainer object's relevant
1462 settings object's origin's effective domain.
1463
1464 user, of type PublicKeyCredentialUserEntity
1465 This member contains data about the user account for which the
1466 Relying Party is requesting attestation.
1467
1468 Its value's name member is required, and contains a name for the
1469 user account (e.g., "john.p.smith@example.com" or
1470 "+14255551234").
1471
1472 Its value's displayName member is required, and contains a
1473 friendly name for the user account (e.g., "John P. Smith").
1474
1475 Its value's id member is required, and contains an identifier
1476 for the account, specified by the Relying Party. This is not
1477 meant to be displayed to the user, but is used by the Relying
1478 Party to control the number of credentials - an authenticator
1479 will never contain more than one credential for a given Relying
1480 Party under the same id.
1481
1482 challenge, of type BufferSource
1483 This member contains a challenge intended to be used for
1484 generating the newly created credential's attestation object.
1485
1486 pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>
1487 This member contains information about the desired properties of
1488 the credential to be created. The sequence is ordered from most
1489 preferred to least preferred. The platform makes a best-effort
1490 to create the most preferred credential that it can.
1491
1492 timeout, of type unsigned long
1493 This member specifies a time, in milliseconds, that the caller
1494 is willing to wait for the call to complete. This is treated as
1495 a hint, and may be overridden by the platform.
1496
1497 excludeCredentials, of type sequence<PublicKeyCredentialDescriptor>,
1498 defaulting to None
1499 This member is intended for use by Relying Parties that wish to
1500 limit the creation of multiple credentials for the same account
1501 on a single authenticator. The platform is requested to return
1502 an error if the new credential would be created on an
1503 authenticator that also contains one of the credentials
1504 enumerated in this parameter.
1505
1506 authenticatorSelection, of type AuthenticatorSelectionCriteria
1507 This member is intended for use by Relying Parties that wish to
1508 select the appropriate authenticators to participate in the
1509 create() or get() operation.
1510
1511 extensions, of type AuthenticationExtensions
1512 This member contains additional parameters requesting additional
1513 processing by the client and authenticator. For example, the
1514 caller may request that only authenticators with certain
1515 capabilities be used to create the credential, or that particular
1516 information be returned in the attestation object. Some
1517 extensions are defined in 8 WebAuthn Extensions; consult the
1518 IANA "WebAuthn Extension Identifier" registry established by
1519 [WebAuthn-Registries] for an up-to-date list of registered
1520 WebAuthn Extensions.
1521

1454 4.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)
 1455
 1456 The PublicKeyCredentialEntity dictionary describes a user account, or a
 1457 Relying Party, with which a public key credential is associated.
 1458 dictionary PublicKeyCredentialEntity {
 1459 DOMString name;
 1460 USVString icon;
 1461 };
 1462
 1463 name, of type DOMString
 1464 A human-friendly identifier for the entity. For example, this
 1465 could be a company name for a Relying Party, or a user's name.
 1466 This identifier is intended for display.
 1467
 1468 icon, of type USVString
 1469 A serialized URL which resolves to an image associated with the
 1470 entity. For example, this could be a user's avatar or a Relying
 1471 Party's logo. This URL MUST be an a priori authenticated URL.
 1472
 1473 4.4.2. RP Parameters for Credential Generation (dictionary
 1474 PublicKeyCredentialRpEntity)
 1475
 1476 The PublicKeyCredentialRpEntity dictionary is used to supply additional
 1477 Relying Party attributes when creating a new credential.
 1478 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
 1479 DOMString id;
 1480 };
 1481
 1482 id, of type DOMString
 1483 A unique identifier for the Relying Party entity, which sets the
 1484 RP ID.
 1485
 1486 4.4.3. User Account Parameters for Credential Generation (dictionary
 1487 PublicKeyCredentialUserEntity)
 1488
 1489 The PublicKeyCredentialUserEntity dictionary is used to supply
 1490 additional user account attributes when creating a new credential.
 1491 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
 1492 BufferSource id;
 1493 DOMString displayName;
 1494 };
 1495
 1496 id, of type BufferSource
 1497 A unique identifier for the user account entity. This is a
 1498 reference to an opaque byte array value specified by the Relying
 1499 Party. The maximum size of this array is 64 bytes.
 1500
 1501 displayName, of type DOMString
 1502 A friendly name for the user account (e.g., "John P. Smith").
 1503
 1504 4.4.4. Authenticator Selection Criteria (dictionary
 1505 AuthenticatorSelectionCriteria)
 1506
 1507 Relying Parties may use the AuthenticatorSelectionCriteria dictionary
 1508 to specify their requirements regarding authenticator attributes.
 1509 dictionary AuthenticatorSelectionCriteria {
 1510 AuthenticatorAttachment authenticatorAttachment;
 1511 boolean requireResidentKey = false;
 1512 boolean requireUserVerification = false;
 1513 };
 1514
 1515 authenticatorAttachment, of type AuthenticatorAttachment
 1516 If this member is present, eligible authenticators are filtered
 1517 to only authenticators attached with the specified 4.4.5
 1518 Authenticator Attachment enumeration (enum
 1519 AuthenticatorAttachment).
 1520
 1521 requireResidentKey, of type boolean, defaulting to false
 1522 This member describes the Relying Parties' requirements
 1523 regarding availability of the Client-side-resident Credential

1522 4.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)
 1523
 1524 The PublicKeyCredentialEntity dictionary describes a user account, or a
 1525 Relying Party, with which a public key credential is associated.
 1526 dictionary PublicKeyCredentialEntity {
 1527 DOMString name;
 1528 USVString icon;
 1529 };
 1530
 1531 name, of type DOMString
 1532 A human-friendly identifier for the entity. For example, this
 1533 could be a company name for a Relying Party, or a user's name.
 1534 This identifier is intended for display.
 1535
 1536 icon, of type USVString
 1537 A serialized URL which resolves to an image associated with the
 1538 entity. For example, this could be a user's avatar or a Relying
 1539 Party's logo. This URL MUST be an a priori authenticated URL.
 1540
 1541 4.4.2. RP Parameters for Credential Generation (dictionary
 1542 PublicKeyCredentialRpEntity)
 1543
 1544 The PublicKeyCredentialRpEntity dictionary is used to supply additional
 1545 Relying Party attributes when creating a new credential.
 1546 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
 1547 DOMString id;
 1548 };
 1549
 1550 id, of type DOMString
 1551 A unique identifier for the Relying Party entity, which sets the
 1552 RP ID.
 1553
 1554 4.4.3. User Account Parameters for Credential Generation (dictionary
 1555 PublicKeyCredentialUserEntity)
 1556
 1557 The PublicKeyCredentialUserEntity dictionary is used to supply
 1558 additional user account attributes when creating a new credential.
 1559 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
 1560 BufferSource id;
 1561 DOMString displayName;
 1562 };
 1563
 1564 id, of type BufferSource
 1565 A unique identifier for the user account entity. This is a
 1566 reference to an opaque byte array value specified by the Relying
 1567 Party. The maximum size of this array is 64 bytes.
 1568
 1569 displayName, of type DOMString
 1570 A friendly name for the user account (e.g., "John P. Smith").
 1571
 1572 4.4.4. Authenticator Selection Criteria (dictionary
 1573 AuthenticatorSelectionCriteria)
 1574
 1575 Relying Parties may use the AuthenticatorSelectionCriteria dictionary
 1576 to specify their requirements regarding authenticator attributes.
 1577 dictionary AuthenticatorSelectionCriteria {
 1578 AuthenticatorAttachment authenticatorAttachment;
 1579 boolean requireResidentKey = false;
 1580 boolean requireUserVerification = false;
 1581 };
 1582
 1583 authenticatorAttachment, of type AuthenticatorAttachment
 1584 If this member is present, eligible authenticators are filtered
 1585 to only authenticators attached with the specified 4.4.5
 1586 Authenticator Attachment enumeration (enum
 1587 AuthenticatorAttachment).
 1588
 1589 requireResidentKey, of type boolean, defaulting to false
 1590 This member describes the Relying Parties' requirements
 1591 regarding availability of the Client-side-resident Credential

1524 Private Key. If the parameter is set to true, the authenticator
 1525 MUST create a Client-side-resident Credential Private Key when
 1526 creating a public key credential.
 1527
 1528 requireUserVerification, of type boolean, defaulting to false
 1529 This member describes the Relying Parties' requirements
 1530 regarding the authenticator being capable of performing user
 1531 verification. If the parameter is set to true, the authenticator
 1532 MUST perform user verification when performing the create()
 1533 operation and future 4.1.4 Use an existing credential to make
 1534 an assertion - PublicKeyCredential's
 1535 [[DiscoverFromExternalSource]](options) method operations when
 1536 it is requested to verify the credential.
 1537
 1538 4.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)
 1539
 1540 enum AuthenticatorAttachment {
 1541 "platform" // Platform attachment
 1542 "cross-platform" // Cross-platform attachment
 1543 };
 1544
 1545 Clients may communicate with authenticators using a variety of
 1546 mechanisms. For example, a client may use a platform-specific API to
 1547 communicate with an authenticator which is physically bound to a
 1548 platform. On the other hand, a client may use a variety of standardized
 1549 cross-platform transport protocols such as Bluetooth (see 4.7.4
 1550 Authenticator Transport enumeration (enum AuthenticatorTransport)) to
 1551 discover and communicate with cross-platform attached authenticators.
 1552 Therefore, we use AuthenticatorAttachment to describe an
 1553 authenticator's attachment modality. We define authenticators that are
 1554 part of the client's platform as having a platform attachment, and
 1555 refer to them as platform authenticators. While those that are
 1556 reachable via cross-platform transport protocols are defined as having
 1557 cross-platform attachment, and refer to them as roaming authenticators.
 1558 * platform attachment - the respective authenticator is attached
 1559 using platform-specific transports. Usually, authenticators of this
 1560 class are non-removable from the platform.
 1561 * cross-platform attachment - the respective authenticator is
 1562 attached using cross-platform transports. Authenticators of this
 1563 class are removable from, and can "roam" among, client platforms.
 1564
 1565 This distinction is important because there are use-cases where only
 1566 platform authenticators are acceptable to a Relying Party, and
 1567 conversely ones where only roaming authenticators are employed. As a
 1568 concrete example of the former, a credential on a platform
 1569 authenticator may be used by Relying Parties to quickly and
 1570 conveniently reauthenticate the user with a minimum of friction, e.g.,
 1571 the user will not have to dig around in their pocket for their key fob
 1572 or phone. As a concrete example of the latter, when the user is
 1573 accessing the Relying Party from a given client for the first time,
 1574 they may be required to use a roaming authenticator which was
 1575 originally registered with the Relying Party using a different client.
 1576
 1577 4.5. Options for Assertion Generation (dictionary
 1578 PublicKeyCredentialRequestOptions)
 1579
 1580 The PublicKeyCredentialRequestOptions dictionary supplies get() with
 1581 the data it needs to generate an assertion. Its challenge member must
 1582 be present, while its other members are optional.
 1583 dictionary PublicKeyCredentialRequestOptions {
 1584 required BufferSource challenge;
 1585 unsigned long timeout;
 1586 USVString rpld;
 1587 sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
 1588 AuthenticationExtensions extensions;
 1589 };
 1590
 1591 challenge, of type BufferSource
 1592 This member represents a challenge that the selected
 1593 authenticator signs, along with other data, when producing an

1592 Private Key. If the parameter is set to true, the authenticator
 1593 MUST create a Client-side-resident Credential Private Key when
 1594 creating a public key credential.
 1595
 1596 requireUserVerification, of type boolean, defaulting to false
 1597 This member describes the Relying Parties' requirements
 1598 regarding the authenticator being capable of performing user
 1599 verification. If the parameter is set to true, the authenticator
 1600 MUST perform user verification when performing the create()
 1601 operation and future 4.1.4 Use an existing credential to make
 1602 an assertion - PublicKeyCredential's
 1603 [[DiscoverFromExternalSource]](options) method operations when
 1604 it is requested to verify the credential.
 1605
 1606 4.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)
 1607
 1608 enum AuthenticatorAttachment {
 1609 "platform" // Platform attachment
 1610 "cross-platform" // Cross-platform attachment
 1611 };
 1612
 1613 Clients may communicate with authenticators using a variety of
 1614 mechanisms. For example, a client may use a platform-specific API to
 1615 communicate with an authenticator which is physically bound to a
 1616 platform. On the other hand, a client may use a variety of standardized
 1617 cross-platform transport protocols such as Bluetooth (see 4.7.4
 1618 Authenticator Transport enumeration (enum AuthenticatorTransport)) to
 1619 discover and communicate with cross-platform attached authenticators.
 1620 Therefore, we use AuthenticatorAttachment to describe an
 1621 authenticator's attachment modality. We define authenticators that are
 1622 part of the client's platform as having a platform attachment, and
 1623 refer to them as platform authenticators. While those that are
 1624 reachable via cross-platform transport protocols are defined as having
 1625 cross-platform attachment, and refer to them as roaming authenticators.
 1626 * platform attachment - the respective authenticator is attached
 1627 using platform-specific transports. Usually, authenticators of this
 1628 class are non-removable from the platform.
 1629 * cross-platform attachment - the respective authenticator is
 1630 attached using cross-platform transports. Authenticators of this
 1631 class are removable from, and can "roam" among, client platforms.
 1632
 1633 This distinction is important because there are use-cases where only
 1634 platform authenticators are acceptable to a Relying Party, and
 1635 conversely ones where only roaming authenticators are employed. As a
 1636 concrete example of the former, a credential on a platform
 1637 authenticator may be used by Relying Parties to quickly and
 1638 conveniently reauthenticate the user with a minimum of friction, e.g.,
 1639 the user will not have to dig around in their pocket for their key fob
 1640 or phone. As a concrete example of the latter, when the user is
 1641 accessing the Relying Party from a given client for the first time,
 1642 they may be required to use a roaming authenticator which was
 1643 originally registered with the Relying Party using a different client.
 1644
 1645 4.5. Options for Assertion Generation (dictionary
 1646 PublicKeyCredentialRequestOptions)
 1647
 1648 The PublicKeyCredentialRequestOptions dictionary supplies get() with
 1649 the data it needs to generate an assertion. Its challenge member must
 1650 be present, while its other members are optional.
 1651 dictionary PublicKeyCredentialRequestOptions {
 1652 required BufferSource challenge;
 1653 unsigned long timeout;
 1654 USVString rpld;
 1655 sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
 1656 AuthenticationExtensions extensions;
 1657 };
 1658
 1659 challenge, of type BufferSource
 1660 This member represents a challenge that the selected
 1661 authenticator signs, along with other data, when producing an

1594 authentication assertion.
 1595
 1596 timeout, of type unsigned long
 1597 This optional member specifies a time, in milliseconds, that the
 1598 caller is willing to wait for the call to complete. The value is
 1599 treated as a hint, and may be overridden by the platform.
 1600
 1601 rpId, of type USVString
 1602 This optional member specifies the relying party identifier
 1603 claimed by the caller. If omitted, its value will be the
 1604 CredentialsContainer object's relevant settings object's
 1605 origin's effective domain.
 1606
 1607 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,
 1608 defaulting to None
 1609 This optional member contains a list of
 1610 PublicKeyCredentialDescriptor object representing public key
 1611 credentials acceptable to the caller, in decending order of the
 1612 caller's preference (the first item in the list is the most
 1613 preferred credential, and so on down the list).
 1614
 1615 extensions, of type AuthenticationExtensions
 1616 This optional member contains additional parameters requesting
 1617 additional processing by the client and authenticator. For
 1618 example, if transaction confirmation is sought from the user,
 1619 then the prompt string might be included as an extension.
 1620
 1621 4.6. Authentication Extensions (typedef AuthenticationExtensions)
 1622
 1623 typedef record<DOMString, any> AuthenticationExtensions;
 1624
 1625 This is a dictionary containing zero or more WebAuthn extensions, as
 1626 defined in 8 WebAuthn Extensions. An AuthenticationExtensions instance
 1627 can contain either client extensions or authenticator extensions,
 1628 depending upon context.
 1629
 1630 4.7. Supporting Data Structures
 1631
 1632 The public key credential type uses certain data structures that are
 1633 specified in supporting specifications. These are as follows.
 1634
 1635 4.7.1. Client data used in WebAuthn signatures (dictionary
 1636 CollectedClientData)
 1637
 1638 The client data represents the contextual bindings of both the Relying
 1639 Party and the client platform. It is a key-value mapping with
 1640 string-valued keys. Values may be any type that has a valid encoding in
 1641 JSON. Its structure is defined by the following Web IDL.
 1642 dictionary CollectedClientData {
 1643 required DOMString challenge;
 1644 required DOMString origin;
 1645 required DOMString hashAlgorithm;
 1646 DOMString tokenBindingId;
 1647 AuthenticationExtensions clientExtensions;
 1648 AuthenticationExtensions authenticatorExtensions;
 1649 };
 1650
 1651 The challenge member contains the base64url encoding of the challenge
 1652 provided by the RP.
 1653
 1654 The origin member contains the fully qualified origin of the requester,
 1655 as provided to the authenticator by the client, in the syntax defined
 1656 by [RFC6454].
 1657
 1658 The hashAlgorithm member is a recognized algorithm name that supports
 1659 the "digest" operation, which specifies the algorithm used to compute
 1660 the hash of the serialized client data. This algorithm is chosen by the
 1661 client at its sole discretion.
 1662
 1663 The tokenBindingId member contains the base64url encoding of the Token

1662 authentication assertion.
 1663
 1664 timeout, of type unsigned long
 1665 This optional member specifies a time, in milliseconds, that the
 1666 caller is willing to wait for the call to complete. The value is
 1667 treated as a hint, and may be overridden by the platform.
 1668
 1669 rpId, of type USVString
 1670 This optional member specifies the relying party identifier
 1671 claimed by the caller. If omitted, its value will be the
 1672 CredentialsContainer object's relevant settings object's
 1673 origin's effective domain.
 1674
 1675 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,
 1676 defaulting to None
 1677 This optional member contains a list of
 1678 PublicKeyCredentialDescriptor object representing public key
 1679 credentials acceptable to the caller, in decending order of the
 1680 caller's preference (the first item in the list is the most
 1681 preferred credential, and so on down the list).
 1682
 1683 extensions, of type AuthenticationExtensions
 1684 This optional member contains additional parameters requesting
 1685 additional processing by the client and authenticator. For
 1686 example, if transaction confirmation is sought from the user,
 1687 then the prompt string might be included as an extension.
 1688
 1689 4.6. Authentication Extensions (typedef AuthenticationExtensions)
 1690
 1691 typedef record<DOMString, any> AuthenticationExtensions;
 1692
 1693 This is a dictionary containing zero or more WebAuthn extensions, as
 1694 defined in 8 WebAuthn Extensions. An AuthenticationExtensions instance
 1695 can contain either client extensions or authenticator extensions,
 1696 depending upon context.
 1697
 1698 4.7. Supporting Data Structures
 1699
 1700 The public key credential type uses certain data structures that are
 1701 specified in supporting specifications. These are as follows.
 1702
 1703 4.7.1. Client data used in WebAuthn signatures (dictionary
 1704 CollectedClientData)
 1705
 1706 The client data represents the contextual bindings of both the Relying
 1707 Party and the client platform. It is a key-value mapping with
 1708 string-valued keys. Values may be any type that has a valid encoding in
 1709 JSON. Its structure is defined by the following Web IDL.
 1710 dictionary CollectedClientData {
 1711 required DOMString challenge;
 1712 required DOMString origin;
 1713 required DOMString hashAlgorithm;
 1714 DOMString tokenBindingId;
 1715 AuthenticationExtensions clientExtensions;
 1716 AuthenticationExtensions authenticatorExtensions;
 1717 };
 1718
 1719 The challenge member contains the base64url encoding of the challenge
 1720 provided by the RP.
 1721
 1722 The origin member contains the fully qualified origin of the requester,
 1723 as provided to the authenticator by the client, in the syntax defined
 1724 by [RFC6454].
 1725
 1726 The hashAlgorithm member is a recognized algorithm name that supports
 1727 the "digest" operation, which specifies the algorithm used to compute
 1728 the hash of the serialized client data. This algorithm is chosen by the
 1729 client at its sole discretion.
 1730
 1731 The tokenBindingId member contains the base64url encoding of the Token

1664 Binding ID that this client uses for the Token Binding protocol when
 1665 communicating with the Relying Party. This can be omitted if no Token
 1666 Binding has been negotiated between the client and the Relying Party.
 1667
 1668 The optional clientExtensions and authenticatorExtensions members
 1669 contain additional parameters generated by processing the extensions
 1670 passed in by the Relying Party. WebAuthn extensions are detailed in
 1671 Section 8 WebAuthn Extensions.
 1672
 1673 This structure is used by the client to compute the following
 1674 quantities:
 1675
 1676 JSON-serialized client data
 1677 This is the UTF-8 encoding of the result of calling the initial
 1678 value of JSON.stringify on a CollectedClientData dictionary.
 1679
 1680 Hash of the serialized client data
 1681 This is the hash (computed using hashAlgorithm) of the
 1682 JSON-serialized client data, as constructed by the client.
 1683
 1684 4.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
 1685
 1686 enum PublicKeyCredentialType {
 1687 "public-key"
 1688 };
 1689
 1690 This enumeration defines the valid credential types. It is an extension
 1691 point; values may be added to it in the future, as more credential
 1692 types are defined. The values of this enumeration are used for
 1693 versioning the Authentication Assertion and attestation structures
 1694 according to the type of the authenticator.
 1695
 1696 Currently one credential type is defined, namely "public-key".
 1697
 1698 4.7.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)
 1699
 1700 dictionary PublicKeyCredentialDescriptor {
 1701 required PublicKeyCredentialType type;
 1702 required BufferSource id;
 1703 sequence<AuthenticatorTransport> transports;
 1704 };
 1705
 1706 This dictionary contains the attributes that are specified by a caller
 1707 when referring to a credential as an input parameter to the create() or
 1708 get() methods. It mirrors the fields of the PublicKeyCredential object
 1709 returned by the latter methods.
 1710
 1711 The type member contains the type of the credential the caller is
 1712 referring to.
 1713
 1714 The id member contains the identifier of the credential that the caller
 1715 is referring to.
 1716
 1717 4.7.4. Authenticator Transport enumeration (enum AuthenticatorTransport)
 1718
 1719 enum AuthenticatorTransport {
 1720 "usb",
 1721 "nfc",
 1722 "ble",
 1723 };
 1724
 1725 Authenticators may communicate with Clients using a variety of
 1726 transports. This enumeration defines a hint as to how Clients might
 1727 communicate with a particular Authenticator in order to obtain an
 1728 assertion for a specific credential. Note that these hints represent
 1729 the Relying Party's best belief as to how an Authenticator may be
 1730 reached. A Relying Party may obtain a list of transports hints from
 1731 some attestation statement formats or via some out-of-band mechanism;
 1732 it is outside the scope of this specification to define that mechanism.
 1733 * usb - the respective Authenticator may be contacted over USB.

1732 Binding ID that this client uses for the Token Binding protocol when
 1733 communicating with the Relying Party. This can be omitted if no Token
 1734 Binding has been negotiated between the client and the Relying Party.
 1735
 1736 The optional clientExtensions and authenticatorExtensions members
 1737 contain additional parameters generated by processing the extensions
 1738 passed in by the Relying Party. WebAuthn extensions are detailed in
 1739 Section 8 WebAuthn Extensions.
 1740
 1741 This structure is used by the client to compute the following
 1742 quantities:
 1743
 1744 JSON-serialized client data
 1745 This is the UTF-8 encoding of the result of calling the initial
 1746 value of JSON.stringify on a CollectedClientData dictionary.
 1747
 1748 Hash of the serialized client data
 1749 This is the hash (computed using hashAlgorithm) of the
 1750 JSON-serialized client data, as constructed by the client.
 1751
 1752 4.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
 1753
 1754 enum PublicKeyCredentialType {
 1755 "public-key"
 1756 };
 1757
 1758 This enumeration defines the valid credential types. It is an extension
 1759 point; values may be added to it in the future, as more credential
 1760 types are defined. The values of this enumeration are used for
 1761 versioning the Authentication Assertion and attestation structures
 1762 according to the type of the authenticator.
 1763
 1764 Currently one credential type is defined, namely "public-key".
 1765
 1766 4.7.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)
 1767
 1768 dictionary PublicKeyCredentialDescriptor {
 1769 required PublicKeyCredentialType type;
 1770 required BufferSource id;
 1771 sequence<AuthenticatorTransport> transports;
 1772 };
 1773
 1774 This dictionary contains the attributes that are specified by a caller
 1775 when referring to a credential as an input parameter to the create() or
 1776 get() methods. It mirrors the fields of the PublicKeyCredential object
 1777 returned by the latter methods.
 1778
 1779 The type member contains the type of the credential the caller is
 1780 referring to.
 1781
 1782 The id member contains the identifier of the credential that the caller
 1783 is referring to.
 1784
 1785 4.7.4. Authenticator Transport enumeration (enum AuthenticatorTransport)
 1786
 1787 enum AuthenticatorTransport {
 1788 "usb",
 1789 "nfc",
 1790 "ble",
 1791 };
 1792
 1793 Authenticators may communicate with Clients using a variety of
 1794 transports. This enumeration defines a hint as to how Clients might
 1795 communicate with a particular Authenticator in order to obtain an
 1796 assertion for a specific credential. Note that these hints represent
 1797 the Relying Party's best belief as to how an Authenticator may be
 1798 reached. A Relying Party may obtain a list of transports hints from
 1799 some attestation statement formats or via some out-of-band mechanism;
 1800 it is outside the scope of this specification to define that mechanism.
 1801 * usb - the respective Authenticator may be contacted over USB.

1734 * nfc - the respective Authenticator may be contacted over Near Field
 1735 Communication (NFC).
 1736 * ble - the respective Authenticator may be contacted over Bluetooth
 1737 Smart (Bluetooth Low Energy / BLE).
 1738
 1739 4.7.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)
 1740
 1741 typedef long COSEAlgorithmIdentifier;
 1742
 1743 A COSEAlgorithmIdentifier's value is a number identifying a
 1744 cryptographic algorithm. The algorithm identifiers SHOULD be values
 1745 registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],
 1746 for instance, -7 for "ES256" and -257 for "RS256".
 1747
 1748 5. WebAuthn Authenticator model
 1749
 1750 The API defined in this specification implies a specific abstract
 1751 functional model for an authenticator. This section describes the
 1752 authenticator model.
 1753
 1754 Client platforms may implement and expose this abstract model in any
 1755 way desired. However, the behavior of the client's Web Authentication
 1756 API implementation, when operating on the authenticators supported by
 1757 that platform, MUST be indistinguishable from the behavior specified in
 1758 4 Web Authentication API.
 1759
 1760 For authenticators, this model defines the logical operations that they
 1761 must support, and the data formats that they expose to the client and
 1762 the Relying Party. However, it does not define the details of how
 1763 authenticators communicate with the client platform, unless they are
 1764 required for interoperability with Relying Parties. For instance, this
 1765 abstract model does not define protocols for connecting authenticators
 1766 to clients over transports such as USB or NFC. Similarly, this abstract
 1767 model does not define specific error codes or methods of returning
 1768 them; however, it does define error behavior in terms of the needs of
 1769 the client. Therefore, specific error codes are mentioned as a means of
 1770 showing which error conditions must be distinguishable (or not) from
 1771 each other in order to enable a compliant and secure client
 1772 implementation.
 1773
 1774 In this abstract model, the authenticator provides key management and
 1775 cryptographic signatures. It may be embedded in the WebAuthn client, or
 1776 housed in a separate device entirely. The authenticator may itself
 1777 contain a cryptographic module which operates at a higher security
 1778 level than the rest of the authenticator. This is particularly
 1779 important for authenticators that are embedded in the WebAuthn client,
 1780 as in those cases this cryptographic module (which may, for example, be
 1781 a TPM) could be considered more trustworthy than the rest of the
 1782 authenticator.
 1783
 1784 Each authenticator stores some number of public key credentials. Each
 1785 public key credential has an identifier which is unique (or extremely
 1786 unlikely to be duplicated) among all public key credentials. Each
 1787 credential is also associated with a Relying Party, whose identity is
 1788 represented by a Relying Party Identifier (RP ID).
 1789
 1790 Each authenticator has an AAGUID, which is a 128-bit identifier that
 1791 indicates the type (e.g. make and model) of the authenticator. The
 1792 AAGUID MUST be chosen by the manufacturer to be identical across all
 1793 substantially identical authenticators made by that manufacturer, and
 1794 different (with probability $1-2^{-128}$ or greater) from the AAGUIDs of
 1795 all other types of authenticators. The RP MAY use the AAGUID to infer
 1796 certain properties of the authenticator, such as certification level
 1797 and strength of key protection, using information from other sources.
 1798
 1799 The primary function of the authenticator is to provide WebAuthn
 1800 signatures, which are bound to various contextual data. These data are
 1801 observed, and added at different levels of the stack as a signature
 1802 request passes from the server to the authenticator. In verifying a
 1803 signature, the server checks these bindings against expected values.

1802 * nfc - the respective Authenticator may be contacted over Near Field
 1803 Communication (NFC).
 1804 * ble - the respective Authenticator may be contacted over Bluetooth
 1805 Smart (Bluetooth Low Energy / BLE).
 1806
 1807 4.7.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)
 1808
 1809 typedef long COSEAlgorithmIdentifier;
 1810
 1811 A COSEAlgorithmIdentifier's value is a number identifying a
 1812 cryptographic algorithm. The algorithm identifiers SHOULD be values
 1813 registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],
 1814 for instance, -7 for "ES256" and -257 for "RS256".
 1815
 1816 5. WebAuthn Authenticator model
 1817
 1818 The API defined in this specification implies a specific abstract
 1819 functional model for an authenticator. This section describes the
 1820 authenticator model.
 1821
 1822 Client platforms may implement and expose this abstract model in any
 1823 way desired. However, the behavior of the client's Web Authentication
 1824 API implementation, when operating on the authenticators supported by
 1825 that platform, MUST be indistinguishable from the behavior specified in
 1826 4 Web Authentication API.
 1827
 1828 For authenticators, this model defines the logical operations that they
 1829 must support, and the data formats that they expose to the client and
 1830 the Relying Party. However, it does not define the details of how
 1831 authenticators communicate with the client platform, unless they are
 1832 required for interoperability with Relying Parties. For instance, this
 1833 abstract model does not define protocols for connecting authenticators
 1834 to clients over transports such as USB or NFC. Similarly, this abstract
 1835 model does not define specific error codes or methods of returning
 1836 them; however, it does define error behavior in terms of the needs of
 1837 the client. Therefore, specific error codes are mentioned as a means of
 1838 showing which error conditions must be distinguishable (or not) from
 1839 each other in order to enable a compliant and secure client
 1840 implementation.
 1841
 1842 In this abstract model, the authenticator provides key management and
 1843 cryptographic signatures. It may be embedded in the WebAuthn client, or
 1844 housed in a separate device entirely. The authenticator may itself
 1845 contain a cryptographic module which operates at a higher security
 1846 level than the rest of the authenticator. This is particularly
 1847 important for authenticators that are embedded in the WebAuthn client,
 1848 as in those cases this cryptographic module (which may, for example, be
 1849 a TPM) could be considered more trustworthy than the rest of the
 1850 authenticator.
 1851
 1852 Each authenticator stores some number of public key credentials. Each
 1853 public key credential has an identifier which is unique (or extremely
 1854 unlikely to be duplicated) among all public key credentials. Each
 1855 credential is also associated with a Relying Party, whose identity is
 1856 represented by a Relying Party Identifier (RP ID).
 1857
 1858 Each authenticator has an AAGUID, which is a 128-bit identifier that
 1859 indicates the type (e.g. make and model) of the authenticator. The
 1860 AAGUID MUST be chosen by the manufacturer to be identical across all
 1861 substantially identical authenticators made by that manufacturer, and
 1862 different (with probability $1-2^{-128}$ or greater) from the AAGUIDs of
 1863 all other types of authenticators. The RP MAY use the AAGUID to infer
 1864 certain properties of the authenticator, such as certification level
 1865 and strength of key protection, using information from other sources.
 1866
 1867 The primary function of the authenticator is to provide WebAuthn
 1868 signatures, which are bound to various contextual data. These data are
 1869 observed, and added at different levels of the stack as a signature
 1870 request passes from the server to the authenticator. In verifying a
 1871 signature, the server checks these bindings against expected values.

1804 These contextual bindings are divided in two: Those added by the RP or
1805 the client, referred to as client data; and those added by the
1806 authenticator, referred to as the authenticator data. The authenticator
1807 signs over the client data, but is otherwise not interested in its
1808 contents. To save bandwidth and processing requirements on the
1809 authenticator, the client hashes the client data and sends only the
1810 result to the authenticator. The authenticator signs over the
1811 combination of the hash of the serialized client data, and its own
1812 authenticator data.

1813
1814 The goals of this design can be summarized as follows.

- 1815 * The scheme for generating signatures should accommodate cases where
- 1816 the link between the client platform and authenticator is very
- 1817 limited, in bandwidth and/or latency. Examples include Bluetooth
- 1818 Low Energy and Near-Field Communication.
- 1819 * The data processed by the authenticator should be small and easy to
- 1820 interpret in low-level code. In particular, authenticators should
- 1821 not have to parse high-level encodings such as JSON.
- 1822 * Both the client platform and the authenticator should have the
- 1823 flexibility to add contextual bindings as needed.
- 1824 * The design aims to reuse as much as possible of existing encoding
- 1825 formats in order to aid adoption and implementation.
- 1826

1827 Authenticators produce cryptographic signatures for two distinct
1828 purposes:

- 1829 1. An attestation signature is produced when a new public key
- 1830 credential is created via an authenticatorMakeCredential operation.
- 1831 An attestation signature provides cryptographic proof of certain
- 1832 properties of the the authenticator and the credential. For
- 1833 instance, an attestation signature asserts the authenticator type
- 1834 (as denoted by its AAGUID) and the credential public key. The
- 1835 attestation signature is signed by an attestation private key,
- 1836 which is chosen depending on the type of attestation desired. For
- 1837 more details on attestation, see 5.3 Attestation.
- 1838 2. An assertion signature is produced when the
- 1839 authenticatorGetAssertion method is invoked. It represents an
- 1840 assertion by the authenticator that the user has consented to a
- 1841 specific transaction, such as logging in, or completing a purchase.
- 1842 Thus, an assertion signature asserts that the authenticator
- 1843 possessing a particular credential private key has established, to
- 1844 the best of its ability, that the user requesting this transaction
- 1845 is the same user who consented to creating that particular public
- 1846 key credential. It also asserts additional information, termed
- 1847 client data, that may be useful to the caller, such as the means by
- 1848 which user consent was provided, and the prompt shown to the user
- 1849 by the authenticator. The assertion signature format is illustrated
- 1850 in Figure 2, below.

1851
1852 The formats of these signatures, as well as the procedures for
1853 generating them, are specified below.

1854 5.1. Authenticator data

1855 The authenticator data structure encodes contextual bindings made by
1856 the authenticator. These bindings are controlled by the authenticator
1857 itself, and derive their trust from the Relying Party's assessment of
1858 the security properties of the authenticator. In one extreme case, the
1859 authenticator may be embedded in the client, and its bindings may be no
1860 more trustworthy than the client data. At the other extreme, the
1861 authenticator may be a discrete entity with high-security hardware and
1862 software, connected to the client over a secure channel. In both cases,
1863 the Relying Party receives the authenticator data in the same format,
1864 and uses its knowledge of the authenticator to make trust decisions.

1865 The authenticator data has a compact but extensible encoding. This is
1866 desired since authenticators can be devices with limited capabilities
1867 and low power requirements, with much simpler software stacks than the
1868 client platform components.

1869 The authenticator data structure is a byte array of 37 bytes or more,

1870
1871
1872
1873

1872 These contextual bindings are divided in two: Those added by the RP or
1873 the client, referred to as client data; and those added by the
1874 authenticator, referred to as the authenticator data. The authenticator
1875 signs over the client data, but is otherwise not interested in its
1876 contents. To save bandwidth and processing requirements on the
1877 authenticator, the client hashes the client data and sends only the
1878 result to the authenticator. The authenticator signs over the
1879 combination of the hash of the serialized client data, and its own
1880 authenticator data.

1881
1882 The goals of this design can be summarized as follows.

- 1883 * The scheme for generating signatures should accommodate cases where
- 1884 the link between the client platform and authenticator is very
- 1885 limited, in bandwidth and/or latency. Examples include Bluetooth
- 1886 Low Energy and Near-Field Communication.
- 1887 * The data processed by the authenticator should be small and easy to
- 1888 interpret in low-level code. In particular, authenticators should
- 1889 not have to parse high-level encodings such as JSON.
- 1890 * Both the client platform and the authenticator should have the
- 1891 flexibility to add contextual bindings as needed.
- 1892 * The design aims to reuse as much as possible of existing encoding
- 1893 formats in order to aid adoption and implementation.
- 1894

1895 Authenticators produce cryptographic signatures for two distinct
1896 purposes:

- 1897 1. An attestation signature is produced when a new public key
- 1898 credential is created via an authenticatorMakeCredential operation.
- 1899 An attestation signature provides cryptographic proof of certain
- 1900 properties of the the authenticator and the credential. For
- 1901 instance, an attestation signature asserts the authenticator type
- 1902 (as denoted by its AAGUID) and the credential public key. The
- 1903 attestation signature is signed by an attestation private key,
- 1904 which is chosen depending on the type of attestation desired. For
- 1905 more details on attestation, see 5.3 Attestation.
- 1906 2. An assertion signature is produced when the
- 1907 authenticatorGetAssertion method is invoked. It represents an
- 1908 assertion by the authenticator that the user has consented to a
- 1909 specific transaction, such as logging in, or completing a purchase.
- 1910 Thus, an assertion signature asserts that the authenticator
- 1911 possessing a particular credential private key has established, to
- 1912 the best of its ability, that the user requesting this transaction
- 1913 is the same user who consented to creating that particular public
- 1914 key credential. It also asserts additional information, termed
- 1915 client data, that may be useful to the caller, such as the means by
- 1916 which user consent was provided, and the prompt shown to the user
- 1917 by the authenticator. The assertion signature format is illustrated
- 1918 in Figure 2, below.

1919
1920 The formats of these signatures, as well as the procedures for
1921 generating them, are specified below.

1922 5.1. Authenticator data

1923 The authenticator data structure encodes contextual bindings made by
1924 the authenticator. These bindings are controlled by the authenticator
1925 itself, and derive their trust from the Relying Party's assessment of
1926 the security properties of the authenticator. In one extreme case, the
1927 authenticator may be embedded in the client, and its bindings may be no
1928 more trustworthy than the client data. At the other extreme, the
1929 authenticator may be a discrete entity with high-security hardware and
1930 software, connected to the client over a secure channel. In both cases,
1931 the Relying Party receives the authenticator data in the same format,
1932 and uses its knowledge of the authenticator to make trust decisions.

1933 The authenticator data has a compact but extensible encoding. This is
1934 desired since authenticators can be devices with limited capabilities
1935 and low power requirements, with much simpler software stacks than the
1936 client platform components.

