

0001 THE_URL:file://localhost/Users/jehodges/documents/work/standards/W3C/WebAuthn/index-master-ee174c2.html

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

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

0008
0009 Editor's Draft, 16 October 2017

0010
0011 This version:
0012 <https://w3c.github.io/webauthn/>

0013
0014 Latest published version:
0015 <https://www.w3.org/TR/webauthn/>

0016
0017 Previous Versions:
0018 <https://www.w3.org/TR/2017/WD-webauthn-20170811/>
0019 <https://www.w3.org/TR/2017/WD-webauthn-20170505/>
0020 <https://www.w3.org/TR/2017/WD-webauthn-20170216/>
0021 <https://www.w3.org/TR/2016/WD-webauthn-20161207/>
0022 <https://www.w3.org/TR/2016/WD-webauthn-20160928/>
0023 <https://www.w3.org/TR/2016/WD-webauthn-20160902/>
0024 <https://www.w3.org/TR/2016/WD-webauthn-20160531/>

0025
0026 Issue Tracking:
0027 Github

0028
0029 Editors:
0030 Vijay Bharadwaj (Microsoft)
0031 Hubert Le Van Gong (PayPal)
0032 Dirk Balfanz (Google)
0033 Alexei Czeskis (Google)
0034 Arnar Birgisson (Google)
0035 Jeff Hodges (PayPal)
0036 Michael B. Jones (Microsoft)
0037 Rolf Lindemann (Nok Nok Labs)
0038 J.C. Jones (Mozilla)

0039
0040 Tests:
0041 web-platform-tests webauthn/ (ongoing work)

0042
0043 Copyright 2017 W3C[^] (MIT, ERCIM, Keio, Beihang). W3C liability,
0044 trademark and document use rules apply.

0045
0046
0047 **Abstract**

0048 This specification defines an API enabling the creation and use of
0049 strong, attested, scoped, public key-based credentials by web
0050 applications, for the purpose of strongly authenticating users.
0051 Conceptually, one or more public key credentials, each scoped to a
0052 given Relying Party, are created and stored on an authenticator by the
0053 user agent in conjunction with the web application. The user agent
0054 mediates access to public key credentials in order to preserve user
0055 privacy. Authenticators are responsible for ensuring that no operation
0056 is performed without user consent. Authenticators provide cryptographic
0057 proof of their properties to relying parties via attestation. This
0058 specification also describes the functional model for WebAuthn
0059 conformant authenticators, including their signature and attestation
0060 functionality.

0061
0062
0063 **Status of this document**

0064 This section describes the status of this document at the time of its
0065 publication. Other documents may supersede this document. A list of
0066 current W3C publications and the latest revision of this technical
0067 report can be found in the W3C technical reports index at
0068 <http://www.w3.org/TR/>.

0001 THE_URL:file://localhost/Users/jehodges/documents/work/standards/W3C/WebAuthn/index-jeffh-fixup-algs-contd-3-7b272f1.html

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

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

0008
0009 Editor's Draft, 16 October 2017

0010
0011 This version:
0012 <https://w3c.github.io/webauthn/>

0013
0014 Latest published version:
0015 <https://www.w3.org/TR/webauthn/>

0016
0017 Previous Versions:
0018 <https://www.w3.org/TR/2017/WD-webauthn-20170811/>
0019 <https://www.w3.org/TR/2017/WD-webauthn-20170505/>
0020 <https://www.w3.org/TR/2017/WD-webauthn-20170216/>
0021 <https://www.w3.org/TR/2016/WD-webauthn-20161207/>
0022 <https://www.w3.org/TR/2016/WD-webauthn-20160928/>
0023 <https://www.w3.org/TR/2016/WD-webauthn-20160902/>
0024 <https://www.w3.org/TR/2016/WD-webauthn-20160531/>