1937 The authenticator data structure is a byte array of 37 bytes or more,

1938
1939
1940
1941

1874 as follows.
1875
1876 Length (in bytes) Description
1877 32 SHA-256 hash of the RP ID associated with the credential.
1878 1 Flags (bit 0 is the least significant bit):
1879 * Bit 0: User Present (UP) result.
1880 + 1 means the user is present.
1881 + 0 means the user is not present.
1882 * Bit 1: Reserved for future use (RFU1).
1883 * Bit 2: User Verified (UV) result.
1884 + 1 means the user is verified.
1885 + 0 means the user is not verified.
1886 * Bits 3-5: Reserved for future use (RFU2).
1887 * Bit 6: Attestation data included (AT).
1888 + Indicates whether the authenticator added attestation data.
1889 * Bit 7: Extension data included (ED).
1890 + Indicates if the authenticator data has extensions.
1891
1892 4 Signature counter (signCount), 32-bit unsigned big-endian integer.
1893 variable (if present) attestation data (if present). See 5.3.1
1894 Attestation data for details. Its length depends on the length of the
1895 credential public key and credential ID being attested.
1896 variable (if present) Extension-defined authenticator data. This is a
1897 CBOR [RFC7049] map with extension identifiers as keys, and
1898 authenticator extension outputs as values. See 8 WebAuthn Extensions
1899 for details.
1900
1901 The RP ID is originally received from the client when the credential is
1902 created, and again when an assertion is generated. However, it differs
1903 from other client data in some important ways. First, unlike the client
1904 data, the RP ID of a credential does not change between operations but
1905 instead remains the same for the lifetime of that credential. Secondly,
1906 it is validated by the authenticator during the
1907 authenticatorGetAssertion operation, by verifying that the RP ID
1908 associated with the requested credential exactly matches the RP ID
1909 supplied by the client, and that the RP ID is a registrable domain
1910 suffix of or is equal to the effective domain of the RP's origin's
1911 effective domain.
1912
1913 The UP flag SHALL be set if and only if the authenticator detected a
1914 user through an authenticator specific gesture. The RFU bits SHALL be
1915 set to zero.
1916
1917 For attestation signatures, the authenticator MUST set the AT flag and
1918 include the attestation data. For authentication signatures, the AT
1919 flag MUST NOT be set and the attestation data MUST NOT be included.
1920
1921 If the authenticator does not include any extension data, it MUST set
1922 the ED flag to zero, and to one if extension data is included.
1923
1924 The figure below shows a visual representation of the authenticator
1925 data structure.
1926 [fido-signature-formats-figure1.svg] Authenticator data layout.
1927
1928 Note that the authenticator data describes its own length: If the AT
1929 and ED flags are not set, it is always 37 bytes long. The attestation
1930 data (which is only present if the AT flag is set) describes its own
1931 length. If the ED flag is set, then the total length is 37 bytes plus
1932 the length of the attestation data, plus the length of the CBOR map
1933 that follows.
1934
1935 5.2. Authenticator operations
1936
1937 A client must connect to an authenticator in order to invoke any of the
1938 operations of that authenticator. This connection defines an
1939 authenticator session. An authenticator must maintain isolation between
1940 sessions. It may do this by only allowing one session to exist at any
1941 particular time, or by providing more complicated session management.
1942
1943 The following operations can be invoked by the client in an

1942 as follows.
1943
1944 Length (in bytes) Description
1945 32 SHA-256 hash of the RP ID associated with the credential.
1946 1 Flags (bit 0 is the least significant bit):
1947 * Bit 0: User Present (UP) result.
1948 + 1 means the user is present.
1949 + 0 means the user is not present.
1950 * Bit 1: Reserved for future use (RFU1).
1951 * Bit 2: User Verified (UV) result.
1952 + 1 means the user is verified.
1953 + 0 means the user is not verified.
1954 * Bits 3-5: Reserved for future use (RFU2).
1955 * Bit 6: Attestation data included (AT).
1956 + Indicates whether the authenticator added attestation data.
1957 * Bit 7: Extension data included (ED).
1958 + Indicates if the authenticator data has extensions.
1959
1960 4 Signature counter (signCount), 32-bit unsigned big-endian integer.
1961 variable (if present) attestation data (if present). See 5.3.1
1962 Attestation data for details. Its length depends on the length of the
1963 credential public key and credential ID being attested.
1964 variable (if present) Extension-defined authenticator data. This is a
1965 CBOR [RFC7049] map with extension identifiers as keys, and
1966 authenticator extension outputs as values. See 8 WebAuthn Extensions
1967 for details.
1968
1969 The RP ID is originally received from the client when the credential is
1970 created, and again when an assertion is generated. However, it differs
1971 from other client data in some important ways. First, unlike the client
1972 data, the RP ID of a credential does not change between operations but
1973 instead remains the same for the lifetime of that credential. Secondly,
1974 it is validated by the authenticator during the
1975 authenticatorGetAssertion operation, by verifying that the RP ID
1976 associated with the requested credential exactly matches the RP ID
1977 supplied by the client, and that the RP ID is a registrable domain
1978 suffix of or is equal to the effective domain of the RP's origin's
1979 effective domain.
1980
1981 The UP flag SHALL be set if and only if the authenticator detected a
1982 user through an authenticator specific gesture. The RFU bits SHALL be
1983 set to zero.
1984
1985 For attestation signatures, the authenticator MUST set the AT flag and
1986 include the attestation data. For authentication signatures, the AT
1987 flag MUST NOT be set and the attestation data MUST NOT be included.
1988
1989 If the authenticator does not include any extension data, it MUST set
1990 the ED flag to zero, and to one if extension data is included.
1991
1992 The figure below shows a visual representation of the authenticator
1993 data structure.
1994 [fido-signature-formats-figure1.svg] Authenticator data layout.
1995
1996 Note that the authenticator data describes its own length: If the AT
1997 and ED flags are not set, it is always 37 bytes long. The attestation
1998 data (which is only present if the AT flag is set) describes its own
1999 length. If the ED flag is set, then the total length is 37 bytes plus
2000 the length of the attestation data, plus the length of the CBOR map
2001 that follows.
2002
2003 5.2. Authenticator operations
2004
2005 A client must connect to an authenticator in order to invoke any of the
2006 operations of that authenticator. This connection defines an
2007 authenticator session. An authenticator must maintain isolation between
2008 sessions. It may do this by only allowing one session to exist at any
2009 particular time, or by providing more complicated session management.
2010
2011 The following operations can be invoked by the client in an

1944 authenticator session.
 1945
 1946 5.2.1. The authenticatorMakeCredential operation
 1947
 1948 This operation must be invoked in an authenticator session which has no
 1949 other operations in progress. It takes the following input parameters:
 1950 * The caller's RP ID, as determined by the user agent and the client.
 1951 * The hash of the serialized client data, provided by the client.
 1952 * The Relying Party's PublicKeyCredentialRpEntity.
 1953
 1954 * The user account's PublicKeyCredentialUserEntity.
 1955 * A sequence of pairs of PublicKeyCredentialType and
 1956 COSEAlgorithmIdentifier requested by the Relying Party. This
 1957 sequence is ordered from most preferred to least preferred. The
 1958 platform makes a best-effort to create the most preferred
 1959 credential that it can.
 1960 * An optional list of PublicKeyCredentialDescriptor objects provided
 1961 by the Relying Party with the intention that, if any of these are
 1962 known to the authenticator, it should not create a new credential.
 1963 excludeCredentialDescriptorList contains a list of known
 1964 credentials.
 1965 * The requireResidentKey member of the options.authenticatorSelection
 1966 dictionary.
 1967 * The requireUserVerification member of the
 1968 options.authenticatorSelection dictionary.
 1969 * Extension data created by the client based on the extensions
 1970 requested by the Relying Party, if any.
 1971
 1972 When this operation is invoked, the authenticator must perform the
 1973 following procedure:
 1974 * Check if all the supplied parameters are syntactically well-formed
 1975 and of the correct length. If not, return an error code equivalent
 1976 to "UnknownError" and terminate the operation.
 1977 * Check if at least one of the specified combinations of
 1978 PublicKeyCredentialType and cryptographic parameters is supported.
 1979 If not, return an error code equivalent to "NotSupportedError" and
 1980 terminate the operation.
 1981 * Check if a credential matching any of the supplied
 1982 PublicKeyCredential identifiers is present on this authenticator.
 1983 If so, return an error code equivalent to "NotAllowedError" and
 1984 terminate the operation.
 1985 * If requireResidentKey is true and the authenticator cannot store a
 1986 Client-side-resident Credential Private Key, return an error code
 1987 equivalent to "ConstraintError" and terminate the operation.
 1988 * If requireUserVerification is true and the authenticator cannot
 1989 perform user verification, return an error code equivalent to
 1990 "ConstraintError" and terminate the operation.
 1991 * Prompt the user for consent to create a new credential. The prompt
 1992 for obtaining this consent is shown by the authenticator if it has
 1993 its own output capability, or by the user agent otherwise. If the
 1994 user denies consent, return an error code equivalent to
 1995 "NotAllowedError" and terminate the operation.
 1996 * Once user consent has been obtained, generate a new credential
 1997 object:
 1998 + Generate a set of cryptographic keys using the most preferred
 1999 combination of PublicKeyCredentialType and cryptographic
 2000 parameters supported by this authenticator.
 2001 + Generate an identifier for this credential, such that this
 2002 identifier is globally unique with high probability across all
 2003 credentials with the same type across all authenticators.
 2004 + Associate the credential with the specified RP ID and the
 2005 user's account identifier user.id.
 2006 + Delete any older credentials with the same RP ID and user.id
 2007 that are stored locally by the authenticator.
 2008 * If any error occurred while creating the new credential object,
 2009 return an error code equivalent to "UnknownError" and terminate the
 2010 operation.
 2011 * Process all the supported extensions requested by the client, and
 2012 generate the authenticator data with attestation data as specified
 in 5.1 Authenticator data. Use this authenticator data and the

2012 authenticator session.
 2013
 2014 5.2.1. The authenticatorMakeCredential operation
 2015
 2016 This operation must be invoked in an authenticator session which has no
 2017 other operations in progress. It takes the following input parameters:
 2018
 2019 * The hash of the serialized client data, provided by the client.
 2020 * The Relying Party's PublicKeyCredentialEntity. This contains the
 2021 caller's RP ID, as determined by the user agent and the client.
 2022 * The user account's PublicKeyCredentialUserEntity.
 2023 * A sequence of pairs of PublicKeyCredentialType and
 2024 COSEAlgorithmIdentifier requested by the Relying Party. This
 2025 sequence is ordered from most preferred to least preferred. The
 2026 platform makes a best-effort to create the most preferred
 2027 credential that it can.
 2028 * An optional list of PublicKeyCredentialDescriptor objects provided
 2029 by the Relying Party with the intention that, if any of these are
 2030 known to the authenticator, it should not create a new credential.
 2031 excludeCredentialDescriptorList contains a list of known
 2032 credentials.
 2033 * The requireResidentKey member of the options.authenticatorSelection
 2034 dictionary.
 2035 * The requireUserVerification member of the
 2036 options.authenticatorSelection dictionary.
 2037 * Extension data created by the client based on the extensions
 2038 requested by the Relying Party, if any.
 2039
 2040 When this operation is invoked, the authenticator must perform the
 2041 following procedure:
 2042 * Check if all the supplied parameters are syntactically well-formed
 2043 and of the correct length. If not, return an error code equivalent
 2044 to "UnknownError" and terminate the operation.
 2045 * Check if at least one of the specified combinations of
 2046 PublicKeyCredentialType and cryptographic parameters is supported.
 2047 If not, return an error code equivalent to "NotSupportedError" and
 2048 terminate the operation.
 2049 * Check if a credential matching any of the supplied
 2050 PublicKeyCredential identifiers is present on this authenticator.
 2051 If so, return an error code equivalent to "NotAllowedError" and
 2052 terminate the operation.
 2053 * If requireResidentKey is true and the authenticator cannot store a
 2054 Client-side-resident Credential Private Key, return an error code
 2055 equivalent to "ConstraintError" and terminate the operation.
 2056 * If requireUserVerification is true and the authenticator cannot
 2057 perform user verification, return an error code equivalent to
 2058 "ConstraintError" and terminate the operation.
 2059 * Prompt the user for consent to create a new credential. The prompt
 2060 for obtaining this consent is shown by the authenticator if it has
 2061 its own output capability, or by the user agent otherwise. If the
 2062 user denies consent, return an error code equivalent to
 2063 "NotAllowedError" and terminate the operation.
 2064 * Once user consent has been obtained, generate a new credential
 2065 object:
 2066 + Generate a set of cryptographic keys using the most preferred
 2067 combination of PublicKeyCredentialType and cryptographic
 2068 parameters supported by this authenticator.
 2069 + Generate an identifier for this credential, such that this
 2070 identifier is globally unique with high probability across all
 2071 credentials with the same type across all authenticators.
 2072 + Associate the credential with the specified RP ID and the
 2073 user's account identifier user.id.
 2074 + Delete any older credentials with the same RP ID and user.id
 2075 that are stored locally by the authenticator.
 2076 * If any error occurred while creating the new credential object,
 2077 return an error code equivalent to "UnknownError" and terminate the
 2078 operation.
 2079 * Process all the supported extensions requested by the client, and
 2080 generate the authenticator data with attestation data as specified
 in 5.1 Authenticator data. Use this authenticator data and the

2013 hash of the serialized client data to create an attestation object
 2014 for the new credential using the procedure specified in 5.3.4
 2015 Generating an Attestation Object. For more details on attestation,
 2016 see 5.3 Attestation.
 2017
 2018 On successful completion of this operation, the authenticator returns
 2019 the attestation object to the client.
 2020
 2021 5.2.2. The authenticatorGetAssertion operation
 2022
 2023 This operation must be invoked in an authenticator session which has no
 2024 other operations in progress. It takes the following input parameters:
 2025 * The caller's RP ID, as determined by the user agent and the client.
 2026 * The hash of the serialized client data, provided by the client.
 2027 * A list of credentials acceptable to the Relying Party (possibly
 2028 filtered by the client), if any.
 2029 * Extension data created by the client based on the extensions
 2030 requested by the Relying Party, if any.
 2031
 2032 When this method is invoked, the authenticator must perform the
 2033 following procedure:
 2034 * Check if all the supplied parameters are syntactically well-formed
 2035 and of the correct length. If not, return an error code equivalent
 2036 to "UnknownError" and terminate the operation.
 2037 * If a list of credentials was supplied by the client, filter it by
 2038 removing those credentials that are not present on this
 2039 authenticator. If no list was supplied, create a list with all
 2040 credentials stored for the caller's RP ID (as determined by an
 2041 exact match of the RP ID).
 2042 * If the previous step resulted in an empty list, return an error
 2043 code equivalent to "NotAllowedError" and terminate the operation.
 2044 * Prompt the user to select a credential from among the above list.
 2045 Obtain user consent for using this credential. The prompt for
 2046 obtaining this consent may be shown by the authenticator if it has
 2047 its own output capability, or by the user agent otherwise.
 2048 * Process all the supported extensions requested by the client, and
 2049 generate the authenticator data as specified in 5.1 Authenticator
 2050 data, though without attestation data. Concatenate this
 2051 authenticator data with the hash of the serialized client data to
 2052 generate an assertion signature using the private key of the
 2053 selected credential as shown in Figure 2, below. A simple,
 2054 undelimited concatenation is safe to use here because the
 2055 authenticator data describes its own length. The hash of the
 2056 serialized client data (which potentially has a variable length) is
 2057 always the last element.
 2058 * If any error occurred while generating the assertion signature,
 2059 return an error code equivalent to "UnknownError" and terminate the
 2060 operation.
 2061
 2062 [fido-signature-formats-figure2.svg] Generating an assertion signature.
 2063
 2064 On successful completion, the authenticator returns to the user agent:
 2065 * The identifier of the credential (credential ID) used to generate
 2066 the assertion signature.
 2067
 2068 * The authenticator data used to generate the assertion signature.
 2069 * The assertion signature.
 2070
 2071 If the authenticator cannot find any credential corresponding to the
 2072 specified Relying Party that matches the specified criteria, it
 2073 terminates the operation and returns an error.
 2074
 2075 If the user refuses consent, the authenticator returns an appropriate
 2076 error status to the client.
 2077
 2078 5.2.3. The authenticatorCancel operation

2081 hash of the serialized client data to create an attestation object
 2082 for the new credential using the procedure specified in 5.3.4
 2083 Generating an Attestation Object. For more details on attestation,
 2084 see 5.3 Attestation.
 2085
 2086 On successful completion of this operation, the authenticator returns
 2087 the attestation object to the client.
 2088
 2089 5.2.2. The authenticatorGetAssertion operation
 2090
 2091 This operation must be invoked in an authenticator session which has no
 2092 other operations in progress. It takes the following input parameters:
 2093 * The caller's RP ID, as determined by the user agent and the client.
 2094 * The hash of the serialized client data, provided by the client.
 2095 * A list of credentials acceptable to the Relying Party (possibly
 2096 filtered by the client), if any.
 2097 * Extension data created by the client based on the extensions
 2098 requested by the Relying Party, if any.
 2099
 2100 When this method is invoked, the authenticator must perform the
 2101 following procedure:
 2102 * Check if all the supplied parameters are syntactically well-formed
 2103 and of the correct length. If not, return an error code equivalent
 2104 to "UnknownError" and terminate the operation.
 2105 * If a list of credentials was supplied by the client, filter it by
 2106 removing those credentials that are not present on this
 2107 authenticator. If no list was supplied, create a list with all
 2108 credentials stored for the caller's RP ID (as determined by an
 2109 exact match of the RP ID).
 2110 * If the previous step resulted in an empty list, return an error
 2111 code equivalent to "NotAllowedError" and terminate the operation.
 2112 * Prompt the user to select a credential from among the above list.
 2113 Obtain user consent for using this credential. The prompt for
 2114 obtaining this consent may be shown by the authenticator if it has
 2115 its own output capability, or by the user agent otherwise.
 2116 * Process all the supported extensions requested by the client, and
 2117 generate the authenticator data as specified in 5.1 Authenticator
 2118 data, though without attestation data. Concatenate this
 2119 authenticator data with the hash of the serialized client data to
 2120 generate an assertion signature using the private key of the
 2121 selected credential as shown in Figure 2, below. A simple,
 2122 undelimited concatenation is safe to use here because the
 2123 authenticator data describes its own length. The hash of the
 2124 serialized client data (which potentially has a variable length) is
 2125 always the last element.
 2126 * If any error occurred while generating the assertion signature,
 2127 return an error code equivalent to "UnknownError" and terminate the
 2128 operation.
 2129
 2130 [fido-signature-formats-figure2.svg] Generating an assertion signature.
 2131
 2132 On successful completion, the authenticator returns to the user agent:
 2133 * The identifier of the credential (credential ID) used to generate
 2134 the assertion signature, if either a list of credentials of length
 2135 2 or greater was supplied by the client, or no such list was
 2136 supplied.
 2137 Note: If the client supplies a list of exactly one credential and
 2138 it was successfully employed, then its credential ID is not
 2139 returned since the client already knows it.
 2140 * The authenticator data used to generate the assertion signature.
 2141 * The assertion signature.
 2142
 2143 If the authenticator cannot find any credential corresponding to the
 2144 specified Relying Party that matches the specified criteria, it
 2145 terminates the operation and returns an error.
 2146
 2147 If the user refuses consent, the authenticator returns an appropriate
 2148 error status to the client.
 2149
 2150 5.2.3. The authenticatorCancel operation

207E This operation takes no input parameters and returns no result.
 207F
 208C
 2081 When this operation is invoked by the client in an authenticator
 2082 session, it has the effect of terminating any
 2083 authenticatorMakeCredential or authenticatorGetAssertion operation
 2084 currently in progress in that authenticator session. The authenticator
 2085 stops prompting for, or accepting, any user input related to
 2086 authorizing the canceled operation. The client ignores any further
 2087 responses from the authenticator for the canceled operation.
 2088
 2089 This operation is ignored if it is invoked in an authenticator session
 2090 which does not have an authenticatorMakeCredential or
 2091 authenticatorGetAssertion operation currently in progress.
 2092
 2093 **5.3. Attestation**
 2094
 2095 Authenticators must also provide some form of attestation. The basic
 2096 requirement is that the authenticator can produce, for each credential
 2097 public key, an attestation statement verifiable by the Relying Party.
 2098 Typically, this attestation statement contains a signature by an
 2099 attestation private key over the attested credential public key and a
 2100 challenge, as well as a certificate or similar data providing
 2101 provenance information for the attestation public key, enabling the
 2102 Relying Party to make a trust decision. However, if an attestation key
 2103 pair is not available, then the authenticator **MUST** perform self
 2104 attestation of the credential public key with the corresponding
 2105 credential private key. All this information is returned by
 2106 authenticators any time a new public key credential is generated, in
 2107 the overall form of an attestation object. The relationship of the
 2108 attestation object with authenticator data (containing attestation
 2109 data) and the attestation statement is illustrated in figure 3, below.
 2110 [fido-attestation-structures.svg] Attestation object layout
 2111 illustrating the included authenticator data (containing attestation
 2112 data) and the attestation statement.
 2113
 2114 This figure illustrates only the packed attestation statement format.
 2115 Several additional attestation statement formats are defined in 7
 2116 Defined Attestation Statement Formats.
 2117
 2118 An important component of the attestation object is the attestation
 2119 statement. This is a specific type of signed data object, containing
 2120 statements about a public key credential itself and the authenticator
 2121 that created it. It contains an attestation signature created using the
 2122 key of the attesting authority (except for the case of self
 2123 attestation, when it is created using the credential private key). In
 2124 order to correctly interpret an attestation statement, a Relying Party
 2125 needs to understand these two aspects of attestation:
 2126 1. The attestation statement format is the manner in which the
 2127 signature is represented and the various contextual bindings are
 2128 incorporated into the attestation statement by the authenticator.
 2129 In other words, this defines the syntax of the statement. Various
 2130 existing devices and platforms (such as TPMs and the Android OS)
 2131 have previously defined attestation statement formats. This
 2132 specification supports a variety of such formats in an extensible
 2133 way, as defined in 5.3.2 Attestation Statement Formats.
 2134 2. The attestation type defines the semantics of attestation
 2135 statements and their underlying trust models. Specifically, it
 2136 defines how a Relying Party establishes trust in a particular
 2137 attestation statement, after verifying that it is cryptographically
 2138 valid. This specification supports a number of attestation types,
 2139 as described in 5.3.3 Attestation Types.
 2140
 2141 In general, there is no simple mapping between attestation statement
 2142 formats and attestation types. For example, the "packed" attestation
 2143 statement format defined in 7.2 Packed Attestation Statement Format
 2144 can be used in conjunction with all attestation types, while other
 2145 formats and types have more limited applicability.
 2146
 2147 The privacy, security and operational characteristics of attestation

2151 This operation takes no input parameters and returns no result.
 2152
 2153
 2154 When this operation is invoked by the client in an authenticator
 2155 session, it has the effect of terminating any
 2156 authenticatorMakeCredential or authenticatorGetAssertion operation
 2157 currently in progress in that authenticator session. The authenticator
 2158 stops prompting for, or accepting, any user input related to
 2159 authorizing the canceled operation. The client ignores any further
 2160 responses from the authenticator for the canceled operation.
 2161
 2162 This operation is ignored if it is invoked in an authenticator session
 2163 which does not have an authenticatorMakeCredential or
 2164 authenticatorGetAssertion operation currently in progress.
 2165
 2166 **5.3. Attestation**
 2167
 2168 Authenticators must also provide some form of attestation. The basic
 2169 requirement is that the authenticator can produce, for each credential
 2170 public key, an attestation statement verifiable by the Relying Party.
 2171 Typically, this attestation statement contains a signature by an
 2172 attestation private key over the attested credential public key and a
 2173 challenge, as well as a certificate or similar data providing
 2174 provenance information for the attestation public key, enabling the
 2175 Relying Party to make a trust decision. However, if an attestation key
 2176 pair is not available, then the authenticator **MUST** perform self
 2177 attestation of the credential public key with the corresponding
 2178 credential private key. All this information is returned by
 2179 authenticators any time a new public key credential is generated, in
 2180 the overall form of an attestation object. The relationship of the
 2181 attestation object with authenticator data (containing attestation
 2182 data) and the attestation statement is illustrated in figure 3, below.
 2183 [fido-attestation-structures.svg] Attestation object layout
 2184 illustrating the included authenticator data (containing attestation
 2185 data) and the attestation statement.
 2186
 2187 This figure illustrates only the packed attestation statement format.
 2188 Several additional attestation statement formats are defined in 7
 2189 Defined Attestation Statement Formats.
 2190
 2191 An important component of the attestation object is the attestation
 2192 statement. This is a specific type of signed data object, containing
 2193 statements about a public key credential itself and the authenticator
 2194 that created it. It contains an attestation signature created using the
 2195 key of the attesting authority (except for the case of self
 2196 attestation, when it is created using the credential private key). In
 2197 order to correctly interpret an attestation statement, a Relying Party
 2198 needs to understand these two aspects of attestation:
 2199 1. The attestation statement format is the manner in which the
 2200 signature is represented and the various contextual bindings are
 2201 incorporated into the attestation statement by the authenticator.
 2202 In other words, this defines the syntax of the statement. Various
 2203 existing devices and platforms (such as TPMs and the Android OS)
 2204 have previously defined attestation statement formats. This
 2205 specification supports a variety of such formats in an extensible
 2206 way, as defined in 5.3.2 Attestation Statement Formats.
 2207 2. The attestation type defines the semantics of attestation
 2208 statements and their underlying trust models. Specifically, it
 2209 defines how a Relying Party establishes trust in a particular
 2210 attestation statement, after verifying that it is cryptographically
 2211 valid. This specification supports a number of attestation types,
 2212 as described in 5.3.3 Attestation Types.
 2213
 2214 In general, there is no simple mapping between attestation statement
 2215 formats and attestation types. For example, the "packed" attestation
 2216 statement format defined in 7.2 Packed Attestation Statement Format
 2217 can be used in conjunction with all attestation types, while other
 2218 formats and types have more limited applicability.
 2219
 2220 The privacy, security and operational characteristics of attestation

2148 depend on:

- 2149 * The attestation type, which determines the trust model,
- 2150 * The attestation statement format, which may constrain the strength
- 2151 of the attestation by limiting what can be expressed in an
- 2152 attestation statement, and
- 2153 * The characteristics of the individual authenticator, such as its
- 2154 construction, whether part or all of it runs in a secure operating
- 2155 environment, and so on.

2157 It is expected that most authenticators will support a small number of

2158 attestation types and attestation statement formats, while Relying

2159 Parties will decide what attestation types are acceptable to them by

2160 policy. Relying Parties will also need to understand the

2161 characteristics of the authenticators that they trust, based on

2162 information they have about these authenticators. For example, the FIDO

2163 Metadata Service [FIDOMetadataService] provides one way to access such

2164 information.

2165

2166 5.3.1. Attestation data

2167

2168 Attestation data is added to the authenticator data when generating an

2169 attestation object for a given credential. It has the following format:

2170

2171 Length (in bytes)	2171 Description
2172 16	2172 The AAGUID of the authenticator.
2173 2	2173 Byte length L of Credential ID
2174 L	2174 Credential ID
2175 variable	2175 The credential public key encoded in COSE_Key format, as
2176 defined in Section 7 of [RFC8152]. The encoded credential public key	
2177 MUST contain the "alg" parameter and MUST NOT contain any other	
2178 optional parameters. The "alg" parameter MUST contain a	
2179 COSEAlgorithmIdentifier value.	

2180

2181 5.3.2. Attestation Statement Formats

2182

2183 As described above, an attestation statement format is a data format

2184 which represents a cryptographic signature by an authenticator over a

2185 set of contextual bindings. Each attestation statement format MUST be

2186 defined using the following template:

- 2187 * Attestation statement format identifier:
- 2188 * Supported attestation types:
- 2189 * Syntax: The syntax of an attestation statement produced in this
- 2190 format, defined using [CDDL] for the extension point \$attStmtFormat
- 2191 defined in 5.3.4 Generating an Attestation Object.
- 2192 * Signing procedure: The signing procedure for computing an
- 2193 attestation statement in this format given the public key
- 2194 credential to be attested, the authenticator data structure
- 2195 containing the authenticator data for the attestation, and the hash
- 2196 of the serialized client data.
- 2197 * Verification procedures: The procedure for verifying an attestation
- 2198 statement, which takes as inputs the authenticator data structure
- 2199 containing the authenticator data claimed to have been used for the
- 2200 attestation and the hash of the serialized client data, and returns
- 2201 either:
- 2202 + An error indicating that the attestation is invalid, or
- 2203 + The attestation type, and the trust path of the attestation.

2204 This trust path is either empty (in case of self attestation),

2205 an identifier of a ECDAA-Issuer public key (in the case of

2206 ECDAA), or a set of X.509 certificates.

2207

2208 The initial list of specified attestation statement formats is in 7

2209 Defined Attestation Statement Formats.

2210

2211 5.3.3. Attestation Types

2212

2213 WebAuthn supports multiple attestation types:

2214

2215 Basic Attestation

2216 In the case of basic attestation [UAFProtocol], the

2217 authenticator's attestation key pair is specific to an

2221 depend on:

- 2222 * The attestation type, which determines the trust model,
- 2223 * The attestation statement format, which may constrain the strength
- 2224 of the attestation by limiting what can be expressed in an
- 2225 attestation statement, and
- 2226 * The characteristics of the individual authenticator, such as its
- 2227 construction, whether part or all of it runs in a secure operating
- 2228 environment, and so on.

2230 It is expected that most authenticators will support a small number of

2231 attestation types and attestation statement formats, while Relying

2232 Parties will decide what attestation types are acceptable to them by

2233 policy. Relying Parties will also need to understand the

2234 characteristics of the authenticators that they trust, based on

2235 information they have about these authenticators. For example, the FIDO

2236 Metadata Service [FIDOMetadataService] provides one way to access such

2237 information.

2238

2239 5.3.1. Attestation data

2240

2241 Attestation data is added to the authenticator data when generating an

2242 attestation object for a given credential. It has the following format:

2243

2244 Length (in bytes)	2244 Description
2245 16	2245 The AAGUID of the authenticator.
2246 2	2246 Byte length L of Credential ID
2247 L	2247 Credential ID
2248 variable	2248 The credential public key encoded in COSE_Key format, as
2249 defined in Section 7 of [RFC8152]. The encoded credential public key	
2250 MUST contain the "alg" parameter and MUST NOT contain any other	
2251 optional parameters. The "alg" parameter MUST contain a	
2252 COSEAlgorithmIdentifier value.	

2253

2254 5.3.2. Attestation Statement Formats

2255

2256 As described above, an attestation statement format is a data format

2257 which represents a cryptographic signature by an authenticator over a

2258 set of contextual bindings. Each attestation statement format MUST be

2259 defined using the following template:

- 2260 * Attestation statement format identifier:
- 2261 * Supported attestation types:
- 2262 * Syntax: The syntax of an attestation statement produced in this
- 2263 format, defined using [CDDL] for the extension point \$attStmtFormat
- 2264 defined in 5.3.4 Generating an Attestation Object.
- 2265 * Signing procedure: The signing procedure for computing an
- 2266 attestation statement in this format given the public key
- 2267 credential to be attested, the authenticator data structure
- 2268 containing the authenticator data for the attestation, and the hash
- 2269 of the serialized client data.
- 2270 * Verification procedures: The procedure for verifying an attestation
- 2271 statement, which takes as inputs the authenticator data structure
- 2272 containing the authenticator data claimed to have been used for the
- 2273 attestation and the hash of the serialized client data, and returns
- 2274 either:
- 2275 + An error indicating that the attestation is invalid, or
- 2276 + The attestation type, and the trust path of the attestation.

2277 This trust path is either empty (in case of self attestation),

2278 an identifier of a ECDAA-Issuer public key (in the case of

2279 ECDAA), or a set of X.509 certificates.

2280

2281 The initial list of specified attestation statement formats is in 7

2282 Defined Attestation Statement Formats.

2283

2284 5.3.3. Attestation Types

2285

2286 WebAuthn supports multiple attestation types:

2287

2288 Basic Attestation

2289 In the case of basic attestation [UAFProtocol], the

2290 authenticator's attestation key pair is specific to an

2218 authenticator model. Thus, authenticators of the same model
 2219 often share the same attestation key pair. See 5.3.5.1 Privacy
 2220 for further information.

2221 **Self Attestation**
 2222 In the case of self attestation, also known as surrogate basic
 2223 attestation [UAFProtocol], the Authenticator does not have any
 2224 specific attestation key. Instead it uses the authentication key
 2225 itself to create the attestation signature. Authenticators
 2226 without meaningful protection measures for an attestation
 2227 private key typically use this attestation type.

2228 **Privacy CA**
 2229 In this case, the Authenticator owns an authenticator-specific
 2230 (endorsement) key. This key is used to securely communicate with
 2231 a trusted third party, the Privacy CA. The Authenticator can
 2232 generate multiple attestation key pairs and asks the Privacy CA
 2233 to issue an attestation certificate for it. Using this approach,
 2234 the Authenticator can limit the exposure of the endorsement key
 2235 (which is a global correlation handle) to Privacy CA(s).
 2236 Attestation keys can be requested for each public key credential
 2237 individually.

2238 **Note:** This concept typically leads to multiple attestation
 2239 certificates. The attestation certificate requested most
 2240 recently is called "active".

2241 **Elliptic Curve based Direct Anonymous Attestation (ECDAA)**
 2242 In this case, the Authenticator receives direct anonymous
 2243 attestation (DAA) credentials from a single DAA-Issuer. These
 2244 DAA credentials are used along with blinding to sign the
 2245 attestation data. The concept of blinding avoids the DAA
 2246 credentials being misused as global correlation handle. WebAuthn
 2247 supports DAA using elliptic curve cryptography and bilinear
 2248 pairings, called ECDAA (see [FIDOEcdaaAlgorithm]) in this
 2249 specification. Consequently we denote the DAA-Issuer as
 2250 ECDAA-Issuer (see [FIDOEcdaaAlgorithm]).

2251 **5.3.4. Generating an Attestation Object**

2252 This section specifies the algorithm for generating an attestation
 2253 object (see: Figure 3) for any attestation statement format.

2254 In order to construct an attestation object for a given public key
 2255 credential using a particular attestation statement format, the
 2256 authenticator MUST first generate the authenticator data.

2257 The authenticator MUST then run the signing procedure for the desired
 2258 attestation statement format with this authenticator data and the hash
 2259 of the serialized client data as input, and use this to construct an
 2260 attestation statement in that attestation statement format.

2261 Finally, the authenticator MUST construct the attestation object as a
 2262 CBOR map with the following syntax:

```
2263 attObj = {
2264   authData: bytes,
2265   $$attStmtType
2266 }
```

```
2267 attStmtTemplate = (
2268   fmt: text,
2269   attStmt: bytes
2270 )
```

2271 ; Every attestation statement format must have the above fields
 2272 attStmtTemplate .within \$\$attStmtType

2273 The semantics of the fields in the attestation object are as follows:

2274 **fmt**

2291 authenticator model. Thus, authenticators of the same model
 2292 often share the same attestation key pair. See 5.3.5.1 Privacy
 2293 for further information.

2294 **Self Attestation**
 2295 In the case of self attestation, also known as surrogate basic
 2296 attestation [UAFProtocol], the Authenticator does not have any
 2297 specific attestation key. Instead it uses the authentication key
 2298 itself to create the attestation signature. Authenticators
 2299 without meaningful protection measures for an attestation
 2300 private key typically use this attestation type.

2301 **Privacy CA**
 2302 In this case, the Authenticator owns an authenticator-specific
 2303 (endorsement) key. This key is used to securely communicate with
 2304 a trusted third party, the Privacy CA. The Authenticator can
 2305 generate multiple attestation key pairs and asks the Privacy CA
 2306 to issue an attestation certificate for it. Using this approach,
 2307 the Authenticator can limit the exposure of the endorsement key
 2308 (which is a global correlation handle) to Privacy CA(s).
 2309 Attestation keys can be requested for each public key credential
 2310 individually.

2311 **Note:** This concept typically leads to multiple attestation
 2312 certificates. The attestation certificate requested most
 2313 recently is called "active".

2314 **Elliptic Curve based Direct Anonymous Attestation (ECDAA)**
 2315 In this case, the Authenticator receives direct anonymous
 2316 attestation (DAA) credentials from a single DAA-Issuer. These
 2317 DAA credentials are used along with blinding to sign the
 2318 attestation data. The concept of blinding avoids the DAA
 2319 credentials being misused as global correlation handle. WebAuthn
 2320 supports DAA using elliptic curve cryptography and bilinear
 2321 pairings, called ECDAA (see [FIDOEcdaaAlgorithm]) in this
 2322 specification. Consequently we denote the DAA-Issuer as
 2323 ECDAA-Issuer (see [FIDOEcdaaAlgorithm]).

2324 **5.3.4. Generating an Attestation Object**

2325 This section specifies the algorithm for generating an attestation
 2326 object (see: Figure 3) for any attestation statement format.

2327 In order to construct an attestation object for a given public key
 2328 credential using a particular attestation statement format, the
 2329 authenticator MUST first generate the authenticator data.

2330 The authenticator MUST then run the signing procedure for the desired
 2331 attestation statement format with this authenticator data and the hash
 2332 of the serialized client data as input, and use this to construct an
 2333 attestation statement in that attestation statement format.

2334 Finally, the authenticator MUST construct the attestation object as a
 2335 CBOR map with the following syntax:

```
2336 attObj = {
2337   authData: bytes,
2338   $$attStmtType
2339 }
```

```
2340 attStmtTemplate = (
2341   fmt: text,
2342   attStmt: bytes
2343 )
```

2344 ; Every attestation statement format must have the above fields
 2345 attStmtTemplate .within \$\$attStmtType

2346 The semantics of the fields in the attestation object are as follows:

2347 **fmt**

228E The attestation statement format identifier associated with the
228F attestation statement. Each attestation statement format defines
2290 its identifier.

2291 authData

2293 The authenticator data used to generate the attestation
2294 statement.

2296 attStmt

2297 The attestation statement constructed above. The syntax of this
2298 is defined by the attestation statement format used.

2300 5.3.5. Security Considerations

2301 5.3.5.1. Privacy

2302 Attestation keys may be used to track users or link various online
2303 identities of the same user together. This may be mitigated in several
2304 ways, including:

- 2307 * A WebAuthn authenticator manufacturer may choose to ship all of
2308 their devices with the same (or a fixed number of) attestation
2309 key(s) (called Basic Attestation). This will anonymize the user at
2310 the risk of not being able to revoke a particular attestation key
2311 should its WebAuthn Authenticator be compromised.
- 2312 * A WebAuthn Authenticator may be capable of dynamically generating
2313 different attestation keys (and requesting related certificates)
2314 per origin (following the Privacy CA approach). For example, a
2315 WebAuthn Authenticator can ship with a master attestation key (and
2316 certificate), and combined with a cloud operated privacy CA, can
2317 dynamically generate per origin attestation keys and attestation
2318 certificates.
- 2319 * A WebAuthn Authenticator can implement Elliptic Curve based direct
2320 anonymous attestation (see [FIDOEcdaaAlgorithm]). Using this
2321 scheme, the authenticator generates a blinded attestation
2322 signature. This allows the Relying Party to verify the signature
2323 using the ECDAAs-Issuer public key, but the attestation signature
2324 does not serve as a global correlation handle.

2326 5.3.5.2. Attestation Certificate and Attestation Certificate CA Compromise

2327 When an intermediate CA or a root CA used for issuing attestation
2328 certificates is compromised, WebAuthn authenticator attestation keys
2329 are still safe although their certificates can no longer be trusted. A
2330 WebAuthn Authenticator manufacturer that has recorded the public
2331 attestation keys for their devices can issue new attestation
2332 certificates for these keys from a new intermediate CA or from a new
2333 root CA. If the root CA changes, the Relying Parties must update their
2334 trusted root certificates accordingly.

2335 A WebAuthn Authenticator attestation certificate must be revoked by the
2336 issuing CA if its key has been compromised. A WebAuthn Authenticator
2337 manufacturer may need to ship a firmware update and inject new
2338 attestation keys and certificates into already manufactured WebAuthn
2339 Authenticators, if the exposure was due to a firmware flaw. (The
2340 process by which this happens is out of scope for this specification.)
2341 If the WebAuthn Authenticator manufacturer does not have this
2342 capability, then it may not be possible for Relying Parties to trust
2343 any further attestation statements from the affected WebAuthn
2344 Authenticators.

2345 If attestation certificate validation fails due to a revoked
2346 intermediate attestation CA certificate, and the Relying Party's policy
2347 requires rejecting the registration/authentication request in these
2348 situations, then it is recommended that the Relying Party also
2349 un-registers (or marks with a trust level equivalent to "self
2350 attestation") public key credentials that were registered after the CA
2351 compromise date using an attestation certificate chaining up to the
2352 same intermediate CA. It is thus recommended that Relying Parties
2353 remember intermediate attestation CA certificates during Authenticator
2354 registration in order to un-register related public key credentials if

2357

2361 The attestation statement format identifier associated with the
2362 attestation statement. Each attestation statement format defines
2363 its identifier.

2364 authData

2365 The authenticator data used to generate the attestation
2366 statement.

2367 attStmt

2368 The attestation statement constructed above. The syntax of this
2369 is defined by the attestation statement format used.

2370 5.3.5. Security Considerations

2371 5.3.5.1. Privacy

2372 Attestation keys may be used to track users or link various online
2373 identities of the same user together. This may be mitigated in several
2374 ways, including:

- 2377 * A WebAuthn authenticator manufacturer may choose to ship all of
2378 their devices with the same (or a fixed number of) attestation
2379 key(s) (called Basic Attestation). This will anonymize the user at
2380 the risk of not being able to revoke a particular attestation key
2381 should its WebAuthn Authenticator be compromised.
- 2382 * A WebAuthn Authenticator may be capable of dynamically generating
2383 different attestation keys (and requesting related certificates)
2384 per origin (following the Privacy CA approach). For example, a
2385 WebAuthn Authenticator can ship with a master attestation key (and
2386 certificate), and combined with a cloud operated privacy CA, can
2387 dynamically generate per origin attestation keys and attestation
2388 certificates.
- 2389 * A WebAuthn Authenticator can implement Elliptic Curve based direct
2390 anonymous attestation (see [FIDOEcdaaAlgorithm]). Using this
2391 scheme, the authenticator generates a blinded attestation
2392 signature. This allows the Relying Party to verify the signature
2393 using the ECDAAs-Issuer public key, but the attestation signature
2394 does not serve as a global correlation handle.

2396 5.3.5.2. Attestation Certificate and Attestation Certificate CA Compromise

2397 When an intermediate CA or a root CA used for issuing attestation
2398 certificates is compromised, WebAuthn authenticator attestation keys
2399 are still safe although their certificates can no longer be trusted. A
2400 WebAuthn Authenticator manufacturer that has recorded the public
2401 attestation keys for their devices can issue new attestation
2402 certificates for these keys from a new intermediate CA or from a new
2403 root CA. If the root CA changes, the Relying Parties must update their
2404 trusted root certificates accordingly.

2405 A WebAuthn Authenticator attestation certificate must be revoked by the
2406 issuing CA if its key has been compromised. A WebAuthn Authenticator
2407 manufacturer may need to ship a firmware update and inject new
2408 attestation keys and certificates into already manufactured WebAuthn
2409 Authenticators, if the exposure was due to a firmware flaw. (The
2410 process by which this happens is out of scope for this specification.)
2411 If the WebAuthn Authenticator manufacturer does not have this
2412 capability, then it may not be possible for Relying Parties to trust
2413 any further attestation statements from the affected WebAuthn
2414 Authenticators.

2415 If attestation certificate validation fails due to a revoked
2416 intermediate attestation CA certificate, and the Relying Party's policy
2417 requires rejecting the registration/authentication request in these
2418 situations, then it is recommended that the Relying Party also
2419 un-registers (or marks with a trust level equivalent to "self
2420 attestation") public key credentials that were registered after the CA
2421 compromise date using an attestation certificate chaining up to the
2422 same intermediate CA. It is thus recommended that Relying Parties
2423 remember intermediate attestation CA certificates during Authenticator
2424 registration in order to un-register related public key credentials if

2430

235E the registration was performed after revocation of such certificates.
235F
236C If an ECDAAs attestation key has been compromised, it can be added to
236D the RogueList (i.e., the list of revoked authenticators) maintained by
236E the related ECDAAs-Issuer. The Relying Party should verify whether an
236F authenticator belongs to the RogueList when performing ECDAAs-Verify
236G (see section 3.6 in [FIDOEcdaaAlgorithm]). For example, the FIDO
236H Metadata Service [FIDOMetadataService] provides one way to access such
236I information.

236J 5.3.5.3. Attestation Certificate Hierarchy

237C A 3-tier hierarchy for attestation certificates is recommended (i.e.,
237D Attestation Root, Attestation Issuing CA, Attestation Certificate). It
237E is also recommended that for each WebAuthn Authenticator device line
237F (i.e., model), a separate issuing CA is used to help facilitate
237G isolating problems with a specific version of a device.

237H If the attestation root certificate is not dedicated to a single
237I WebAuthn Authenticator device line (i.e., AAGUID), the AAGUID should be
237J specified in the attestation certificate itself, so that it can be
237K verified against the authenticator data.

237L 6. Relying Party Operations

238C Upon successful execution of create() or get(), the Relying Party's
238D script receives a PublicKeyCredential containing an
238E AuthenticatorAttestationResponse or AuthenticatorAssertionResponse
238F structure, respectively, from the client. It must then deliver the
238G contents of this structure to the Relying Party server, using methods
238H outside the scope of this specification. This section describes the
238I operations that the Relying Party must perform upon receipt of these
238J structures.

238K 6.1. Registering a new credential

239A When registering a new credential, represented by a
239B AuthenticatorAttestationResponse structure, as part of a registration
239C ceremony, a Relying Party MUST proceed as follows:

- 239D 1. Perform JSON deserialization on the clientDataJSON field of the
239E AuthenticatorAttestationResponse object to extract the client data
239F C claimed as collected during the credential creation.
- 2400 2. Verify that the challenge in C matches the challenge that was sent
2401 to the authenticator in the create() call.
- 2402 3. Verify that the origin in C matches the Relying Party's origin.
- 2403 4. Verify that the tokenBindingId in C matches the Token Binding ID
2404 for the TLS connection over which the attestation was obtained.
- 2405 5. Verify that the clientExtensions in C is a subset of the extensions
2406 requested by the RP and that the authenticatorExtensions in C is
2407 also a subset of the extensions requested by the RP.
- 2408 6. Compute the hash of clientDataJSON using the algorithm identified
2409 by C.hashAlgorithm.
- 2410 7. Perform CBOR decoding on the attestationObject field of the
2411 AuthenticatorAttestationResponse structure to obtain the
2412 attestation statement format fmt, the authenticator data authData,
2413 and the attestation statement attStmt.
- 2414 8. Verify that the RP ID hash in authData is indeed the SHA-256 hash
2415 of the RP ID expected by the RP.
- 2416 9. Determine the attestation statement format by performing an USASCII
2417 case-sensitive match on fmt against the set of supported WebAuthn
2418 Attestation Statement Format Identifier values. The up-to-date list
2419 of registered WebAuthn Attestation Statement Format Identifier
2420 values is maintained in the in the IANA registry of the same name
2421 [WebAuthn-Registries].
- 2422 10. Verify that attStmt is a correct, validly-signed attestation
2423 statement, using the attestation statement format fmt's
2424 verification procedure given authenticator data authData and the
2425 hash of the serialized client data computed in step 6.
- 2426 11. If validation is successful, obtain a list of acceptable trust
2427 anchors (attestation root certificates or ECDAAs-Issuer public keys)

2431 the registration was performed after revocation of such certificates.

2432
2433 If an ECDAAs attestation key has been compromised, it can be added to
2434 the RogueList (i.e., the list of revoked authenticators) maintained by
2435 the related ECDAAs-Issuer. The Relying Party should verify whether an
2436 authenticator belongs to the RogueList when performing ECDAAs-Verify
2437 (see section 3.6 in [FIDOEcdaaAlgorithm]). For example, the FIDO
2438 Metadata Service [FIDOMetadataService] provides one way to access such
2439 information.

244C 5.3.5.3. Attestation Certificate Hierarchy

244D A 3-tier hierarchy for attestation certificates is recommended (i.e.,
244E Attestation Root, Attestation Issuing CA, Attestation Certificate). It
244F is also recommended that for each WebAuthn Authenticator device line
244G (i.e., model), a separate issuing CA is used to help facilitate
244H isolating problems with a specific version of a device.

244I If the attestation root certificate is not dedicated to a single
244J WebAuthn Authenticator device line (i.e., AAGUID), the AAGUID should be
244K specified in the attestation certificate itself, so that it can be
244L verified against the authenticator data.

244M 6. Relying Party Operations

245C Upon successful execution of create() or get(), the Relying Party's
245D script receives a PublicKeyCredential containing an
245E AuthenticatorAttestationResponse or AuthenticatorAssertionResponse
245F structure, respectively, from the client. It must then deliver the
245G contents of this structure to the Relying Party server, using methods
245H outside the scope of this specification. This section describes the
245I operations that the Relying Party must perform upon receipt of these
245J structures.

245K 6.1. Registering a new credential

246A When registering a new credential, represented by a
246B AuthenticatorAttestationResponse structure, as part of a registration
246C ceremony, a Relying Party MUST proceed as follows:

- 246D 1. Perform JSON deserialization on the clientDataJSON field of the
246E AuthenticatorAttestationResponse object to extract the client data
246F C claimed as collected during the credential creation.
- 246G 2. Verify that the challenge in C matches the challenge that was sent
246H to the authenticator in the create() call.
- 246I 3. Verify that the origin in C matches the Relying Party's origin.
- 246J 4. Verify that the tokenBindingId in C matches the Token Binding ID
246K for the TLS connection over which the attestation was obtained.
- 246L 5. Verify that the clientExtensions in C is a subset of the extensions
246M requested by the RP and that the authenticatorExtensions in C is
246N also a subset of the extensions requested by the RP.
- 246O 6. Compute the hash of clientDataJSON using the algorithm identified
246P by C.hashAlgorithm.
- 246Q 7. Perform CBOR decoding on the attestationObject field of the
246R AuthenticatorAttestationResponse structure to obtain the
246S attestation statement format fmt, the authenticator data authData,
246T and the attestation statement attStmt.
- 246U 8. Verify that the RP ID hash in authData is indeed the SHA-256 hash
246V of the RP ID expected by the RP.
- 246W 9. Determine the attestation statement format by performing an USASCII
246X case-sensitive match on fmt against the set of supported WebAuthn
246Y Attestation Statement Format Identifier values. The up-to-date list
246Z of registered WebAuthn Attestation Statement Format Identifier
246AA values is maintained in the in the IANA registry of the same name
246AB [WebAuthn-Registries].
- 246AC 10. Verify that attStmt is a correct, validly-signed attestation
246AD statement, using the attestation statement format fmt's
246AE verification procedure given authenticator data authData and the
246AF hash of the serialized client data computed in step 6.
- 246AG 11. If validation is successful, obtain a list of acceptable trust
246AH anchors (attestation root certificates or ECDAAs-Issuer public keys)

242E for that attestation type and attestation statement format fmt,
 242F from a trusted source or from policy. For example, the FIDO
 243C Metadata Service [FIDOMetadataService] provides one way to obtain
 2431 such information, using the AAGUID in the attestation data
 2432 contained in authData.
 2433 12. Assess the attestation trustworthiness using the outputs of the
 2434 verification procedure in step 10, as follows:
 2435 + If self attestation was used, check if self attestation is
 2436 acceptable under Relying Party policy.
 2437 + If ECDAAs were used, verify that the identifier of the
 2438 ECDAAs-issuer public key used is included in the set of
 2439 acceptable trust anchors obtained in step 11.
 2440 + Otherwise, use the X.509 certificates returned by the
 2441 verification procedure to verify that the attestation public
 2442 key correctly chains up to an acceptable root certificate.
 2443 13. If the attestation statement attStmt verified successfully and is
 2444 found to be trustworthy, then register the new credential with the
 2445 account that was denoted in the options.user passed to create(), by
 2446 associating it with the credential ID and credential public key
 2447 contained in authData's attestation data, as appropriate for the
 2448 Relying Party's systems.
 2449 14. If the attestation statement attStmt successfully verified but is
 2450 not trustworthy per step 12 above, the Relying Party SHOULD fail
 2451 the registration ceremony.
 2452 NOTE: However, if permitted by policy, the Relying Party MAY
 2453 register the credential ID and credential public key but treat the
 2454 credential as one with self attestation (see 5.3.3 Attestation
 2455 Types). If doing so, the Relying Party is asserting there is no
 2456 cryptographic proof that the public key credential has been
 2457 generated by a particular authenticator model. See [FIDOsecRef] and
 2458 [UAFProtocol] for a more detailed discussion.
 2459 15. If verification of the attestation statement failed, the Relying
 2460 Party MUST fail the registration ceremony.