0025
0026 Issue Tracking:
0027 Github

0028
0029 Editors:
0030 Vijay Bharadwaj (Microsoft)
0031 Hubert Le Van Gong (PayPal)
0032 Dirk Balfanz (Google)
0033 Alexei Czeskis (Google)
0034 Arnar Birgisson (Google)
0035 Jeff Hodges (PayPal)
0036 Michael B. Jones (Microsoft)
0037 Rolf Lindemann (Nok Nok Labs)
0038 J.C. Jones (Mozilla)

0039
0040 Tests:
0041 web-platform-tests webauthn/ (ongoing work)

0042
0043 Copyright 2017 W3C[^] (MIT, ERCIM, Keio, Beihang). W3C liability,
0044 trademark and document use rules apply.

0045
0046
0047 **Abstract**

0048 This specification defines an API enabling the creation and use of
0049 strong, attested, scoped, public key-based credentials by web
0050 applications, for the purpose of strongly authenticating users.
0051 Conceptually, one or more public key credentials, each scoped to a
0052 given Relying Party, are created and stored on an authenticator by the
0053 user agent in conjunction with the web application. The user agent
0054 mediates access to public key credentials in order to preserve user
0055 privacy. Authenticators are responsible for ensuring that no operation
0056 is performed without user consent. Authenticators provide cryptographic
0057 proof of their properties to relying parties via attestation. This
0058 specification also describes the functional model for WebAuthn
0059 conformant authenticators, including their signature and attestation
0060 functionality.

0061
0062
0063 **Status of this document**

0064 This section describes the status of this document at the time of its
0065 publication. Other documents may supersede this document. A list of
0066 current W3C publications and the latest revision of this technical
0067 report can be found in the W3C technical reports index at
0068 <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 User Agents
0101 2. 2.2 Authenticators
0102 3. 2.3 Relying Parties
0103 3. 3 Dependencies
0104 4. 4 Terminology
0105 5. 5 Web Authentication API
0106 1. 5.1 PublicKeyCredential Interface
0107 1. 5.1.1 CredentialCreationOptions Extension
0108 2. 5.1.2 CredentialRequestOptions Extension
0109 3. 5.1.3 Create a new credential - PublicKeyCredential's
0110 [[Create]](options) method
0111 4. 5.1.4 Use an existing credential to make an assertion
0112 1. 5.1.4.1 PublicKeyCredential's
0113 [[DiscoverFromExternalSource]](options) method
0114 5. 5.1.5 Store an existing credential -
0115 PublicKeyCredential's [[Store]](credential) method
0116 6. 5.1.6 Platform Authenticator Availability -
0117 PublicKeyCredential's isPlatformAuthenticatorAvailable()
0118 method
0119 2. 5.2 Authenticator Responses (interface AuthenticatorResponse)
0120 1. 5.2.1 Information about Public Key Credential (interface
0121 AuthenticatorAttestationResponse)
0122 2. 5.2.2 Web Authentication Assertion (interface
0123 AuthenticatorAssertionResponse)
0124 3. 5.3 Parameters for Credential Generation (dictionary
0125 PublicKeyCredentialParameters)
0126 4. 5.4 Options for Credential Creation (dictionary
0127 MakePublicKeyCredentialOptions)
0128 1. 5.4.1 Public Key Entity Description (dictionary
0129 PublicKeyCredentialEntity)
0130 2. 5.4.2 RP Parameters for Credential Generation (dictionary
0131 PublicKeyCredentialRpEntity)
0132 3. 5.4.3 User Account Parameters for Credential Generation
0133 (dictionary PublicKeyCredentialUserEntity)
0134 4. 5.4.4 Authenticator Selection Criteria (dictionary
0135 AuthenticatorSelectionCriteria)
0136 5. 5.4.5 Authenticator Attachment enumeration (enum
0137 AuthenticatorAttachment)
0138 5. 5.5 Options for Assertion Generation (dictionary
0139 PublicKeyCredentialRequestOptions)