2461 Verification of attestation objects requires that the Relying Party has
 2462 a trusted method of determining acceptable trust anchors in step 11
 2463 above. Also, if certificates are being used, the Relying Party must
 2464 have access to certificate status information for the intermediate CA
 2465 certificates. The Relying Party must also be able to build the
 2466 attestation certificate chain if the client did not provide this chain
 2467 in the attestation information.
 2468

2469 To avoid ambiguity during authentication, the Relying Party SHOULD
 2470 check that each credential is registered to no more than one user. If
 2471 registration is requested for a credential that is already registered
 2472 to a different user, the Relying Party SHOULD fail this ceremony, or it
 2473 MAY decide to accept the registration, e.g. while deleting the older
 2474 registration.
 2475

2476 6.2. Verifying an authentication assertion

2477 When verifying a given PublicKeyCredential structure (credential) as
 2478 part of an authentication ceremony, the Relying Party MUST proceed as
 2479 follows:
 2480

- 2481 1. Using credential's id attribute (or the corresponding rawId, if
 2482 base64url encoding is inappropriate for your use case), look up the
 2483 corresponding credential public key.
- 2484 2. Let cData, aData and sig denote the value of credential's
 2485 response's clientDataJSON, authenticatorData, and signature
 2486 respectively.
- 2487 3. Perform JSON deserialization on cData to extract the client data C
 2488 used for the signature.
- 2489 4. Verify that the challenge member of C matches the challenge that
 2490 was sent to the authenticator in the
 2491 PublicKeyCredentialRequestOptions passed to the get() call.
- 2492 5. Verify that the origin member of C matches the Relying Party's
 2493 origin.
- 2494 6. Verify that the tokenBindingId member of C (if present) matches the
 2495 Token Binding ID for the TLS connection over which the signature
 2496 was obtained.
 2497

2501 for that attestation type and attestation statement format fmt,
 2502 from a trusted source or from policy. For example, the FIDO
 2503 Metadata Service [FIDOMetadataService] provides one way to obtain
 2504 such information, using the AAGUID in the attestation data
 2505 contained in authData.
 2506 12. Assess the attestation trustworthiness using the outputs of the
 2507 verification procedure in step 10, as follows:
 2508 + If self attestation was used, check if self attestation is
 2509 acceptable under Relying Party policy.
 2510 + If ECDAAs were used, verify that the identifier of the
 2511 ECDAAs-issuer public key used is included in the set of
 2512 acceptable trust anchors obtained in step 11.
 2513 + Otherwise, use the X.509 certificates returned by the
 2514 verification procedure to verify that the attestation public
 2515 key correctly chains up to an acceptable root certificate.
 2516 13. If the attestation statement attStmt verified successfully and is
 2517 found to be trustworthy, then register the new credential with the
 2518 account that was denoted in the options.user passed to create(), by
 2519 associating it with the credential ID and credential public key
 2520 contained in authData's attestation data, as appropriate for the
 2521 Relying Party's systems.
 2522 14. If the attestation statement attStmt successfully verified but is
 2523 not trustworthy per step 12 above, the Relying Party SHOULD fail
 2524 the registration ceremony.
 2525 NOTE: However, if permitted by policy, the Relying Party MAY
 2526 register the credential ID and credential public key but treat the
 2527 credential as one with self attestation (see 5.3.3 Attestation
 2528 Types). If doing so, the Relying Party is asserting there is no
 2529 cryptographic proof that the public key credential has been
 2530 generated by a particular authenticator model. See [FIDOsecRef] and
 2531 [UAFProtocol] for a more detailed discussion.
 2532 15. If verification of the attestation statement failed, the Relying
 2533 Party MUST fail the registration ceremony.

2534 Verification of attestation objects requires that the Relying Party has
 2535 a trusted method of determining acceptable trust anchors in step 11
 2536 above. Also, if certificates are being used, the Relying Party must
 2537 have access to certificate status information for the intermediate CA
 2538 certificates. The Relying Party must also be able to build the
 2539 attestation certificate chain if the client did not provide this chain
 2540 in the attestation information.
 2541

2542 To avoid ambiguity during authentication, the Relying Party SHOULD
 2543 check that each credential is registered to no more than one user. If
 2544 registration is requested for a credential that is already registered
 2545 to a different user, the Relying Party SHOULD fail this ceremony, or it
 2546 MAY decide to accept the registration, e.g. while deleting the older
 2547 registration.
 2548

2549 6.2. Verifying an authentication assertion

2550 When verifying a given PublicKeyCredential structure (credential) as
 2551 part of an authentication ceremony, the Relying Party MUST proceed as
 2552 follows:
 2553

- 2554 1. Using credential's id attribute (or the corresponding rawId, if
 2555 base64url encoding is inappropriate for your use case), look up the
 2556 corresponding credential public key.
- 2557 2. Let cData, aData and sig denote the value of credential's
 2558 response's clientDataJSON, authenticatorData, and signature
 2559 respectively.
- 2560 3. Perform JSON deserialization on cData to extract the client data C
 2561 used for the signature.
- 2562 4. Verify that the challenge member of C matches the challenge that
 2563 was sent to the authenticator in the
 2564 PublicKeyCredentialRequestOptions passed to the get() call.
- 2565 5. Verify that the origin member of C matches the Relying Party's
 2566 origin.
- 2567 6. Verify that the tokenBindingId member of C (if present) matches the
 2568 Token Binding ID for the TLS connection over which the signature
 2569 was obtained.
 2570

- 249E 7. Verify that the clientExtensions member of C is a subset of the
- 249F extensions requested by the Relying Party and that the
- 250C authenticatorExtensions in C is also a subset of the extensions
- 2501 requested by the Relying Party.
- 2502 8. Verify that the RP ID hash in aData is the SHA-256 hash of the RP
- 2503 ID expected by the Relying Party.
- 2504 9. Let hash be the result of computing a hash over the cData using the
- 2505 algorithm represented by the hashAlgorithm member of C.
- 2506 10. Using the credential public key looked up in step 1, verify that
- 2507 sig is a valid signature over the binary concatenation of aData and
- 2508 hash.
- 2509 11. If all the above steps are successful, continue with the
- 251C authentication ceremony as appropriate. Otherwise, fail the
- 2511 authentication ceremony.

2512 7. Defined Attestation Statement Formats

2514 WebAuthn supports pluggable attestation statement formats. This section

2515 defines an initial set of such formats.

2516 7.1. Attestation Statement Format Identifiers

2518 Attestation statement formats are identified by a string, called a

252C attestation statement format identifier, chosen by the author of the

2521 attestation statement format.

2522 Attestation statement format identifiers SHOULD be registered per

2523 [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)".

2524 All registered attestation statement format identifiers are unique

2525 amongst themselves as a matter of course.

2526 Unregistered attestation statement format identifiers SHOULD use

2527 lowercase reverse domain-name naming, using a domain name registered by

2528 the developer, in order to assure uniqueness of the identifier. All

253C attestation statement format identifiers MUST be a maximum of 32 octets

2532 in length and MUST consist only of printable USASCII characters,

2533 excluding backslash and doublequote, i.e., VCHAR as defined in

2534 [RFC5234] but without %x22 and %x5c.

2535 Note: This means attestation statement format identifiers based on

2536 domain names MUST incorporate only LDH Labels [RFC5890].

2537 Implementations MUST match WebAuthn attestation statement format

2538 identifiers in a case-sensitive fashion.

2539 Attestation statement formats that may exist in multiple versions

254C SHOULD include a version in their identifier. In effect, different

2544 versions are thus treated as different formats, e.g., packed2 as a new

2545 version of the packed attestation statement format.

2546 The following sections present a set of currently-defined and

2547 registered attestation statement formats and their identifiers. The

2548 up-to-date list of registered WebAuthn Extensions is maintained in the

2549 IANA "WebAuthn Attestation Statement Format Identifier" registry

255C established by [WebAuthn-Registries].

2551 7.2. Packed Attestation Statement Format

2552 This is a WebAuthn optimized attestation statement format. It uses a

2553 very compact but still extensible encoding method. It is implementable

2554 by authenticators with limited resources (e.g., secure elements).

2555 Attestation statement format identifier

2556 packed

2557 Attestation types supported

2558 All

2559 Syntax

256C The syntax of a Packed Attestation statement is defined by the

2567

- 2571 7. Verify that the clientExtensions member of C is a subset of the
- 2572 extensions requested by the Relying Party and that the
- 2573 authenticatorExtensions in C is also a subset of the extensions
- 2574 requested by the Relying Party.
- 2575 8. Verify that the RP ID hash in aData is the SHA-256 hash of the RP
- 2576 ID expected by the Relying Party.
- 2577 9. Let hash be the result of computing a hash over the cData using the
- 2578 algorithm represented by the hashAlgorithm member of C.
- 2579 10. Using the credential public key looked up in step 1, verify that
- 258C sig is a valid signature over the binary concatenation of aData and
- 2581 hash.
- 2582 11. If all the above steps are successful, continue with the
- 2583 authentication ceremony as appropriate. Otherwise, fail the
- 2584 authentication ceremony.

2585 7. Defined Attestation Statement Formats

2586 WebAuthn supports pluggable attestation statement formats. This section

2587 defines an initial set of such formats.

2588 7.1. Attestation Statement Format Identifiers

2589 Attestation statement formats are identified by a string, called a

2594 attestation statement format identifier, chosen by the author of the

2595 attestation statement format.

2596 Attestation statement format identifiers SHOULD be registered per

2597 [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)".

2598 All registered attestation statement format identifiers are unique

2599 amongst themselves as a matter of course.

260C Unregistered attestation statement format identifiers SHOULD use

2600 lowercase reverse domain-name naming, using a domain name registered by

2604 the developer, in order to assure uniqueness of the identifier. All

2605 attestation statement format identifiers MUST be a maximum of 32 octets

2606 in length and MUST consist only of printable USASCII characters,

2607 excluding backslash and doublequote, i.e., VCHAR as defined in

2608 [RFC5234] but without %x22 and %x5c.

2609 Note: This means attestation statement format identifiers based on

261C domain names MUST incorporate only LDH Labels [RFC5890].

2611 Implementations MUST match WebAuthn attestation statement format

2612 identifiers in a case-sensitive fashion.

2613 Attestation statement formats that may exist in multiple versions

2614 SHOULD include a version in their identifier. In effect, different

2615 versions are thus treated as different formats, e.g., packed2 as a new

2616 version of the packed attestation statement format.

2617 The following sections present a set of currently-defined and

2618 registered attestation statement formats and their identifiers. The

2619 up-to-date list of registered WebAuthn Extensions is maintained in the

262C IANA "WebAuthn Attestation Statement Format Identifier" registry

2621 established by [WebAuthn-Registries].

2622 7.2. Packed Attestation Statement Format

2623 This is a WebAuthn optimized attestation statement format. It uses a

2624 very compact but still extensible encoding method. It is implementable

2625 by authenticators with limited resources (e.g., secure elements).

2626 Attestation statement format identifier

2627 packed

2628 Attestation types supported

2629 All

263C Syntax

2631 The syntax of a Packed Attestation statement is defined by the

264C

```

2568 following CDDL:
2569
2570 $$attStmtType ::= (
2571     fmt: "packed",
2572     attStmt: packedStmtFormat
2573 )
2574
2575 packedStmtFormat = {
2576     alg: COSEAlgorithmIdentifier,
2577     sig: bytes,
2578     x5c: [ attestnCert: bytes, * (caCert: bytes) ]
2579 } //
2580     alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261
2581 for ED512)
2582     sig: bytes,
2583     ecdaaKeyId: bytes
2584 }

```

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 ECDAAs-Issuer public key. This is the BigIntegerToB encoding of the component "c" of the ECDAAs-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.

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

If Basic or Privacy CA 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.

If ECDAAs is in use, the authenticator produces sig by concatenating authenticatorData and clientDataHash, and signing the result using ECDAAs-Sign (see section 3.5 of [FIDOEcdaaAlgorithm]) with a ECDAAs-Issuer public key selected through an authenticator-specific mechanism (see [FIDOEcdaaAlgorithm]). It sets alg to the algorithm of the ECDAAs-Issuer public key and ecdaaKeyId to the identifier of the ECDAAs-Issuer public key (see above).

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.

```

2641 following CDDL:
2642
2643 $$attStmtType ::= (
2644     fmt: "packed",
2645     attStmt: packedStmtFormat
2646 )
2647
2648 packedStmtFormat = {
2649     alg: COSEAlgorithmIdentifier,
2650     sig: bytes,
2651     x5c: [ attestnCert: bytes, * (caCert: bytes) ]
2652 } //
2653     alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261
2654 for ED512)
2655     sig: bytes,
2656     ecdaaKeyId: bytes
2657 }

```

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 ECDAAs-Issuer public key. This is the BigIntegerToB encoding of the component "c" of the ECDAAs-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.

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

If Basic or Privacy CA 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.

If ECDAAs is in use, the authenticator produces sig by concatenating authenticatorData and clientDataHash, and signing the result using ECDAAs-Sign (see section 3.5 of [FIDOEcdaaAlgorithm]) with a ECDAAs-Issuer public key selected through an authenticator-specific mechanism (see [FIDOEcdaaAlgorithm]). It sets alg to the algorithm of the ECDAAs-Issuer public key and ecdaaKeyId to the identifier of the ECDAAs-Issuer public key (see above).

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.

263E
263F
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707

Verification procedure

Verify that the given attestation statement is valid CBOR conforming to the syntax defined above.

Let authenticatorData denote the authenticator data claimed to have been used for the attestation, and let clientDataHash denote the hash of the serialized client data.

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

- + 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.
- + Verify that x5c meets the requirements in 7.2.1 Packed 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 Basic and trust path x5c.

If ecdaaKeyId is present, then the attestation type is ECDA. In this case:

- + Verify that sig is a valid signature over the concatenation of authenticatorData and clientDataHash using ECDA-Verify with ECDA-Issuer public key identified by ecdaaKeyId (see [FIDOEcdaaAlgorithm]).
- + If successful, return attestation type ECDA and trust path ecdaaKeyId.

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

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

7.2.1. Packed attestation statement certificate requirements

The attestation certificate MUST have the following fields/extensions:

- * Version must be set to 3.
- * Subject field MUST be set to:

- Subject-C
Country where the Authenticator vendor is incorporated
- Subject-O
Legal name of the Authenticator vendor
- Subject-OU
Authenticator Attestation
- Subject-CN
No stipulation.

- * If the related attestation root certificate is used for multiple authenticator models, the Extension OID 1 3 6 1 4 1 45724 1 1 4 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as value.
- * The Basic Constraints extension MUST have the CA component set to false
- * An Authority Information Access (AIA) extension with entry id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are

2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780

Verification procedure

Verify that the given attestation statement is valid CBOR conforming to the syntax defined above.

Let authenticatorData denote the authenticator data claimed to have been used for the attestation, and let clientDataHash denote the hash of the serialized client data.

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

- + 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.
- + Verify that x5c meets the requirements in 7.2.1 Packed 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 Basic and trust path x5c.

If ecdaaKeyId is present, then the attestation type is ECDA. In this case:

- + Verify that sig is a valid signature over the concatenation of authenticatorData and clientDataHash using ECDA-Verify with ECDA-Issuer public key identified by ecdaaKeyId (see [FIDOEcdaaAlgorithm]).
- + If successful, return attestation type ECDA and trust path ecdaaKeyId.

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

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

7.2.1. Packed attestation statement certificate requirements

The attestation certificate MUST have the following fields/extensions:

- * Version must be set to 3.
- * Subject field MUST be set to:

- Subject-C
Country where the Authenticator vendor is incorporated
- Subject-O
Legal name of the Authenticator vendor
- Subject-OU
Authenticator Attestation
- Subject-CN
No stipulation.

- * If the related attestation root certificate is used for multiple authenticator models, the Extension OID 1 3 6 1 4 1 45724 1 1 4 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as value.
- * The Basic Constraints extension MUST have the CA component set to false
- * An Authority Information Access (AIA) extension with entry id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are

2708 both optional as the status of many attestation certificates is
 2709 available through authenticator metadata services. See, for
 2710 example, the FIDO Metadata Service [FIDOMetadataService].
 2711

2712 7.3. TPM Attestation Statement Format

2713

2714 This attestation statement format is generally used by authenticators
 2715 that use a Trusted Platform Module as their cryptographic engine.
 2716

2717 **Attestation statement format identifier**
 2718 tpm
 2719

2720 **Attestation types supported**
 2721 Privacy CA, ECDA
 2722

2723 **Syntax**
 2724 The syntax of a TPM Attestation statement is as follows:
 2725

```

    2726 $$attStmtType // = (
    2727     fmt: "tpm",
    2728     attStmt: tpmStmtFormat
    2729 )
    2730
    2731 tpmStmtFormat = {
    2732     ver: "2.0",
    2733     (
    2734         alg: COSEAlgorithmIdentifier,
    2735         x5c: [ aikCert: bytes, * (caCert: bytes) ]
    2736     ) //
    2737     (
    2738         alg: COSEAlgorithmIdentifier, (-260 for ED256 / -26
    2739 1 for ED512)
    2740         ecdaaKeyId: bytes
    2741     ),
    2742     sig: bytes,
    2743     certInfo: bytes,
    2744     pubArea: bytes
    2745 }
    
```

2746

2747 The semantics of the above fields are as follows:

2748

2749 **ver**
 2750 The version of the TPM specification to which the
 2751 signature conforms.
 2752

2753 **alg**
 2754 A COSEAlgorithmIdentifier containing the identifier of the
 2755 algorithm used to generate the attestation signature.
 2756

2757 **x5c**
 2758 The AIK certificate used for the attestation and its
 2759 certificate chain, in X.509 encoding.
 2760

2761 **ecdaaKeyId**
 2762 The identifier of the ECDA-issuer public key. This is the
 2763 BigNumberToB encoding of the component "c" as defined
 2764 section 3.3, step 3.5 in [FIDOecdaaAlgorithm].
 2765

2766 **sig**
 2767 The attestation signature, in the form of a TPMT_SIGNATURE
 2768 structure as specified in [TPMv2-Part2] section 11.3.4.
 2769

2770 **certInfo**
 2771 The TPMS_ATTEST structure over which the above signature
 2772 was computed, as specified in [TPMv2-Part2] section
 2773 10.12.8.
 2774

2775 **pubArea**
 2776 The TPMT_PUBLIC structure (see [TPMv2-Part2] section
 2777 12.2.4) used by the TPM to represent the credential public

2781 both optional as the status of many attestation certificates is
 2782 available through authenticator metadata services. See, for
 2783 example, the FIDO Metadata Service [FIDOMetadataService].
 2784

2785 7.3. TPM Attestation Statement Format

2786

2787 This attestation statement format is generally used by authenticators
 2788 that use a Trusted Platform Module as their cryptographic engine.
 2789

2790 **Attestation statement format identifier**
 2791 tpm
 2792

2793 **Attestation types supported**
 2794 Privacy CA, ECDA
 2795

2796 **Syntax**
 2797 The syntax of a TPM Attestation statement is as follows:
 2798

```

    2799 $$attStmtType // = (
    2800     fmt: "tpm",
    2801     attStmt: tpmStmtFormat
    2802 )
    2803
    2804 tpmStmtFormat = {
    2805     ver: "2.0",
    2806     (
    2807         alg: COSEAlgorithmIdentifier,
    2808         x5c: [ aikCert: bytes, * (caCert: bytes) ]
    2809     ) //
    2810     (
    2811         alg: COSEAlgorithmIdentifier, (-260 for ED256 / -26
    2812 1 for ED512)
    2813         ecdaaKeyId: bytes
    2814     ),
    2815     sig: bytes,
    2816     certInfo: bytes,
    2817     pubArea: bytes
    2818 }
    
```

2819

2820 The semantics of the above fields are as follows:

2821

2822 **ver**
 2823 The version of the TPM specification to which the
 2824 signature conforms.
 2825

2826 **alg**
 2827 A COSEAlgorithmIdentifier containing the identifier of the
 2828 algorithm used to generate the attestation signature.
 2829

2830 **x5c**
 2831 The AIK certificate used for the attestation and its
 2832 certificate chain, in X.509 encoding.
 2833

2834 **ecdaaKeyId**
 2835 The identifier of the ECDA-issuer public key. This is the
 2836 BigNumberToB encoding of the component "c" as defined
 2837 section 3.3, step 3.5 in [FIDOecdaaAlgorithm].
 2838

2839 **sig**
 2840 The attestation signature, in the form of a TPMT_SIGNATURE
 2841 structure as specified in [TPMv2-Part2] section 11.3.4.
 2842

2843 **certInfo**
 2844 The TPMS_ATTEST structure over which the above signature
 2845 was computed, as specified in [TPMv2-Part2] section
 2846 10.12.8.
 2847

2848 **pubArea**
 2849 The TPMT_PUBLIC structure (see [TPMv2-Part2] section
 2850 12.2.4) used by the TPM to represent the credential public

2778 key.
 2779
 2780
 2781 **Signing procedure**
 2782 Let authenticatorData denote the authenticator data for the
 2783 attestation, and let clientDataHash denote the hash of the
 2784 serialized client data.
 2785
 2786 Concatenate authenticatorData and clientDataHash to form
 2787 attToBeSigned.
 2788
 2789 Generate a signature using the procedure specified in
 2790 [TPMv2-Part3] Section 18.2, using the attestation private key
 2791 and setting the qualifyingData parameter to attToBeSigned.
 2792
 2793 Set the pubArea field to the public area of the credential
 2794 public key, the certInfo field to the output parameter of the
 2795 same name, and the sig field to the signature obtained from the
 2796 above procedure.
 2797
 2798 **Verification procedure**
 2799 Verify that the given attestation statement is valid CBOR
 2800 conforming to the syntax defined above.
 2801
 2802 Let authenticatorData denote the authenticator data claimed to
 2803 have been used for the attestation, and let clientDataHash
 2804 denote the hash of the serialized client data.
 2805
 2806 Verify that the public key specified by the parameters and
 2807 unique fields of pubArea is identical to the public key
 2808 contained in the attestation data inside authenticatorData.
 2809
 2810 Concatenate authenticatorData and clientDataHash to form
 2811 attToBeSigned.
 2812
 2813 Validate that certInfo is valid:
 2814
 2815 + Verify that magic is set to TPM_GENERATED_VALUE.
 2816 + Verify that type is set to TPM_ST_ATTEST_CERTIFY.
 2817 + Verify that extraData is set to attToBeSigned.
 2818 + Verify that attested contains a TPMS_CERTIFY_INFO structure,
 2819 whose name field contains a valid Name for pubArea, as
 2820 computed using the algorithm in the nameAlg field of pubArea
 2821 using the procedure specified in [TPMv2-Part1] section 16.
 2822
 2823 If x5c is present, this indicates that the attestation type is
 2824 not ECDA. In this case:
 2825
 2826 + Verify the sig is a valid signature over certInfo using the
 2827 attestation public key in x5c with the algorithm specified in
 2828 alg.
 2829 + Verify that x5c meets the requirements in 7.3.1 TPM
 2830 attestation statement certificate requirements.
 2831 + If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4
 2832 (id-fido-gen-ce-aaguid) verify that the value of this
 2833 extension matches the AAGUID in authenticatorData.
 2834 + If successful, return attestation type Privacy CA and trust
 2835 path x5c.
 2836
 2837 If ecdAaKeyId is present, then the attestation type is ECDA.
 2838
 2839 + Perform ECDA-Verify on sig to verify that it is a valid
 2840 signature over certInfo (see [FIDOecdaaAlgorithm]).
 2841 + If successful, return attestation type ECDA and the
 2842 identifier of the ECDA-Issuer public key ecdAaKeyId.
 2843
 2844 **7.3.1. TPM attestation statement certificate requirements**
 2845
 2846 TPM attestation certificate MUST have the following fields/extensions:
 2847 * Version must be set to 3.
 * Subject field MUST be set to empty.

2851 key.
 2852
 2853
 2854 **Signing procedure**
 2855 Let authenticatorData denote the authenticator data for the
 2856 attestation, and let clientDataHash denote the hash of the
 2857 serialized client data.
 2858
 2859 Concatenate authenticatorData and clientDataHash to form
 2860 attToBeSigned.
 2861
 2862 Generate a signature using the procedure specified in
 2863 [TPMv2-Part3] Section 18.2, using the attestation private key
 2864 and setting the qualifyingData parameter to attToBeSigned.
 2865
 2866 Set the pubArea field to the public area of the credential
 2867 public key, the certInfo field to the output parameter of the
 2868 same name, and the sig field to the signature obtained from the
 2869 above procedure.
 2870
 2871 **Verification procedure**
 2872 Verify that the given attestation statement is valid CBOR
 2873 conforming to the syntax defined above.
 2874
 2875 Let authenticatorData denote the authenticator data claimed to
 2876 have been used for the attestation, and let clientDataHash
 2877 denote the hash of the serialized client data.
 2878
 2879 Verify that the public key specified by the parameters and
 2880 unique fields of pubArea is identical to the public key
 2881 contained in the attestation data inside authenticatorData.
 2882
 2883 Concatenate authenticatorData and clientDataHash to form
 2884 attToBeSigned.
 2885
 2886 Validate that certInfo is valid:
 2887
 2888 + Verify that magic is set to TPM_GENERATED_VALUE.
 2889 + Verify that type is set to TPM_ST_ATTEST_CERTIFY.
 2890 + Verify that extraData is set to attToBeSigned.
 2891 + Verify that attested contains a TPMS_CERTIFY_INFO structure,
 2892 whose name field contains a valid Name for pubArea, as
 2893 computed using the algorithm in the nameAlg field of pubArea
 2894 using the procedure specified in [TPMv2-Part1] section 16.
 2895
 2896 If x5c is present, this indicates that the attestation type is
 2897 not ECDA. In this case:
 2898
 2899 + Verify the sig is a valid signature over certInfo using the
 2900 attestation public key in x5c with the algorithm specified in
 2901 alg.
 2902 + Verify that x5c meets the requirements in 7.3.1 TPM
 2903 attestation statement certificate requirements.
 2904 + If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4
 2905 (id-fido-gen-ce-aaguid) verify that the value of this
 2906 extension matches the AAGUID in authenticatorData.
 2907 + If successful, return attestation type Privacy CA and trust
 2908 path x5c.
 2909
 2910 If ecdAaKeyId is present, then the attestation type is ECDA.
 2911
 2912 + Perform ECDA-Verify on sig to verify that it is a valid
 2913 signature over certInfo (see [FIDOecdaaAlgorithm]).
 2914 + If successful, return attestation type ECDA and the
 2915 identifier of the ECDA-Issuer public key ecdAaKeyId.
 2916
 2917 **7.3.1. TPM attestation statement certificate requirements**
 2918
 2919 TPM attestation certificate MUST have the following fields/extensions:
 2920 * Version must be set to 3.
 * Subject field MUST be set to empty.

- 2848 * The Subject Alternative Name extension must be set as defined in [TPMv2-EK-Profile] section 3.2.9.
- 2849
- 2850 * The Extended Key Usage extension MUST contain the "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8) tcg-kp-AIKCertificate(3)" OID.
- 2851
- 2852 * The Basic Constraints extension MUST have the CA component set to false.
- 2853
- 2854 * 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].
- 2855

7.4. Android Key Attestation Statement Format

When the authenticator in question is a platform-provided Authenticator on the Android "N" or later platform, the attestation statement is based on the Android key attestation. In these cases, the attestation statement is produced by a component 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:

```

2885 $$attStmtType ::= (
2886     fmt: "android-key",
2887     attStmt: androidStmtFormat
2888 )
2889
2890 androidStmtFormat = {
2891     alg: COSEAlgorithmIdentifier,
2892     sig: bytes,
2893     x5c: [ credCert: bytes, * (caCert: bytes) ]
2894 }
    
```

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

Verification is performed as follows:

- + Let authenticatorData denote the authenticator data claimed to have been used for the attestation, and let clientDataHash denote the hash of the serialized client data.

- 2921 * The Subject Alternative Name extension must be set as defined in [TPMv2-EK-Profile] section 3.2.9.
- 2922
- 2923 * The Extended Key Usage extension MUST contain the "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8) tcg-kp-AIKCertificate(3)" OID.
- 2924
- 2925 * The Basic Constraints extension MUST have the CA component set to false.
- 2926
- 2927 * 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].
- 2928

7.4. Android Key Attestation Statement Format

When the authenticator in question is a platform-provided Authenticator on the Android "N" or later platform, the attestation statement is based on the Android key attestation. In these cases, the attestation statement is produced by a component 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:

```

2951 $$attStmtType ::= (
2952     fmt: "android-key",
2953     attStmt: androidStmtFormat
2954 )
2955
2956 androidStmtFormat = {
2957     alg: COSEAlgorithmIdentifier,
2958     sig: bytes,
2959     x5c: [ credCert: bytes, * (caCert: bytes) ]
2960 }
    
```

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

Verification is performed as follows:

- + Let authenticatorData denote the authenticator data claimed to have been used for the attestation, and let clientDataHash denote the hash of the serialized client data.

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

7.5. Android SafetyNet Attestation Statement Format

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

Attestation statement format identifier
android-safetynet

Attestation types supported
Basic

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

```

$$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 to form attToBeSigned.

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

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

7.5. Android SafetyNet Attestation Statement Format

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

Attestation statement format identifier
android-safetynet

Attestation types supported
Basic

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

```

$$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 to form attToBeSigned.

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

298E Verification procedure
 298F Verification is performed as follows:
 2990
 2991 + Verify that the given attestation statement is valid CBOR
 2992 conforming to the syntax defined above.
 2993 + Verify that response is a valid SafetyNet response of version
 2994 ver.
 2995 + Verify that the nonce in the response is identical to the
 2996 concatenation of the authenticatorData and clientDataHash.
 2997 + Verify that the attestation certificate is issued to the
 2998 hostname "attest.android.com" (see SafetyNet online
 2999 documentation).
 3000 + Verify that the ctsProfileMatch attribute in the payload of
 3001 response is true.
 3002 + If successful, return attestation type Basic with the trust
 3003 path set to the above attestation certificate.
 3004
 3005
 3006 7.6. FIDO U2F Attestation Statement Format
 3007
 3008 This attestation statement format is used with FIDO U2F authenticators
 3009 using the formats defined in [FIDO-U2F-Message-Formats].
 3010
 3011 Attestation statement format identifier
 3012 fido-u2f
 3013
 3014 Attestation types supported
 3015 Basic, self attestation
 3016
 3017 Syntax
 3018 The syntax of a FIDO U2F attestation statement is defined as
 3019 follows:
 3020
 3021 \$\$attStmtType ::= (
 3022 fmt: "fido-u2f",
 3023 attStmt: u2fStmtFormat
 3024)
 3025
 3026 u2fStmtFormat = {
 3027 x5c: [attestnCert: bytes, * (caCert: bytes)],
 3028 sig: bytes
 3029 }
 3030
 3031 The semantics of the above fields are as follows:
 3032
 3033 x5c
 3034 The elements of this array contain the attestation
 3035 certificate and its certificate chain, each encoded in
 3036 X.509 format. The attestation certificate must be the
 3037 first element in the array.
 3038
 3039 sig
 3040 The attestation signature.
 3041
 3042 Signing procedure
 3043 If the credential public key of the given credential is not of
 3044 algorithm -7 ("ES256"), stop and return an error.
 3045
 3046 Let authenticatorData denote the authenticator data for the
 3047 attestation, and let clientDataHash denote the hash of the
 3048 serialized client data.
 3049
 3050 If clientDataHash is 256 bits long, set tbsHash to this value.
 3051 Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
 3052
 3053 Generate a Registration Response Message as specified in
 3054 [FIDO-U2F-Message-Formats] section 4.3, with the application
 3055 parameter set to the SHA-256 hash of the RP ID associated with
 3056 the given credential, the challenge parameter set to tbsHash,
 3057 and the key handle parameter set to the credential ID of the

3061 Verification procedure
 3062 Verification is performed as follows:
 3063
 3064 + Verify that the given attestation statement is valid CBOR
 3065 conforming to the syntax defined above.
 3066 + Verify that response is a valid SafetyNet response of version
 3067 ver.
 3068 + Verify that the nonce in the response is identical to the
 3069 concatenation of the authenticatorData and clientDataHash.
 3070 + Verify that the attestation certificate is issued to the
 3071 hostname "attest.android.com" (see SafetyNet online
 3072 documentation).
 3073 + Verify that the ctsProfileMatch attribute in the payload of
 3074 response is true.
 3075 + If successful, return attestation type Basic with the trust
 3076 path set to the above attestation certificate.
 3077
 3078
 3079 7.6. FIDO U2F Attestation Statement Format
 3080
 3081 This attestation statement format is used with FIDO U2F authenticators
 3082 using the formats defined in [FIDO-U2F-Message-Formats].
 3083
 3084 Attestation statement format identifier
 3085 fido-u2f
 3086
 3087 Attestation types supported
 3088 Basic, self attestation
 3089
 3090 Syntax
 3091 The syntax of a FIDO U2F attestation statement is defined as
 3092 follows:
 3093
 3094 \$\$attStmtType ::= (
 3095 fmt: "fido-u2f",
 3096 attStmt: u2fStmtFormat
 3097)
 3098
 3099 u2fStmtFormat = {
 3100 x5c: [attestnCert: bytes, * (caCert: bytes)],
 3101 sig: bytes
 3102 }
 3103
 3104 The semantics of the above fields are as follows:
 3105
 3106 x5c
 3107 The elements of this array contain the attestation
 3108 certificate and its certificate chain, each encoded in
 3109 X.509 format. The attestation certificate must be the
 3110 first element in the array.
 3111
 3112 sig
 3113 The attestation signature.
 3114
 3115 Signing procedure
 3116 If the credential public key of the given credential is not of
 3117 algorithm -7 ("ES256"), stop and return an error.
 3118
 3119 Let authenticatorData denote the authenticator data for the
 3120 attestation, and let clientDataHash denote the hash of the
 3121 serialized client data.
 3122
 3123 If clientDataHash is 256 bits long, set tbsHash to this value.
 3124 Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
 3125
 3126 Generate a Registration Response Message as specified in
 3127 [FIDO-U2F-Message-Formats] section 4.3, with the application
 3128 parameter set to the SHA-256 hash of the RP ID associated with
 3129 the given credential, the challenge parameter set to tbsHash,
 3130 and the key handle parameter set to the credential ID of the

305E given credential. Set the raw signature part of this
305F Registration Response Message (i.e., without the user public
306C key, key handle, and attestation certificates) as sig and set
3061 the attestation certificates of the attestation public key as
3062 x5c.
3063

3064 Verification procedure

3065 Verification is performed as follows:
3066

- 3067 + Verify that the given attestation statement is valid CBOR
3068 conforming to the syntax defined above.
- 3069 + If x5c is not a certificate for an ECDSA public key over the
307C P-256 curve, stop verification and return an error.
- 3071 + Let authenticatorData denote the authenticator data claimed to
3072 have been used for the attestation, and let clientDataHash
3073 denote the hash of the serialized client data.
- 3074 + If clientDataHash is 256 bits long, set tbsHash to this value.
3075 Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
- 3076 + From authenticatorData, extract the claimed RP ID hash, the
3077 claimed credential ID and the claimed credential public key.
- 307E + Generate the claimed to-be-signed data as specified in
307F [FIDO-U2F-Message-Formats] section 4.3, with the application
308C parameter set to the claimed RP ID hash, the challenge
3081 parameter set to tbsHash, the key handle parameter set to the
3082 claimed credential ID of the given credential, and the user
3083 public key parameter set to the claimed credential public key.
- 3084 + Verify that the sig is a valid ECDSA P-256 signature over the
3085 to-be-signed data constructed above.
- 308E + If successful, return attestation type Basic with the trust
3087 path set to x5c.
308E

308E 8. WebAuthn Extensions
309C

3091 The mechanism for generating public key credentials, as well as
3092 requesting and generating Authentication assertions, as defined in 4
3093 Web Authentication API, can be extended to suit particular use cases.
3094 Each case is addressed by defining a registration extension and/or an
3095 authentication extension.
309E

3097 Every extension is a client extension, meaning that the extension
309E involves communication with and processing by the client. Client
3095 extensions define the following steps and data:

- 310C * navigator.credentials.create() extension request parameters and
3101 response values for registration extensions.
- 3102 * navigator.credentials.get() extension request parameters and
3103 response values for authentication extensions.
- 3104 * Client extension processing for registration extensions and
3105 authentication extensions.
310E

3107 When creating a public key credential or requesting an authentication
310E assertion, a Relying Party can request the use of a set of extensions.
3105 These extensions will be invoked during the requested operation if they
3110 are supported by the client and/or the authenticator. The Relying Party
3111 sends the client extension input for each extension in the get() call
3112 (for authentication extensions) or create() call (for registration
3113 extensions) to the client platform. The client platform performs client
3114 extension processing for each extension that it supports, and augments
3115 the client data as specified by each extension, by including the
3116 extension identifier and client extension output values.
3117

3118 An extension can also be an authenticator extension, meaning that the
3119 extension involves communication with and processing by the
312C authenticator. Authenticator extensions define the following steps and
3121 data:

- 3122 * authenticatorMakeCredential extension request parameters and
3123 response values for registration extensions.
- 3124 * authenticatorGetAssertion extension request parameters and response
3125 values for authentication extensions.
- 312E * Authenticator extension processing for registration extensions and
3127 authentication extensions.

3131 given credential. Set the raw signature part of this
3132 Registration Response Message (i.e., without the user public
3133 key, key handle, and attestation certificates) as sig and set
3134 the attestation certificates of the attestation public key as
3135 x5c.
313E

3137 Verification procedure

313E Verification is performed as follows:
313F

- 314C + Verify that the given attestation statement is valid CBOR
3141 conforming to the syntax defined above.
- 3142 + If x5c is not a certificate for an ECDSA public key over the
3143 P-256 curve, stop verification and return an error.
- 3144 + Let authenticatorData denote the authenticator data claimed to
314E have been used for the attestation, and let clientDataHash
314F denote the hash of the serialized client data.
- 3147 + If clientDataHash is 256 bits long, set tbsHash to this value.
314E Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
- 314E + From authenticatorData, extract the claimed RP ID hash, the
315C claimed credential ID and the claimed credential public key.
- 3151 + Generate the claimed to-be-signed data as specified in
3152 [FIDO-U2F-Message-Formats] section 4.3, with the application
3153 parameter set to the claimed RP ID hash, the challenge
3154 parameter set to tbsHash, the key handle parameter set to the
3155 claimed credential ID of the given credential, and the user
315E public key parameter set to the claimed credential public key.
- 3157 + Verify that the sig is a valid ECDSA P-256 signature over the
315E to-be-signed data constructed above.
- 315E + If successful, return attestation type Basic with the trust
316C path set to x5c.
3161

3162 8. WebAuthn Extensions
316C

3164 The mechanism for generating public key credentials, as well as
3165 requesting and generating Authentication assertions, as defined in 4
3166 Web Authentication API, can be extended to suit particular use cases.
3167 Each case is addressed by defining a registration extension and/or an
316E authentication extension.
316E

317C Every extension is a client extension, meaning that the extension
3171 involves communication with and processing by the client. Client
3172 extensions define the following steps and data:

- 3173 * navigator.credentials.create() extension request parameters and
3174 response values for registration extensions.
- 317E * navigator.credentials.get() extension request parameters and
317F response values for authentication extensions.
- 3177 * Client extension processing for registration extensions and
317E authentication extensions.
317E

318C When creating a public key credential or requesting an authentication
3181 assertion, a Relying Party can request the use of a set of extensions.
3182 These extensions will be invoked during the requested operation if they
3183 are supported by the client and/or the authenticator. The Relying Party
3184 sends the client extension input for each extension in the get() call
318E (for authentication extensions) or create() call (for registration
318E extensions) to the client platform. The client platform performs client
3187 extension processing for each extension that it supports, and augments
318E the client data as specified by each extension, by including the
318E extension identifier and client extension output values.
319C

3191 An extension can also be an authenticator extension, meaning that the
3192 extension involves communication with and processing by the
319C authenticator. Authenticator extensions define the following steps and
3194 data:

- 319E * authenticatorMakeCredential extension request parameters and
319E response values for registration extensions.
- 3197 * authenticatorGetAssertion extension request parameters and response
319E values for authentication extensions.
- 319E * Authenticator extension processing for registration extensions and
320C authentication extensions.

3128 For authenticator extensions, as part of the client extension
3129 processing, the client also creates the CBOR authenticator extension
3130 input value for each extension (often based on the corresponding client
3131 extension input value), and passes them to the authenticator in the
3132 create() call (for registration extensions) or the get() call (for
3133 authentication extensions). These authenticator extension input values
3134 are represented in CBOR and passed as name-value pairs, with the
3135 extension identifier as the name, and the corresponding authenticator
3136 extension input as the value. The authenticator, in turn, performs
3137 additional processing for the extensions that it supports, and returns
3138 the CBOR authenticator extension output for each as specified by the
3139 extension. Part of the client extension processing for authenticator
3140 extensions is to use the authenticator extension output as an input to
3141 creating the client extension output.

3142 All WebAuthn extensions are optional for both clients and
3143 authenticators. Thus, any extensions requested by a Relying Party may
3144 be ignored by the client browser or OS and not passed to the
3145 authenticator at all, or they may be ignored by the authenticator.
3146 Ignoring an extension is never considered a failure in WebAuthn API
3147 processing, so when Relying Parties include extensions with any API
3148 calls, they must be prepared to handle cases where some or all of those
3149 extensions are ignored.

3150 Clients wishing to support the widest possible range of extensions may
3151 choose to pass through any extensions that they do not recognize to
3152 authenticators, generating the authenticator extension input by simply
3153 encoding the client extension input in CBOR. All WebAuthn extensions
3154 MUST be defined in such a way that this implementation choice does not
3155 endanger the user's security or privacy. For instance, if an extension
3156 requires client processing, it could be defined in a manner that
3157 ensures such a nave pass-through will produce a semantically invalid
3158 authenticator extension input value, resulting in the extension being
3159 ignored by the authenticator. Since all extensions are optional, this
3160 will not cause a functional failure in the API operation. Likewise,
3161 clients can choose to produce a client extension output value for an
3162 extension that it does not understand by encoding the authenticator
3163 extension output value into JSON, provided that the CBOR output uses
3164 only types present in JSON.

3165 The IANA "WebAuthn Extension Identifier" registry established by
3166 [WebAuthn-Registries] should be consulted for an up-to-date list of
3167 registered WebAuthn Extensions.

3168 8.1. Extension Identifiers

3169 Extensions are identified by a string, called an extension identifier,
3170 chosen by the extension author.

3171 Extension identifiers SHOULD be registered per [WebAuthn-Registries]
3172 "Registries for Web Authentication (WebAuthn)". All registered
3173 extension identifiers are unique amongst themselves as a matter of
3174 course.

3175 Unregistered extension identifiers should aim to be globally unique,
3176 e.g., by including the defining entity such as myCompany_extension.

3177 All extension identifiers MUST be a maximum of 32 octets in length and
3178 MUST consist only of printable USASCII characters, excluding backslash
3179 and doublequote, i.e., VCHAR as defined in [RFC5234] but without %x22
3180 and %x5c. Implementations MUST match WebAuthn extension identifiers in
3181 a case-sensitive fashion.

3182 Extensions that may exist in multiple versions should take care to
3183 include a version in their identifier. In effect, different versions
3184 are thus treated as different extensions, e.g., myCompany_extension_01

3185 9 Defined Extensions defines an initial set of extensions and their
3186 identifiers. See the IANA "WebAuthn Extension Identifier" registry

3187

3201 For authenticator extensions, as part of the client extension
3202 processing, the client also creates the CBOR authenticator extension
3203 input value for each extension (often based on the corresponding client
3204 extension input value), and passes them to the authenticator in the
3205 create() call (for registration extensions) or the get() call (for
3206 authentication extensions). These authenticator extension input values
3207 are represented in CBOR and passed as name-value pairs, with the
3208 extension identifier as the name, and the corresponding authenticator
3209 extension input as the value. The authenticator, in turn, performs
3210 additional processing for the extensions that it supports, and returns
3211 the CBOR authenticator extension output for each as specified by the
3212 extension. Part of the client extension processing for authenticator
3213 extensions is to use the authenticator extension output as an input to
3214 creating the client extension output.

3215 All WebAuthn extensions are optional for both clients and
3216 authenticators. Thus, any extensions requested by a Relying Party may
3217 be ignored by the client browser or OS and not passed to the
3218 authenticator at all, or they may be ignored by the authenticator.
3219 Ignoring an extension is never considered a failure in WebAuthn API
3220 processing, so when Relying Parties include extensions with any API
3221 calls, they must be prepared to handle cases where some or all of those
3222 extensions are ignored.

3223 Clients wishing to support the widest possible range of extensions may
3224 choose to pass through any extensions that they do not recognize to
3225 authenticators, generating the authenticator extension input by simply
3226 encoding the client extension input in CBOR. All WebAuthn extensions
3227 MUST be defined in such a way that this implementation choice does not
3228 endanger the user's security or privacy. For instance, if an extension
3229 requires client processing, it could be defined in a manner that
3230 ensures such a nave pass-through will produce a semantically invalid
3231 authenticator extension input value, resulting in the extension being
3232 ignored by the authenticator. Since all extensions are optional, this
3233 will not cause a functional failure in the API operation. Likewise,
3234 clients can choose to produce a client extension output value for an
3235 extension that it does not understand by encoding the authenticator
3236 extension output value into JSON, provided that the CBOR output uses
3237 only types present in JSON.

3238 The IANA "WebAuthn Extension Identifier" registry established by
3239 [WebAuthn-Registries] should be consulted for an up-to-date list of
3240 registered WebAuthn Extensions.

3241 8.1. Extension Identifiers

3242 Extensions are identified by a string, called an extension identifier,
3243 chosen by the extension author.

3244 Extension identifiers SHOULD be registered per [WebAuthn-Registries]
3245 "Registries for Web Authentication (WebAuthn)". All registered
3246 extension identifiers are unique amongst themselves as a matter of
3247 course.

3248 Unregistered extension identifiers should aim to be globally unique,
3249 e.g., by including the defining entity such as myCompany_extension.

3250 All extension identifiers MUST be a maximum of 32 octets in length and
3251 MUST consist only of printable USASCII characters, excluding backslash
3252 and doublequote, i.e., VCHAR as defined in [RFC5234] but without %x22
3253 and %x5c. Implementations MUST match WebAuthn extension identifiers in
3254 a case-sensitive fashion.

3255 Extensions that may exist in multiple versions should take care to
3256 include a version in their identifier. In effect, different versions
3257 are thus treated as different extensions, e.g., myCompany_extension_01

3258 9 Defined Extensions defines an initial set of extensions and their
3259 identifiers. See the IANA "WebAuthn Extension Identifier" registry

3260

319E established by [WebAuthn-Registries] for an up-to-date list of
 319F registered WebAuthn Extension Identifiers.
 3200

3201 8.2. Defining extensions

3202
 3203 A definition of an extension must specify an extension identifier, a
 3204 client extension input argument to be sent via the `get()` or `create()`
 3205 call, the client extension processing rules, and a client extension
 3206 output value. If the extension communicates with the authenticator
 3207 (meaning it is an authenticator extension), it must also specify the
 3208 CBOR authenticator extension input argument sent via the
 3209 `authenticatorGetAssertion` or `authenticatorMakeCredential` call, the
 3210 authenticator extension processing rules, and the CBOR authenticator
 3211 extension output value.
 3212

3213 Any client extension that is processed by the client MUST return a
 3214 client extension output value so that the Relying Party knows that the
 3215 extension was honored by the client. Similarly, any extension that
 3216 requires authenticator processing MUST return an authenticator
 3217 extension output to let the Relying Party know that the extension was
 3218 honored by the authenticator. If an extension does not otherwise
 3219 require any result values, it SHOULD be defined as returning a JSON
 3220 Boolean client extension output result, set to true to signify that the
 3221 extension was understood and processed. Likewise, any authenticator
 3222 extension that does not otherwise require any result values MUST return
 3223 a value and SHOULD return a CBOR Boolean authenticator extension output
 3224 result, set to true to signify that the extension was understood and
 3225 processed.
 3226

3227 8.3. Extending request parameters

3228
 3229 An extension defines one or two request arguments. The client extension
 3230 input, which is a value that can be encoded in JSON, is passed from the
 3231 Relying Party to the client in the `get()` or `create()` call, while the
 3232 CBOR authenticator extension input is passed from the client to the
 3233 authenticator for authenticator extensions during the processing of
 3234 these calls.
 3235

3236 A Relying Party simultaneously requests the use of an extension and
 3237 sets its client extension input by including an entry in the extensions
 3238 option to the `create()` or `get()` call. The entry key is the extension
 3239 identifier and the value is the client extension input.

```
3240 var assertionPromise = navigator.credentials.get({
3241   publicKey: {
3242     challenge: "...",
3243     extensions: {
3244       "webauthnExample_foobar": 42
3245     }
3246   }
3247 });
```

3248

3249 Extension definitions MUST specify the valid values for their client
 3250 extension input. Clients SHOULD ignore extensions with an invalid
 3251 client extension input. If an extension does not require any parameters
 3252 from the Relying Party, it SHOULD be defined as taking a Boolean client
 3253 argument, set to true to signify that the extension is requested by the
 3254 Relying Party.
 3255

3256 Extensions that only affect client processing need not specify
 3257 authenticator extension input. Extensions that have authenticator
 3258 processing MUST specify the method of computing the authenticator
 3259 extension input from the client extension input. For extensions that do
 3260 not require input parameters and are defined as taking a Boolean client
 3261 extension input value set to true, this method SHOULD consist of
 3262 passing an authenticator extension input value of true (CBOR major type
 3263 7, value 21).
 3264

3265 Note: Extensions should aim to define authenticator arguments that are
 3266 as small as possible. Some authenticators communicate over
 3267 low-bandwidth links such as Bluetooth Low-Energy or NFC.

3271 established by [WebAuthn-Registries] for an up-to-date list of
 3272 registered WebAuthn Extension Identifiers.
 3273

3274 8.2. Defining extensions

3275
 3276 A definition of an extension must specify an extension identifier, a
 3277 client extension input argument to be sent via the `get()` or `create()`
 3278 call, the client extension processing rules, and a client extension
 3279 output value. If the extension communicates with the authenticator
 3280 (meaning it is an authenticator extension), it must also specify the
 3281 CBOR authenticator extension input argument sent via the
 3282 `authenticatorGetAssertion` or `authenticatorMakeCredential` call, the
 3283 authenticator extension processing rules, and the CBOR authenticator
 3284 extension output value.
 3285

3286 Any client extension that is processed by the client MUST return a
 3287 client extension output value so that the Relying Party knows that the
 3288 extension was honored by the client. Similarly, any extension that
 3289 requires authenticator processing MUST return an authenticator
 3290 extension output to let the Relying Party know that the extension was
 3291 honored by the authenticator. If an extension does not otherwise
 3292 require any result values, it SHOULD be defined as returning a JSON
 3293 Boolean client extension output result, set to true to signify that the
 3294 extension was understood and processed. Likewise, any authenticator
 3295 extension that does not otherwise require any result values MUST return
 3296 a value and SHOULD return a CBOR Boolean authenticator extension output
 3297 result, set to true to signify that the extension was understood and
 3298 processed.
 3299

3300 8.3. Extending request parameters

3301
 3302 An extension defines one or two request arguments. The client extension
 3303 input, which is a value that can be encoded in JSON, is passed from the
 3304 Relying Party to the client in the `get()` or `create()` call, while the
 3305 CBOR authenticator extension input is passed from the client to the
 3306 authenticator for authenticator extensions during the processing of
 3307 these calls.
 3308

3309 A Relying Party simultaneously requests the use of an extension and
 3310 sets its client extension input by including an entry in the extensions
 3311 option to the `create()` or `get()` call. The entry key is the extension
 3312 identifier and the value is the client extension input.

```
3313 var assertionPromise = navigator.credentials.get({
3314   publicKey: {
3315     challenge: "...",
3316     extensions: {
3317       "webauthnExample_foobar": 42
3318     }
3319   }
3320 });
```

3321

3322 Extension definitions MUST specify the valid values for their client
 3323 extension input. Clients SHOULD ignore extensions with an invalid
 3324 client extension input. If an extension does not require any parameters
 3325 from the Relying Party, it SHOULD be defined as taking a Boolean client
 3326 argument, set to true to signify that the extension is requested by the
 3327 Relying Party.
 3328

3329 Extensions that only affect client processing need not specify
 3330 authenticator extension input. Extensions that have authenticator
 3331 processing MUST specify the method of computing the authenticator
 3332 extension input from the client extension input. For extensions that do
 3333 not require input parameters and are defined as taking a Boolean client
 3334 extension input value set to true, this method SHOULD consist of
 3335 passing an authenticator extension input value of true (CBOR major type
 3336 7, value 21).
 3337

3338 Note: Extensions should aim to define authenticator arguments that are
 3339 as small as possible. Some authenticators communicate over
 3340 low-bandwidth links such as Bluetooth Low-Energy or NFC.

8.4. Client extension processing

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

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

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

8.5. Authenticator extension processing

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

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

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

8.6. Example Extension

This section is not normative.

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

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

```
var assertionPromise =
  navigator.credentials.get({
    publicKey: {
      challenge: "SGFuIFNvbG8gc2hvdCBmaXJzdC4",
      allowCredentials: [], /* Empty filter */
      extensions: { 'webauthnExample_geo': true }
    }
  });
```

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

The extension requires the authenticator to specify its geolocation in the authenticator extension output, if known. The extension e.g. specifies that the location shall be encoded as a two-element array of floating point numbers, encoded with CBOR. An authenticator does this

8.4. Client extension processing

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

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

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

8.5. Authenticator extension processing

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

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

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

8.6. Example Extension

This section is not normative.

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

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

```
var assertionPromise =
  navigator.credentials.get({
    publicKey: {
      challenge: "SGFuIFNvbG8gc2hvdCBmaXJzdC4",
      allowCredentials: [], /* Empty filter */
      extensions: { 'webauthnExample_geo': true }
    }
  });
```

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

The extension requires the authenticator to specify its geolocation in the authenticator extension output, if known. The extension e.g. specifies that the location shall be encoded as a two-element array of floating point numbers, encoded with CBOR. An authenticator does this

```

333E by including it in the authenticator data. As an example, authenticator
333F data may be as follows (notation taken from [RFC7049]):
3340 81 (hex) -- Flags, ED and UP both set.
3341 20 05 58 1F -- Signature counter
3342 A1 -- CBOR map of one element
3343 73 -- Key 1: CBOR text string of 19 byt
3344 es
3345 77 65 62 61 75 74 68 6E 45 78 61
3346 6D 70 6C 65 5F 67 65 6F -- "webauthnExample_geo" [=UTF-8 enc
3347 oded=] string
3348 82 -- Value 1: CBOR array of two elemen
3349 ts
3350 FA 42 82 1E B3 -- Element 1: Latitude as CBOR encod
3351 ed float
3352 FA C1 5F E3 7F -- Element 2: Longitude as CBOR enco
3353 ded float
3354
3355 The extension defines the client extension output to be the geolocation
3356 information, if known, as a GeoJSON [GeoJSON] point. The client
3357 constructs the following client data:
3358 {
3359   'extensions': {
3360     'webauthnExample_geo': {
3361       'type': 'Point',
3362       'coordinates': [65.059962, -13.993041]
3363     }
3364   }
3365 }
3366
3367 9. Defined Extensions
3368
3369 This section defines the initial set of extensions to be registered in
3370 the IANA "WebAuthn Extension Identifier" registry established by
3371 [WebAuthn-Registries]. These are recommended for implementation by user
3372 agents targeting broad interoperability.
3373
3374 9.1. FIDO AppId Extension (appid)
3375
3376 This authentication extension allows Relying Parties that have
3377 previously registered a credential using the legacy FIDO JavaScript
3378 APIs to request an assertion. Specifically, this extension allows
3379 Relying Parties to specify an appid [FIDO-APPID] to overwrite the
3380 otherwise computed rpId. This extension is only valid if used during
3381 the get() call; other usage will result in client error.
3382
3383 Extension identifier
3384 appid
3385
3386 Client extension input
3387 A single JSON string specifying a FIDO appid.
3388
3389 Client extension processing
3390 If rpId is present, reject promise with a DOMException whose
3391 name is "NotAllowedError", and terminate this algorithm. Replace
3392 the calculation of rpId in Step 3 of 4.1.4 Use an existing
3393 credential to make an assertion - PublicKeyCredential's
3394 [[DiscoverFromExternalSource]](options) method with the
3395 following procedure: The client uses the value of appid to
3396 perform the Appid validation procedure (as defined by
3397 [FIDO-APPID]). If valid, the value of rpId for all client
3398 processing should be replaced by the value of appid.
3399
3400 Client extension output
3401 Returns the JSON value true to indicate to the RP that the
3402 extension was acted upon
3403
3404 Authenticator extension input
3405 None.
3406
3407

```

```

3411 by including it in the authenticator data. As an example, authenticator
3412 data may be as follows (notation taken from [RFC7049]):
3413 81 (hex) -- Flags, ED and UP both set.
3414 20 05 58 1F -- Signature counter
3415 A1 -- CBOR map of one element
3416 73 -- Key 1: CBOR text string of 19 byt
3417 es
3418 77 65 62 61 75 74 68 6E 45 78 61
3419 6D 70 6C 65 5F 67 65 6F -- "webauthnExample_geo" [=UTF-8 enc
3420 oded=] string
3421 82 -- Value 1: CBOR array of two elemen
3422 ts
3423 FA 42 82 1E B3 -- Element 1: Latitude as CBOR encod
3424 ed float
3425 FA C1 5F E3 7F -- Element 2: Longitude as CBOR enco
3426 ded float
3427
3428 The extension defines the client extension output to be the geolocation
3429 information, if known, as a GeoJSON [GeoJSON] point. The client
3430 constructs the following client data:
3431 {
3432   'extensions': {
3433     'webauthnExample_geo': {
3434       'type': 'Point',
3435       'coordinates': [65.059962, -13.993041]
3436     }
3437   }
3438 }
3439
3440 9. Defined Extensions
3441
3442 This section defines the initial set of extensions to be registered in
3443 the IANA "WebAuthn Extension Identifier" registry established by
3444 [WebAuthn-Registries]. These are recommended for implementation by user
3445 agents targeting broad interoperability.
3446
3447 9.1. FIDO AppId Extension (appid)
3448
3449 This authentication extension allows Relying Parties that have
3450 previously registered a credential using the legacy FIDO JavaScript
3451 APIs to request an assertion. Specifically, this extension allows
3452 Relying Parties to specify an appid [FIDO-APPID] to overwrite the
3453 otherwise computed rpId. This extension is only valid if used during
3454 the get() call; other usage will result in client error.
3455
3456 Extension identifier
3457 appid
3458
3459 Client extension input
3460 A single JSON string specifying a FIDO appid.
3461
3462 Client extension processing
3463 If rpId is present, reject promise with a DOMException whose
3464 name is "NotAllowedError", and terminate this algorithm. Replace
3465 the calculation of rpId in Step 3 of 4.1.4 Use an existing
3466 credential to make an assertion - PublicKeyCredential's
3467 [[DiscoverFromExternalSource]](options) method with the
3468 following procedure: The client uses the value of appid to
3469 perform the Appid validation procedure (as defined by
3470 [FIDO-APPID]). If valid, the value of rpId for all client
3471 processing should be replaced by the value of appid.
3472
3473 Client extension output
3474 Returns the JSON value true to indicate to the RP that the
3475 extension was acted upon
3476
3477 Authenticator extension input
3478 None.
3479
3480

```

340E Authenticator extension processing
 340F None.
 341C
 3411 Authenticator extension output
 3412 None.
 3413
 3414 9.2. Simple Transaction Authorization Extension (txAuthSimple)
 341E
 341F This registration extension and authentication extension allows for a
 3420 simple form of transaction authorization. A Relying Party can specify a
 3421 prompt string, intended for display on a trusted device on the
 3422 authenticator.
 3423
 3424 Extension identifier
 3425 txAuthSimple
 3426
 3427 Client extension input
 3428 A single JSON string prompt.
 3429
 342A Client extension processing
 342B None, except creating the authenticator extension input from the
 342C client extension input.
 342D
 342E Client extension output
 342F Returns the authenticator extension output string UTF-8 decoded
 3430 into a JSON string
 3431
 3432 Authenticator extension input
 3433 The client extension input encoded as a CBOR text string (major
 3434 type 3).
 3435
 3436 Authenticator extension processing
 3437 The authenticator MUST display the prompt to the user before
 3438 performing either user verification or test of user presence.
 3439 The authenticator may insert line breaks if needed.
 3440
 3441 Authenticator extension output
 3442 A single CBOR string, representing the prompt as displayed
 3443 (including any eventual line breaks).
 3444
 3445 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
 3446
 3447 This registration extension and authentication extension allows images
 3448 to be used as transaction authorization prompts as well. This allows
 3449 authenticators without a font rendering engine to be used and also
 3450 supports a richer visual appearance.
 3451
 3452 Extension identifier
 3453 txAuthGeneric
 3454
 3455 Client extension input
 3456 A CBOR map defined as follows:
 3457
 3458 txAuthGenericArg = {
 3459 contentType: text, ; MIME-Type of the content, e.g.
 3460 "image/png"
 3461 content: bytes
 3462 }
 3463
 3464 Client extension processing
 3465 None, except creating the authenticator extension input from the
 3466 client extension input.
 3467
 3468 Client extension output
 3469 Returns the base64url encoding of the authenticator extension
 3470 output value as a JSON string
 3471
 3472 Authenticator extension input
 3473 The client extension input encoded as a CBOR map.
 3474
 3475
 3476
 3477

3481 Authenticator extension processing
 3482 None.
 3483
 3484 Authenticator extension output
 3485 None.
 3486
 3487 9.2. Simple Transaction Authorization Extension (txAuthSimple)
 3488
 3489 This registration extension and authentication extension allows for a
 3490 simple form of transaction authorization. A Relying Party can specify a
 3491 prompt string, intended for display on a trusted device on the
 3492 authenticator.
 3493
 3494 Extension identifier
 3495 txAuthSimple
 3496
 3497 Client extension input
 3498 A single JSON string prompt.
 3499
 3500 Client extension processing
 3501 None, except creating the authenticator extension input from the
 3502 client extension input.
 3503
 3504 Client extension output
 3505 Returns the authenticator extension output string UTF-8 decoded
 3506 into a JSON string
 3507
 3508 Authenticator extension input
 3509 The client extension input encoded as a CBOR text string (major
 3510 type 3).
 3511
 3512 Authenticator extension processing
 3513 The authenticator MUST display the prompt to the user before
 3514 performing either user verification or test of user presence.
 3515 The authenticator may insert line breaks if needed.
 3516
 3517 Authenticator extension output
 3518 A single CBOR string, representing the prompt as displayed
 3519 (including any eventual line breaks).
 3520
 3521 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
 3522
 3523 This registration extension and authentication extension allows images
 3524 to be used as transaction authorization prompts as well. This allows
 3525 authenticators without a font rendering engine to be used and also
 3526 supports a richer visual appearance.
 3527
 3528 Extension identifier
 3529 txAuthGeneric
 3530
 3531 Client extension input
 3532 A CBOR map defined as follows:
 3533
 3534 txAuthGenericArg = {
 3535 contentType: text, ; MIME-Type of the content, e.g.
 3536 "image/png"
 3537 content: bytes
 3538 }
 3539
 3540 Client extension processing
 3541 None, except creating the authenticator extension input from the
 3542 client extension input.
 3543
 3544 Client extension output
 3545 Returns the base64url encoding of the authenticator extension
 3546 output value as a JSON string
 3547
 3548 Authenticator extension input
 3549 The client extension input encoded as a CBOR map.
 3550
 3551

347E Authenticator extension processing
 347F The authenticator MUST display the content to the user before
 348C performing either user verification or test of user presence.
 3481 The authenticator may add other information below the content.
 3482 No changes are allowed to the content itself, i.e., inside
 3483 content boundary box.
 3484

348E Authenticator extension output
 348E The hash value of the content which was displayed. The
 3487 authenticator MUST use the same hash algorithm as it uses for
 348E the signature itself.
 348E

349C 9.4. Authenticator Selection Extension (authnSel)
 3491

3492 This registration extension allows a Relying Party to guide the
 3493 selection of the authenticator that will be leveraged when creating the
 3494 credential. It is intended primarily for Relying Parties that wish to
 349E tightly control the experience around credential creation.
 349E

3497 Extension identifier
 349E authnSel
 349E

350C Client extension input
 3501 A sequence of AAGUIDs:
 3502

3503 typedef sequence<AAGUID> AuthenticatorSelectionList;
 3504

350E Each AAGUID corresponds to an authenticator model that is
 350E acceptable to the Relying Party for this credential creation.
 3507 The list is ordered by decreasing preference.
 350E

350E An AAGUID is defined as an array containing the globally unique
 351C identifier of the authenticator model being sought.
 3511

3512 typedef BufferSource AAGUID;
 3513

3514 Client extension processing
 351E This extension can only be used during create(). If the client
 351E supports the Authenticator Selection Extension, it MUST use the
 3517 first available authenticator whose AAGUID is present in the
 351E AuthenticatorSelectionList. If none of the available
 351E authenticators match a provided AAGUID, the client MUST select
 352C an authenticator from among the available authenticators to
 3521 generate the credential.
 3522

3523 Client extension output
 3524 Returns the JSON value true to indicate to the RP that the
 352E extension was acted upon
 352E

3527 Authenticator extension input
 352E None.
 352E

353C Authenticator extension processing
 3531 None.
 3532

3533 Authenticator extension output
 3534 None.
 353E

353E 9.5. Supported Extensions Extension (exts)
 3537

353E This registration extension enables the Relying Party to determine
 353E which extensions the authenticator supports.
 354C

3541 Extension identifier
 3542 exts
 3543

3544 Client extension input
 354E The Boolean value true to indicate that this extension is
 354E requested by the Relying Party.
 3547

3551 Authenticator extension processing
 3552 The authenticator MUST display the content to the user before
 3553 performing either user verification or test of user presence.
 3554 The authenticator may add other information below the content.
 355E No changes are allowed to the content itself, i.e., inside
 355E content boundary box.
 3557

355E Authenticator extension output
 355E The hash value of the content which was displayed. The
 356C authenticator MUST use the same hash algorithm as it uses for
 3561 the signature itself.
 3562

3563 9.4. Authenticator Selection Extension (authnSel)
 3564

356E This registration extension allows a Relying Party to guide the
 356E selection of the authenticator that will be leveraged when creating the
 3567 credential. It is intended primarily for Relying Parties that wish to
 356E tightly control the experience around credential creation.
 356E

357C Extension identifier
 3571 authnSel
 3572

3573 Client extension input
 3574 A sequence of AAGUIDs:
 357E

357E typedef sequence<AAGUID> AuthenticatorSelectionList;
 357E

357E Each AAGUID corresponds to an authenticator model that is
 357E acceptable to the Relying Party for this credential creation.
 358C The list is ordered by decreasing preference.
 3581

358E An AAGUID is defined as an array containing the globally unique
 358E identifier of the authenticator model being sought.
 3584

358E typedef BufferSource AAGUID;
 358E

3587 Client extension processing
 358E This extension can only be used during create(). If the client
 358E supports the Authenticator Selection Extension, it MUST use the
 359C first available authenticator whose AAGUID is present in the
 3591 AuthenticatorSelectionList. If none of the available
 3592 authenticators match a provided AAGUID, the client MUST select
 359E an authenticator from among the available authenticators to
 359E generate the credential.
 359E

359E Client extension output
 3597 Returns the JSON value true to indicate to the RP that the
 359E extension was acted upon
 359E

360C Authenticator extension input
 3601 None.
 3602

3603 Authenticator extension processing
 3604 None.
 360E

360E Authenticator extension output
 3607 None.
 360E

360E 9.5. Supported Extensions Extension (exts)
 360E

361C This registration extension enables the Relying Party to determine
 3611 which extensions the authenticator supports.
 3612

3613 Extension identifier
 3614 exts
 361E

361E Client extension input
 361E The Boolean value true to indicate that this extension is
 361E requested by the Relying Party.
 362C

354E Client extension processing
 354F None, except creating the authenticator extension input from the
 3550 client extension input.
 3551
 3552 Client extension output
 3553 Returns the list of supported extensions as a JSON array of
 3554 extension identifier strings
 3555
 3556 Authenticator extension input
 3557 The Boolean value true, encoded in CBOR (major type 7, value
 3558 21).
 3559
 3560 Authenticator extension processing
 3561 The authenticator sets the authenticator extension output to be
 3562 a list of extensions that the authenticator supports, as defined
 3563 below. This extension can be added to attestation objects.
 3564
 3565 Authenticator extension output
 3566 The SupportedExtensions extension is a list (CBOR array) of
 3567 extension identifier (UTF-8 encoded strings).
 3568
 3569 9.6. User Verification Index Extension (uvi)
 3570
 3571 This registration extension and authentication extension enables use of
 3572 a user verification index.
 3573
 3574 Extension identifier
 3575 uvi
 3576
 3577 Client extension input
 3578 The Boolean value true to indicate that this extension is
 3579 requested by the Relying Party.
 3580
 3581 Client extension processing
 3582 None, except creating the authenticator extension input from the
 3583 client extension input.
 3584
 3585 Client extension output
 3586 Returns a JSON string containing the base64url encoding of the
 3587 authenticator extension output
 3588
 3589 Authenticator extension input
 3590 The Boolean value true, encoded in CBOR (major type 7, value
 3591 21).
 3592
 3593 Authenticator extension processing
 3594 The authenticator sets the authenticator extension output to be
 3595 a user verification index indicating the method used by the user
 3596 to authorize the operation, as defined below. This extension can
 3597 be added to attestation objects and assertions.
 3598
 3599 Authenticator extension output
 3600 The user verification index (UVI) is a value uniquely
 3601 identifying a user verification data record. The UVI is encoded
 3602 as CBOR byte string (type 0x58). Each UVI value MUST be specific
 3603 to the related key (in order to provide unlinkability). It also
 3604 must contain sufficient entropy that makes guessing impractical.
 3605 UVI values MUST NOT be reused by the Authenticator (for other
 3606 biometric data or users).
 3607
 3608 The UVI data can be used by servers to understand whether an
 3609 authentication was authorized by the exact same biometric data
 3610 as the initial key generation. This allows the detection and
 3611 prevention of "friendly fraud".
 3612
 3613 As an example, the UVI could be computed as SHA256(KeyID |
 3614 SHA256(rawUVI)), where the rawUVI reflects (a) the biometric
 3615 reference data, (b) the related OS level user ID and (c) an
 3616 identifier which changes whenever a factory reset is performed
 3617 for the device, e.g. rawUVI = biometricReferenceData |

3621 Client extension processing
 3622 None, except creating the authenticator extension input from the
 3623 client extension input.
 3624
 3625 Client extension output
 3626 Returns the list of supported extensions as a JSON array of
 3627 extension identifier strings
 3628
 3629 Authenticator extension input
 3630 The Boolean value true, encoded in CBOR (major type 7, value
 3631 21).
 3632
 3633 Authenticator extension processing
 3634 The authenticator sets the authenticator extension output to be
 3635 a list of extensions that the authenticator supports, as defined
 3636 below. This extension can be added to attestation objects.
 3637
 3638 Authenticator extension output
 3639 The SupportedExtensions extension is a list (CBOR array) of
 3640 extension identifier (UTF-8 encoded strings).
 3641
 3642 9.6. User Verification Index Extension (uvi)
 3643
 3644 This registration extension and authentication extension enables use of
 3645 a user verification index.
 3646
 3647 Extension identifier
 3648 uvi
 3649
 3650 Client extension input
 3651 The Boolean value true to indicate that this extension is
 3652 requested by the Relying Party.
 3653
 3654 Client extension processing
 3655 None, except creating the authenticator extension input from the
 3656 client extension input.
 3657
 3658 Client extension output
 3659 Returns a JSON string containing the base64url encoding of the
 3660 authenticator extension output
 3661
 3662 Authenticator extension input
 3663 The Boolean value true, encoded in CBOR (major type 7, value
 3664 21).
 3665
 3666 Authenticator extension processing
 3667 The authenticator sets the authenticator extension output to be
 3668 a user verification index indicating the method used by the user
 3669 to authorize the operation, as defined below. This extension can
 3670 be added to attestation objects and assertions.
 3671
 3672 Authenticator extension output
 3673 The user verification index (UVI) is a value uniquely
 3674 identifying a user verification data record. The UVI is encoded
 3675 as CBOR byte string (type 0x58). Each UVI value MUST be specific
 3676 to the related key (in order to provide unlinkability). It also
 3677 must contain sufficient entropy that makes guessing impractical.
 3678 UVI values MUST NOT be reused by the Authenticator (for other
 3679 biometric data or users).
 3680
 3681 The UVI data can be used by servers to understand whether an
 3682 authentication was authorized by the exact same biometric data
 3683 as the initial key generation. This allows the detection and
 3684 prevention of "friendly fraud".
 3685
 3686 As an example, the UVI could be computed as SHA256(KeyID |
 3687 SHA256(rawUVI)), where the rawUVI reflects (a) the biometric
 3688 reference data, (b) the related OS level user ID and (c) an
 3689 identifier which changes whenever a factory reset is performed
 3690 for the device, e.g. rawUVI = biometricReferenceData |

```

3618 OSLevelUserID | FactoryResetCounter.
3619
3620 Servers supporting UVI extensions MUST support a length of up to
3621 32 bytes for the UVI value.
3622
3623 Example for authenticator data containing one UVI extension
3624
3625 ... -- [=RP ID=] hash (32 bytes)
3626 81 -- UP and ED set
3627 00 00 00 01 -- (initial) signature counter
3628 -- all public key alg etc.
3629 A1 -- extension: CBOR map of one elemen
3630 t
3631 63 -- Key 1: CBOR text string of 3 byte
3632 s
3633 75 76 69 -- "uvi" [=UTF-8 encoded=] string
3634 58 20 -- Value 1: CBOR byte string with 0x
3635 20 bytes
3636 00 43 B8 E3 BE 27 95 8C -- the UVI value itself
3637 28 D5 74 BF 46 8A 85 CF
3638 46 9A 14 F0 E5 16 69 31
3639 DA 4B CF FF C1 BB 11 32
3640 82
3641
3642 9.7. Location Extension (loc)
3643
3644 The location registration extension and authentication extension
3645 provides the client device's current location to the WebAuthn Relying
3646 Party.
3647
3648 Extension identifier
3649 loc
3650
3651 Client extension input
3652 The Boolean value true to indicate that this extension is
3653 requested by the Relying Party.
3654
3655 Client extension processing
3656 None, except creating the authenticator extension input from the
3657 client extension input.
3658
3659 Client extension output
3660 Returns a JSON object that encodes the location information in
3661 the authenticator extension output as a Coordinates value, as
3662 defined by The W3C Geolocation API Specification.
3663
3664 Authenticator extension input
3665 The Boolean value true, encoded in CBOR (major type 7, value
3666 21).
3667
3668 Authenticator extension processing
3669 If the authenticator does not support the extension, then the
3670 authenticator MUST ignore the extension request. If the
3671 authenticator accepts the extension, then the authenticator
3672 SHOULD only add this extension data to a packed attestation or
3673 assertion.
3674
3675 Authenticator extension output
3676 If the authenticator accepts the extension request, then
3677 authenticator extension output SHOULD provide location data in
3678 the form of a CBOR-encoded map, with the first value being the
3679 extension identifier and the second being an array of returned
3680 values. The array elements SHOULD be derived from (key,value)
3681 pairings for each location attribute that the authenticator
3682 supports. The following is an example of authenticator data
3683 where the returned array is comprised of a {longitude, latitude,
3684 altitude} triplet, following the coordinate representation
3685 defined in The W3C Geolocation API Specification.
3686
3687 ... -- [=RP ID=] hash (32 bytes)

```

```

3691 OSLevelUserID | FactoryResetCounter.
3692
3693 Servers supporting UVI extensions MUST support a length of up to
3694 32 bytes for the UVI value.
3695
3696 Example for authenticator data containing one UVI extension
3697
3698 ... -- [=RP ID=] hash (32 bytes)
3699 81 -- UP and ED set
3700 00 00 00 01 -- (initial) signature counter
3701 -- all public key alg etc.
3702 A1 -- extension: CBOR map of one elemen
3703 t
3704 63 -- Key 1: CBOR text string of 3 byte
3705 s
3706 75 76 69 -- "uvi" [=UTF-8 encoded=] string
3707 58 20 -- Value 1: CBOR byte string with 0x
3708 20 bytes
3709 00 43 B8 E3 BE 27 95 8C -- the UVI value itself
3710 28 D5 74 BF 46 8A 85 CF
3711 46 9A 14 F0 E5 16 69 31
3712 DA 4B CF FF C1 BB 11 32
3713 82
3714
3715 9.7. Location Extension (loc)
3716
3717 The location registration extension and authentication extension
3718 provides the client device's current location to the WebAuthn Relying
3719 Party.
3720
3721 Extension identifier
3722 loc
3723
3724 Client extension input
3725 The Boolean value true to indicate that this extension is
3726 requested by the Relying Party.
3727
3728 Client extension processing
3729 None, except creating the authenticator extension input from the
3730 client extension input.
3731
3732 Client extension output
3733 Returns a JSON object that encodes the location information in
3734 the authenticator extension output as a Coordinates value, as
3735 defined by The W3C Geolocation API Specification.
3736
3737 Authenticator extension input
3738 The Boolean value true, encoded in CBOR (major type 7, value
3739 21).
3740
3741 Authenticator extension processing
3742 If the authenticator does not support the extension, then the
3743 authenticator MUST ignore the extension request. If the
3744 authenticator accepts the extension, then the authenticator
3745 SHOULD only add this extension data to a packed attestation or
3746 assertion.
3747
3748 Authenticator extension output
3749 If the authenticator accepts the extension request, then
3750 authenticator extension output SHOULD provide location data in
3751 the form of a CBOR-encoded map, with the first value being the
3752 extension identifier and the second being an array of returned
3753 values. The array elements SHOULD be derived from (key,value)
3754 pairings for each location attribute that the authenticator
3755 supports. The following is an example of authenticator data
3756 where the returned array is comprised of a {longitude, latitude,
3757 altitude} triplet, following the coordinate representation
3758 defined in The W3C Geolocation API Specification.
3759
3760 ... -- [=RP ID=] hash (32 bytes)

```

```

368E 81 -- UP and ED set
368F 00 00 00 01 -- (initial) signature counter
3690 ... -- all public key alg etc.
3691 A1 -- extension: CBOR map of one elemen
3692 t
3693 63 -- Value 1: CBOR text string of 3 by
3694 tes
3695 6C 6F 63 -- "loc" [=UTF-8 encoded=] string
3696 86 -- Value 2: array of 6 elements
3697 68 -- Element 1: CBOR text string of 8 bytes
3698 6C 61 74 69 74 75 64 65 -- "latitude" [=UTF-8 encoded=] stri
3699 ng
3700 FB ... -- Element 2: Latitude as CBOR encoded double-p
3701 recision float
3702 69 -- Element 3: CBOR text string of 9 bytes
3703 6C 6F 6E 67 69 74 75 64 65 -- "longitude" [=UTF-8 encoded=] str
3704 ing
3705 FB ... -- Element 4: Longitude as CBOR encoded double-
3706 precision float
3707 68 -- Element 5: CBOR text string of 8 bytes
3708 61 6C 74 69 74 75 64 65 -- "altitude" [=UTF-8 encoded=] stri
3709 ng
3710 FB ... -- Element 6: Altitude as CBOR encoded double-p
3711 recision float
3712
3713 9.8. User Verification Method Extension (uvm)
3714
3715 This registration extension and authentication extension enables use of
3716 a user verification method.
3717
3718 Extension identifier
3719 uvm
3720
3721 Client extension input
3722 The Boolean value true to indicate that this extension is
3723 requested by the WebAuthn Relying Party.
3724
3725 Client extension processing
3726 None, except creating the authenticator extension input from the
3727 client extension input.
3728
3729 Client extension output
3730 Returns a JSON array of 3-element arrays of numbers that encodes
3731 the factors in the authenticator extension output
3732
3733 Authenticator extension input
3734 The Boolean value true, encoded in CBOR (major type 7, value
3735 21).
3736
3737 Authenticator extension processing
3738 The authenticator sets the authenticator extension output to be
3739 a user verification index indicating the method used by the user
3740 to authorize the operation, as defined below. This extension can
3741 be added to attestation objects and assertions.
3742
3743 Authenticator extension output
3744 Authenticators can report up to 3 different user verification
3745 methods (factors) used in a single authentication instance,
3746 using the CBOR syntax defined below:
3747
3748 uvmFormat = [ 1*3 uvmEntry ]
3749 uvmEntry = [
3750 userVerificationMethod: uint .size 4,
3751 keyProtectionType: uint .size 2,
3752 matcherProtectionType: uint .size 2
3753 ]
3754
3755 The semantics of the fields in each uvmEntry are as follows:
3756
3757 userVerificationMethod

```

```

3761 81 -- UP and ED set
3762 00 00 00 01 -- (initial) signature counter
3763 ... -- all public key alg etc.
3764 A1 -- extension: CBOR map of one elemen
3765 t
3766 63 -- Value 1: CBOR text string of 3 by
3767 tes
3768 6C 6F 63 -- "loc" [=UTF-8 encoded=] string
3769 86 -- Value 2: array of 6 elements
3770 68 -- Element 1: CBOR text string of 8 bytes
3771 6C 61 74 69 74 75 64 65 -- "latitude" [=UTF-8 encoded=] stri
3772 ng
3773 FB ... -- Element 2: Latitude as CBOR encoded double-p
3774 recision float
3775 69 -- Element 3: CBOR text string of 9 bytes
3776 6C 6F 6E 67 69 74 75 64 65 -- "longitude" [=UTF-8 encoded=] str
3777 ing
3778 FB ... -- Element 4: Longitude as CBOR encoded double-
3779 precision float
3780 68 -- Element 5: CBOR text string of 8 bytes
3781 61 6C 74 69 74 75 64 65 -- "altitude" [=UTF-8 encoded=] stri
3782 ng
3783 FB ... -- Element 6: Altitude as CBOR encoded double-p
3784 recision float
3785
3786 9.8. User Verification Method Extension (uvm)
3787
3788 This registration extension and authentication extension enables use of
3789 a user verification method.
3790
3791 Extension identifier
3792 uvm
3793
3794 Client extension input
3795 The Boolean value true to indicate that this extension is
3796 requested by the WebAuthn Relying Party.
3797
3798 Client extension processing
3799 None, except creating the authenticator extension input from the
3800 client extension input.
3801
3802 Client extension output
3803 Returns a JSON array of 3-element arrays of numbers that encodes
3804 the factors in the authenticator extension output
3805
3806 Authenticator extension input
3807 The Boolean value true, encoded in CBOR (major type 7, value
3808 21).
3809
3810 Authenticator extension processing
3811 The authenticator sets the authenticator extension output to be
3812 a user verification index indicating the method used by the user
3813 to authorize the operation, as defined below. This extension can
3814 be added to attestation objects and assertions.
3815
3816 Authenticator extension output
3817 Authenticators can report up to 3 different user verification
3818 methods (factors) used in a single authentication instance,
3819 using the CBOR syntax defined below:
3820
3821 uvmFormat = [ 1*3 uvmEntry ]
3822 uvmEntry = [
3823 userVerificationMethod: uint .size 4,
3824 keyProtectionType: uint .size 2,
3825 matcherProtectionType: uint .size 2
3826 ]
3827
3828 The semantics of the fields in each uvmEntry are as follows:
3829
3830 userVerificationMethod

```

375E The authentication method/factor used by the authenticator
375F to verify the user. Available values are defined in
376C [FIDOReg], "User Verification Methods" section.

3761
3762 keyProtectionType
3763 The method used by the authenticator to protect the FIDO
3764 registration private key material. Available values are
3765 defined in [FIDOReg], "Key Protection Types" section.

3766
3767 matcherProtectionType
3768 The method used by the authenticator to protect the
3769 matcher that performs user verification. Available values
3770 are defined in [FIDOReg], "Matcher Protection Types"
3771 section.

3772
3773 If >3 factors can be used in an authentication instance the
3774 authenticator vendor must select the 3 factors it believes will
3775 be most relevant to the Server to include in the UVM.

3776
3777 Example for authenticator data containing one UVM extension for
3778 a multi-factor authentication instance where 2 factors were
3779 used:

```

3780 ... -- [=RP ID=] hash (32 bytes)
3781 81 -- UP and ED set
3782 00 00 00 01 -- (initial) signature counter
3783 ... -- all public key alg etc.
3784 A1 -- extension: CBOR map of one element
3785 63 -- Key 1: CBOR text string of 3 bytes
3786 75 76 6d -- "uvm" [=UTF-8 encoded=] string
3787 82 -- Value 1: CBOR array of length 2 indicating two factor
3788 usage
3789 83 -- Item 1: CBOR array of length 3
3790 02 -- Subitem 1: CBOR integer for User Verification Method
3791 Fingerprint
3792 04 -- Subitem 2: CBOR short for Key Protection Type TEE
3793 02 -- Subitem 3: CBOR short for Matcher Protection Type TE
3794 E
3795 83 -- Item 2: CBOR array of length 3
3796 04 -- Subitem 1: CBOR integer for User Verification Method
3797 Passcode
3798 01 -- Subitem 2: CBOR short for Key Protection Type Softwa
3799 re
3800 01 -- Subitem 3: CBOR short for Matcher Protection Type So
3801 ftware

```

3802
3803 10. IANA Considerations

3804 10.1. WebAuthn Attestation Statement Format Identifier Registrations

3805
3806 This section registers the attestation statement formats defined in
3807 Section 7 Defined Attestation Statement Formats in the IANA "WebAuthn
3808 Attestation Statement Format Identifier" registry established by
3809 [WebAuthn-Registries].

3810 * WebAuthn Attestation Statement Format Identifier: packed
3811 * Description: The "packed" attestation statement format is a
3812 WebAuthn-optimized format for attestation data. It uses a very
3813 compact but still extensible encoding method. This format is
3814 implementable by authenticators with limited resources (e.g.,
3815 secure elements).

3816 * Specification Document: Section 7.2 Packed Attestation Statement
3817 Format of this specification

3818 * WebAuthn Attestation Statement Format Identifier: tpm
3819 * Description: The TPM attestation statement format returns an
3820 attestation statement in the same format as the packed attestation
3821 statement format, although the the rawData and signature fields are
3822 computed differently.

3823 * Specification Document: Section 7.3 TPM Attestation Statement
3824 Format of this specification

3825 * WebAuthn Attestation Statement Format Identifier: android-key

3831 The authentication method/factor used by the authenticator
3832 to verify the user. Available values are defined in
3833 [FIDOReg], "User Verification Methods" section.

3834
3835 keyProtectionType
3836 The method used by the authenticator to protect the FIDO
3837 registration private key material. Available values are
3838 defined in [FIDOReg], "Key Protection Types" section.

3839
3840 matcherProtectionType
3841 The method used by the authenticator to protect the
3842 matcher that performs user verification. Available values
3843 are defined in [FIDOReg], "Matcher Protection Types"
3844 section.

3845
3846 If >3 factors can be used in an authentication instance the
3847 authenticator vendor must select the 3 factors it believes will
3848 be most relevant to the Server to include in the UVM.

3849
3850 Example for authenticator data containing one UVM extension for
3851 a multi-factor authentication instance where 2 factors were
3852 used:

```

3853 ... -- [=RP ID=] hash (32 bytes)
3854 81 -- UP and ED set
3855 00 00 00 01 -- (initial) signature counter
3856 ... -- all public key alg etc.
3857 A1 -- extension: CBOR map of one element
3858 63 -- Key 1: CBOR text string of 3 bytes
3859 75 76 6d -- "uvm" [=UTF-8 encoded=] string
3860 82 -- Value 1: CBOR array of length 2 indicating two factor
3861 usage
3862 83 -- Item 1: CBOR array of length 3
3863 02 -- Subitem 1: CBOR integer for User Verification Method
3864 Fingerprint
3865 04 -- Subitem 2: CBOR short for Key Protection Type TEE
3866 02 -- Subitem 3: CBOR short for Matcher Protection Type TE
3867 E
3868 83 -- Item 2: CBOR array of length 3
3869 04 -- Subitem 1: CBOR integer for User Verification Method
3870 Passcode
3871 01 -- Subitem 2: CBOR short for Key Protection Type Softwa
3872 re
3873 01 -- Subitem 3: CBOR short for Matcher Protection Type So
3874 ftware

```

3875
3876 10. IANA Considerations

3877 10.1. WebAuthn Attestation Statement Format Identifier Registrations

3878
3879 This section registers the attestation statement formats defined in
3880 Section 7 Defined Attestation Statement Formats in the IANA "WebAuthn
3881 Attestation Statement Format Identifier" registry established by
3882 [WebAuthn-Registries].

3883 * WebAuthn Attestation Statement Format Identifier: packed
3884 * Description: The "packed" attestation statement format is a
3885 WebAuthn-optimized format for attestation data. It uses a very
3886 compact but still extensible encoding method. This format is
3887 implementable by authenticators with limited resources (e.g.,
3888 secure elements).

3889 * Specification Document: Section 7.2 Packed Attestation Statement
3890 Format of this specification

3891 * WebAuthn Attestation Statement Format Identifier: tpm
3892 * Description: The TPM attestation statement format returns an
3893 attestation statement in the same format as the packed attestation
3894 statement format, although the the rawData and signature fields are
3895 computed differently.

3896 * Specification Document: Section 7.3 TPM Attestation Statement
3897 Format of this specification

3898 * WebAuthn Attestation Statement Format Identifier: android-key

3828 * Description: Platform-provided authenticators based on versions
 3829 "N", and later, may provide this proprietary "hardware attestation"
 3830 statement.
 3831 * Specification Document: Section 7.4 Android Key Attestation
 3832 Statement Format of this specification
 3833 * WebAuthn Attestation Statement Format Identifier: android-safetynet
 3834 * Description: Android-based, platform-provided authenticators may
 3835 produce an attestation statement based on the Android SafetyNet
 3836 API.
 3837 * Specification Document: Section 7.5 Android SafetyNet Attestation
 3838 Statement Format of this specification
 3839 * WebAuthn Attestation Statement Format Identifier: fido-u2f
 3840 * Description: Used with FIDO U2F authenticators
 3841 * Specification Document: Section 7.6 FIDO U2F Attestation Statement
 3842 Format of this specification

10.2. WebAuthn Extension Identifier Registrations

3843
 3844 This section registers the extension identifier values defined in
 3845 Section 8 WebAuthn Extensions in the IANA "WebAuthn Extension
 3846 Identifier" registry established by [WebAuthn-Registries].
 3847 * WebAuthn Extension Identifier: appid
 3848 * Description: This authentication extension allows Relying Parties
 3849 that have previously registered a credential using the legacy FIDO
 3850 JavaScript APIs to request an assertion.
 3851 * Specification Document: Section 9.1 FIDO AppId Extension (appid)
 3852 of this specification
 3853 * WebAuthn Extension Identifier: txAuthSimple
 3854 * Description: This registration extension and authentication
 3855 extension allows for a simple form of transaction authorization. A
 3856 WebAuthn Relying Party can specify a prompt string, intended for
 3857 display on a trusted device on the authenticator
 3858 * Specification Document: Section 9.2 Simple Transaction
 3859 Authorization Extension (txAuthSimple) of this specification
 3860 * WebAuthn Extension Identifier: txAuthGeneric
 3861 * Description: This registration extension and authentication
 3862 extension allows images to be used as transaction authorization
 3863 prompts as well. This allows authenticators without a font
 3864 rendering engine to be used and also supports a richer visual
 3865 appearance than accomplished with the webauthn.txauth.simple
 3866 extension.
 3867 * Specification Document: Section 9.3 Generic Transaction
 3868 Authorization Extension (txAuthGeneric) of this specification
 3869 * WebAuthn Extension Identifier: authnSel
 3870 * Description: This registration extension allows a WebAuthn Relying
 3871 Party to guide the selection of the authenticator that will be
 3872 leveraged when creating the credential. It is intended primarily
 3873 for WebAuthn Relying Parties that wish to tightly control the
 3874 experience around credential creation.
 3875 * Specification Document: Section 9.4 Authenticator Selection
 3876 Extension (authnSel) of this specification
 3877 * WebAuthn Extension Identifier: exts
 3878 * Description: This registration extension enables the Relying Party
 3879 to determine which extensions the authenticator supports. The
 3880 extension data is a list (CBOR array) of extension identifiers
 3881 encoded as UTF-8 Strings. This extension is added automatically by
 3882 the authenticator. This extension can be added to attestation
 3883 statements.
 3884 * Specification Document: Section 9.5 Supported Extensions Extension
 3885 (exts) of this specification
 3886 * WebAuthn Extension Identifier: uvi
 3887 * Description: This registration extension and authentication
 3888 extension enables use of a user verification index. The user
 3889 verification index is a value uniquely identifying a user
 3890 verification data record. The UVI data can be used by servers to
 3891 understand whether an authentication was authorized by the exact
 3892 same biometric data as the initial key generation. This allows the
 3893 detection and prevention of "friendly fraud".
 3894 * Specification Document: Section 9.6 User Verification Index
 3895 Extension (uvi) of this specification
 3896
 3897

3901 * Description: Platform-provided authenticators based on versions
 3902 "N", and later, may provide this proprietary "hardware attestation"
 3903 statement.
 3904 * Specification Document: Section 7.4 Android Key Attestation
 3905 Statement Format of this specification
 3906 * WebAuthn Attestation Statement Format Identifier: android-safetynet
 3907 * Description: Android-based, platform-provided authenticators may
 3908 produce an attestation statement based on the Android SafetyNet
 3909 API.
 3910 * Specification Document: Section 7.5 Android SafetyNet Attestation
 3911 Statement Format of this specification
 3912 * WebAuthn Attestation Statement Format Identifier: fido-u2f
 3913 * Description: Used with FIDO U2F authenticators
 3914 * Specification Document: Section 7.6 FIDO U2F Attestation Statement
 3915 Format of this specification

10.2. WebAuthn Extension Identifier Registrations

3916
 3917 This section registers the extension identifier values defined in
 3918 Section 8 WebAuthn Extensions in the IANA "WebAuthn Extension
 3919 Identifier" registry established by [WebAuthn-Registries].
 3920 * WebAuthn Extension Identifier: appid
 3921 * Description: This authentication extension allows Relying Parties
 3922 that have previously registered a credential using the legacy FIDO
 3923 JavaScript APIs to request an assertion.
 3924 * Specification Document: Section 9.1 FIDO AppId Extension (appid)
 3925 of this specification
 3926 * WebAuthn Extension Identifier: txAuthSimple
 3927 * Description: This registration extension and authentication
 3928 extension allows for a simple form of transaction authorization. A
 3929 WebAuthn Relying Party can specify a prompt string, intended for
 3930 display on a trusted device on the authenticator
 3931 * Specification Document: Section 9.2 Simple Transaction
 3932 Authorization Extension (txAuthSimple) of this specification
 3933 * WebAuthn Extension Identifier: txAuthGeneric
 3934 * Description: This registration extension and authentication
 3935 extension allows images to be used as transaction authorization
 3936 prompts as well. This allows authenticators without a font
 3937 rendering engine to be used and also supports a richer visual
 3938 appearance than accomplished with the webauthn.txauth.simple
 3939 extension.
 3940 * Specification Document: Section 9.3 Generic Transaction
 3941 Authorization Extension (txAuthGeneric) of this specification
 3942 * WebAuthn Extension Identifier: authnSel
 3943 * Description: This registration extension allows a WebAuthn Relying
 3944 Party to guide the selection of the authenticator that will be
 3945 leveraged when creating the credential. It is intended primarily
 3946 for WebAuthn Relying Parties that wish to tightly control the
 3947 experience around credential creation.
 3948 * Specification Document: Section 9.4 Authenticator Selection
 3949 Extension (authnSel) of this specification
 3950 * WebAuthn Extension Identifier: exts
 3951 * Description: This registration extension enables the Relying Party
 3952 to determine which extensions the authenticator supports. The
 3953 extension data is a list (CBOR array) of extension identifiers
 3954 encoded as UTF-8 Strings. This extension is added automatically by
 3955 the authenticator. This extension can be added to attestation
 3956 statements.
 3957 * Specification Document: Section 9.5 Supported Extensions Extension
 3958 (exts) of this specification
 3959 * WebAuthn Extension Identifier: uvi
 3960 * Description: This registration extension and authentication
 3961 extension enables use of a user verification index. The user
 3962 verification index is a value uniquely identifying a user
 3963 verification data record. The UVI data can be used by servers to
 3964 understand whether an authentication was authorized by the exact
 3965 same biometric data as the initial key generation. This allows the
 3966 detection and prevention of "friendly fraud".
 3967 * Specification Document: Section 9.6 User Verification Index
 3968 Extension (uvi) of this specification
 3969
 3970

389E * WebAuthn Extension Identifier: loc
 389F * Description: The location registration extension and authentication
 3900 extension provides the client device's current location to the
 3901 WebAuthn relying party, if supported by the client device and
 3902 subject to user consent.
 3903 * Specification Document: Section 9.7 Location Extension (loc) of
 3904 this specification
 3905 * WebAuthn Extension Identifier: uvm
 3906 * Description: This registration extension and authentication
 3907 extension enables use of a user verification method. The user
 3908 verification method extension returns to the Webauthn relying party
 3909 which user verification methods (factors) were used for the
 3910 WebAuthn operation.
 3911 * Specification Document: Section 9.8 User Verification Method
 3912 Extension (uvm) of this specification

10.3. COSE Algorithm Registrations

3914 This section registers identifiers for RSASSA-PKCS1-v1_5 [RFC8017]
 3915 algorithms using SHA-2 hash functions in the IANA COSE Algorithms
 3916 registry [IANA-COSE-ALGS-REG].

- 3917 * Name: RS256
- 3918 * Value: -257
- 3919 * Description: RSASSA-PKCS1-v1_5 w/ SHA-256
- 3920 * Reference: Section 8.2 of [RFC8017]
- 3921 * Recommended: No
- 3922 * Name: RS384
- 3923 * Value: -258
- 3924 * Description: RSASSA-PKCS1-v1_5 w/ SHA-384
- 3925 * Reference: Section 8.2 of [RFC8017]
- 3926 * Recommended: No
- 3927 * Name: RS512
- 3928 * Value: -259
- 3929 * Description: RSASSA-PKCS1-v1_5 w/ SHA-512
- 3930 * Reference: Section 8.2 of [RFC8017]
- 3931 * Recommended: No
- 3932 * Name: ED256
- 3933 * Value: -260
- 3934 * Description: TPM_ECC_BN_P256 curve w/ SHA-256
- 3935 * Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
- 3936 * Recommended: Yes
- 3937 * Name: ED512
- 3938 * Value: -261
- 3939 * Description: ECC_BN_ISOP512 curve w/ SHA-512
- 3940 * Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
- 3941 * Recommended: Yes

11. Sample scenarios

3944 This section is not normative.

3945 In this section, we walk through some events in the lifecycle of a
 3946 public key credential, along with the corresponding sample code for
 3947 using this API. Note that this is an example flow, and does not limit
 3948 the scope of how the API can be used.

3949 As was the case in earlier sections, this flow focuses on a use case
 3950 involving an external first-factor authenticator with its own display.
 3951 One example of such an authenticator would be a smart phone. Other
 3952 authenticator types are also supported by this API, subject to
 3953 implementation by the platform. For instance, this flow also works
 3954 without modification for the case of an authenticator that is embedded
 3955 in the client platform. The flow also works for the case of an
 3956 authenticator without its own display (similar to a smart card) subject
 3957 to specific implementation considerations. Specifically, the client
 3958 platform needs to display any prompts that would otherwise be shown by
 3959 the authenticator, and the authenticator needs to allow the client
 3960 platform to enumerate all the authenticator's credentials so that the
 3961 client can have information to show appropriate prompts.

3962

3971 * WebAuthn Extension Identifier: loc
 3972 * Description: The location registration extension and authentication
 3973 extension provides the client device's current location to the
 3974 WebAuthn relying party, if supported by the client device and
 3975 subject to user consent.
 3976 * Specification Document: Section 9.7 Location Extension (loc) of
 3977 this specification
 3978 * WebAuthn Extension Identifier: uvm
 3979 * Description: This registration extension and authentication
 3980 extension enables use of a user verification method. The user
 3981 verification method extension returns to the Webauthn relying party
 3982 which user verification methods (factors) were used for the
 3983 WebAuthn operation.
 3984 * Specification Document: Section 9.8 User Verification Method
 3985 Extension (uvm) of this specification

10.3. COSE Algorithm Registrations

3987 This section registers identifiers for RSASSA-PKCS1-v1_5 [RFC8017]
 3988 algorithms using SHA-2 hash functions in the IANA COSE Algorithms
 3989 registry [IANA-COSE-ALGS-REG].

- 3990 * Name: RS256
- 3991 * Value: -257
- 3992 * Description: RSASSA-PKCS1-v1_5 w/ SHA-256
- 3993 * Reference: Section 8.2 of [RFC8017]
- 3994 * Recommended: No
- 3995 * Name: RS384
- 3996 * Value: -258
- 3997 * Description: RSASSA-PKCS1-v1_5 w/ SHA-384
- 3998 * Reference: Section 8.2 of [RFC8017]
- 3999 * Recommended: No
- 4000 * Name: RS512
- 4001 * Value: -259
- 4002 * Description: RSASSA-PKCS1-v1_5 w/ SHA-512
- 4003 * Reference: Section 8.2 of [RFC8017]
- 4004 * Recommended: No
- 4005 * Name: ED256
- 4006 * Value: -260
- 4007 * Description: TPM_ECC_BN_P256 curve w/ SHA-256
- 4008 * Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
- 4009 * Recommended: Yes
- 4010 * Name: ED512
- 4011 * Value: -261
- 4012 * Description: ECC_BN_ISOP512 curve w/ SHA-512
- 4013 * Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
- 4014 * Recommended: Yes

11. Sample scenarios

4015 This section is not normative.

4016 In this section, we walk through some events in the lifecycle of a
 4017 public key credential, along with the corresponding sample code for
 4018 using this API. Note that this is an example flow, and does not limit
 4019 the scope of how the API can be used.

4020 As was the case in earlier sections, this flow focuses on a use case
 4021 involving an external first-factor authenticator with its own display.
 4022 One example of such an authenticator would be a smart phone. Other
 4023 authenticator types are also supported by this API, subject to
 4024 implementation by the platform. For instance, this flow also works
 4025 without modification for the case of an authenticator that is embedded
 4026 in the client platform. The flow also works for the case of an
 4027 authenticator without its own display (similar to a smart card) subject
 4028 to specific implementation considerations. Specifically, the client
 4029 platform needs to display any prompts that would otherwise be shown by
 4030 the authenticator, and the authenticator needs to allow the client
 4031 platform to enumerate all the authenticator's credentials so that the
 4032 client can have information to show appropriate prompts.

404C

```

396E 11.1. Registration
396F
397C This is the first-time flow, in which a new credential is created and
397D registered with the server. In this flow, the Relying Party does not
397E have a preference for platform authenticator or roaming authenticators.
397F 1. The user visits example.com, which serves up a script. At this
397G point, the user may already be logged in using a legacy username
397H and password, or additional authenticator, or other means
397I acceptable to the Relying Party. Or the user may be in the process
397J of creating a new account.
397K 2. The Relying Party script runs the code snippet below.
397L 3. The client platform searches for and locates the authenticator.
397M 4. The client platform connects to the authenticator, performing any
397N pairing actions if necessary.
397O 5. The authenticator shows appropriate UI for the user to select the
397P authenticator on which the new credential will be created, and
397Q obtains a biometric or other authorization gesture from the user.
397R 6. The authenticator returns a response to the client platform, which
397S in turn returns a response to the Relying Party script. If the user
397T declined to select an authenticator or provide authorization, an
397U appropriate error is returned.
397V 7. If a new credential was created,
397W + The Relying Party script sends the newly generated credential
397X public key to the server, along with additional information
397Y such as attestation regarding the provenance and
397Z characteristics of the authenticator.
3980 + The server stores the credential public key in its database
3981 and associates it with the user as well as with the
3982 characteristics of authentication indicated by attestation,
3983 also storing a friendly name for later use.
3984 + The script may store data such as the credential ID in local
3985 storage, to improve future UX by narrowing the choice of
3986 credential for the user.
3987
3988 The sample code for generating and registering a new key follows:
3989 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
3990
3991 var publicKey = {
3992   challenge: Uint8Array.from(window.atob("PGifxAoBwCkWkm4b1Cill5otCphilh6Mijdbw
3993 FjomA="), c=>c.charCodeAt(0)),
3994
3995   // Relying Party:
3996   rp: {
3997     name: "Acme"
3998   },
3999
4000   // User:
4001   user: {
4002     id: "1098237235409872",
4003     name: "john.p.smith@example.com",
4004     displayName: "John P. Smith",
4005     icon: "https://pics.acme.com/00/p/aBjjjpqPb.png"
4006   },
4007
4008   // This Relying Party will accept either an ES256 or RS256 credential, but
4009   // prefers an ES256 credential.
4010   pubKeyCredParams: [
4011     {
4012       type: "public-key",
4013       alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
4014     },
4015     {
4016       type: "public-key",
4017       alg: -257 // Value registered by this specification for "RS256"
4018     }
4019   ],
4020
4021   timeout: 60000, // 1 minute
4022   excludeCredentials: [], // No exclude list of PKCredDescriptors
4023   extensions: {"webauthn.location": true} // Include location information
4024
4025
4026
4027
4028
4029
4030
4031
4032
4033
4034
4035
4036
4037

```

```

4041 11.1. Registration
4042
4043 This is the first-time flow, in which a new credential is created and
4044 registered with the server. In this flow, the Relying Party does not
4045 have a preference for platform authenticator or roaming authenticators.
4046 1. The user visits example.com, which serves up a script. At this
4047 point, the user may already be logged in using a legacy username
4048 and password, or additional authenticator, or other means
4049 acceptable to the Relying Party. Or the user may be in the process
4050 of creating a new account.
4051 2. The Relying Party script runs the code snippet below.
4052 3. The client platform searches for and locates the authenticator.
4053 4. The client platform connects to the authenticator, performing any
4054 pairing actions if necessary.
4055 5. The authenticator shows appropriate UI for the user to select the
4056 authenticator on which the new credential will be created, and
4057 obtains a biometric or other authorization gesture from the user.
4058 6. The authenticator returns a response to the client platform, which
4059 in turn returns a response to the Relying Party script. If the user
4060 declined to select an authenticator or provide authorization, an
4061 appropriate error is returned.
4062 7. If a new credential was created,
4063 + The Relying Party script sends the newly generated credential
4064 public key to the server, along with additional information
4065 such as attestation regarding the provenance and
4066 characteristics of the authenticator.
4067 + The server stores the credential public key in its database
4068 and associates it with the user as well as with the
4069 characteristics of authentication indicated by attestation,
4070 also storing a friendly name for later use.
4071 + The script may store data such as the credential ID in local
4072 storage, to improve future UX by narrowing the choice of
4073 credential for the user.
4074
4075 The sample code for generating and registering a new key follows:
4076 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4077
4078 var publicKey = {
4079   challenge: Uint8Array.from(window.atob("PGifxAoBwCkWkm4b1Cill5otCphilh6Mijdbw
4080 FjomA="), c=>c.charCodeAt(0)),
4081
4082   // Relying Party:
4083   rp: {
4084     name: "Acme"
4085   },
4086
4087   // User:
4088   user: {
4089     id: "1098237235409872",
4090     name: "john.p.smith@example.com",
4091     displayName: "John P. Smith",
4092     icon: "https://pics.acme.com/00/p/aBjjjpqPb.png"
4093   },
4094
4095   // This Relying Party will accept either an ES256 or RS256 credential, but
4096   // prefers an ES256 credential.
4097   pubKeyCredParams: [
4098     {
4099       type: "public-key",
4100       alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
4101     },
4102     {
4103       type: "public-key",
4104       alg: -257 // Value registered by this specification for "RS256"
4105     }
4106   ],
4107
4108   timeout: 60000, // 1 minute
4109   excludeCredentials: [], // No exclude list of PKCredDescriptors
4110   extensions: {"webauthn.location": true} // Include location information
4111
4112
4113
4114
4115
4116
4117
4118
4119
4120
4121
4122
4123
4124
4125

```

```

403E // in attestation
403F };
404C
4041 // Note: The following call will cause the authenticator to display UI.
4042 navigator.credentials.create({ publicKey })
4043 .then(function (newCredentialInfo) {
4044 // Send new credential info to server for verification and registration.
4045 }).catch(function (err) {
4046 // No acceptable authenticator or user refused consent. Handle appropriately
4047
4048 });
4049
405C 11.2. Registration Specifically with Platform Authenticator
4051
4052 This is flow for when the Relying Party is specifically interested in
4053 creating a public key credential with a platform authenticator.
4054 1. The user visits example.com and clicks on the login button, which
4055 redirects the user to login.example.com.
4056 2. The user enters a username and password to log in. After successful
4057 login, the user is redirected back to example.com.
4058 3. The Relying Party script runs the code snippet below.
4059 4. The user agent asks the user whether they are willing to register
4060 with the Relying Party using an available platform authenticator.
4061 5. If the user is not willing, terminate this flow.
4062 6. The user is shown appropriate UI and guided in creating a
4063 credential using one of the available platform authenticators. Upon
4064 successful credential creation, the RP script conveys the new
4065 credential to the server.
4066 if (!PublicKeyCredential) { /* Platform not capable of the API. Handle error. */
4067 }
4068
4069 PublicKeyCredential.isPlatformAuthenticatorAvailable()
4070 .then(function (userIntent) {
4071
4072 // If the user has affirmed willingness to register with RP using an ava
4073 ilable platform authenticator
4074 if (userIntent) {
4075 var publicKeyOptions = { /* Public key credential creation options.
4076 */};
4077
4078 // Create and register credentials.
4079 return navigator.credentials.create({ "publicKey": publicKeyOptions
4080 });
4081 } else {
4082
4083 // Record that the user does not intend to use a platform authentica
4084 tor
4085 // and default the user to a password-based flow in the future.
4086 }
4087
4088 }).then(function (newCredentialInfo) {
4089 // Send new credential info to server for verification and registration.
4090 }).catch( function(err) {
4091 // Something went wrong. Handle appropriately.
4092 });
4093
4094 11.3. Authentication
4095
4096 This is the flow when a user with an already registered credential
4097 visits a website and wants to authenticate using the credential.
4098 1. The user visits example.com, which serves up a script.
4099 2. The script asks the client platform for an Authentication
4100 Assertion, providing as much information as possible to narrow the
4101 choice of acceptable credentials for the user. This may be obtained
4102 from the data that was stored locally after registration, or by
4103 other means such as prompting the user for a username.
4104 3. The Relying Party script runs one of the code snippets below.
4105 4. The client platform searches for and locates the authenticator.
4106 5. The client platform connects to the authenticator, performing any
4107 pairing actions if necessary.

```

```

4111 // in attestation
4112 };
4113
4114 // Note: The following call will cause the authenticator to display UI.
4115 navigator.credentials.create({ publicKey })
4116 .then(function (newCredentialInfo) {
4117 // Send new credential info to server for verification and registration.
4118 }).catch(function (err) {
4119 // No acceptable authenticator or user refused consent. Handle appropriately
4120
4121 });
4122
4123 11.2. Registration Specifically with Platform Authenticator
4124
4125 This is flow for when the Relying Party is specifically interested in
4126 creating a public key credential with a platform authenticator.
4127 1. The user visits example.com and clicks on the login button, which
4128 redirects the user to login.example.com.
4129 2. The user enters a username and password to log in. After successful
4130 login, the user is redirected back to example.com.
4131 3. The Relying Party script runs the code snippet below.
4132 4. The user agent asks the user whether they are willing to register
4133 with the Relying Party using an available platform authenticator.
4134 5. If the user is not willing, terminate this flow.
4135 6. The user is shown appropriate UI and guided in creating a
4136 credential using one of the available platform authenticators. Upon
4137 successful credential creation, the RP script conveys the new
4138 credential to the server.
4139 if (!PublicKeyCredential) { /* Platform not capable of the API. Handle error. */
4140 }
4141
4142 PublicKeyCredential.isPlatformAuthenticatorAvailable()
4143 .then(function (userIntent) {
4144
4145 // If the user has affirmed willingness to register with RP using an ava
4146 ilable platform authenticator
4147 if (userIntent) {
4148 var publicKeyOptions = { /* Public key credential creation options.
4149 */};
4150
4151 // Create and register credentials.
4152 return navigator.credentials.create({ "publicKey": publicKeyOptions
4153 });
4154 } else {
4155
4156 // Record that the user does not intend to use a platform authentica
4157 tor
4158 // and default the user to a password-based flow in the future.
4159 }
4160
4161 }).then(function (newCredentialInfo) {
4162 // Send new credential info to server for verification and registration.
4163 }).catch( function(err) {
4164 // Something went wrong. Handle appropriately.
4165 });
4166
4167 11.3. Authentication
4168
4169 This is the flow when a user with an already registered credential
4170 visits a website and wants to authenticate using the credential.
4171 1. The user visits example.com, which serves up a script.
4172 2. The script asks the client platform for an Authentication
4173 Assertion, providing as much information as possible to narrow the
4174 choice of acceptable credentials for the user. This may be obtained
4175 from the data that was stored locally after registration, or by
4176 other means such as prompting the user for a username.
4177 3. The Relying Party script runs one of the code snippets below.
4178 4. The client platform searches for and locates the authenticator.
4179 5. The client platform connects to the authenticator, performing any
4180 pairing actions if necessary.

```

410E 6. The authenticator presents the user with a notification that their
410F attention is required. On opening the notification, the user is
4110 shown a friendly selection menu of acceptable credentials using the
4111 account information provided when creating the credentials, along
4112 with some information on the origin that is requesting these keys.
4113 7. The authenticator obtains a biometric or other authorization
4114 gesture from the user.
4115 8. The authenticator returns a response to the client platform, which
4116 in turn returns a response to the Relying Party script. If the user
4117 declined to select a credential or provide an authorization, an
4118 appropriate error is returned.
4119 9. If an assertion was successfully generated and returned,
4120 + The script sends the assertion to the server.
4121 + The server examines the assertion, extracts the credential ID,
4122 looks up the registered credential public key it is database,
4123 and verifies the assertion's authentication signature. If
4124 valid, it looks up the identity associated with the
4125 assertion's credential ID; that identity is now authenticated.
4126 If the credential ID is not recognized by the server (e.g., it
4127 has been deregistered due to inactivity) then the
4128 authentication has failed; each Relying Party will handle this
4129 in its own way.
4130 + The server now does whatever it would otherwise do upon
4131 successful authentication -- return a success page, set
4132 authentication cookies, etc.
4133
4134 If the Relying Party script does not have any hints available (e.g.,
4135 from locally stored data) to help it narrow the list of credentials,
4136 then the sample code for performing such an authentication might look
4137 like this:
4138 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4139
4140 var options = {
4141 challenge: new TextEncoder().encode("climb a mountain"),
4142 timeout: 60000, // 1 minute
4143 allowCredentials: [{ type: "public-key" }]
4144 };
4145 navigator.credentials.get({ "publicKey": options })
4146 .then(function (assertion) {
4147 // Send assertion to server for verification
4148 }).catch(function (err) {
4149 // No acceptable credential or user refused consent. Handle appropriately.
4150 });
4151
4152 On the other hand, if the Relying Party script has some hints to help
4153 it narrow the list of credentials, then the sample code for performing
4154 such an authentication might look like the following. Note that this
4155 sample also demonstrates how to use the extension for transaction
4156 authorization.
4157 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4158
4159 var encoder = new TextEncoder();
4160 var acceptableCredential1 = {
4161 type: "public-key",
4162 id: encoder.encode("!!!!!!!hi there!!!!!!!\n")
4163 };
4164 var acceptableCredential2 = {
4165 type: "public-key",
4166 id: encoder.encode("roses are red, violets are blue\n")
4167 };
4168
4169 var options = {
4170 challenge: encoder.encode("climb a mountain"),
4171 timeout: 60000, // 1 minute
4172 allowCredentials: [acceptableCredential1, acceptableCredential2]
4173 ,
4174 extensions: { 'webauthn.txauth.simple':
4175 "Wave your hands in the air like you just don't care" }
4176 };
4177

4181 6. The authenticator presents the user with a notification that their
4182 attention is required. On opening the notification, the user is
4183 shown a friendly selection menu of acceptable credentials using the
4184 account information provided when creating the credentials, along
4185 with some information on the origin that is requesting these keys.
4186 7. The authenticator obtains a biometric or other authorization
4187 gesture from the user.
4188 8. The authenticator returns a response to the client platform, which
4189 in turn returns a response to the Relying Party script. If the user
4190 declined to select a credential or provide an authorization, an
4191 appropriate error is returned.
4192 9. If an assertion was successfully generated and returned,
4193 + The script sends the assertion to the server.
4194 + The server examines the assertion, extracts the credential ID,
4195 looks up the registered credential public key it is database,
4196 and verifies the assertion's authentication signature. If
4197 valid, it looks up the identity associated with the
4198 assertion's credential ID; that identity is now authenticated.
4199 If the credential ID is not recognized by the server (e.g., it
4200 has been deregistered due to inactivity) then the
4201 authentication has failed; each Relying Party will handle this
4202 in its own way.
4203 + The server now does whatever it would otherwise do upon
4204 successful authentication -- return a success page, set
4205 authentication cookies, etc.
4206
4207 If the Relying Party script does not have any hints available (e.g.,
4208 from locally stored data) to help it narrow the list of credentials,
4209 then the sample code for performing such an authentication might look
4210 like this:
4211 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4212
4213 var options = {
4214 challenge: new TextEncoder().encode("climb a mountain"),
4215 timeout: 60000, // 1 minute
4216 allowCredentials: [{ type: "public-key" }]
4217 };
4218 navigator.credentials.get({ "publicKey": options })
4219 .then(function (assertion) {
4220 // Send assertion to server for verification
4221 }).catch(function (err) {
4222 // No acceptable credential or user refused consent. Handle appropriately.
4223 });
4224
4225 On the other hand, if the Relying Party script has some hints to help
4226 it narrow the list of credentials, then the sample code for performing
4227 such an authentication might look like the following. Note that this
4228 sample also demonstrates how to use the extension for transaction
4229 authorization.
4230 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4231
4232 var encoder = new TextEncoder();
4233 var acceptableCredential1 = {
4234 type: "public-key",
4235 id: encoder.encode("!!!!!!!hi there!!!!!!!\n")
4236 };
4237 var acceptableCredential2 = {
4238 type: "public-key",
4239 id: encoder.encode("roses are red, violets are blue\n")
4240 };
4241
4242 var options = {
4243 challenge: encoder.encode("climb a mountain"),
4244 timeout: 60000, // 1 minute
4245 allowCredentials: [acceptableCredential1, acceptableCredential2]
4246 ,
4247 extensions: { 'webauthn.txauth.simple':
4248 "Wave your hands in the air like you just don't care" }
4249 };
4250

```

417E navigator.credentials.get({ "publicKey": options })
417F .then(function (assertion) {
4180 // Send assertion to server for verification
4181 }).catch(function (err) {
4182 // No acceptable credential or user refused consent. Handle appropriately.
4183 });

```

11.4. 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.

12. Acknowledgements

We thank the following for their contributions to, and thorough review of, this specification: Richard Barnes, Dominic Battr, Domenic Denicola, Rahul Ghosh, Brad Hill, Jing Jin, Angelo Liao, Anne van Kesteren, Ian Kilpatrick, Giridhar Mandyam, Axel Nennker, Kimberly Paulhamus, Adam Powers, Yaron Sheffer, Mike West, Jeffrey Yasskin, Boris Zbarsky.

Index

Terms defined by this specification

- * AAGUID, in 9.4
- * alg, in 4.3
- * allowCredentials, in 4.5
- * Assertion, in 3
- * assertion signature, in 5
- * attachment modality, in 4.4.5
- * Attestation, in 3
- * Attestation Certificate, in 3
- * Attestation data, in 5.3.1
- * attestation key pair, in 3
- * attestationObject, in 4.2.1
- * attestation object, in 5.3
- * attestation private key, in 3
- * attestation public key, in 3
- * attestation signature, in 5
- * attestation statement, in 5.3
- * attestation statement format, in 5.3

```

4251 navigator.credentials.get({ "publicKey": options })
4252 .then(function (assertion) {
4253 // Send assertion to server for verification
4254 }).catch(function (err) {
4255 // No acceptable credential or user refused consent. Handle appropriately.
4256 });

```

11.4. 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.

12. Acknowledgements

We thank the following for their contributions to, and thorough review of, this specification: Richard Barnes, Dominic Battr, Domenic Denicola, Rahul Ghosh, Brad Hill, Jing Jin, Angelo Liao, Anne van Kesteren, Ian Kilpatrick, Giridhar Mandyam, Axel Nennker, Kimberly Paulhamus, Adam Powers, Yaron Sheffer, Mike West, Jeffrey Yasskin, Boris Zbarsky.

Index

Terms defined by this specification

- * AAGUID, in 9.4
- * alg, in 4.3
- * allowCredentials, in 4.5
- * Assertion, in 3
- * assertion signature, in 5
- * attachment modality, in 4.4.5
- * Attestation, in 3
- * Attestation Certificate, in 3
- * Attestation data, in 5.3.1
- * attestation key pair, in 3
- * attestationObject, in 4.2.1
- * attestation object, in 5.3
- * attestationObjectResult, in 4.1.3
- * attestation private key, in 3
- * attestation public key, in 3
- * attestation signature, in 5
- * attestation statement, in 5.3
- * attestation statement format, in 5.3

4247 * attestation statement format identifier, in 7.1
 4248 * attestation type, in 5.3
 4249 * Authentication, in 3
 4250 * Authentication Assertion, in 3
 4251 * authentication extension, in 8
 4252 * AuthenticationExtensions
 4253 + definition of, in 4.6
 4254 + (typedef), in 4.6
 4255 * Authenticator, in 3
 4256 * AuthenticatorAssertionResponse, in 4.2.2
 4257 * AuthenticatorAttachment, in 4.4.5
 4258 * authenticatorAttachment, in 4.4.4
 4259 * AuthenticatorAttestationResponse, in 4.2.1
 4260 * authenticatorCancel, in 5.2.3
 4261 * authenticator data, in 5.1
 4262 * authenticatorData, in 4.2.2
 4263 * authenticator data claimed to have been used for the attestation,
 4264 in 5.3.2
 4265 * authenticator data for the attestation, in 5.3.2
 4266 * authenticator extension, in 8
 4267 * authenticator extension input, in 8.3
 4268 * authenticator extension output, in 8.5
 4269 * Authenticator extension processing, in 8.5
 4270 * authenticatorExtensions, in 4.7.1
 4271 * authenticatorGetAssertion, in 5.2.2
 4272 * authenticatorMakeCredential, in 5.2.1
 4273 * AuthenticatorResponse, in 4.2
 4274 * authenticatorSelection, in 4.4
 4275 * AuthenticatorSelectionCriteria, in 4.4.4
 4276 * AuthenticatorSelectionList, in 9.4
 4277 * AuthenticatorTransport, in 4.7.4
 4278 * Authorization Gesture, in 3
 4279 * Base64url Encoding, in 2.1
 4280 * Basic Attestation, in 5.3.3
 4281 * Biometric Recognition, in 3
 4282 * ble, in 4.7.4
 4283 * CBOR, in 2.1
 4284 * Ceremony, in 3
 4285 * challenge
 4286 + dict-member for MakePublicKeyCredentialOptions, in 4.4
 4287 + dict-member for PublicKeyCredentialRequestOptions, in 4.5
 4288 + dict-member for CollectedClientData, in 4.7.1
 4289 * Client, in 3
 4290 * client data, in 4.7.1
 4291 * clientDataJSON, in 4.2

 4292 * client extension, in 8
 4293 * client extension input, in 8.3
 4294 * client extension output, in 8.4
 4295 * Client extension processing, in 8.4
 4296 * clientExtensionResults, in 4.1
 4297 * clientExtensions, in 4.7.1
 4298 * Client-Side, in 3
 4299 * client-side credential private key storage, in 3
 4300 * Client-side-resident Credential Private Key, in 3
 4301 * CollectedClientData, in 4.7.1
 4302 * Conforming User Agent, in 3

 4303 * COSEAlgorithmIdentifier
 4304 + definition of, in 4.7.5
 4305 + (typedef), in 4.7.5
 4306 * [[Create]](options), in 4.1.3
 4307 * credential key pair, in 3
 4308 * credential private key, in 3
 4309 * Credential Public Key, in 3
 4310 * "cross-platform", in 4.4.5
 4311 * cross-platform, in 4.4.5
 4312 * cross-platform attached, in 4.4.5
 4313 * cross-platform attachment, in 4.4.5
 4314 * DAA, in 5.3.3

4321 * attestation statement format identifier, in 7.1
 4322 * attestation type, in 5.3
 4323 * Authentication, in 3
 4324 * Authentication Assertion, in 3
 4325 * authentication extension, in 8
 4326 * AuthenticationExtensions
 4327 + definition of, in 4.6
 4328 + (typedef), in 4.6
 4329 * Authenticator, in 3
 4330 * AuthenticatorAssertionResponse, in 4.2.2
 4331 * AuthenticatorAttachment, in 4.4.5
 4332 * authenticatorAttachment, in 4.4.4
 4333 * AuthenticatorAttestationResponse, in 4.2.1
 4334 * authenticatorCancel, in 5.2.3
 4335 * authenticator data, in 5.1
 4336 * authenticatorData, in 4.2.2
 4337 * authenticator data claimed to have been used for the attestation,
 4338 in 5.3.2
 4339 * authenticator data for the attestation, in 5.3.2
 4340 * authenticator extension, in 8
 4341 * authenticator extension input, in 8.3
 4342 * authenticator extension output, in 8.5
 4343 * Authenticator extension processing, in 8.5
 4344 * authenticatorExtensions, in 4.7.1
 4345 * authenticatorGetAssertion, in 5.2.2
 4346 * authenticatorMakeCredential, in 5.2.1
 4347 * AuthenticatorResponse, in 4.2
 4348 * authenticatorSelection, in 4.4
 4349 * AuthenticatorSelectionCriteria, in 4.4.4
 4350 * AuthenticatorSelectionList, in 9.4
 4351 * AuthenticatorTransport, in 4.7.4
 4352 * Authorization Gesture, in 3
 4353 * Base64url Encoding, in 2.1
 4354 * Basic Attestation, in 5.3.3
 4355 * Biometric Recognition, in 3
 4356 * ble, in 4.7.4
 4357 * CBOR, in 2.1
 4358 * Ceremony, in 3
 4359 * challenge
 4360 + dict-member for MakePublicKeyCredentialOptions, in 4.4
 4361 + dict-member for PublicKeyCredentialRequestOptions, in 4.5
 4362 + dict-member for CollectedClientData, in 4.7.1
 4363 * Client, in 3
 4364 * client data, in 4.7.1
 4365 * clientDataJSON, in 4.2
 4366 * **clientDataJSONResult, in 4.1.3**
 4367 * client extension, in 8
 4368 * client extension input, in 8.3
 4369 * client extension output, in 8.4
 4370 * Client extension processing, in 8.4
 4371 * clientExtensionResults, in 4.1
 4372 * clientExtensions, in 4.7.1
 4373 * Client-Side, in 3
 4374 * client-side credential private key storage, in 3
 4375 * Client-side-resident Credential Private Key, in 3
 4376 * CollectedClientData, in 4.7.1
 4377 * Conforming User Agent, in 3
 4378 * **constructCredentialCallback, in 4.1.3**
 4379 * COSEAlgorithmIdentifier
 4380 + definition of, in 4.7.5
 4381 + (typedef), in 4.7.5
 4382 * [[Create]](options), in 4.1.3
 4383 * credential key pair, in 3
 4384 * credential private key, in 3
 4385 * Credential Public Key, in 3
 4386 * "cross-platform", in 4.4.5
 4387 * cross-platform, in 4.4.5
 4388 * cross-platform attached, in 4.4.5
 4389 * cross-platform attachment, in 4.4.5
 4390 * DAA, in 5.3.3

4315 * [[DiscoverFromExternalSource]](options), in 4.1.4
 4316 * [[discovery]], in 4.1
 4317 * displayName, in 4.4.3
 4318 * ECDSA, in 5.3.3
 4319 * ECDSA-Issuer public key, in 7.2
 4320 * Elliptic Curve based Direct Anonymous Attestation, in 5.3.3
 4321 * excludeCredentials, in 4.4
 4322 * extension identifier, in 8.1
 4323 * extensions
 4324 + dict-member for MakePublicKeyCredentialOptions, in 4.4
 4325 + dict-member for PublicKeyCredentialRequestOptions, in 4.5
 4326 * hashAlgorithm, in 4.7.1
 4327 * Hash of the serialized client data, in 4.7.1
 4328 * icon, in 4.4.1
 4329 * id
 4330 + dict-member for PublicKeyCredentialRpEntity, in 4.4.2
 4331 + dict-member for PublicKeyCredentialUserEntity, in 4.4.3
 4332 + dict-member for PublicKeyCredentialDescriptor, in 4.7.3
 4333 * [[identifier]], in 4.1
 4334 * identifier of the ECDSA-Issuer public key, in 7.2
 4335 * isPlatformAuthenticatorAvailable(), in 4.1.5
 4336 * JSON-serialized client data, in 4.7.1
 4337 * MakePublicKeyCredentialOptions, in 4.4
 4338 * name, in 4.4.1
 4339 * nfc, in 4.7.4
 4340 * origin, in 4.7.1
 4341 * platform, in 4.4.5
 4342 * "platform", in 4.4.5
 4343 * platform attachment, in 4.4.5
 4344 * platform authenticators, in 4.4.5
 4345 * Privacy CA, in 5.3.3
 4346 * pubKeyCredParams, in 4.4
 4347 * publicKey
 4348 + dict-member for CredentialCreationOptions, in 4.1.1
 4349 + dict-member for CredentialRequestOptions, in 4.1.2
 4350 * public-key, in 4.7.2
 4351 * Public Key Credential, in 3
 4352 * PublicKeyCredential, in 4.1
 4353 * PublicKeyCredentialDescriptor, in 4.7.3
 4354 * PublicKeyCredentialEntity, in 4.4.1
 4355 * PublicKeyCredentialParameters, in 4.3
 4356 * PublicKeyCredentialRequestOptions, in 4.5
 4357 * PublicKeyCredentialRpEntity, in 4.4.2
 4358 * PublicKeyCredentialType, in 4.7.2
 4359 * PublicKeyCredentialUserEntity, in 4.4.3
 4360 * Rate Limiting, in 3
 4361 * rawId, in 4.1
 4362 * Registration, in 3
 4363 * registration extension, in 8
 4364 * Relying Party, in 3
 4365 * Relying Party Identifier, in 3
 4366 * requireResidentKey, in 4.4.4
 4367 * requireUserVerification, in 4.4.4
 4368 * response, in 4.1
 4369 * roaming authenticators, in 4.4.5
 4370 * rp, in 4.4
 4371 * rpId, in 4.5
 4372 * RP ID, in 3
 4373 * Self Attestation, in 5.3.3
 4374 * signature, in 4.2.2
 4375 * Signing procedure, in 5.3.2
 4376 * Test of User Presence, in 3
 4377 * timeout
 4378 + dict-member for MakePublicKeyCredentialOptions, in 4.4
 4379 + dict-member for PublicKeyCredentialRequestOptions, in 4.5
 4380 * tokenBindingId, in 4.7.1
 4381 * transports, in 4.7.3
 4382 * [[type]], in 4.1
 4383 * type

4391 * [[DiscoverFromExternalSource]](options), in 4.1.4
 4392 * [[discovery]], in 4.1
 4393 * displayName, in 4.4.3
 4394 * ECDSA, in 5.3.3
 4395 * ECDSA-Issuer public key, in 7.2
 4396 * Elliptic Curve based Direct Anonymous Attestation, in 5.3.3
 4397 * excludeCredentials, in 4.4
 4398 * extension identifier, in 8.1
 4399 * extensionOutputsMap, in 4.1.3
 4400 * extensions
 4401 + dict-member for MakePublicKeyCredentialOptions, in 4.4
 4402 + dict-member for PublicKeyCredentialRequestOptions, in 4.5
 4403 * hashAlgorithm, in 4.7.1
 4404 * Hash of the serialized client data, in 4.7.1
 4405 * icon, in 4.4.1
 4406 * id
 4407 + dict-member for PublicKeyCredentialRpEntity, in 4.4.2
 4408 + dict-member for PublicKeyCredentialUserEntity, in 4.4.3
 4409 + dict-member for PublicKeyCredentialDescriptor, in 4.7.3
 4410 * [[identifier]], in 4.1
 4411 * identifier of the ECDSA-Issuer public key, in 7.2
 4412 * isPlatformAuthenticatorAvailable(), in 4.1.5
 4413 * JSON-serialized client data, in 4.7.1
 4414 * MakePublicKeyCredentialOptions, in 4.4
 4415 * name, in 4.4.1
 4416 * nfc, in 4.7.4
 4417 * origin, in 4.7.1
 4418 * platform, in 4.4.5
 4419 * "platform", in 4.4.5
 4420 * platform attachment, in 4.4.5
 4421 * platform authenticators, in 4.4.5
 4422 * Privacy CA, in 5.3.3
 4423 * pubKeyCredParams, in 4.4
 4424 * publicKey
 4425 + dict-member for CredentialCreationOptions, in 4.1.1
 4426 + dict-member for CredentialRequestOptions, in 4.1.2
 4427 * public-key, in 4.7.2
 4428 * Public Key Credential, in 3
 4429 * PublicKeyCredential, in 4.1
 4430 * PublicKeyCredentialDescriptor, in 4.7.3
 4431 * PublicKeyCredentialEntity, in 4.4.1
 4432 * PublicKeyCredentialParameters, in 4.3
 4433 * PublicKeyCredentialRequestOptions, in 4.5
 4434 * PublicKeyCredentialRpEntity, in 4.4.2
 4435 * PublicKeyCredentialType, in 4.7.2
 4436 * PublicKeyCredentialUserEntity, in 4.4.3
 4437 * Rate Limiting, in 3
 4438 * rawId, in 4.1
 4439 * Registration, in 3
 4440 * registration extension, in 8
 4441 * Relying Party, in 3
 4442 * Relying Party Identifier, in 3
 4443 * requireResidentKey, in 4.4.4
 4444 * requireUserVerification, in 4.4.4
 4445 * response, in 4.1
 4446 * roaming authenticators, in 4.4.5
 4447 * rp, in 4.4
 4448 * rpId, in 4.5
 4449 * RP ID, in 3
 4450 * Self Attestation, in 5.3.3
 4451 * signature, in 4.2.2
 4452 * Signing procedure, in 5.3.2
 4453 * Test of User Presence, in 3
 4454 * timeout
 4455 + dict-member for MakePublicKeyCredentialOptions, in 4.4
 4456 + dict-member for PublicKeyCredentialRequestOptions, in 4.5
 4457 * tokenBindingId, in 4.7.1
 4458 * transports, in 4.7.3
 4459 * [[type]], in 4.1
 4460 * type

- 4384 + dict-member for PublicKeyCredentialParameters, in 4.3
- 4385 + dict-member for PublicKeyCredentialDescriptor, in 4.7.3
- 4386 * UP, in 3
- 4387 * usb, in 4.7.4
- 4388 * user, in 4.4
- 4389 * User Consent, in 3
- 4390 * User Present, in 3
- 4391 * User Verification, in 3
- 4392 * User Verified, in 3
- 4393 * UV, in 3
- 4394 * Verification procedures, in 5.3.2
- 4395 * Web Authentication API, in 4
- 4396 * WebAuthn Client, in 3

Terms defined by reference

* [CREDENTIAL-MANAGEMENT-1] defines the following terms:

- + Credential
- + CredentialCreationOptions
- + CredentialRequestOptions
- + CredentialsContainer
- + [[CollectFromCredentialStore]](options)
- + [[Store]](credential)
- + [[discovery]]
- + [[type]]
- + create()
- + get()
- + id
- + remote
- + type

* [ECMAScript] defines the following terms:

- + %arraybuffer%
- + internal slot
- + stringify

* [ENCODING] defines the following terms:

- + utf-8 encode

* [encoding-1] defines the following terms:

- + utf-8 encode

* [HTML] defines the following terms:

- + ascii serialization of an origin

- + dom manipulation task source
- + effective domain

- + global object
- + in parallel
- + is a registrable domain suffix of or is equal to
- + is not a registrable domain suffix of and is not equal to
- + origin
- + promise

- + relevant settings object
- + task
- + task source

* [HTML52] defines the following terms:

- + document.domain
- + opaque origin
- + origin

* [INFRA] defines the following terms:

- + append (for list)
- + append (for set)
- + continue

- + for each (for list)
- + for each (for map)
- + is empty
- + is not empty
- + item

- 4461 + dict-member for PublicKeyCredentialParameters, in 4.3
- 4462 + dict-member for PublicKeyCredentialDescriptor, in 4.7.3
- 4463 * UP, in 3
- 4464 * usb, in 4.7.4
- 4465 * user, in 4.4
- 4466 * User Consent, in 3
- 4467 * User Present, in 3
- 4468 * User Verification, in 3
- 4469 * User Verified, in 3
- 4470 * UV, in 3
- 4471 * Verification procedures, in 5.3.2
- 4472 * Web Authentication API, in 4
- 4473 * WebAuthn Client, in 3

Terms defined by reference

* [CREDENTIAL-MANAGEMENT-1] defines the following terms:

- + Credential
- + CredentialCreationOptions
- + CredentialRequestOptions
- + CredentialsContainer
- + [[CollectFromCredentialStore]](options)
- + [[Store]](credential)
- + [[discovery]]
- + [[type]]
- + create()
- + get()
- + id
- + remote
- + type

* [ECMAScript] defines the following terms:

- + %arraybuffer%
- + internal slot
- + stringify

* [ENCODING] defines the following terms:

- + utf-8 encode

* [HTML] defines the following terms:

- + ascii serialization of an origin
- + current settings object
- + dom manipulation task source
- + effective domain

- + event loop
- + global object
- + in parallel
- + is a registrable domain suffix of or is equal to
- + is not a registrable domain suffix of and is not equal to
- + origin
- + promise

- + relevant global object
- + relevant settings object
- + task
- + task source

* [HTML52] defines the following terms:

- + document.domain
- + opaque origin
- + origin

* [INFRA] defines the following terms:

- + append (for list)
- + append (for set)
- + continue

- + entry
- + for each (for list)
- + for each (for map)
- + is empty
- + is not empty
- + item (for list)
- + item (for struct)
- + key

```

444E + list
444F + map

445C + ordered set
4451 + remove
4452 + set

4453 * [mixed-content] defines the following terms:
4454 + a priori authenticated url
4455 * [secure-contexts] defines the following terms:
4456 + secure context
4457 * [TokenBinding] defines the following terms:
4458 + token binding
4459 + token binding id
4460 * [URL] defines the following terms:
4461 + domain
4462 + empty host
4463 + host
4464 + ipv4 address
4465 + ipv6 address
4466 + opaque host
4467 + url serializer
4468 + valid domain
4469 + valid domain string
4470 * [WebCryptoAPI] defines the following terms:
4471 + recognized algorithm name
4472 * [WebIDL] defines the following terms:
4473 + ArrayBuffer
4474 + BufferSource
4475 + ConstraintError
4476 + DOMException
4477 + DOMString

4478 + NotAllowedError
4479 + NotFoundError
4480 + NotSupportedError
4481 + Promise
4482 + SameObject
4483 + SecureContext
4484 + SecurityError
4485 + TypeError
4486 + USVString
4487 + UnknownError
4488 + Unscopable
4489 + boolean

449C + interface object
4491 + long
4492 + present

4493 + simple exception
4494 + unsigned long

449E References
449F Normative References
450C [CDDL]
4501 C. Vigano; H. Birkholz. CBOR data definition language (CDDL): a
4502 notational convention to express CBOR data structures. 21
4503 September 2016. Internet Draft (work in progress). URL:
4504 https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl
4505
4506 [CREDENTIAL-MANAGEMENT-1]
4507 Mike West. Credential Management Level 1. 4 August 2017. WD.
4508 URL: https://www.w3.org/TR/credential-management-1/
4509
451C [DOM4]

```

```

452E + list
453C + map
4531 + ordered map
4532 + ordered set
4533 + remove
4534 + set
4535 + struct
4536 + value
4537 + while

4538 * [mixed-content] defines the following terms:
4539 + a priori authenticated url
454C * [secure-contexts] defines the following terms:
4541 + secure context
4542 * [TokenBinding] defines the following terms:
4543 + token binding
4544 + token binding id
4545 * [URL] defines the following terms:
4546 + domain
4547 + empty host
4548 + host
4549 + ipv4 address
455C + ipv6 address
4551 + opaque host
4552 + url serializer
4553 + valid domain
4554 + valid domain string
4555 * [WebCryptoAPI] defines the following terms:
4556 + recognized algorithm name
4557 * [WebIDL] defines the following terms:
4558 + ArrayBuffer
4559 + BufferSource
456C + ConstraintError
4561 + DOMException
4562 + DOMString
4563 + Function
4564 + NotAllowedError
4565 + NotFoundError
4566 + NotSupportedError
4567 + Promise
4568 + SameObject
4569 + SecureContext
457C + SecurityError
4571 + TypeError
4572 + USVString
4573 + UnknownError
4574 + Unscopable
4575 + boolean
4576 + callback function types
4577 + interface object
4578 + long
4579 + present
458C + record type
4581 + simple exception
4582 + unsigned long

4584 References
4585 Normative References
4586 [CDDL]
4587 C. Vigano; H. Birkholz. CBOR data definition language (CDDL): a
4588 notational convention to express CBOR data structures. 21
459C September 2016. Internet Draft (work in progress). URL:
4591 https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl
4592
4593 [CREDENTIAL-MANAGEMENT-1]
4594 Mike West. Credential Management Level 1. 4 August 2017. WD.
4595 URL: https://www.w3.org/TR/credential-management-1/
4596
4597 [DOM4]

```

4511 Anne van Kesteren. DOM Standard. Living Standard. URL:
4512 <https://dom.spec.whatwg.org/>
4513
4514 [ECMAScript]
4515 ECMAScript Language Specification. URL:
4516 <https://tc39.github.io/ecma262/>
4517
4518 [ENCODING]
4519 Anne van Kesteren. Encoding Standard. Living Standard. URL:
4520 <https://encoding.spec.whatwg.org/>
4521
4522 [FIDOEcdaaAlgorithm]
4523 R. Lindemann; et al. FIDO ECDAA Algorithm. FIDO Alliance
4524 Implementation Draft. URL:
4525 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ecdaa-algorithm-v1.1-id-20170202.html>
4526
4527 [FIDOReg]
4528 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO UAF Registry of
4529 Predefined Values. FIDO Alliance Proposed Standard. URL:
4530 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-reg-v1.0-ps-20141208.html>
4531
4532 [HTML]
4533 Anne van Kesteren; et al. HTML Standard. Living Standard. URL:
4534 <https://html.spec.whatwg.org/multipage/>
4535
4536 [HTML52]
4537 Steve Faulkner; et al. HTML 5.2. 8 August 2017. CR. URL:
4538 <https://www.w3.org/TR/html52/>
4539
4540 [IANA-COSE-ALGS-REG]
4541 IANA CBOR Object Signing and Encryption (COSE) Algorithms
4542 Registry. URL:
4543 <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>
4544
4545 [INFRA]
4546 Anne van Kesteren; Domenic Denicola. Infra Standard. Living
4547 Standard. URL: <https://infra.spec.whatwg.org/>
4548
4549 [MIXED-CONTENT]
4550 Mike West. Mixed Content. 2 August 2016. CR. URL:
4551 <https://www.w3.org/TR/mixed-content/>
4552
4553 [RFC2119]
4554 S. Bradner. Key words for use in RFCs to Indicate Requirement
4555 Levels. March 1997. Best Current Practice. URL:
4556 <https://tools.ietf.org/html/rfc2119>
4557
4558 [RFC4648]
4559 S. Josefsson. The Base16, Base32, and Base64 Data Encodings.
4560 October 2006. Proposed Standard. URL:
4561 <https://tools.ietf.org/html/rfc4648>
4562
4563 [RFC5234]
4564 D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax
4565 Specifications: ABNF. January 2008. Internet Standard. URL:
4566 <https://tools.ietf.org/html/rfc5234>
4567
4568 [RFC5890]
4569 J. Klensin. Internationalized Domain Names for Applications
4570 (IDNA): Definitions and Document Framework. August 2010.
4571 Proposed Standard. URL: <https://tools.ietf.org/html/rfc5890>
4572
4573 [RFC7049]
4574 C. Bormann; P. Hoffman. Concise Binary Object Representation
4575 (CBOR). October 2013. Proposed Standard. URL:
4576 <https://tools.ietf.org/html/rfc7049>
4577
4578 [RFC8152]

4599 Anne van Kesteren. DOM Standard. Living Standard. URL:
4600 <https://dom.spec.whatwg.org/>
4601
4602 [ECMAScript]
4603 ECMAScript Language Specification. URL:
4604 <https://tc39.github.io/ecma262/>
4605
4606 [ENCODING]
4607 Anne van Kesteren. Encoding Standard. Living Standard. URL:
4608 <https://encoding.spec.whatwg.org/>
4609
4610 [FIDOEcdaaAlgorithm]
4611 R. Lindemann; et al. FIDO ECDAA Algorithm. FIDO Alliance
4612 Implementation Draft. URL:
4613 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ecdaa-algorithm-v1.1-id-20170202.html>
4614
4615 [FIDOReg]
4616 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO UAF Registry of
4617 Predefined Values. FIDO Alliance Proposed Standard. URL:
4618 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-reg-v1.0-ps-20141208.html>
4619
4620 [HTML]
4621 Anne van Kesteren; et al. HTML Standard. Living Standard. URL:
4622 <https://html.spec.whatwg.org/multipage/>
4623
4624 [HTML52]
4625 Steve Faulkner; et al. HTML 5.2. 8 August 2017. CR. URL:
4626 <https://www.w3.org/TR/html52/>
4627
4628 [IANA-COSE-ALGS-REG]
4629 IANA CBOR Object Signing and Encryption (COSE) Algorithms
4630 Registry. URL:
4631 <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>
4632
4633 [INFRA]
4634 Anne van Kesteren; Domenic Denicola. Infra Standard. Living
4635 Standard. URL: <https://infra.spec.whatwg.org/>
4636
4637 [MIXED-CONTENT]
4638 Mike West. Mixed Content. 2 August 2016. CR. URL:
4639 <https://www.w3.org/TR/mixed-content/>
4640
4641 [RFC2119]
4642 S. Bradner. Key words for use in RFCs to Indicate Requirement
4643 Levels. March 1997. Best Current Practice. URL:
4644 <https://tools.ietf.org/html/rfc2119>
4645
4646 [RFC4648]
4647 S. Josefsson. The Base16, Base32, and Base64 Data Encodings.
4648 October 2006. Proposed Standard. URL:
4649 <https://tools.ietf.org/html/rfc4648>
4650
4651 [RFC5234]
4652 D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax
4653 Specifications: ABNF. January 2008. Internet Standard. URL:
4654 <https://tools.ietf.org/html/rfc5234>
4655
4656 [RFC5890]
4657 J. Klensin. Internationalized Domain Names for Applications
4658 (IDNA): Definitions and Document Framework. August 2010.
4659 Proposed Standard. URL: <https://tools.ietf.org/html/rfc5890>
4660
4661 [RFC7049]
4662 C. Bormann; P. Hoffman. Concise Binary Object Representation
4663 (CBOR). October 2013. Proposed Standard. URL:
4664 <https://tools.ietf.org/html/rfc7049>
4665
4666 [RFC8152]

4581 J. Schaad. CBOR Object Signing and Encryption (COSE). July 2017.
4582 Proposed Standard. URL: <https://tools.ietf.org/html/rfc8152>
4583

[SECURE-CONTEXTS]
4584 Mike West. Secure Contexts. 15 September 2016. CR. URL:
4585 <https://www.w3.org/TR/secure-contexts/>
4586

[TokenBinding]
4587
4588 A. Popov; et al. The Token Binding Protocol Version 1.0.
4589 February 16, 2017. Internet-Draft. URL:
4590 <https://tools.ietf.org/html/draft-ietf-tokbind-protocol>
4591
4592

[URL]
4593 Anne van Kesteren. URL Standard. Living Standard. URL:
4594 <https://url.spec.whatwg.org/>
4595

[WebAuthn-Registries]
4596 Jeff Hodges; Giridhar Mandyam; Michael B. Jones. Registries for
4597 Web Authentication (WebAuthn). March 2017. Active
4598 Internet-Draft. URL:
4599 <https://xml2rfc.tools.ietf.org/cgi-bin/xml2rfc.cgi?modeAsFormat=html&ascii&url=https://raw.githubusercontent.com/w3c/webauthn/master/draft-hodges-webauthn-registries.xml>
4600
4601
4602
4603

[WebCryptoAPI]
4604 Mark Watson. Web Cryptography API. 26 January 2017. REC. URL:
4605 <https://www.w3.org/TR/WebCryptoAPI/>
4606

[WebIDL]
4607 Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15
4608 December 2016. ED. URL: <https://heycam.github.io/webidl/>
4609

[WebIDL-1]
4610 Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL:
4611 <https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/>
4612
4613

Informative References
4614

[Ceremony]
4615 Carl Ellison. Ceremony Design and Analysis. 2007. URL:
4616 <https://eprint.iacr.org/2007/399.pdf>
4617

[FIDO-APPID]
4618 D. Balfanz; et al. FIDO AppID and Facets. FIDO Alliance Review
4619 Draft. URL:
4620 <https://fidoalliance.org/specs/fido-uaf-v1.1-rd-20161005/fido-appid-and-facets-v1.1-rd-20161005.html>
4621
4622

[FIDO-U2F-Message-Formats]
4623 D. Balfanz; J. Ehrensvar; J. Lang. FIDO U2F Raw Message
4624 Formats. FIDO Alliance Implementation Draft. URL:
4625 <https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-message-formats-v1.1-id-20160915.html>
4626
4627

[FIDOMetadataService]
4628 R. Lindemann; B. Hill; D. Baghdasaryan. FIDO Metadata Service
4629 v1.0. FIDO Alliance Proposed Standard. URL:
4630 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-metadata-service-v1.0-ps-20141208.html>
4631
4632
4633
4634

[FIDOSecRef]
4635 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference.
4636 FIDO Alliance Proposed Standard. URL:
4637 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-security-ref-v1.0-ps-20141208.html>
4638
4639
4640

[GeoJSON]
4641 The GeoJSON Format Specification. URL:
4642 <http://geojson.org/geojson-spec.html>
4643
4644
4645
4646

4669 J. Schaad. CBOR Object Signing and Encryption (COSE). July 2017.
4670 Proposed Standard. URL: <https://tools.ietf.org/html/rfc8152>
4671

[SECURE-CONTEXTS]
4672 Mike West. Secure Contexts. 15 September 2016. CR. URL:
4673 <https://www.w3.org/TR/secure-contexts/>
4674
4675

[TokenBinding]
4676
4677 A. Popov; et al. The Token Binding Protocol Version 1.0.
4678 February 16, 2017. Internet-Draft. URL:
4679 <https://tools.ietf.org/html/draft-ietf-tokbind-protocol>
4680
4681

[URL]
4682 Anne van Kesteren. URL Standard. Living Standard. URL:
4683 <https://url.spec.whatwg.org/>
4684

[WebAuthn-Registries]
4685 Jeff Hodges; Giridhar Mandyam; Michael B. Jones. Registries for
4686 Web Authentication (WebAuthn). March 2017. Active
4687 Internet-Draft. URL:
4688 <https://xml2rfc.tools.ietf.org/cgi-bin/xml2rfc.cgi?modeAsFormat=html&ascii&url=https://raw.githubusercontent.com/w3c/webauthn/master/draft-hodges-webauthn-registries.xml>
4689
4690
4691
4692

[WebCryptoAPI]
4693 Mark Watson. Web Cryptography API. 26 January 2017. REC. URL:
4694 <https://www.w3.org/TR/WebCryptoAPI/>
4695

[WebIDL]
4696 Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15
4697 December 2016. ED. URL: <https://heycam.github.io/webidl/>
4698

[WebIDL-1]
4699 Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL:
4700 <https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/>
4701
4702

Informative References
4703

[Ceremony]
4704 Carl Ellison. Ceremony Design and Analysis. 2007. URL:
4705 <https://eprint.iacr.org/2007/399.pdf>
4706

[FIDO-APPID]
4707 D. Balfanz; et al. FIDO AppID and Facets. FIDO Alliance Review
4708 Draft. URL:
4709 <https://fidoalliance.org/specs/fido-uaf-v1.1-rd-20161005/fido-appid-and-facets-v1.1-rd-20161005.html>
4710
4711

[FIDO-U2F-Message-Formats]
4712 D. Balfanz; J. Ehrensvar; J. Lang. FIDO U2F Raw Message
4713 Formats. FIDO Alliance Implementation Draft. URL:
4714 <https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-message-formats-v1.1-id-20160915.html>
4715
4716

[FIDOMetadataService]
4717 R. Lindemann; B. Hill; D. Baghdasaryan. FIDO Metadata Service
4718 v1.0. FIDO Alliance Proposed Standard. URL:
4719 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-metadata-service-v1.0-ps-20141208.html>
4720
4721
4722

[FIDOSecRef]
4723 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference.
4724 FIDO Alliance Proposed Standard. URL:
4725 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-security-ref-v1.0-ps-20141208.html>
4726
4727
4728

[GeoJSON]
4729 The GeoJSON Format Specification. URL:
4730 <http://geojson.org/geojson-spec.html>
4731
4732
4733
4734

4651 [ISOBiometricVocabulary]
4652 ISO/IEC JTC1/SC37. Information technology -- Vocabulary --
4653 Biometrics. 15 December 2012. International Standard: ISO/IEC
4654 2382-37:2012(E) First Edition. URL:
4655 http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194
4656 _ISOIEC_2382-37_2012.zip

4657
4658 [RFC4949]
4659 R. Shirey. Internet Security Glossary, Version 2. August 2007.
4660 Informational. URL: https://tools.ietf.org/html/rfc4949

4661
4662 [RFC5280]
4663 D. Cooper; et al. Internet X.509 Public Key Infrastructure
4664 Certificate and Certificate Revocation List (CRL) Profile. May
4665 2008. Proposed Standard. URL:
4666 https://tools.ietf.org/html/rfc5280

4667
4668 [RFC6265]
4669 A. Barth. HTTP State Management Mechanism. April 2011. Proposed
4670 Standard. URL: https://tools.ietf.org/html/rfc6265

4671
4672 [RFC6454]
4673 A. Barth. The Web Origin Concept. December 2011. Proposed
4674 Standard. URL: https://tools.ietf.org/html/rfc6454

4675
4676 [RFC7515]
4677 M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May
4678 2015. Proposed Standard. URL:
4679 https://tools.ietf.org/html/rfc7515

4680
4681 [RFC8017]
4682 K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography
4683 Specifications Version 2.2. November 2016. Informational. URL:
4684 https://tools.ietf.org/html/rfc8017

4685
4686 [TPMv2-EK-Profile]
4687 TCG EK Credential Profile for TPM Family 2.0. URL:
4688 http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti
4689 al_Profile_EK_V2.0_R14_published.pdf

4690
4691 [TPMv2-Part1]
4692 Trusted Platform Module Library, Part 1: Architecture. URL:
4693 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4694 2.0-Part-1-Architecture-01.38.pdf

4695
4696 [TPMv2-Part2]
4697 Trusted Platform Module Library, Part 2: Structures. URL:
4698 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4699 2.0-Part-2-Structures-01.38.pdf

4700
4701 [TPMv2-Part3]
4702 Trusted Platform Module Library, Part 3: Commands. URL:
4703 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4704 2.0-Part-3-Commands-01.38.pdf

4705
4706 [UAFProtocol]
4707 R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO
4708 Alliance Proposed Standard. URL:
4709 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua
4710 f-protocol-v1.0-ps-20141208.html

4711
4712 IDL Index

4713
4714 [SecureContext]
4715 interface PublicKeyCredential : Credential {
4716 [SameObject] readonly attribute ArrayBuffer rawId;
4717 [SameObject] readonly attribute AuthenticatorResponse response;
4718 [SameObject] readonly attribute AuthenticationExtensions clientExtensionResu
4719 lts;
4720 };

4739 [ISOBiometricVocabulary]
4740 ISO/IEC JTC1/SC37. Information technology -- Vocabulary --
4741 Biometrics. 15 December 2012. International Standard: ISO/IEC
4742 2382-37:2012(E) First Edition. URL:
4743 http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194
4744 _ISOIEC_2382-37_2012.zip

4745
4746 [RFC4949]
4747 R. Shirey. Internet Security Glossary, Version 2. August 2007.
4748 Informational. URL: https://tools.ietf.org/html/rfc4949

4749
4750 [RFC5280]
4751 D. Cooper; et al. Internet X.509 Public Key Infrastructure
4752 Certificate and Certificate Revocation List (CRL) Profile. May
4753 2008. Proposed Standard. URL:
4754 https://tools.ietf.org/html/rfc5280

4755
4756 [RFC6265]
4757 A. Barth. HTTP State Management Mechanism. April 2011. Proposed
4758 Standard. URL: https://tools.ietf.org/html/rfc6265

4759
4760 [RFC6454]
4761 A. Barth. The Web Origin Concept. December 2011. Proposed
4762 Standard. URL: https://tools.ietf.org/html/rfc6454

4763
4764 [RFC7515]
4765 M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May
4766 2015. Proposed Standard. URL:
4767 https://tools.ietf.org/html/rfc7515

4768
4769 [RFC8017]
4770 K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography
4771 Specifications Version 2.2. November 2016. Informational. URL:
4772 https://tools.ietf.org/html/rfc8017

4773
4774 [TPMv2-EK-Profile]
4775 TCG EK Credential Profile for TPM Family 2.0. URL:
4776 http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti
4777 al_Profile_EK_V2.0_R14_published.pdf

4778
4779 [TPMv2-Part1]
4780 Trusted Platform Module Library, Part 1: Architecture. URL:
4781 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4782 2.0-Part-1-Architecture-01.38.pdf

4783
4784 [TPMv2-Part2]
4785 Trusted Platform Module Library, Part 2: Structures. URL:
4786 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4787 2.0-Part-2-Structures-01.38.pdf

4788
4789 [TPMv2-Part3]
4790 Trusted Platform Module Library, Part 3: Commands. URL:
4791 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4792 2.0-Part-3-Commands-01.38.pdf

4793
4794 [UAFProtocol]
4795 R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO
4796 Alliance Proposed Standard. URL:
4797 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua
4798 f-protocol-v1.0-ps-20141208.html

4799
4800 IDL Index

4801
4802 [SecureContext]
4803 interface PublicKeyCredential : Credential {
4804 [SameObject] readonly attribute ArrayBuffer rawId;
4805 [SameObject] readonly attribute AuthenticatorResponse response;
4806 [SameObject] readonly attribute AuthenticationExtensions clientExtensionResu
4807 lts;
4808 };

```

4721 partial dictionary CredentialCreationOptions {
4722   MakePublicKeyCredentialOptions publicKey;
4723 };
4724
4725 partial dictionary CredentialRequestOptions {
4726   PublicKeyCredentialRequestOptions publicKey;
4727 };
4728
4729 [SecureContext]
4730 partial interface PublicKeyCredential {
4731   [Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();
4732 };
4733
4734 [SecureContext]
4735 interface AuthenticatorResponse {
4736   [SameObject] readonly attribute ArrayBuffer clientDataJSON;
4737 };
4738
4739 [SecureContext]
4740 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
4741   [SameObject] readonly attribute ArrayBuffer attestationObject;
4742 };
4743
4744 [SecureContext]
4745 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
4746   [SameObject] readonly attribute ArrayBuffer authenticatorData;
4747   [SameObject] readonly attribute ArrayBuffer signature;
4748 };
4749
4750 dictionary PublicKeyCredentialParameters {
4751   required PublicKeyCredentialType type;
4752   required COSEAlgorithmIdentifier alg;
4753 };
4754
4755 dictionary MakePublicKeyCredentialOptions {
4756   required PublicKeyCredentialRpEntity rp;
4757   required PublicKeyCredentialUserEntity user;
4758
4759   required BufferSource challenge;
4760   required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
4761
4762   unsigned long timeout;
4763   sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
4764   AuthenticatorSelectionCriteria authenticatorSelection;
4765   AuthenticationExtensions extensions;
4766 };
4767
4768 dictionary PublicKeyCredentialEntity {
4769   DOMString name;
4770   USVString icon;
4771 };
4772
4773 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
4774   DOMString id;
4775 };
4776
4777 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
4778   BufferSource id;
4779   DOMString displayName;
4780 };
4781
4782 dictionary AuthenticatorSelectionCriteria {
4783   AuthenticatorAttachment authenticatorAttachment;
4784   boolean requireResidentKey = false;
4785   boolean requireUserVerification = false;
4786 };
4787
4788 enum AuthenticatorAttachment {
4789   "platform", // Platform attachment
4790

```

```

4809 partial dictionary CredentialCreationOptions {
4810   MakePublicKeyCredentialOptions publicKey;
4811 };
4812
4813 partial dictionary CredentialRequestOptions {
4814   PublicKeyCredentialRequestOptions publicKey;
4815 };
4816
4817 [SecureContext]
4818 partial interface PublicKeyCredential {
4819   [Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();
4820 };
4821
4822 [SecureContext]
4823 interface AuthenticatorResponse {
4824   [SameObject] readonly attribute ArrayBuffer clientDataJSON;
4825 };
4826
4827 [SecureContext]
4828 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
4829   [SameObject] readonly attribute ArrayBuffer attestationObject;
4830 };
4831
4832 [SecureContext]
4833 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
4834   [SameObject] readonly attribute ArrayBuffer authenticatorData;
4835   [SameObject] readonly attribute ArrayBuffer signature;
4836 };
4837
4838 dictionary PublicKeyCredentialParameters {
4839   required PublicKeyCredentialType type;
4840   required COSEAlgorithmIdentifier alg;
4841 };
4842
4843 dictionary MakePublicKeyCredentialOptions {
4844   required PublicKeyCredentialRpEntity rp;
4845   required PublicKeyCredentialUserEntity user;
4846
4847   required BufferSource challenge;
4848   required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
4849
4850   unsigned long timeout;
4851   sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
4852   AuthenticatorSelectionCriteria authenticatorSelection;
4853   AuthenticationExtensions extensions;
4854 };
4855
4856 dictionary PublicKeyCredentialEntity {
4857   DOMString name;
4858   USVString icon;
4859 };
4860
4861 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
4862   DOMString id;
4863 };
4864
4865 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
4866   BufferSource id;
4867   DOMString displayName;
4868 };
4869
4870 dictionary AuthenticatorSelectionCriteria {
4871   AuthenticatorAttachment authenticatorAttachment;
4872   boolean requireResidentKey = false;
4873   boolean requireUserVerification = false;
4874 };
4875
4876 enum AuthenticatorAttachment {
4877   "platform", // Platform attachment
4878

```

```

4791 "cross-platform" // Cross-platform attachment
4792 };
4793
4794 dictionary PublicKeyCredentialRequestOptions {
4795     required BufferSource challenge;
4796     unsigned long timeout;
4797     USVString rpId;
4798     sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
4799     AuthenticationExtensions extensions;
4800 };
4801
4802 typedef record<DOMString, any> AuthenticationExtensions;
4803
4804 dictionary CollectedClientData {
4805     required DOMString challenge;
4806     required DOMString origin;
4807     required DOMString hashAlgorithm;
4808     DOMString tokenBindingId;
4809     AuthenticationExtensions clientExtensions;
4810     AuthenticationExtensions authenticatorExtensions;
4811 };
4812
4813 enum PublicKeyCredentialType {
4814     "public-key"
4815 };
4816
4817 dictionary PublicKeyCredentialDescriptor {
4818     required PublicKeyCredentialType type;
4819     required BufferSource id;
4820     sequence<AuthenticatorTransport> transports;
4821 };
4822
4823 enum AuthenticatorTransport {
4824     "usb",
4825     "nfc",
4826     "ble"
4827 };
4828
4829 typedef long COSEAlgorithmIdentifier;
4830
4831 typedef sequence<AAGUID> AuthenticatorSelectionList;
4832
4833 typedef BufferSource AAGUID;
4834
4835

```

```

4836 #base64url-encodingReferenced in:
4837 * 4.1. PublicKeyCredential Interface
4838 * 4.1.3. Create a new credential - PublicKeyCredential's
4839 [[Create]](options) method (2)
4840 * 4.1.4. Use an existing credential to make an assertion -
4841 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
4842 method (2)
4843 * 6.2. Verifying an authentication assertion
4844
4845 #cborReferenced in:
4846 * 4.1.3. Create a new credential - PublicKeyCredential's
4847 [[Create]](options) method
4848 * 4.1.4. Use an existing credential to make an assertion -
4849 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
4850 method
4851 * 5.1. Authenticator data (2)
4852 * 8. WebAuthn Extensions (2) (3)
4853 * 8.2. Defining extensions (2)
4854 * 8.3. Extending request parameters

```

```

4879 "cross-platform" // Cross-platform attachment
4880 };
4881
4882 dictionary PublicKeyCredentialRequestOptions {
4883     required BufferSource challenge;
4884     unsigned long timeout;
4885     USVString rpId;
4886     sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
4887     AuthenticationExtensions extensions;
4888 };
4889
4890 typedef record<DOMString, any> AuthenticationExtensions;
4891
4892 dictionary CollectedClientData {
4893     required DOMString challenge;
4894     required DOMString origin;
4895     required DOMString hashAlgorithm;
4896     DOMString tokenBindingId;
4897     AuthenticationExtensions clientExtensions;
4898     AuthenticationExtensions authenticatorExtensions;
4899 };
4900
4901 enum PublicKeyCredentialType {
4902     "public-key"
4903 };
4904
4905 dictionary PublicKeyCredentialDescriptor {
4906     required PublicKeyCredentialType type;
4907     required BufferSource id;
4908     sequence<AuthenticatorTransport> transports;
4909 };
4910
4911 enum AuthenticatorTransport {
4912     "usb",
4913     "nfc",
4914     "ble"
4915 };
4916
4917 typedef long COSEAlgorithmIdentifier;
4918
4919 typedef sequence<AAGUID> AuthenticatorSelectionList;
4920
4921 typedef BufferSource AAGUID;
4922
4923

```

[Issues Index](#)

is there any special way to declare credentialCreationData such that it properly exists when constructCredentialCallback (defined just below) is invoked? RET

```

4930 #base64url-encodingReferenced in:
4931 * 4.1. PublicKeyCredential Interface
4932 * 4.1.3. Create a new credential - PublicKeyCredential's
4933 [[Create]](options) method (2)
4934 * 4.1.4. Use an existing credential to make an assertion -
4935 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
4936 method (2)
4937 * 6.2. Verifying an authentication assertion
4938
4939 #cborReferenced in:
4940 * 4.1.3. Create a new credential - PublicKeyCredential's
4941 [[Create]](options) method
4942 * 4.1.4. Use an existing credential to make an assertion -
4943 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
4944 method
4945 * 5.1. Authenticator data (2)
4946 * 8. WebAuthn Extensions (2) (3)
4947 * 8.2. Defining extensions (2)
4948 * 8.3. Extending request parameters

```

4855 * 8.4. Client extension processing (2)
4856 * 8.5. Authenticator extension processing (2) (3) (4) (5)
4857
4858 #attestationReferenced in:
4859 * 3. Terminology
4860 * 5. WebAuthn Authenticator model (2)
4861 * 5.3. Attestation (2) (3) (4)
4862
4863 #attestation-certificateReferenced in:
4864 * 3. Terminology (2)
4865 * 7.3.1. TPM attestation statement certificate requirements
4866
4867 #attestation-key-pairReferenced in:
4868 * 3. Terminology (2)
4869 * 5.3. Attestation
4870
4871 #attestation-private-keyReferenced in:
4872 * 5. WebAuthn Authenticator model
4873 * 5.3. Attestation
4874
4875 #attestation-public-keyReferenced in:
4876 * 5.3. Attestation
4877
4878 #authenticationReferenced in:
4879 * 1. Introduction (2)
4880 * 3. Terminology (2) (3) (4) (5) (6) (7)
4881 * 6.2. Verifying an authentication assertion
4882
4883 #authentication-assertionReferenced in:
4884 * 1. Introduction
4885 * 3. Terminology (2) (3)
4886 * 4.1. PublicKeyCredential Interface
4887 * 4.2.2. Web Authentication Assertion (interface
4888 AuthenticatorAssertionResponse)
4889 * 4.5. Options for Assertion Generation (dictionary
4890 PublicKeyCredentialRequestOptions)
4891 * 8. WebAuthn Extensions
4892
4893 #authenticatorReferenced in:
4894 * 1. Introduction (2) (3) (4)
4895 * 1.1. Use Cases
4896 * 2. Conformance
4897 * 3. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
4898 (14) (15)
4899 * 4. Web Authentication API (2) (3)
4900 * 4.1. PublicKeyCredential Interface
4901 * 4.1.3. Create a new credential - PublicKeyCredential's
4902 [[Create]](options) method (2)
4903 * 4.1.4. Use an existing credential to make an assertion -
4904 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
4905 method (2) (3)
4906 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
4907 * 4.2.1. Information about Public Key Credential (interface
4908 AuthenticatorAttestationResponse) (2)
4909 * 4.2.2. Web Authentication Assertion (interface
4910 AuthenticatorAssertionResponse)
4911 * 4.4.5. Authenticator Attachment enumeration (enum
4912 AuthenticatorAttachment)
4913 * 4.5. Options for Assertion Generation (dictionary
4914 PublicKeyCredentialRequestOptions)
4915 * 5. WebAuthn Authenticator model (2) (3) (4) (5) (6)
4916 * 5.1. Authenticator data
4917 * 5.2.1. The authenticatorMakeCredential operation
4918 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
4919 * 5.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)
4920 * 5.3.2. Attestation Statement Formats
4921 * 5.3.4. Generating an Attestation Object (2)
4922 * 5.3.5.1. Privacy
4923 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
4924 Compromise

4949 * 8.4. Client extension processing (2)
4950 * 8.5. Authenticator extension processing (2) (3) (4) (5)
4951
4952 #attestationReferenced in:
4953 * 3. Terminology
4954 * 5. WebAuthn Authenticator model (2)
4955 * 5.3. Attestation (2) (3) (4)
4956
4957 #attestation-certificateReferenced in:
4958 * 3. Terminology (2)
4959 * 7.3.1. TPM attestation statement certificate requirements
4960
4961 #attestation-key-pairReferenced in:
4962 * 3. Terminology (2)
4963 * 5.3. Attestation
4964
4965 #attestation-private-keyReferenced in:
4966 * 5. WebAuthn Authenticator model
4967 * 5.3. Attestation
4968
4969 #attestation-public-keyReferenced in:
4970 * 5.3. Attestation
4971
4972 #authenticationReferenced in:
4973 * 1. Introduction (2)
4974 * 3. Terminology (2) (3) (4) (5) (6) (7)
4975 * 6.2. Verifying an authentication assertion
4976
4977 #authentication-assertionReferenced in:
4978 * 1. Introduction
4979 * 3. Terminology (2) (3)
4980 * 4.1. PublicKeyCredential Interface
4981 * 4.2.2. Web Authentication Assertion (interface
4982 AuthenticatorAssertionResponse)
4983 * 4.5. Options for Assertion Generation (dictionary
4984 PublicKeyCredentialRequestOptions)
4985 * 8. WebAuthn Extensions
4986
4987 #authenticatorReferenced in:
4988 * 1. Introduction (2) (3) (4)
4989 * 1.1. Use Cases
4990 * 2. Conformance
4991 * 3. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
4992 (14) (15)
4993 * 4. Web Authentication API (2) (3)
4994 * 4.1. PublicKeyCredential Interface
4995 * 4.1.3. Create a new credential - PublicKeyCredential's
4996 [[Create]](options) method (2)
4997 * 4.1.4. Use an existing credential to make an assertion -
4998 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
4999 method (2) (3)
5000 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
5001 * 4.2.1. Information about Public Key Credential (interface
5002 AuthenticatorAttestationResponse) (2)
5003 * 4.2.2. Web Authentication Assertion (interface
5004 AuthenticatorAssertionResponse)
5005 * 4.4.5. Authenticator Attachment enumeration (enum
5006 AuthenticatorAttachment)
5007 * 4.5. Options for Assertion Generation (dictionary
5008 PublicKeyCredentialRequestOptions)
5009 * 5. WebAuthn Authenticator model (2) (3) (4) (5) (6)
5010 * 5.1. Authenticator data
5011 * 5.2.1. The authenticatorMakeCredential operation
5012 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
5013 * 5.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)
5014 * 5.3.2. Attestation Statement Formats
5015 * 5.3.4. Generating an Attestation Object (2)
5016 * 5.3.5.1. Privacy
5017 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
5018 Compromise

4925 * 6.1. Registering a new credential
4926 * 7.2. Packed Attestation Statement Format
4927 * 7.4. Android Key Attestation Statement Format
4928 * 7.5. Android SafetyNet Attestation Statement Format
4929 * 9.5. Supported Extensions Extension (exts)
4930 * 9.6. User Verification Index Extension (uvi)
4931 * 9.7. Location Extension (loc) (2) (3) (4)
4932 * 9.8. User Verification Method Extension (uvm)
4933 * 11. Sample scenarios
4934
4935 #authorization-gestureReferenced in:
4936 * 1.1.1. Registration
4937 * 1.1.2. Authentication
4938 * 1.1.3. Other use cases and configurations
4939 * 3. Terminology (2) (3) (4) (5) (6)
4940
4941 #biometric-recognitionReferenced in:
4942 * 3. Terminology (2)
4943
4944 #ceremonyReferenced in:
4945 * 1. Introduction
4946 * 3. Terminology (2) (3) (4) (5) (6) (7)
4947 * 6.1. Registering a new credential
4948 * 6.2. Verifying an authentication assertion
4949
4950 #clientReferenced in:
4951 * 3. Terminology
4952 * 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
4953 isPlatformAuthenticatorAvailable() method (2) (3) (4)
4954
4955 #client-side-resident-credential-private-keyReferenced in:
4956 * 3. Terminology (2)
4957 * 4.1.3. Create a new credential - PublicKeyCredential's
4958 [[Create]](options) method
4959 * 4.4.4. Authenticator Selection Criteria (dictionary
4960 AuthenticatorSelectionCriteria) (2)
4961 * 5.2.1. The authenticatorMakeCredential operation
4962
4963 #conforming-user-agentReferenced in:
4964 * 1. Introduction
4965 * 2. Conformance (2) (3)
4966 * 3. Terminology (2)
4967
4968 #credential-public-keyReferenced in:
4969 * 3. Terminology (2) (3)
4970 * 4.2.1. Information about Public Key Credential (interface
4971 AuthenticatorAttestationResponse)
4972 * 5. WebAuthn Authenticator model
4973 * 5.1. Authenticator data
4974 * 5.3. Attestation (2) (3)
4975 * 5.3.1. Attestation data (2)
4976 * 7.4. Android Key Attestation Statement Format
4977 * 11.1. Registration (2)
4978
4979 #credential-key-pairReferenced in:
4980 * 3. Terminology (2) (3)
4981 * 4.1.3. Create a new credential - PublicKeyCredential's
4982 [[Create]](options) method
4983
4984 #credential-private-keyReferenced in:
4985 * 3. Terminology (2) (3) (4)
4986 * 4.1. PublicKeyCredential Interface
4987 * 4.2.2. Web Authentication Assertion (interface
4988 AuthenticatorAssertionResponse)
4989 * 5. WebAuthn Authenticator model
4990 * 5.2.2. The authenticatorGetAssertion operation
4991 * 5.3. Attestation (2)
4992
4993 #registrationReferenced in:
4994 * 1. Introduction (2)

5019 * 6.1. Registering a new credential
5020 * 7.2. Packed Attestation Statement Format
5021 * 7.4. Android Key Attestation Statement Format
5022 * 7.5. Android SafetyNet Attestation Statement Format
5023 * 9.5. Supported Extensions Extension (exts)
5024 * 9.6. User Verification Index Extension (uvi)
5025 * 9.7. Location Extension (loc) (2) (3) (4)
5026 * 9.8. User Verification Method Extension (uvm)
5027 * 11. Sample scenarios
5028
5029 #authorization-gestureReferenced in:
5030 * 1.1.1. Registration
5031 * 1.1.2. Authentication
5032 * 1.1.3. Other use cases and configurations
5033 * 3. Terminology (2) (3) (4) (5) (6)
5034
5035 #biometric-recognitionReferenced in:
5036 * 3. Terminology (2)
5037
5038 #ceremonyReferenced in:
5039 * 1. Introduction
5040 * 3. Terminology (2) (3) (4) (5) (6) (7)
5041 * 6.1. Registering a new credential
5042 * 6.2. Verifying an authentication assertion
5043
5044 #clientReferenced in:
5045 * 3. Terminology
5046 * 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
5047 isPlatformAuthenticatorAvailable() method (2) (3) (4)
5048
5049 #client-side-resident-credential-private-keyReferenced in:
5050 * 3. Terminology (2)
5051 * 4.1.3. Create a new credential - PublicKeyCredential's
5052 [[Create]](options) method
5053 * 4.4.4. Authenticator Selection Criteria (dictionary
5054 AuthenticatorSelectionCriteria) (2)
5055 * 5.2.1. The authenticatorMakeCredential operation
5056
5057 #conforming-user-agentReferenced in:
5058 * 1. Introduction
5059 * 2. Conformance (2) (3)
5060 * 3. Terminology (2)
5061
5062 #credential-public-keyReferenced in:
5063 * 3. Terminology (2) (3)
5064 * 4.2.1. Information about Public Key Credential (interface
5065 AuthenticatorAttestationResponse)
5066 * 5. WebAuthn Authenticator model
5067 * 5.1. Authenticator data
5068 * 5.3. Attestation (2) (3)
5069 * 5.3.1. Attestation data (2)
5070 * 7.4. Android Key Attestation Statement Format
5071 * 11.1. Registration (2)
5072
5073 #credential-key-pairReferenced in:
5074 * 3. Terminology (2) (3)
5075 * 4.1.3. Create a new credential - PublicKeyCredential's
5076 [[Create]](options) method
5077
5078 #credential-private-keyReferenced in:
5079 * 3. Terminology (2) (3) (4)
5080 * 4.1. PublicKeyCredential Interface
5081 * 4.2.2. Web Authentication Assertion (interface
5082 AuthenticatorAssertionResponse)
5083 * 5. WebAuthn Authenticator model
5084 * 5.2.2. The authenticatorGetAssertion operation
5085 * 5.3. Attestation (2)
5086
5087 #registrationReferenced in:
5088 * 1. Introduction (2)

4995 * 3. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
4996 * 6.1. Registering a new credential
4997
4998 #relying-partyReferenced in:
4999 * 1. Introduction (2) (3) (4) (5) (6) (7)
5000 * 1.1.3. Other use cases and configurations
5001 * 3. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
5002 (14) (15) (16) (17) (18) (19) (20) (21) (22)
5003 * 4. Web Authentication API (2) (3) (4) (5) (6) (7)
5004 * 4.1.4. Use an existing credential to make an assertion -
5005 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5006 method (2)
5007 * 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
5008 isPlatformAuthenticatorAvailable() method (2) (3)
5009 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
5010 * 4.2.1. Information about Public Key Credential (interface
5011 AuthenticatorAttestationResponse) (2)
5012 * 4.2.2. Web Authentication Assertion (interface
5013 AuthenticatorAssertionResponse)
5014 * 4.4. Options for Credential Creation (dictionary
5015 MakePublicKeyCredentialOptions) (2) (3) (4) (5) (6) (7) (8)
5016 * 4.4.1. Public Key Entity Description (dictionary
5017 PublicKeyCredentialEntity) (2) (3)
5018 * 4.4.2. RP Parameters for Credential Generation (dictionary
5019 PublicKeyCredentialRpEntity) (2)
5020 * 4.4.3. User Account Parameters for Credential Generation
5021 (dictionary PublicKeyCredentialUserEntity)
5022 * 4.4.4. Authenticator Selection Criteria (dictionary
5023 AuthenticatorSelectionCriteria) (2) (3)
5024 * 4.4.5. Authenticator Attachment enumeration (enum
5025 AuthenticatorAttachment) (2) (3) (4)
5026 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5027 CollectedClientData) (2) (3) (4)
5028 * 4.7.4. Authenticator Transport enumeration (enum
5029 AuthenticatorTransport) (2)
5030 * 5. WebAuthn Authenticator model (2)
5031 * 5.1. Authenticator data (2)
5032 * 5.2.1. The authenticatorMakeCredential operation (2) (3) (4)
5033 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
5034 * 5.3. Attestation (2) (3) (4) (5) (6)
5035 * 5.3.5.1. Privacy
5036 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
5037 Compromise (2) (3) (4) (5) (6)
5038 * 6. Relying Party Operations (2) (3) (4)
5039 * 6.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)
5040 (10) (11) (12) (13)
5041 * 6.2. Verifying an authentication assertion (2) (3) (4) (5)
5042 * 7.4. Android Key Attestation Statement Format
5043 * 8. WebAuthn Extensions (2) (3) (4)
5044 * 8.2. Defining extensions (2)
5045 * 8.3. Extending request parameters (2) (3) (4)
5046 * 8.6. Example Extension (2) (3)
5047 * 9.1. FIDO Appid Extension (appid) (2)
5048 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
5049 * 9.4. Authenticator Selection Extension (authnSel) (2) (3)
5050 * 9.5. Supported Extensions Extension (exts) (2)
5051 * 9.6. User Verification Index Extension (uvi)
5052 * 9.7. Location Extension (loc) (2)
5053 * 10.2. WebAuthn Extension Identifier Registrations (2)
5054 * 11.1. Registration (2) (3) (4) (5)
5055 * 11.2. Registration Specifically with Platform Authenticator (2) (3)
5056 * 11.3. Authentication (2) (3) (4) (5)
5057 * 11.4. Decommissioning (2)
5058
5059 #relying-party-identifierReferenced in:
5060 * 4. Web Authentication API
5061 * 4.4. Options for Credential Creation (dictionary
5062 MakePublicKeyCredentialOptions)
5063 * 4.5. Options for Assertion Generation (dictionary
5064 PublicKeyCredentialRequestOptions)

5089 * 3. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
5090 * 6.1. Registering a new credential
5091
5092 #relying-partyReferenced in:
5093 * 1. Introduction (2) (3) (4) (5) (6) (7)
5094 * 1.1.3. Other use cases and configurations
5095 * 3. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
5096 (14) (15) (16) (17) (18) (19) (20) (21) (22)
5097 * 4. Web Authentication API (2) (3) (4) (5) (6) (7)
5098 * 4.1.4. Use an existing credential to make an assertion -
5099 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5100 method (2)
5101 * 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
5102 isPlatformAuthenticatorAvailable() method (2) (3)
5103 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
5104 * 4.2.1. Information about Public Key Credential (interface
5105 AuthenticatorAttestationResponse) (2)
5106 * 4.2.2. Web Authentication Assertion (interface
5107 AuthenticatorAssertionResponse)
5108 * 4.4. Options for Credential Creation (dictionary
5109 MakePublicKeyCredentialOptions) (2) (3) (4) (5) (6) (7) (8)
5110 * 4.4.1. Public Key Entity Description (dictionary
5111 PublicKeyCredentialEntity) (2) (3)
5112 * 4.4.2. RP Parameters for Credential Generation (dictionary
5113 PublicKeyCredentialRpEntity) (2)
5114 * 4.4.3. User Account Parameters for Credential Generation
5115 (dictionary PublicKeyCredentialUserEntity)
5116 * 4.4.4. Authenticator Selection Criteria (dictionary
5117 AuthenticatorSelectionCriteria) (2) (3)
5118 * 4.4.5. Authenticator Attachment enumeration (enum
5119 AuthenticatorAttachment) (2) (3) (4)
5120 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5121 CollectedClientData) (2) (3) (4)
5122 * 4.7.4. Authenticator Transport enumeration (enum
5123 AuthenticatorTransport) (2)
5124 * 5. WebAuthn Authenticator model (2)
5125 * 5.1. Authenticator data (2)
5126 * 5.2.1. The authenticatorMakeCredential operation (2) (3) (4)
5127 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
5128 * 5.3. Attestation (2) (3) (4) (5) (6)
5129 * 5.3.5.1. Privacy
5130 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
5131 Compromise (2) (3) (4) (5) (6)
5132 * 6. Relying Party Operations (2) (3) (4)
5133 * 6.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)
5134 (10) (11) (12) (13)
5135 * 6.2. Verifying an authentication assertion (2) (3) (4) (5)
5136 * 7.4. Android Key Attestation Statement Format
5137 * 8. WebAuthn Extensions (2) (3) (4)
5138 * 8.2. Defining extensions (2)
5139 * 8.3. Extending request parameters (2) (3) (4)
5140 * 8.6. Example Extension (2) (3)
5141 * 9.1. FIDO Appid Extension (appid) (2)
5142 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
5143 * 9.4. Authenticator Selection Extension (authnSel) (2) (3)
5144 * 9.5. Supported Extensions Extension (exts) (2)
5145 * 9.6. User Verification Index Extension (uvi)
5146 * 9.7. Location Extension (loc) (2)
5147 * 10.2. WebAuthn Extension Identifier Registrations (2)
5148 * 11.1. Registration (2) (3) (4) (5)
5149 * 11.2. Registration Specifically with Platform Authenticator (2) (3)
5150 * 11.3. Authentication (2) (3) (4) (5)
5151 * 11.4. Decommissioning (2)
5152
5153 #relying-party-identifierReferenced in:
5154 * 4. Web Authentication API
5155 * 4.4. Options for Credential Creation (dictionary
5156 MakePublicKeyCredentialOptions)
5157 * 4.5. Options for Assertion Generation (dictionary
5158 PublicKeyCredentialRequestOptions)

5065 * 5. WebAuthn Authenticator model
5066
5067 #rp-idReferenced in:
5068 * 3. Terminology (2) (3) (4) (5) (6)
5069 * 4. Web Authentication API (2) (3) (4) (5)
5070 * 4.1.3. Create a new credential - PublicKeyCredential's
5071 [[Create]](options) method (2)
5072 * 4.1.4. Use an existing credential to make an assertion -
5073 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5074 method (2)
5075 * 4.4.2. RP Parameters for Credential Generation (dictionary
5076 PublicKeyCredentialRpEntity)
5077 * 5. WebAuthn Authenticator model
5078 * 5.1. Authenticator data (2) (3) (4) (5) (6)
5079 * 5.2.1. The authenticatorMakeCredential operation (2) (3)
5080 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
5081 * 6.1. Registering a new credential (2)
5082 * 6.2. Verifying an authentication assertion (2)
5083 * 7.4. Android Key Attestation Statement Format
5084 * 7.6. FIDO U2F Attestation Statement Format (2) (3)
5085
5086 #public-key-credentialReferenced in:
5087 * 1. Introduction (2) (3) (4) (5)
5088 * 3. Terminology (2) (3) (4) (5) (6) (7) (8)
5089 * 4. Web Authentication API (2) (3) (4)
5090 * 4.1. PublicKeyCredential Interface
5091 * 4.1.3. Create a new credential - PublicKeyCredential's
5092 [[Create]](options) method
5093 * 4.1.4. Use an existing credential to make an assertion -
5094 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5095 method (2)
5096 * 4.2.1. Information about Public Key Credential (interface
5097 AuthenticatorAttestationResponse)
5098 * 4.4.1. Public Key Entity Description (dictionary
5099 PublicKeyCredentialEntity)
5100 * 4.4.4. Authenticator Selection Criteria (dictionary
5101 AuthenticatorSelectionCriteria)
5102 * 4.5. Options for Assertion Generation (dictionary
5103 PublicKeyCredentialRequestOptions)
5104 * 4.7. Supporting Data Structures
5105 * 5. WebAuthn Authenticator model (2) (3) (4) (5)
5106 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
5107 * 5.3. Attestation (2)
5108 * 5.3.2. Attestation Statement Formats
5109 * 5.3.3. Attestation Types
5110 * 5.3.4. Generating an Attestation Object
5111 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
5112 Compromise (2)
5113 * 6.1. Registering a new credential
5114 * 8. WebAuthn Extensions (2)
5115 * 11. Sample scenarios
5116
5117 #test-of-user-presenceReferenced in:
5118 * 3. Terminology (2) (3) (4) (5) (6)
5119 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
5120 * 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
5121
5122 #user-consentReferenced in:
5123 * 1. Introduction
5124 * 3. Terminology (2)
5125 * 4.2.2. Web Authentication Assertion (interface
5126 AuthenticatorAssertionResponse)
5127 * 5. WebAuthn Authenticator model (2) (3)
5128 * 5.2.2. The authenticatorGetAssertion operation (2)
5129
5130 #user-verificationReferenced in:
5131 * 1. Introduction
5132 * 3. Terminology (2) (3) (4) (5) (6) (7) (8)
5133 * 4.1.3. Create a new credential - PublicKeyCredential's
5134 [[Create]](options) method

5159 * 5. WebAuthn Authenticator model
5160
5161 #rp-idReferenced in:
5162 * 3. Terminology (2) (3) (4) (5) (6)
5163 * 4. Web Authentication API (2) (3) (4) (5)
5164 * 4.1.3. Create a new credential - PublicKeyCredential's
5165 [[Create]](options) method (2)
5166 * 4.1.4. Use an existing credential to make an assertion -
5167 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5168 method (2)
5169 * 4.4.2. RP Parameters for Credential Generation (dictionary
5170 PublicKeyCredentialRpEntity)
5171 * 5. WebAuthn Authenticator model
5172 * 5.1. Authenticator data (2) (3) (4) (5) (6)
5173 * 5.2.1. The authenticatorMakeCredential operation (2) (3)
5174 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
5175 * 6.1. Registering a new credential (2)
5176 * 6.2. Verifying an authentication assertion (2)
5177 * 7.4. Android Key Attestation Statement Format
5178 * 7.6. FIDO U2F Attestation Statement Format (2) (3)
5179
5180 #public-key-credentialReferenced in:
5181 * 1. Introduction (2) (3) (4) (5)
5182 * 3. Terminology (2) (3) (4) (5) (6) (7) (8)
5183 * 4. Web Authentication API (2) (3) (4)
5184 * 4.1. PublicKeyCredential Interface
5185 * 4.1.3. Create a new credential - PublicKeyCredential's
5186 [[Create]](options) method
5187 * 4.1.4. Use an existing credential to make an assertion -
5188 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5189 method (2)
5190 * 4.2.1. Information about Public Key Credential (interface
5191 AuthenticatorAttestationResponse)
5192 * 4.4.1. Public Key Entity Description (dictionary
5193 PublicKeyCredentialEntity)
5194 * 4.4.4. Authenticator Selection Criteria (dictionary
5195 AuthenticatorSelectionCriteria)
5196 * 4.5. Options for Assertion Generation (dictionary
5197 PublicKeyCredentialRequestOptions)
5198 * 4.7. Supporting Data Structures
5199 * 5. WebAuthn Authenticator model (2) (3) (4) (5)
5200 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
5201 * 5.3. Attestation (2)
5202 * 5.3.2. Attestation Statement Formats
5203 * 5.3.3. Attestation Types
5204 * 5.3.4. Generating an Attestation Object
5205 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
5206 Compromise (2)
5207 * 6.1. Registering a new credential
5208 * 8. WebAuthn Extensions (2)
5209 * 11. Sample scenarios
5210
5211 #test-of-user-presenceReferenced in:
5212 * 3. Terminology (2) (3) (4) (5) (6)
5213 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
5214 * 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
5215
5216 #user-consentReferenced in:
5217 * 1. Introduction
5218 * 3. Terminology (2)
5219 * 4.2.2. Web Authentication Assertion (interface
5220 AuthenticatorAssertionResponse)
5221 * 5. WebAuthn Authenticator model (2) (3)
5222 * 5.2.2. The authenticatorGetAssertion operation (2)
5223
5224 #user-verificationReferenced in:
5225 * 1. Introduction
5226 * 3. Terminology (2) (3) (4) (5) (6) (7) (8)
5227 * 4.1.3. Create a new credential - PublicKeyCredential's
5228 [[Create]](options) method

5135 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
5136 * 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
5137
5138 #concept-user-presentReferenced in:
5139 * 3. Terminology
5140 * 5.1. Authenticator data (2) (3)
5141
5142 #upReferenced in:
5143 * 5.1. Authenticator data
5144
5145 #concept-user-verifiedReferenced in:
5146 * 3. Terminology
5147 * 5.1. Authenticator data (2) (3)
5148
5149 #uvReferenced in:
5150 * 5.1. Authenticator data
5151
5152 #webauthn-clientReferenced in:
5153 * 3. Terminology (2)
5154
5155 #web-authentication-apiReferenced in:
5156 * 1. Introduction (2) (3)
5157 * 3. Terminology (2)
5158
5159 #publickeycredentialReferenced in:
5160 * 1. Introduction
5161 * 4.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
5162 * 4.1.3. Create a new credential - PublicKeyCredential's
5163 [[Create]](options) method (2) (3) (4) (5) (6)

5164 * 4.1.4. Use an existing credential to make an assertion -
5165 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5166 method (2) (3)
5167 * 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
5168 isPlatformAuthenticatorAvailable() method
5169 * 4.7.3. Credential Descriptor (dictionary
5170 PublicKeyCredentialDescriptor)
5171 * 5.2.1. The authenticatorMakeCredential operation
5172 * 6. Relying Party Operations
5173 * 6.2. Verifying an authentication assertion
5174
5175 #dom-publickeycredential-rawidReferenced in:
5176 * 4.1. PublicKeyCredential Interface
5177 * 6.2. Verifying an authentication assertion
5178
5179 #dom-publickeycredential-responseReferenced in:
5180 * 4.1. PublicKeyCredential Interface
5181 * 4.1.3. Create a new credential - PublicKeyCredential's
5182 [[Create]](options) method
5183 * 4.1.4. Use an existing credential to make an assertion -
5184 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5185 method
5186 * 6.2. Verifying an authentication assertion
5187
5188 #dom-publickeycredential-clientextensionresultsReferenced in:
5189 * 4.1. PublicKeyCredential Interface
5190 * 4.1.3. Create a new credential - PublicKeyCredential's
5191 [[Create]](options) method
5192 * 4.1.4. Use an existing credential to make an assertion -
5193 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5194 method
5195 * 8.4. Client extension processing
5196
5197 #dom-publickeycredential-identifier-slotReferenced in:
5198 * 4.1. PublicKeyCredential Interface (2)
5199 * 4.1.3. Create a new credential - PublicKeyCredential's
5200 [[Create]](options) method
5201 * 4.1.4. Use an existing credential to make an assertion -
5202 PublicKeyCredential's [[DiscoverFromExternalSource]](options)

5229 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
5230 * 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
5231
5232 #concept-user-presentReferenced in:
5233 * 3. Terminology
5234 * 5.1. Authenticator data (2) (3)
5235
5236 #upReferenced in:
5237 * 5.1. Authenticator data
5238
5239 #concept-user-verifiedReferenced in:
5240 * 3. Terminology
5241 * 5.1. Authenticator data (2) (3)
5242
5243 #uvReferenced in:
5244 * 5.1. Authenticator data
5245
5246 #webauthn-clientReferenced in:
5247 * 3. Terminology (2)
5248
5249 #web-authentication-apiReferenced in:
5250 * 1. Introduction (2) (3)
5251 * 3. Terminology (2)
5252
5253 #publickeycredentialReferenced in:
5254 * 1. Introduction
5255 * 4.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
5256 * 4.1.3. Create a new credential - PublicKeyCredential's
5257 [[Create]](options) method (2) (3) (4) (5)
5258 * 4.1.3.1. Construct the credential - constructCredentialCallback
5259 method (2)
5260 * 4.1.4. Use an existing credential to make an assertion -
5261 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5262 method (2) (3)
5263 * 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
5264 isPlatformAuthenticatorAvailable() method
5265 * 4.7.3. Credential Descriptor (dictionary
5266 PublicKeyCredentialDescriptor)
5267 * 5.2.1. The authenticatorMakeCredential operation
5268 * 6. Relying Party Operations
5269 * 6.2. Verifying an authentication assertion
5270
5271 #dom-publickeycredential-rawidReferenced in:
5272 * 4.1. PublicKeyCredential Interface
5273 * 6.2. Verifying an authentication assertion
5274
5275 #dom-publickeycredential-responseReferenced in:
5276 * 4.1. PublicKeyCredential Interface
5277 * 4.1.3.1. Construct the credential - constructCredentialCallback
5278 method
5279 * 4.1.4. Use an existing credential to make an assertion -
5280 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5281 method
5282 * 6.2. Verifying an authentication assertion
5283
5284 #dom-publickeycredential-clientextensionresultsReferenced in:
5285 * 4.1. PublicKeyCredential Interface
5286 * 4.1.3.1. Construct the credential - constructCredentialCallback
5287 method
5288 * 4.1.4. Use an existing credential to make an assertion -
5289 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5290 method
5291 * 8.4. Client extension processing
5292
5293 #dom-publickeycredential-identifier-slotReferenced in:
5294 * 4.1. PublicKeyCredential Interface (2)
5295 * 4.1.3.1. Construct the credential - constructCredentialCallback
5296 method
5297 * 4.1.4. Use an existing credential to make an assertion -
5298 PublicKeyCredential's [[DiscoverFromExternalSource]](options)

5203 method
5204
5205 #dom-credentialcreationoptions-publickeyReferenced in:
5206 * 4.1.3. Create a new credential - PublicKeyCredential's
5207 [[Create]](options) method (2) (3)
5208
5209 #dom-credentialrequestoptions-publickeyReferenced in:
5210 * 4.1.4. Use an existing credential to make an assertion -
5211 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5212 method (2) (3)
5213
5214 #dom-publickeycredential-create-slotReferenced in:
5215 * 4.1. PublicKeyCredential Interface
5216
5217 #dom-publickeycredential-create-options-optionsReferenced in:
5218 * 6.1. Registering a new credential
5219
5220
5221 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
5222 * 4.1. PublicKeyCredential Interface
5223
5224 #authenticatorresponseReferenced in:
5225 * 4.1. PublicKeyCredential Interface (2)
5226 * 4.2. Authenticator Responses (interface AuthenticatorResponse) (2)
5227 * 4.2.1. Information about Public Key Credential (interface
5228 AuthenticatorAttestationResponse) (2)
5229 * 4.2.2. Web Authentication Assertion (interface
5230 AuthenticatorAssertionResponse) (2)
5231
5232 #dom-authenticatorresponse-clientdatajsonReferenced in:
5233 * 4.1.3. Create a new credential - PublicKeyCredential's
5234 [[Create]](options) method (2)
5235
5236 * 4.1.4. Use an existing credential to make an assertion -
5237 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5238 method (2)
5239 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
5240 * 4.2.1. Information about Public Key Credential (interface
5241 AuthenticatorAttestationResponse)
5242 * 4.2.2. Web Authentication Assertion (interface
5243 AuthenticatorAssertionResponse)
5244 * 6.1. Registering a new credential (2)
5245 * 6.2. Verifying an authentication assertion
5246
5247 #authenticatorattestationresponseReferenced in:
5248 * 4.1. PublicKeyCredential Interface
5249 * 4.1.3. Create a new credential - PublicKeyCredential's
5250 [[Create]](options) method (2)
5251
5252 * 4.2.1. Information about Public Key Credential (interface
5253 AuthenticatorAttestationResponse) (2)

5299 method
5300
5301 #dom-credentialcreationoptions-publickeyReferenced in:
5302 * 4.1.3. Create a new credential - PublicKeyCredential's
5303 [[Create]](options) method (2) (3)
5304
5305 #dom-credentialrequestoptions-publickeyReferenced in:
5306 * 4.1.4. Use an existing credential to make an assertion -
5307 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5308 method (2) (3)
5309
5310 #dom-publickeycredential-create-slotReferenced in:
5311 * 4.1. PublicKeyCredential Interface
5312
5313 #dom-publickeycredential-create-options-optionsReferenced in:
5314 * 6.1. Registering a new credential
5315
5316 #attestationobjectresultReferenced in:
5317 * 4.1.3.1. Construct the credential - constructCredentialCallback
5318 method
5319
5320 #clientdatajsonresultReferenced in:
5321 * 4.1.3.1. Construct the credential - constructCredentialCallback
5322 method
5323
5324 #extensionoutputsmapReferenced in:
5325 * 4.1.3. Create a new credential - PublicKeyCredential's
5326 [[Create]](options) method
5327 * 4.1.3.1. Construct the credential - constructCredentialCallback
5328 method (2)
5329
5330 #constructcredentialcallbackReferenced in:
5331 * 4.1.3. Create a new credential - PublicKeyCredential's
5332 [[Create]](options) method
5333
5334 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
5335 * 4.1. PublicKeyCredential Interface
5336
5337 #authenticatorresponseReferenced in:
5338 * 4.1. PublicKeyCredential Interface (2)
5339 * 4.2. Authenticator Responses (interface AuthenticatorResponse) (2)
5340 * 4.2.1. Information about Public Key Credential (interface
5341 AuthenticatorAttestationResponse) (2)
5342 * 4.2.2. Web Authentication Assertion (interface
5343 AuthenticatorAssertionResponse) (2)
5344
5345 #dom-authenticatorresponse-clientdatajsonReferenced in:
5346 * 4.1.3. Create a new credential - PublicKeyCredential's
5347 [[Create]](options) method
5348 * 4.1.3.1. Construct the credential - constructCredentialCallback
5349 method
5350 * 4.1.4. Use an existing credential to make an assertion -
5351 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5352 method (2)
5353 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
5354 * 4.2.1. Information about Public Key Credential (interface
5355 AuthenticatorAttestationResponse)
5356 * 4.2.2. Web Authentication Assertion (interface
5357 AuthenticatorAssertionResponse)
5358 * 6.1. Registering a new credential (2)
5359 * 6.2. Verifying an authentication assertion
5360
5361 #authenticatorattestationresponseReferenced in:
5362 * 4.1. PublicKeyCredential Interface
5363 * 4.1.3. Create a new credential - PublicKeyCredential's
5364 [[Create]](options) method
5365 * 4.1.3.1. Construct the credential - constructCredentialCallback
5366 method
5367 * 4.2.1. Information about Public Key Credential (interface
5368 AuthenticatorAttestationResponse) (2)

5251 * 6. Relying Party Operations
5252 * 6.1. Registering a new credential (2) (3)
5253
5254 #dom-authenticatorattestationresponse-attestationobjectReferenced in:
5255 * 4.1.3. Create a new credential - PublicKeyCredential's
5256 [[Create]](options) method
5257 * 4.2.1. Information about Public Key Credential (interface
5258 AuthenticatorAttestationResponse)
5259 * 6.1. Registering a new credential
5260
5261 #authenticatorassertionresponseReferenced in:
5262 * 3. Terminology
5263 * 4.1. PublicKeyCredential Interface
5264 * 4.1.4. Use an existing credential to make an assertion -
5265 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5266 method
5267 * 4.2.2. Web Authentication Assertion (interface
5268 AuthenticatorAssertionResponse) (2)
5269 * 6. Relying Party Operations
5270
5271 #dom-authenticatorassertionresponse-authenticatordataReferenced in:
5272 * 4.1.4. Use an existing credential to make an assertion -
5273 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5274 method (2)
5275 * 4.2.2. Web Authentication Assertion (interface
5276 AuthenticatorAssertionResponse)
5277 * 6.2. Verifying an authentication assertion
5278
5279 #dom-authenticatorassertionresponse-signatureReferenced in:
5280 * 4.1.4. Use an existing credential to make an assertion -
5281 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5282 method (2)
5283 * 4.2.2. Web Authentication Assertion (interface
5284 AuthenticatorAssertionResponse)
5285 * 6.2. Verifying an authentication assertion
5286
5287 #dictdef-publickeycredentialparametersReferenced in:
5288 * 4.3. Parameters for Credential Generation (dictionary
5289 PublicKeyCredentialParameters)
5290 * 4.4. Options for Credential Creation (dictionary
5291 MakePublicKeyCredentialOptions) (2)
5292
5293 #dom-publickeycredentialparameters-typeReferenced in:
5294 * 4.1.3. Create a new credential - PublicKeyCredential's
5295 [[Create]](options) method (2)
5296 * 4.3. Parameters for Credential Generation (dictionary
5297 PublicKeyCredentialParameters)
5298
5299 #dom-publickeycredentialparameters-algReferenced in:
5300 * 4.1.3. Create a new credential - PublicKeyCredential's
5301 [[Create]](options) method
5302 * 4.3. Parameters for Credential Generation (dictionary
5303 PublicKeyCredentialParameters)
5304
5305 #dictdef-makepublickeycredentialoptionsReferenced in:
5306 * 4.1.1. CredentialCreationOptions Extension
5307 * 4.1.3. Create a new credential - PublicKeyCredential's
5308 [[Create]](options) method
5309 * 4.4. Options for Credential Creation (dictionary
5310 MakePublicKeyCredentialOptions)
5311
5312 #dom-makepublickeycredentialoptions-rpReferenced in:
5313 * 4.1.3. Create a new credential - PublicKeyCredential's
5314 [[Create]](options) method (2) (3) (4) (5) (6)
5315 * 4.4. Options for Credential Creation (dictionary
5316 MakePublicKeyCredentialOptions)
5317
5318 #dom-makepublickeycredentialoptions-userReferenced in:
5319 * 4.1.3. Create a new credential - PublicKeyCredential's
5320 [[Create]](options) method (2) (3) (4)

5369 * 6. Relying Party Operations
5370 * 6.1. Registering a new credential (2) (3)
5371
5372 #dom-authenticatorattestationresponse-attestationobjectReferenced in:
5373 * 4.1.3.1. Construct the credential - constructCredentialCallback
5374 method
5375 * 4.2.1. Information about Public Key Credential (interface
5376 AuthenticatorAttestationResponse)
5377 * 6.1. Registering a new credential
5378
5379 #authenticatorassertionresponseReferenced in:
5380 * 3. Terminology
5381 * 4.1. PublicKeyCredential Interface
5382 * 4.1.4. Use an existing credential to make an assertion -
5383 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5384 method
5385 * 4.2.2. Web Authentication Assertion (interface
5386 AuthenticatorAssertionResponse) (2)
5387 * 6. Relying Party Operations
5388
5389 #dom-authenticatorassertionresponse-authenticatordataReferenced in:
5390 * 4.1.4. Use an existing credential to make an assertion -
5391 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5392 method (2)
5393 * 4.2.2. Web Authentication Assertion (interface
5394 AuthenticatorAssertionResponse)
5395 * 6.2. Verifying an authentication assertion
5396
5397 #dom-authenticatorassertionresponse-signatureReferenced in:
5398 * 4.1.4. Use an existing credential to make an assertion -
5399 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5400 method (2)
5401 * 4.2.2. Web Authentication Assertion (interface
5402 AuthenticatorAssertionResponse)
5403 * 6.2. Verifying an authentication assertion
5404
5405 #dictdef-publickeycredentialparametersReferenced in:
5406 * 4.3. Parameters for Credential Generation (dictionary
5407 PublicKeyCredentialParameters)
5408 * 4.4. Options for Credential Creation (dictionary
5409 MakePublicKeyCredentialOptions) (2)
5410
5411 #dom-publickeycredentialparameters-typeReferenced in:
5412 * 4.1.3. Create a new credential - PublicKeyCredential's
5413 [[Create]](options) method (2)
5414 * 4.3. Parameters for Credential Generation (dictionary
5415 PublicKeyCredentialParameters)
5416
5417 #dom-publickeycredentialparameters-algReferenced in:
5418 * 4.1.3. Create a new credential - PublicKeyCredential's
5419 [[Create]](options) method
5420 * 4.3. Parameters for Credential Generation (dictionary
5421 PublicKeyCredentialParameters)
5422
5423 #dictdef-makepublickeycredentialoptionsReferenced in:
5424 * 4.1.1. CredentialCreationOptions Extension
5425 * 4.1.3. Create a new credential - PublicKeyCredential's
5426 [[Create]](options) method
5427 * 4.4. Options for Credential Creation (dictionary
5428 MakePublicKeyCredentialOptions)
5429
5430 #dom-makepublickeycredentialoptions-rpReferenced in:
5431 * 4.1.3. Create a new credential - PublicKeyCredential's
5432 [[Create]](options) method (2) (3) (4) (5) (6) (7)
5433 * 4.4. Options for Credential Creation (dictionary
5434 MakePublicKeyCredentialOptions)
5435
5436 #dom-makepublickeycredentialoptions-userReferenced in:
5437 * 4.1.3. Create a new credential - PublicKeyCredential's
5438 [[Create]](options) method (2) (3) (4)

5321 * 4.4. Options for Credential Creation (dictionary
5322 MakePublicKeyCredentialOptions)
5323 * 5.2.1. The authenticatorMakeCredential operation (2)
5324 * 6.1. Registering a new credential
5325
5326 #dom-makepublickeycredentialoptions-challengeReferenced in:
5327 * 4.1.3. Create a new credential - PublicKeyCredential's
5328 [[Create]](options) method
5329 * 4.4. Options for Credential Creation (dictionary
5330 MakePublicKeyCredentialOptions)
5331
5332 #dom-makepublickeycredentialoptions-pubkeycredparamsReferenced in:
5333 * 4.1.3. Create a new credential - PublicKeyCredential's
5334 [[Create]](options) method (2)
5335 * 4.4. Options for Credential Creation (dictionary
5336 MakePublicKeyCredentialOptions)
5337
5338 #dom-makepublickeycredentialoptions-timeoutReferenced in:
5339 * 4.1.3. Create a new credential - PublicKeyCredential's
5340 [[Create]](options) method (2)
5341 * 4.4. Options for Credential Creation (dictionary
5342 MakePublicKeyCredentialOptions)
5343
5344 #dom-makepublickeycredentialoptions-excludecredentialsReferenced in:
5345 * 4.1.3. Create a new credential - PublicKeyCredential's
5346 [[Create]](options) method
5347 * 4.4. Options for Credential Creation (dictionary
5348 MakePublicKeyCredentialOptions)
5349
5350 #dom-makepublickeycredentialoptions-authenticatorselectionReferenced
5351 in:
5352 * 4.1.3. Create a new credential - PublicKeyCredential's
5353 [[Create]](options) method (2)
5354 * 4.4. Options for Credential Creation (dictionary
5355 MakePublicKeyCredentialOptions)
5356 * 5.2.1. The authenticatorMakeCredential operation (2)
5357
5358 #dom-makepublickeycredentialoptions-extensionsReferenced in:
5359 * 4.1.3. Create a new credential - PublicKeyCredential's
5360 [[Create]](options) method (2)
5361 * 4.4. Options for Credential Creation (dictionary
5362 MakePublicKeyCredentialOptions)
5363 * 8.3. Extending request parameters
5364
5365 #dictdef-publickeycredentialentityReferenced in:
5366 * 4.4.1. Public Key Entity Description (dictionary
5367 PublicKeyCredentialEntity) (2)
5368 * 4.4.2. RP Parameters for Credential Generation (dictionary
5369 PublicKeyCredentialRpEntity)
5370 * 4.4.3. User Account Parameters for Credential Generation
5371 (dictionary PublicKeyCredentialUserEntity)
5372
5373 #dom-publickeycredentialentity-nameReferenced in:
5374 * 4.1.3. Create a new credential - PublicKeyCredential's
5375 [[Create]](options) method (2)
5376 * 4.4. Options for Credential Creation (dictionary
5377 MakePublicKeyCredentialOptions) (2)
5378 * 4.4.1. Public Key Entity Description (dictionary
5379 PublicKeyCredentialEntity)
5380
5381 #dom-publickeycredentialentity-iconReferenced in:
5382 * 4.4.1. Public Key Entity Description (dictionary
5383 PublicKeyCredentialEntity)
5384
5385 #dictdef-publickeycredentialrpentityReferenced in:
5386 * 4.4. Options for Credential Creation (dictionary
5387 MakePublicKeyCredentialOptions) (2)
5388 * 4.4.2. RP Parameters for Credential Generation (dictionary
5389 PublicKeyCredentialRpEntity) (2)

5439 * 4.4. Options for Credential Creation (dictionary
5440 MakePublicKeyCredentialOptions)
5441 * 5.2.1. The authenticatorMakeCredential operation (2)
5442 * 6.1. Registering a new credential
5443
5444 #dom-makepublickeycredentialoptions-challengeReferenced in:
5445 * 4.1.3. Create a new credential - PublicKeyCredential's
5446 [[Create]](options) method
5447 * 4.4. Options for Credential Creation (dictionary
5448 MakePublicKeyCredentialOptions)
5449
5450 #dom-makepublickeycredentialoptions-pubkeycredparamsReferenced in:
5451 * 4.1.3. Create a new credential - PublicKeyCredential's
5452 [[Create]](options) method (2)
5453 * 4.4. Options for Credential Creation (dictionary
5454 MakePublicKeyCredentialOptions)
5455
5456 #dom-makepublickeycredentialoptions-timeoutReferenced in:
5457 * 4.1.3. Create a new credential - PublicKeyCredential's
5458 [[Create]](options) method (2)
5459 * 4.4. Options for Credential Creation (dictionary
5460 MakePublicKeyCredentialOptions)
5461
5462 #dom-makepublickeycredentialoptions-excludecredentialsReferenced in:
5463 * 4.1.3. Create a new credential - PublicKeyCredential's
5464 [[Create]](options) method
5465 * 4.4. Options for Credential Creation (dictionary
5466 MakePublicKeyCredentialOptions)
5467
5468 #dom-makepublickeycredentialoptions-authenticatorselectionReferenced
5469 in:
5470 * 4.1.3. Create a new credential - PublicKeyCredential's
5471 [[Create]](options) method (2)
5472 * 4.4. Options for Credential Creation (dictionary
5473 MakePublicKeyCredentialOptions)
5474 * 5.2.1. The authenticatorMakeCredential operation (2)
5475
5476 #dom-makepublickeycredentialoptions-extensionsReferenced in:
5477 * 4.1.3. Create a new credential - PublicKeyCredential's
5478 [[Create]](options) method (2)
5479 * 4.4. Options for Credential Creation (dictionary
5480 MakePublicKeyCredentialOptions)
5481 * 8.3. Extending request parameters
5482
5483 #dictdef-publickeycredentialentityReferenced in:
5484 * 4.4.1. Public Key Entity Description (dictionary
5485 PublicKeyCredentialEntity) (2)
5486 * 4.4.2. RP Parameters for Credential Generation (dictionary
5487 PublicKeyCredentialRpEntity)
5488 * 4.4.3. User Account Parameters for Credential Generation
5489 (dictionary PublicKeyCredentialUserEntity)
5490 * 5.2.1. The authenticatorMakeCredential operation
5491
5492 #dom-publickeycredentialentity-nameReferenced in:
5493 * 4.1.3. Create a new credential - PublicKeyCredential's
5494 [[Create]](options) method (2)
5495 * 4.4. Options for Credential Creation (dictionary
5496 MakePublicKeyCredentialOptions) (2)
5497 * 4.4.1. Public Key Entity Description (dictionary
5498 PublicKeyCredentialEntity)
5499
5500 #dom-publickeycredentialentity-iconReferenced in:
5501 * 4.4.1. Public Key Entity Description (dictionary
5502 PublicKeyCredentialEntity)
5503
5504 #dictdef-publickeycredentialrpentityReferenced in:
5505 * 4.4. Options for Credential Creation (dictionary
5506 MakePublicKeyCredentialOptions) (2)
5507 * 4.4.2. RP Parameters for Credential Generation (dictionary
5508 PublicKeyCredentialRpEntity) (2)

5390 * 5.2.1. The authenticatorMakeCredential operation
5391
5392 #dom-publickeycredentialrpentity-idReferenced in:
5393 * 4.1.3. Create a new credential - PublicKeyCredential's
5394 [[Create]](options) method (2) (3) (4)
5395 * 4.4. Options for Credential Creation (dictionary
5396 MakePublicKeyCredentialOptions)
5397 * 4.4.2. RP Parameters for Credential Generation (dictionary
5398 PublicKeyCredentialRpEntity)
5399
5400 #dictdef-publickeycredentialuserentityReferenced in:
5401 * 4.4. Options for Credential Creation (dictionary
5402 MakePublicKeyCredentialOptions) (2)
5403 * 4.4.3. User Account Parameters for Credential Generation
5404 (dictionary PublicKeyCredentialUserEntity) (2)
5405 * 5.2.1. The authenticatorMakeCredential operation
5406
5407 #dom-publickeycredentialuserentity-idReferenced in:
5408 * 4.1.3. Create a new credential - PublicKeyCredential's
5409 [[Create]](options) method
5410 * 4.4. Options for Credential Creation (dictionary
5411 MakePublicKeyCredentialOptions) (2)
5412 * 4.4.3. User Account Parameters for Credential Generation
5413 (dictionary PublicKeyCredentialUserEntity)
5414 * 5.2.1. The authenticatorMakeCredential operation (2)
5415
5416 #dom-publickeycredentialuserentity-displaynameReferenced in:
5417 * 4.1.3. Create a new credential - PublicKeyCredential's
5418 [[Create]](options) method
5419 * 4.4. Options for Credential Creation (dictionary
5420 MakePublicKeyCredentialOptions)
5421 * 4.4.3. User Account Parameters for Credential Generation
5422 (dictionary PublicKeyCredentialUserEntity)
5423
5424 #dictdef-authenticatorselectioncriteriaReferenced in:
5425 * 4.4. Options for Credential Creation (dictionary
5426 MakePublicKeyCredentialOptions) (2)
5427 * 4.4.4. Authenticator Selection Criteria (dictionary
5428 AuthenticatorSelectionCriteria) (2)
5429
5430 #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced
5431 in:
5432 * 4.1.3. Create a new credential - PublicKeyCredential's
5433 [[Create]](options) method
5434 * 4.4.4. Authenticator Selection Criteria (dictionary
5435 AuthenticatorSelectionCriteria)
5436
5437 #dom-authenticatorselectioncriteria-requireresidentkeyReferenced in:
5438 * 4.1.3. Create a new credential - PublicKeyCredential's
5439 [[Create]](options) method (2)
5440 * 4.4.4. Authenticator Selection Criteria (dictionary
5441 AuthenticatorSelectionCriteria)
5442
5443 #dom-authenticatorselectioncriteria-requireuserverificationReferenced
5444 in:
5445 * 4.1.3. Create a new credential - PublicKeyCredential's
5446 [[Create]](options) method
5447 * 4.4.4. Authenticator Selection Criteria (dictionary
5448 AuthenticatorSelectionCriteria)
5449
5450 #enumdef-authenticatorattachmentReferenced in:
5451 * 4.4.4. Authenticator Selection Criteria (dictionary
5452 AuthenticatorSelectionCriteria) (2)
5453 * 4.4.5. Authenticator Attachment enumeration (enum
5454 AuthenticatorAttachment) (2)
5455
5456 #platform-authenticatorsReferenced in:
5457 * 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
5458 isPlatformAuthenticatorAvailable() method (2) (3) (4) (5)
5459 * 4.4.5. Authenticator Attachment enumeration (enum

5509 #dom-publickeycredentialrpentity-idReferenced in:
5510
5511 * 4.4. Options for Credential Creation (dictionary
5512 MakePublicKeyCredentialOptions)
5513 * 4.4.2. RP Parameters for Credential Generation (dictionary
5514 PublicKeyCredentialRpEntity)
5515
5516 #dictdef-publickeycredentialuserentityReferenced in:
5517 * 4.4. Options for Credential Creation (dictionary
5518 MakePublicKeyCredentialOptions) (2)
5519 * 4.4.3. User Account Parameters for Credential Generation
5520 (dictionary PublicKeyCredentialUserEntity) (2)
5521 * 5.2.1. The authenticatorMakeCredential operation
5522
5523 #dom-publickeycredentialuserentity-idReferenced in:
5524
5525 * 4.4. Options for Credential Creation (dictionary
5526 MakePublicKeyCredentialOptions) (2)
5527 * 4.4.3. User Account Parameters for Credential Generation
5528 (dictionary PublicKeyCredentialUserEntity)
5529 * 5.2.1. The authenticatorMakeCredential operation (2)
5530
5531 #dom-publickeycredentialuserentity-displaynameReferenced in:
5532 * 4.1.3. Create a new credential - PublicKeyCredential's
5533 [[Create]](options) method
5534 * 4.4. Options for Credential Creation (dictionary
5535 MakePublicKeyCredentialOptions)
5536 * 4.4.3. User Account Parameters for Credential Generation
5537 (dictionary PublicKeyCredentialUserEntity)
5538
5539 #dictdef-authenticatorselectioncriteriaReferenced in:
5540 * 4.4. Options for Credential Creation (dictionary
5541 MakePublicKeyCredentialOptions) (2)
5542 * 4.4.4. Authenticator Selection Criteria (dictionary
5543 AuthenticatorSelectionCriteria) (2)
5544
5545 #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced
5546 in:
5547 * 4.1.3. Create a new credential - PublicKeyCredential's
5548 [[Create]](options) method
5549 * 4.4.4. Authenticator Selection Criteria (dictionary
5550 AuthenticatorSelectionCriteria)
5551
5552 #dom-authenticatorselectioncriteria-requireresidentkeyReferenced in:
5553 * 4.1.3. Create a new credential - PublicKeyCredential's
5554 [[Create]](options) method
5555 * 4.4.4. Authenticator Selection Criteria (dictionary
5556 AuthenticatorSelectionCriteria)
5557
5558 #dom-authenticatorselectioncriteria-requireuserverificationReferenced
5559 in:
5560 * 4.1.3. Create a new credential - PublicKeyCredential's
5561 [[Create]](options) method
5562 * 4.4.4. Authenticator Selection Criteria (dictionary
5563 AuthenticatorSelectionCriteria)
5564
5565 #enumdef-authenticatorattachmentReferenced in:
5566 * 4.4.4. Authenticator Selection Criteria (dictionary
5567 AuthenticatorSelectionCriteria) (2)
5568 * 4.4.5. Authenticator Attachment enumeration (enum
5569 AuthenticatorAttachment) (2)
5570
5571 #platform-authenticatorsReferenced in:
5572 * 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
5573 isPlatformAuthenticatorAvailable() method (2) (3) (4) (5)
5574 * 4.4.5. Authenticator Attachment enumeration (enum

5460 AuthenticatorAttachment) (2)
5461 * 11.1. Registration
5462 * 11.2. Registration Specifically with Platform Authenticator (2)
5463
5464 #roaming-authenticatorsReferenced in:
5465 * 1.1.3. Other use cases and configurations
5466 * 4.4.5. Authenticator Attachment enumeration (enum
5467 AuthenticatorAttachment) (2)
5468 * 11.1. Registration
5469
5470 #platform-attachmentReferenced in:
5471 * 4.4.5. Authenticator Attachment enumeration (enum
5472 AuthenticatorAttachment)
5473
5474 #cross-platform-attachedReferenced in:
5475 * 4.4.5. Authenticator Attachment enumeration (enum
5476 AuthenticatorAttachment) (2)
5477
5478 #dictdef-publickeycredentialrequestoptionsReferenced in:
5479 * 4.1.2. CredentialRequestOptions Extension
5480 * 4.5. Options for Assertion Generation (dictionary
5481 PublicKeyCredentialRequestOptions) (2)
5482 * 6.2. Verifying an authentication assertion
5483
5484 #dom-publickeycredentialrequestoptions-challengeReferenced in:
5485 * 4.1.4. Use an existing credential to make an assertion -
5486 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5487 method
5488 * 4.5. Options for Assertion Generation (dictionary
5489 PublicKeyCredentialRequestOptions) (2)
5490
5491 #dom-publickeycredentialrequestoptions-timeoutReferenced in:
5492 * 4.1.4. Use an existing credential to make an assertion -
5493 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5494 method (2)
5495 * 4.5. Options for Assertion Generation (dictionary
5496 PublicKeyCredentialRequestOptions)
5497
5498 #dom-publickeycredentialrequestoptions-rpidReferenced in:
5499 * 4.1.4. Use an existing credential to make an assertion -
5500 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5501 method (2) (3) (4)
5502 * 4.5. Options for Assertion Generation (dictionary
5503 PublicKeyCredentialRequestOptions)
5504 * 9.1. FIDO Appld Extension (appid)
5505
5506 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:
5507 * 4.1.4. Use an existing credential to make an assertion -
5508 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5509 method (2) (3) (4)
5510 * 4.5. Options for Assertion Generation (dictionary
5511 PublicKeyCredentialRequestOptions)
5512
5513 #dom-publickeycredentialrequestoptions-extensionsReferenced in:
5514 * 4.1.4. Use an existing credential to make an assertion -
5515 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5516 method (2)
5517 * 4.5. Options for Assertion Generation (dictionary
5518 PublicKeyCredentialRequestOptions)
5519
5520 #typedefdef-authenticationextensionsReferenced in:
5521 * 4.1. PublicKeyCredential Interface (2)
5522 * 4.1.3. Create a new credential - PublicKeyCredential's
5523 [[Create]](options) method
5524 * 4.1.4. Use an existing credential to make an assertion -
5525 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5526 method
5527 * 4.4. Options for Credential Creation (dictionary
5528 MakePublicKeyCredentialOptions) (2)
5529 * 4.5. Options for Assertion Generation (dictionary

5574 AuthenticatorAttachment) (2)
5575 * 11.1. Registration
5576 * 11.2. Registration Specifically with Platform Authenticator (2)
5577
5578 #roaming-authenticatorsReferenced in:
5579 * 1.1.3. Other use cases and configurations
5580 * 4.4.5. Authenticator Attachment enumeration (enum
5581 AuthenticatorAttachment) (2)
5582 * 11.1. Registration
5583
5584 #platform-attachmentReferenced in:
5585 * 4.4.5. Authenticator Attachment enumeration (enum
5586 AuthenticatorAttachment)
5587
5588 #cross-platform-attachedReferenced in:
5589 * 4.4.5. Authenticator Attachment enumeration (enum
5590 AuthenticatorAttachment) (2)
5591
5592 #dictdef-publickeycredentialrequestoptionsReferenced in:
5593 * 4.1.2. CredentialRequestOptions Extension
5594 * 4.5. Options for Assertion Generation (dictionary
5595 PublicKeyCredentialRequestOptions) (2)
5596 * 6.2. Verifying an authentication assertion
5597
5598 #dom-publickeycredentialrequestoptions-challengeReferenced in:
5599 * 4.1.4. Use an existing credential to make an assertion -
5600 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5601 method
5602 * 4.5. Options for Assertion Generation (dictionary
5603 PublicKeyCredentialRequestOptions) (2)
5604
5605 #dom-publickeycredentialrequestoptions-timeoutReferenced in:
5606 * 4.1.4. Use an existing credential to make an assertion -
5607 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5608 method (2)
5609 * 4.5. Options for Assertion Generation (dictionary
5610 PublicKeyCredentialRequestOptions)
5611
5612 #dom-publickeycredentialrequestoptions-rpidReferenced in:
5613 * 4.1.4. Use an existing credential to make an assertion -
5614 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5615 method (2) (3) (4)
5616 * 4.5. Options for Assertion Generation (dictionary
5617 PublicKeyCredentialRequestOptions)
5618 * 9.1. FIDO Appld Extension (appid)
5619
5620 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:
5621 * 4.1.4. Use an existing credential to make an assertion -
5622 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5623 method (2) (3) (4)
5624 * 4.5. Options for Assertion Generation (dictionary
5625 PublicKeyCredentialRequestOptions)
5626
5627 #dom-publickeycredentialrequestoptions-extensionsReferenced in:
5628 * 4.1.4. Use an existing credential to make an assertion -
5629 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5630 method (2)
5631 * 4.5. Options for Assertion Generation (dictionary
5632 PublicKeyCredentialRequestOptions)
5633
5634 #typedefdef-authenticationextensionsReferenced in:
5635 * 4.1. PublicKeyCredential Interface (2)
5636 * 4.1.3.1. Construct the credential - constructCredentialCallback
5637 method (2)
5638 * 4.1.4. Use an existing credential to make an assertion -
5639 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5640 method
5641 * 4.4. Options for Credential Creation (dictionary
5642 MakePublicKeyCredentialOptions) (2)
5643 * 4.5. Options for Assertion Generation (dictionary

5530 PublicKeyCredentialRequestOptions) (2)
5531 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5532 CollectedClientData) (2)
5533
5534 #dictdef-collectedclientdataReferenced in:
5535 * 4.1.3. Create a new credential - PublicKeyCredential's
5536 [[Create]](options) method
5537 * 4.1.4. Use an existing credential to make an assertion -
5538 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5539 method
5540 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5541 CollectedClientData) (2)
5542
5543 #client-dataReferenced in:
5544 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
5545 * 5. WebAuthn Authenticator model (2) (3) (4)
5546 * 5.1. Authenticator data (2)
5547 * 6.1. Registering a new credential
5548 * 6.2. Verifying an authentication assertion
5549 * 8. WebAuthn Extensions
5550 * 8.4. Client extension processing
5551 * 8.6. Example Extension
5552
5553 #dom-collectedclientdata-challengeReferenced in:
5554 * 4.1.3. Create a new credential - PublicKeyCredential's
5555 [[Create]](options) method
5556 * 4.1.4. Use an existing credential to make an assertion -
5557 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5558 method
5559 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5560 CollectedClientData)
5561 * 6.1. Registering a new credential
5562 * 6.2. Verifying an authentication assertion
5563
5564 #dom-collectedclientdata-originReferenced in:
5565 * 4.1.3. Create a new credential - PublicKeyCredential's
5566 [[Create]](options) method
5567 * 4.1.4. Use an existing credential to make an assertion -
5568 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5569 method
5570 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5571 CollectedClientData)
5572 * 6.1. Registering a new credential
5573 * 6.2. Verifying an authentication assertion
5574
5575 #dom-collectedclientdata-hashalgorithmReferenced in:
5576 * 4.1.3. Create a new credential - PublicKeyCredential's
5577 [[Create]](options) method
5578 * 4.1.4. Use an existing credential to make an assertion -
5579 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5580 method
5581 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5582 CollectedClientData) (2)
5583 * 6.1. Registering a new credential
5584 * 6.2. Verifying an authentication assertion
5585
5586 #dom-collectedclientdata-tokenbindingidReferenced in:
5587 * 4.1.3. Create a new credential - PublicKeyCredential's
5588 [[Create]](options) method
5589 * 4.1.4. Use an existing credential to make an assertion -
5590 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5591 method
5592 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5593 CollectedClientData)
5594 * 6.1. Registering a new credential
5595 * 6.2. Verifying an authentication assertion
5596
5597 #dom-collectedclientdata-clientextensionsReferenced in:
5598 * 4.1.3. Create a new credential - PublicKeyCredential's
5599 [[Create]](options) method

5644 PublicKeyCredentialRequestOptions) (2)
5645 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5646 CollectedClientData) (2)
5647
5648 #dictdef-collectedclientdataReferenced in:
5649 * 4.1.3. Create a new credential - PublicKeyCredential's
5650 [[Create]](options) method
5651 * 4.1.4. Use an existing credential to make an assertion -
5652 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5653 method
5654 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5655 CollectedClientData) (2)
5656
5657 #client-dataReferenced in:
5658 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
5659 * 5. WebAuthn Authenticator model (2) (3) (4)
5660 * 5.1. Authenticator data (2)
5661 * 6.1. Registering a new credential
5662 * 6.2. Verifying an authentication assertion
5663 * 8. WebAuthn Extensions
5664 * 8.4. Client extension processing
5665 * 8.6. Example Extension
5666
5667 #dom-collectedclientdata-challengeReferenced in:
5668 * 4.1.3. Create a new credential - PublicKeyCredential's
5669 [[Create]](options) method
5670 * 4.1.4. Use an existing credential to make an assertion -
5671 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5672 method
5673 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5674 CollectedClientData)
5675 * 6.1. Registering a new credential
5676 * 6.2. Verifying an authentication assertion
5677
5678 #dom-collectedclientdata-originReferenced in:
5679 * 4.1.3. Create a new credential - PublicKeyCredential's
5680 [[Create]](options) method
5681 * 4.1.4. Use an existing credential to make an assertion -
5682 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5683 method
5684 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5685 CollectedClientData)
5686 * 6.1. Registering a new credential
5687 * 6.2. Verifying an authentication assertion
5688
5689 #dom-collectedclientdata-hashalgorithmReferenced in:
5690 * 4.1.3. Create a new credential - PublicKeyCredential's
5691 [[Create]](options) method
5692 * 4.1.4. Use an existing credential to make an assertion -
5693 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5694 method
5695 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5696 CollectedClientData) (2)
5697 * 6.1. Registering a new credential
5698 * 6.2. Verifying an authentication assertion
5699
5700 #dom-collectedclientdata-tokenbindingidReferenced in:
5701 * 4.1.3. Create a new credential - PublicKeyCredential's
5702 [[Create]](options) method
5703 * 4.1.4. Use an existing credential to make an assertion -
5704 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5705 method
5706 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5707 CollectedClientData)
5708 * 6.1. Registering a new credential
5709 * 6.2. Verifying an authentication assertion
5710
5711 #dom-collectedclientdata-clientextensionsReferenced in:
5712 * 4.1.3. Create a new credential - PublicKeyCredential's
5713 [[Create]](options) method

5600 * 4.1.4. Use an existing credential to make an assertion -
5601 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5602 method
5603 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5604 CollectedClientData)
5605 * 6.1. Registering a new credential
5606 * 6.2. Verifying an authentication assertion
5607 * 8.4. Client extension processing
5608
5609 #dom-collectedclientdata-authenticatorextensionsReferenced in:
5610 * 4.1.3. Create a new credential - PublicKeyCredential's
5611 [[Create]](options) method
5612 * 4.1.4. Use an existing credential to make an assertion -
5613 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5614 method
5615 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5616 CollectedClientData)
5617 * 6.1. Registering a new credential
5618 * 6.2. Verifying an authentication assertion
5619
5620 #collectedclientdata-json-serialized-client-dataReferenced in:
5621 * 4.1.3. Create a new credential - PublicKeyCredential's
5622 [[Create]](options) method
5623 * 4.1.4. Use an existing credential to make an assertion -
5624 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5625 method
5626 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
5627 * 4.2.1. Information about Public Key Credential (interface
5628 AuthenticatorAttestationResponse) (2)
5629 * 4.2.2. Web Authentication Assertion (interface
5630 AuthenticatorAssertionResponse)
5631 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5632 CollectedClientData)
5633
5634 #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:
5635 * 4.1.3. Create a new credential - PublicKeyCredential's
5636 [[Create]](options) method (2)
5637 * 4.1.4. Use an existing credential to make an assertion -
5638 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5639 method (2)
5640 * 4.2.1. Information about Public Key Credential (interface
5641 AuthenticatorAttestationResponse)
5642 * 4.2.2. Web Authentication Assertion (interface
5643 AuthenticatorAssertionResponse)
5644 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5645 CollectedClientData)
5646 * 5. WebAuthn Authenticator model
5647 * 5.2.1. The authenticatorMakeCredential operation (2)
5648 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
5649 * 5.3.2. Attestation Statement Formats (2)
5650 * 5.3.4. Generating an Attestation Object
5651 * 6.1. Registering a new credential
5652 * 7.2. Packed Attestation Statement Format (2)
5653 * 7.3. TPM Attestation Statement Format (2)
5654 * 7.4. Android Key Attestation Statement Format (2)
5655 * 7.5. Android SafetyNet Attestation Statement Format
5656 * 7.6. FIDO U2F Attestation Statement Format (2)
5657
5658 #enumdef-publickeycredentialtypeReferenced in:
5659 * 4.1.3. Create a new credential - PublicKeyCredential's
5660 [[Create]](options) method (2)
5661 * 4.3. Parameters for Credential Generation (dictionary
5662 PublicKeyCredentialParameters)
5663 * 4.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
5664 * 4.7.3. Credential Descriptor (dictionary
5665 PublicKeyCredentialDescriptor)
5666 * 5.2.1. The authenticatorMakeCredential operation (2) (3)
5667
5668 #dom-publickeycredentialtype-public-keyReferenced in:
5669 * 4.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
5670

5714 * 4.1.4. Use an existing credential to make an assertion -
5715 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5716 method
5717 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5718 CollectedClientData)
5719 * 6.1. Registering a new credential
5720 * 6.2. Verifying an authentication assertion
5721 * 8.4. Client extension processing
5722
5723 #dom-collectedclientdata-authenticatorextensionsReferenced in:
5724 * 4.1.3. Create a new credential - PublicKeyCredential's
5725 [[Create]](options) method
5726 * 4.1.4. Use an existing credential to make an assertion -
5727 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5728 method
5729 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5730 CollectedClientData)
5731 * 6.1. Registering a new credential
5732 * 6.2. Verifying an authentication assertion
5733
5734 #collectedclientdata-json-serialized-client-dataReferenced in:
5735 * 4.1.3. Create a new credential - PublicKeyCredential's
5736 [[Create]](options) method
5737 * 4.1.4. Use an existing credential to make an assertion -
5738 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5739 method
5740 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
5741 * 4.2.1. Information about Public Key Credential (interface
5742 AuthenticatorAttestationResponse) (2)
5743 * 4.2.2. Web Authentication Assertion (interface
5744 AuthenticatorAssertionResponse)
5745 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5746 CollectedClientData)
5747
5748 #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:
5749 * 4.1.3. Create a new credential - PublicKeyCredential's
5750 [[Create]](options) method (2)
5751 * 4.1.4. Use an existing credential to make an assertion -
5752 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5753 method (2)
5754 * 4.2.1. Information about Public Key Credential (interface
5755 AuthenticatorAttestationResponse)
5756 * 4.2.2. Web Authentication Assertion (interface
5757 AuthenticatorAssertionResponse)
5758 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5759 CollectedClientData)
5760 * 5. WebAuthn Authenticator model
5761 * 5.2.1. The authenticatorMakeCredential operation (2)
5762 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
5763 * 5.3.2. Attestation Statement Formats (2)
5764 * 5.3.4. Generating an Attestation Object
5765 * 6.1. Registering a new credential
5766 * 7.2. Packed Attestation Statement Format (2)
5767 * 7.3. TPM Attestation Statement Format (2)
5768 * 7.4. Android Key Attestation Statement Format (2)
5769 * 7.5. Android SafetyNet Attestation Statement Format
5770 * 7.6. FIDO U2F Attestation Statement Format (2)
5771
5772 #enumdef-publickeycredentialtypeReferenced in:
5773 * 4.1.3. Create a new credential - PublicKeyCredential's
5774 [[Create]](options) method (2)
5775 * 4.3. Parameters for Credential Generation (dictionary
5776 PublicKeyCredentialParameters)
5777 * 4.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
5778 * 4.7.3. Credential Descriptor (dictionary
5779 PublicKeyCredentialDescriptor)
5780 * 5.2.1. The authenticatorMakeCredential operation (2) (3)
5781
5782 #dom-publickeycredentialtype-public-keyReferenced in:
5783 * 4.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
5784

```

5670 #dictdef-publickeycredentialdescriptorReferenced in:
5671 * 4.4. Options for Credential Creation (dictionary
5672 MakePublicKeyCredentialOptions) (2)
5673 * 4.5. Options for Assertion Generation (dictionary
5674 PublicKeyCredentialRequestOptions) (2) (3)
5675 * 4.7.3. Credential Descriptor (dictionary
5676 PublicKeyCredentialDescriptor)
5677 * 5.2.1. The authenticatorMakeCredential operation
5678
5679 #dom-publickeycredentialdescriptor-transportReferenced in:
5680 * 4.1.3. Create a new credential - PublicKeyCredential's
5681 [[Create]](options) method (2)
5682 * 4.1.4. Use an existing credential to make an assertion -
5683 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5684 method (2)
5685
5686 #dom-publickeycredentialdescriptor-typeReferenced in:
5687 * 4.1.4. Use an existing credential to make an assertion -
5688 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5689 method
5690 * 4.7.3. Credential Descriptor (dictionary
5691 PublicKeyCredentialDescriptor)
5692
5693 #dom-publickeycredentialdescriptor-idReferenced in:
5694 * 4.1.4. Use an existing credential to make an assertion -
5695 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5696 method
5697 * 4.7.3. Credential Descriptor (dictionary
5698 PublicKeyCredentialDescriptor)
5699
5700 #enumdef-authenticatortransportReferenced in:
5701 * 4.7.3. Credential Descriptor (dictionary
5702 PublicKeyCredentialDescriptor)
5703 * 4.7.4. Authenticator Transport enumeration (enum
5704 AuthenticatorTransport)
5705
5706 #dom-authenticatortransport-usbReferenced in:
5707 * 4.7.4. Authenticator Transport enumeration (enum
5708 AuthenticatorTransport)
5709
5710 #dom-authenticatortransport-nfcReferenced in:
5711 * 4.7.4. Authenticator Transport enumeration (enum
5712 AuthenticatorTransport)
5713
5714 #dom-authenticatortransport-bleReferenced in:
5715 * 4.7.4. Authenticator Transport enumeration (enum
5716 AuthenticatorTransport)
5717
5718 #typedefdef-cosealgorithmidentifierReferenced in:
5719 * 4.1.3. Create a new credential - PublicKeyCredential's
5720 [[Create]](options) method
5721 * 4.3. Parameters for Credential Generation (dictionary
5722 PublicKeyCredentialParameters)
5723 * 4.7.5. Cryptographic Algorithm Identifier (typedef
5724 COSEAlgorithmIdentifier)
5725 * 5.2.1. The authenticatorMakeCredential operation
5726 * 5.3.1. Attestation data
5727 * 7.2. Packed Attestation Statement Format
5728 * 7.3. TPM Attestation Statement Format
5729
5730 #attestation-signatureReferenced in:
5731 * 3. Terminology
5732 * 5. WebAuthn Authenticator model (2) (3)
5733 * 5.3. Attestation
5734 * 7.6. FIDO U2F Attestation Statement Format
5735
5736 #assertion-signatureReferenced in:
5737 * 5. WebAuthn Authenticator model (2)
5738 * 5.2.2. The authenticatorGetAssertion operation (2) (3) (4) (5) (6)
5739

```

```

5784 #dictdef-publickeycredentialdescriptorReferenced in:
5785 * 4.4. Options for Credential Creation (dictionary
5786 MakePublicKeyCredentialOptions) (2)
5787 * 4.5. Options for Assertion Generation (dictionary
5788 PublicKeyCredentialRequestOptions) (2) (3)
5789 * 4.7.3. Credential Descriptor (dictionary
5790 PublicKeyCredentialDescriptor)
5791 * 5.2.1. The authenticatorMakeCredential operation
5792
5793 #dom-publickeycredentialdescriptor-transportReferenced in:
5794 * 4.1.3. Create a new credential - PublicKeyCredential's
5795 [[Create]](options) method (2)
5796 * 4.1.4. Use an existing credential to make an assertion -
5797 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5798 method (2)
5799
5800 #dom-publickeycredentialdescriptor-typeReferenced in:
5801 * 4.1.4. Use an existing credential to make an assertion -
5802 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5803 method
5804 * 4.7.3. Credential Descriptor (dictionary
5805 PublicKeyCredentialDescriptor)
5806
5807 #dom-publickeycredentialdescriptor-idReferenced in:
5808 * 4.1.4. Use an existing credential to make an assertion -
5809 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5810 method
5811 * 4.7.3. Credential Descriptor (dictionary
5812 PublicKeyCredentialDescriptor)
5813
5814 #enumdef-authenticatortransportReferenced in:
5815 * 4.7.3. Credential Descriptor (dictionary
5816 PublicKeyCredentialDescriptor)
5817 * 4.7.4. Authenticator Transport enumeration (enum
5818 AuthenticatorTransport)
5819
5820 #dom-authenticatortransport-usbReferenced in:
5821 * 4.7.4. Authenticator Transport enumeration (enum
5822 AuthenticatorTransport)
5823
5824 #dom-authenticatortransport-nfcReferenced in:
5825 * 4.7.4. Authenticator Transport enumeration (enum
5826 AuthenticatorTransport)
5827
5828 #dom-authenticatortransport-bleReferenced in:
5829 * 4.7.4. Authenticator Transport enumeration (enum
5830 AuthenticatorTransport)
5831
5832 #typedefdef-cosealgorithmidentifierReferenced in:
5833 * 4.1.3. Create a new credential - PublicKeyCredential's
5834 [[Create]](options) method
5835 * 4.3. Parameters for Credential Generation (dictionary
5836 PublicKeyCredentialParameters)
5837 * 4.7.5. Cryptographic Algorithm Identifier (typedef
5838 COSEAlgorithmIdentifier)
5839 * 5.2.1. The authenticatorMakeCredential operation
5840 * 5.3.1. Attestation data
5841 * 7.2. Packed Attestation Statement Format
5842 * 7.3. TPM Attestation Statement Format
5843
5844 #attestation-signatureReferenced in:
5845 * 3. Terminology
5846 * 5. WebAuthn Authenticator model (2) (3)
5847 * 5.3. Attestation
5848 * 7.6. FIDO U2F Attestation Statement Format
5849
5850 #assertion-signatureReferenced in:
5851 * 5. WebAuthn Authenticator model (2)
5852 * 5.2.2. The authenticatorGetAssertion operation (2) (3) (4) (5) (6)
5853

```

5740
5741 #authenticator-dataReferenced in:
5742 * 4.2.1. Information about Public Key Credential (interface
5743 AuthenticatorAttestationResponse) (2)
5744 * 4.2.2. Web Authentication Assertion (interface
5745 AuthenticatorAssertionResponse)
5746 * 5. WebAuthn Authenticator model (2)
5747 * 5.1. Authenticator data (2) (3) (4) (5) (6) (7) (8)
5748 * 5.2.1. The authenticatorMakeCredential operation (2)
5749 * 5.2.2. The authenticatorGetAssertion operation (2) (3) (4)
5750 * 5.3. Attestation (2)
5751 * 5.3.1. Attestation data
5752 * 5.3.2. Attestation Statement Formats (2)
5753 * 5.3.4. Generating an Attestation Object (2) (3)
5754 * 5.3.5.3. Attestation Certificate Hierarchy
5755 * 6.1. Registering a new credential (2)
5756 * 7.5. Android SafetyNet Attestation Statement Format
5757 * 8.5. Authenticator extension processing
5758 * 8.6. Example Extension (2)
5759 * 9.6. User Verification Index Extension (uvi)
5760 * 9.7. Location Extension (loc)
5761 * 9.8. User Verification Method Extension (uvm)
5762
5763 #authenticatormakecredentialReferenced in:
5764 * 3. Terminology (2) (3)
5765 * 4.1.3. Create a new credential - PublicKeyCredential's
5766 [[Create]](options) method (2)
5767 * 5. WebAuthn Authenticator model
5768 * 5.2.3. The authenticatorCancel operation (2)
5769 * 8. WebAuthn Extensions
5770 * 8.2. Defining extensions
5771
5772 #authenticatorgetassertionReferenced in:
5773 * 3. Terminology (2) (3)
5774 * 4.1.4. Use an existing credential to make an assertion -
5775 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5776 method (2) (3) (4)
5777 * 5. WebAuthn Authenticator model
5778 * 5.1. Authenticator data
5779 * 5.2.3. The authenticatorCancel operation (2)
5780 * 8. WebAuthn Extensions
5781 * 8.2. Defining extensions
5782
5783 #authenticatorcancelReferenced in:
5784 * 4.1.3. Create a new credential - PublicKeyCredential's
5785 [[Create]](options) method (2) (3)
5786 * 4.1.4. Use an existing credential to make an assertion -
5787 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5788 method (2) (3)
5789
5790 #attestation-objectReferenced in:
5791 * 3. Terminology
5792 * 4. Web Authentication API
5793 * 4.2.1. Information about Public Key Credential (interface
5794 AuthenticatorAttestationResponse) (2)
5795 * 4.4. Options for Credential Creation (dictionary
5796 MakePublicKeyCredentialOptions) (2)
5797 * 5.2.1. The authenticatorMakeCredential operation (2)
5798 * 5.3. Attestation (2) (3)
5799 * 5.3.1. Attestation data
5800 * 5.3.4. Generating an Attestation Object (2) (3) (4)
5801 * 6.1. Registering a new credential
5802
5803 #attestation-statementReferenced in:
5804 * 3. Terminology
5805 * 4.2.1. Information about Public Key Credential (interface
5806 AuthenticatorAttestationResponse) (2) (3)
5807 * 5.3. Attestation (2) (3) (4) (5) (6) (7) (8)
5808 * 5.3.2. Attestation Statement Formats (2) (3)
5809

5854
5855 #authenticator-dataReferenced in:
5856 * 4.2.1. Information about Public Key Credential (interface
5857 AuthenticatorAttestationResponse) (2)
5858 * 4.2.2. Web Authentication Assertion (interface
5859 AuthenticatorAssertionResponse)
5860 * 5. WebAuthn Authenticator model (2)
5861 * 5.1. Authenticator data (2) (3) (4) (5) (6) (7) (8)
5862 * 5.2.1. The authenticatorMakeCredential operation (2)
5863 * 5.2.2. The authenticatorGetAssertion operation (2) (3) (4)
5864 * 5.3. Attestation (2)
5865 * 5.3.1. Attestation data
5866 * 5.3.2. Attestation Statement Formats (2)
5867 * 5.3.4. Generating an Attestation Object (2) (3)
5868 * 5.3.5.3. Attestation Certificate Hierarchy
5869 * 6.1. Registering a new credential (2)
5870 * 7.5. Android SafetyNet Attestation Statement Format
5871 * 8.5. Authenticator extension processing
5872 * 8.6. Example Extension (2)
5873 * 9.6. User Verification Index Extension (uvi)
5874 * 9.7. Location Extension (loc)
5875 * 9.8. User Verification Method Extension (uvm)
5876
5877 #authenticatormakecredentialReferenced in:
5878 * 3. Terminology (2) (3)
5879 * 4.1.3. Create a new credential - PublicKeyCredential's
5880 [[Create]](options) method (2)
5881 * 5. WebAuthn Authenticator model
5882 * 5.2.3. The authenticatorCancel operation (2)
5883 * 8. WebAuthn Extensions
5884 * 8.2. Defining extensions
5885
5886 #authenticatorgetassertionReferenced in:
5887 * 3. Terminology (2) (3)
5888 * 4.1.4. Use an existing credential to make an assertion -
5889 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5890 method (2) (3) (4)
5891 * 5. WebAuthn Authenticator model
5892 * 5.1. Authenticator data
5893 * 5.2.3. The authenticatorCancel operation (2)
5894 * 8. WebAuthn Extensions
5895 * 8.2. Defining extensions
5896
5897 #authenticatorcancelReferenced in:
5898 * 4.1.3. Create a new credential - PublicKeyCredential's
5899 [[Create]](options) method (2) (3)
5900 * 4.1.4. Use an existing credential to make an assertion -
5901 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5902 method (2) (3)
5903
5904 #attestation-objectReferenced in:
5905 * 3. Terminology
5906 * 4. Web Authentication API
5907 * 4.2.1. Information about Public Key Credential (interface
5908 AuthenticatorAttestationResponse) (2)
5909 * 4.4. Options for Credential Creation (dictionary
5910 MakePublicKeyCredentialOptions) (2)
5911 * 5.2.1. The authenticatorMakeCredential operation (2)
5912 * 5.3. Attestation (2) (3)
5913 * 5.3.1. Attestation data
5914 * 5.3.4. Generating an Attestation Object (2) (3) (4)
5915 * 6.1. Registering a new credential
5916
5917 #attestation-statementReferenced in:
5918 * 3. Terminology
5919 * 4.2.1. Information about Public Key Credential (interface
5920 AuthenticatorAttestationResponse) (2) (3)
5921 * 5.3. Attestation (2) (3) (4) (5) (6) (7) (8)
5922 * 5.3.2. Attestation Statement Formats (2) (3)
5923

5810 #attestation-statement-formatReferenced in:
 5811 * 4.2.1. Information about Public Key Credential (interface
 5812 AuthenticatorAttestationResponse)
 5813 * 4.7.4. Authenticator Transport enumeration (enum
 5814 AuthenticatorTransport)
 5815 * 5.3. Attestation (2) (3) (4) (5) (6) (7)
 5816 * 5.3.2. Attestation Statement Formats (2) (3) (4)
 5817 * 5.3.4. Generating an Attestation Object (2)
 5818
 5819 #attestation-typeReferenced in:
 5820 * 5.3. Attestation (2) (3) (4) (5) (6)
 5821 * 5.3.2. Attestation Statement Formats
 5822
 5823 #attestation-dataReferenced in:
 5824 * 5.1. Authenticator data (2) (3) (4) (5) (6) (7)
 5825 * 5.2.1. The authenticatorMakeCredential operation
 5826 * 5.2.2. The authenticatorGetAssertion operation
 5827 * 5.3. Attestation (2)
 5828 * 5.3.3. Attestation Types
 5829 * 6.1. Registering a new credential (2)
 5830 * 7.3. TPM Attestation Statement Format
 5831 * 7.4. Android Key Attestation Statement Format
 5832
 5833 #signing-procedureReferenced in:
 5834 * 5.3.2. Attestation Statement Formats
 5835
 5836 #authenticator-data-for-the-attestationReferenced in:
 5837 * 7.2. Packed Attestation Statement Format
 5838 * 7.3. TPM Attestation Statement Format
 5839 * 7.4. Android Key Attestation Statement Format (2)
 5840 * 7.5. Android SafetyNet Attestation Statement Format
 5841 * 7.6. FIDO U2F Attestation Statement Format
 5842
 5843 #authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:
 5844 * 7.2. Packed Attestation Statement Format
 5845 * 7.3. TPM Attestation Statement Format
 5846 * 7.4. Android Key Attestation Statement Format (2)
 5847 * 7.6. FIDO U2F Attestation Statement Format
 5848
 5849 #basic-attestationReferenced in:
 5850 * 5.3.5.1. Privacy
 5851
 5852 #self-attestationReferenced in:
 5853 * 3. Terminology (2) (3) (4)
 5854 * 5.3. Attestation (2)
 5855 * 5.3.2. Attestation Statement Formats
 5856 * 5.3.3. Attestation Types
 5857 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
 5858 Compromise
 5859 * 6.1. Registering a new credential (2) (3)
 5860 * 7.2. Packed Attestation Statement Format (2)
 5861 * 7.6. FIDO U2F Attestation Statement Format
 5862
 5863 #privacy-caReferenced in:
 5864 * 5.3.5.1. Privacy
 5865
 5866 #elliptic-curve-based-direct-anonymous-attestationReferenced in:
 5867 * 5.3.5.1. Privacy
 5868
 5869 #ecdaaReferenced in:
 5870 * 5.3.2. Attestation Statement Formats
 5871 * 5.3.3. Attestation Types
 5872 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
 5873 Compromise
 5874 * 6.1. Registering a new credential
 5875 * 7.2. Packed Attestation Statement Format (2)
 5876 * 7.3. TPM Attestation Statement Format (2)
 5877
 5878 #attestation-statement-format-identifierReferenced in:

5924 #attestation-statement-formatReferenced in:
 5925 * 4.2.1. Information about Public Key Credential (interface
 5926 AuthenticatorAttestationResponse)
 5927 * 4.7.4. Authenticator Transport enumeration (enum
 5928 AuthenticatorTransport)
 5929 * 5.3. Attestation (2) (3) (4) (5) (6) (7)
 5930 * 5.3.2. Attestation Statement Formats (2) (3) (4)
 5931 * 5.3.4. Generating an Attestation Object (2)
 5932
 5933 #attestation-typeReferenced in:
 5934 * 5.3. Attestation (2) (3) (4) (5) (6)
 5935 * 5.3.2. Attestation Statement Formats
 5936
 5937 #attestation-dataReferenced in:
 5938 * 5.1. Authenticator data (2) (3) (4) (5) (6) (7)
 5939 * 5.2.1. The authenticatorMakeCredential operation
 5940 * 5.2.2. The authenticatorGetAssertion operation
 5941 * 5.3. Attestation (2)
 5942 * 5.3.3. Attestation Types
 5943 * 6.1. Registering a new credential (2)
 5944 * 7.3. TPM Attestation Statement Format
 5945 * 7.4. Android Key Attestation Statement Format
 5946
 5947 #signing-procedureReferenced in:
 5948 * 5.3.2. Attestation Statement Formats
 5949
 5950 #authenticator-data-for-the-attestationReferenced in:
 5951 * 7.2. Packed Attestation Statement Format
 5952 * 7.3. TPM Attestation Statement Format
 5953 * 7.4. Android Key Attestation Statement Format (2)
 5954 * 7.5. Android SafetyNet Attestation Statement Format
 5955 * 7.6. FIDO U2F Attestation Statement Format
 5956
 5957 #authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:
 5958 * 7.2. Packed Attestation Statement Format
 5959 * 7.3. TPM Attestation Statement Format
 5960 * 7.4. Android Key Attestation Statement Format (2)
 5961 * 7.6. FIDO U2F Attestation Statement Format
 5962
 5963 #basic-attestationReferenced in:
 5964 * 5.3.5.1. Privacy
 5965
 5966 #self-attestationReferenced in:
 5967 * 3. Terminology (2) (3) (4)
 5968 * 5.3. Attestation (2)
 5969 * 5.3.2. Attestation Statement Formats
 5970 * 5.3.3. Attestation Types
 5971 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
 5972 Compromise
 5973 * 6.1. Registering a new credential (2) (3)
 5974 * 7.2. Packed Attestation Statement Format (2)
 5975 * 7.6. FIDO U2F Attestation Statement Format
 5976
 5977 #privacy-caReferenced in:
 5978 * 5.3.5.1. Privacy
 5979
 5980 #elliptic-curve-based-direct-anonymous-attestationReferenced in:
 5981 * 5.3.5.1. Privacy
 5982
 5983 #ecdaaReferenced in:
 5984 * 5.3.2. Attestation Statement Formats
 5985 * 5.3.3. Attestation Types
 5986 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
 5987 Compromise
 5988 * 6.1. Registering a new credential
 5989 * 7.2. Packed Attestation Statement Format (2)
 5990 * 7.3. TPM Attestation Statement Format (2)
 5991
 5992 #attestation-statement-format-identifierReferenced in:

5880 * 5.3.2. Attestation Statement Formats
5881 * 5.3.4. Generating an Attestation Object
5882
5883 #identifier-of-the-eccdaa-issuer-public-keyReferenced in:
5884 * 6.1. Registering a new credential
5885 * 7.2. Packed Attestation Statement Format
5886 * 7.3. TPM Attestation Statement Format (2)
5887
5888 #eccdaa-issuer-public-keyReferenced in:
5889 * 5.3.2. Attestation Statement Formats
5890 * 5.3.5.1. Privacy
5891 * 6.1. Registering a new credential
5892 * 7.2. Packed Attestation Statement Format (2) (3)
5893
5894 #registration-extensionReferenced in:
5895 * 4.1.3. Create a new credential - PublicKeyCredential's
5896 [[Create]](options) method
5897 * 8. WebAuthn Extensions (2) (3) (4) (5) (6)
5898 * 8.6. Example Extension
5899 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
5900 * 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
5901 * 9.4. Authenticator Selection Extension (authnSel)
5902 * 9.5. Supported Extensions Extension (exts)
5903 * 9.6. User Verification Index Extension (uvi)
5904 * 9.7. Location Extension (loc)
5905 * 9.8. User Verification Method Extension (uvm)
5906 * 10.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
5907 (6) (7)
5908
5909 #authentication-extensionReferenced in:
5910 * 4.1.4. Use an existing credential to make an assertion -
5911 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5912 method
5913 * 8. WebAuthn Extensions (2) (3) (4) (5) (6)
5914 * 8.6. Example Extension
5915 * 9.1. FIDO Appld Extension (appid)
5916 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
5917 * 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
5918 * 9.6. User Verification Index Extension (uvi)
5919 * 9.7. Location Extension (loc)
5920 * 9.8. User Verification Method Extension (uvm)
5921 * 10.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
5922 (6)
5923
5924 #client-extensionReferenced in:
5925 * 4.1.3. Create a new credential - PublicKeyCredential's
5926 [[Create]](options) method
5927 * 4.1.4. Use an existing credential to make an assertion -
5928 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5929 method
5930 * 4.6. Authentication Extensions (typedef AuthenticationExtensions)
5931 * 8. WebAuthn Extensions
5932 * 8.2. Defining extensions
5933 * 8.4. Client extension processing
5934
5935 #authenticator-extensionReferenced in:
5936 * 4.1.3. Create a new credential - PublicKeyCredential's
5937 [[Create]](options) method
5938 * 4.1.4. Use an existing credential to make an assertion -
5939 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5940 method
5941 * 4.6. Authentication Extensions (typedef AuthenticationExtensions)
5942 * 8. WebAuthn Extensions (2) (3)
5943 * 8.2. Defining extensions (2)
5944 * 8.3. Extending request parameters
5945 * 8.5. Authenticator extension processing
5946
5947 #extension-identifierReferenced in:
5948 * 4.1. PublicKeyCredential Interface
5949 * 4.1.3. Create a new credential - PublicKeyCredential's

5994 * 5.3.2. Attestation Statement Formats
5995 * 5.3.4. Generating an Attestation Object
5996
5997 #identifier-of-the-eccdaa-issuer-public-keyReferenced in:
5998 * 6.1. Registering a new credential
5999 * 7.2. Packed Attestation Statement Format
6000 * 7.3. TPM Attestation Statement Format (2)
6001
6002 #eccdaa-issuer-public-keyReferenced in:
6003 * 5.3.2. Attestation Statement Formats
6004 * 5.3.5.1. Privacy
6005 * 6.1. Registering a new credential
6006 * 7.2. Packed Attestation Statement Format (2) (3)
6007
6008 #registration-extensionReferenced in:
6009 * 4.1.3. Create a new credential - PublicKeyCredential's
6010 [[Create]](options) method
6011 * 8. WebAuthn Extensions (2) (3) (4) (5) (6)
6012 * 8.6. Example Extension
6013 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
6014 * 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
6015 * 9.4. Authenticator Selection Extension (authnSel)
6016 * 9.5. Supported Extensions Extension (exts)
6017 * 9.6. User Verification Index Extension (uvi)
6018 * 9.7. Location Extension (loc)
6019 * 9.8. User Verification Method Extension (uvm)
6020 * 10.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
6021 (6) (7)
6022
6023 #authentication-extensionReferenced in:
6024 * 4.1.4. Use an existing credential to make an assertion -
6025 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
6026 method
6027 * 8. WebAuthn Extensions (2) (3) (4) (5) (6)
6028 * 8.6. Example Extension
6029 * 9.1. FIDO Appld Extension (appid)
6030 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
6031 * 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
6032 * 9.6. User Verification Index Extension (uvi)
6033 * 9.7. Location Extension (loc)
6034 * 9.8. User Verification Method Extension (uvm)
6035 * 10.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
6036 (6)
6037
6038 #client-extensionReferenced in:
6039 * 4.1.3. Create a new credential - PublicKeyCredential's
6040 [[Create]](options) method
6041 * 4.1.4. Use an existing credential to make an assertion -
6042 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
6043 method
6044 * 4.6. Authentication Extensions (typedef AuthenticationExtensions)
6045 * 8. WebAuthn Extensions
6046 * 8.2. Defining extensions
6047 * 8.4. Client extension processing
6048
6049 #authenticator-extensionReferenced in:
6050 * 4.1.3. Create a new credential - PublicKeyCredential's
6051 [[Create]](options) method
6052 * 4.1.4. Use an existing credential to make an assertion -
6053 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
6054 method
6055 * 4.6. Authentication Extensions (typedef AuthenticationExtensions)
6056 * 8. WebAuthn Extensions (2) (3)
6057 * 8.2. Defining extensions (2)
6058 * 8.3. Extending request parameters
6059 * 8.5. Authenticator extension processing
6060
6061 #extension-identifierReferenced in:
6062 * 4.1. PublicKeyCredential Interface
6063 * 4.1.3. Create a new credential - PublicKeyCredential's

5950 [[Create]](options) method

5951 * 4.1.4. Use an existing credential to make an assertion -
 5952 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
 5953 method

5954 * 5.1. Authenticator data
 5955 * 8. WebAuthn Extensions (2)
 5956 * 8.2. Defining extensions
 5957 * 8.3. Extending request parameters
 5958 * 8.4. Client extension processing (2)
 5959 * 8.5. Authenticator extension processing (2)
 5960 * 8.6. Example Extension
 5961 * 9.5. Supported Extensions Extension (exts) (2)
 5962 * 9.7. Location Extension (loc)
 5963 * 10.2. WebAuthn Extension Identifier Registrations

5964 #client-extension-inputReferenced in:
 5965 * 8. WebAuthn Extensions (2) (3)
 5966 * 8.2. Defining extensions
 5967 * 8.3. Extending request parameters (2) (3) (4) (5) (6)
 5968 * 8.4. Client extension processing (2) (3) (4)
 5969 * 8.6. Example Extension

5970 #authenticator-extension-inputReferenced in:
 5971 * 8. WebAuthn Extensions (2) (3) (4) (5)
 5972 * 8.2. Defining extensions
 5973 * 8.3. Extending request parameters (2) (3)
 5974 * 8.4. Client extension processing
 5975 * 8.5. Authenticator extension processing (2) (3)

5976 #client-extension-processingReferenced in:
 5977 * 4.1. PublicKeyCredential Interface
 5978 * 4.1.3. Create a new credential - PublicKeyCredential's
 5979 [[Create]](options) method (2)
 5980 * 4.1.4. Use an existing credential to make an assertion -
 5981 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
 5982 method (2)
 5983 * 8. WebAuthn Extensions (2) (3) (4)
 5984 * 8.2. Defining extensions

5985 #client-extension-outputReferenced in:
 5986 * 4.1. PublicKeyCredential Interface
 5987 * 4.1.3. Create a new credential - PublicKeyCredential's
 5988 [[Create]](options) method (2)

5989 * 4.1.4. Use an existing credential to make an assertion -
 5990 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
 5991 method (2)
 5992 * 8. WebAuthn Extensions (2) (3)
 5993 * 8.2. Defining extensions (2) (3)
 5994 * 8.4. Client extension processing (2) (3)
 5995 * 8.6. Example Extension

6000 #authenticator-extension-processingReferenced in:
 6001 * 8. WebAuthn Extensions
 6002 * 8.2. Defining extensions
 6003 * 8.5. Authenticator extension processing

6004 #authenticator-extension-outputReferenced in:
 6005 * 5.1. Authenticator data
 6006 * 8. WebAuthn Extensions (2) (3)
 6007 * 8.2. Defining extensions (2) (3)
 6008 * 8.4. Client extension processing
 6009 * 8.5. Authenticator extension processing
 6010 * 8.6. Example Extension
 6011 * 9.5. Supported Extensions Extension (exts)
 6012 * 9.6. User Verification Index Extension (uvi)
 6013 * 9.7. Location Extension (loc)

6064 [[Create]](options) method
 6065 * 4.1.3.1. Construct the credential - constructCredentialCallback
 6066 method

6067 * 4.1.4. Use an existing credential to make an assertion -
 6068 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
 6069 method

6070 * 5.1. Authenticator data
 6071 * 8. WebAuthn Extensions (2)
 6072 * 8.2. Defining extensions
 6073 * 8.3. Extending request parameters
 6074 * 8.4. Client extension processing (2)
 6075 * 8.5. Authenticator extension processing (2)
 6076 * 8.6. Example Extension
 6077 * 9.5. Supported Extensions Extension (exts) (2)
 6078 * 9.7. Location Extension (loc)
 6079 * 10.2. WebAuthn Extension Identifier Registrations

6080 #client-extension-inputReferenced in:
 6081 * 8. WebAuthn Extensions (2) (3)
 6082 * 8.2. Defining extensions
 6083 * 8.3. Extending request parameters (2) (3) (4) (5) (6)
 6084 * 8.4. Client extension processing (2) (3) (4)
 6085 * 8.6. Example Extension

6086 #authenticator-extension-inputReferenced in:
 6087 * 8. WebAuthn Extensions (2) (3) (4) (5)
 6088 * 8.2. Defining extensions
 6089 * 8.3. Extending request parameters (2) (3)
 6090 * 8.4. Client extension processing
 6091 * 8.5. Authenticator extension processing (2) (3)

6092 #client-extension-processingReferenced in:
 6093 * 4.1. PublicKeyCredential Interface
 6094 * 4.1.3. Create a new credential - PublicKeyCredential's
 6095 [[Create]](options) method (2)
 6096 * 4.1.4. Use an existing credential to make an assertion -
 6097 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
 6098 method (2)
 6099 * 8. WebAuthn Extensions (2) (3) (4)
 6100 * 8.2. Defining extensions

6101 #client-extension-outputReferenced in:
 6102 * 4.1. PublicKeyCredential Interface
 6103 * 4.1.3. Create a new credential - PublicKeyCredential's
 6104 [[Create]](options) method (2)
 6105 * 4.1.3.1. Construct the credential - constructCredentialCallback
 6106 method

6107 * 4.1.4. Use an existing credential to make an assertion -
 6108 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
 6109 method (2)
 6110 * 8. WebAuthn Extensions (2) (3)
 6111 * 8.2. Defining extensions (2) (3)
 6112 * 8.4. Client extension processing (2) (3)
 6113 * 8.6. Example Extension

6114 #authenticator-extension-processingReferenced in:
 6115 * 8. WebAuthn Extensions
 6116 * 8.2. Defining extensions
 6117 * 8.5. Authenticator extension processing

6118 #authenticator-extension-outputReferenced in:
 6119 * 5.1. Authenticator data
 6120 * 8. WebAuthn Extensions (2) (3)
 6121 * 8.2. Defining extensions (2) (3)
 6122 * 8.4. Client extension processing
 6123 * 8.5. Authenticator extension processing
 6124 * 8.6. Example Extension
 6125 * 9.5. Supported Extensions Extension (exts)
 6126 * 9.6. User Verification Index Extension (uvi)
 6127 * 9.7. Location Extension (loc)

6016 * 9.8. User Verification Method Extension (uvm)
6017
6018 #typedefdef-authenticatorselectionlistReferenced in:
6019 * 9.4. Authenticator Selection Extension (authnSel)
6020
6021 #typedefdef-aaguidReferenced in:
6022 * 9.4. Authenticator Selection Extension (authnSel)
6023

6134 * 9.8. User Verification Method Extension (uvm)
6135
6136 #typedefdef-authenticatorselectionlistReferenced in:
6137 * 9.4. Authenticator Selection Extension (authnSel)
6138
6139 #typedefdef-aaguidReferenced in:
6140 * 9.4. Authenticator Selection Extension (authnSel)
6141