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 User Agents
0101 2. 2.2 Authenticators
0102 3. 2.3 Relying Parties
0103 3. 3 Dependencies
0104 4. 4 Terminology
0105 5. 5 Web Authentication API
0106 1. 5.1 PublicKeyCredential Interface
0107 1. 5.1.1 CredentialCreationOptions Extension
0108 2. 5.1.2 CredentialRequestOptions Extension
0109 3. 5.1.3 Create a new credential - PublicKeyCredential's
0110 [[Create]](options) method
0111 4. 5.1.4 Use an existing credential to make an assertion
0112 1. 5.1.4.1 PublicKeyCredential's
0113 [[DiscoverFromExternalSource]](options) method
0114 5. 5.1.5 Store an existing credential -
0115 PublicKeyCredential's [[Store]](credential) method
0116 6. 5.1.6 Platform Authenticator Availability -
0117 PublicKeyCredential's isPlatformAuthenticatorAvailable()
0118 method
0119 2. 5.2 Authenticator Responses (interface AuthenticatorResponse)
0120 1. 5.2.1 Information about Public Key Credential (interface
0121 AuthenticatorAttestationResponse)
0122 2. 5.2.2 Web Authentication Assertion (interface
0123 AuthenticatorAssertionResponse)
0124 3. 5.3 Parameters for Credential Generation (dictionary
0125 PublicKeyCredentialParameters)
0126 4. 5.4 Options for Credential Creation (dictionary
0127 MakePublicKeyCredentialOptions)
0128 1. 5.4.1 Public Key Entity Description (dictionary
0129 PublicKeyCredentialEntity)
0130 2. 5.4.2 RP Parameters for Credential Generation (dictionary
0131 PublicKeyCredentialRpEntity)
0132 3. 5.4.3 User Account Parameters for Credential Generation
0133 (dictionary PublicKeyCredentialUserEntity)
0134 4. 5.4.4 Authenticator Selection Criteria (dictionary
0135 AuthenticatorSelectionCriteria)
0136 5. 5.4.5 Authenticator Attachment enumeration (enum
0137 AuthenticatorAttachment)
0138 5. 5.5 Options for Assertion Generation (dictionary
0139 PublicKeyCredentialRequestOptions)

- 0140 6. 5.6 Authentication Extensions (typedef
- 0141 AuthenticationExtensions)
- 0142 7. 5.7 Supporting Data Structures
- 0143 1. 5.7.1 Client data used in WebAuthn signatures (dictionary
- 0144 CollectedClientData)
- 0145 2. 5.7.2 Credential Type enumeration (enum
- 0146 PublicKeyCredentialType)
- 0147 3. 5.7.3 Credential Descriptor (dictionary
- 0148 PublicKeyCredentialDescriptor)
- 0149 4. 5.7.4 Authenticator Transport enumeration (enum
- 0150 AuthenticatorTransport)
- 0151 5. 5.7.5 Cryptographic Algorithm Identifier (typedef
- 0152 COSEAlgorithmIdentifier)
- 0153 6. 6 WebAuthn Authenticator model
- 0154 1. 6.1 Authenticator data
- 0155 1. 6.1.1 Signature Counter Considerations
- 0156 2. 6.2 Authenticator operations
- 0157 1. 6.2.1 The authenticatorMakeCredential operation
- 0158 2. 6.2.2 The authenticatorGetAssertion operation
- 0159 3. 6.2.3 The authenticatorCancel operation
- 0160 3. 6.3 Attestation
- 0161 1. 6.3.1 Attested credential data
- 0162 2. 6.3.2 Attestation Statement Formats
- 0163 3. 6.3.3 Attestation Types
- 0164 4. 6.3.4 Generating an Attestation Object
- 0165 5. 6.3.5 Security Considerations
- 0166 1. 6.3.5.1 Privacy
- 0167 2. 6.3.5.2 Attestation Certificate and Attestation
- 0168 Certificate CA Compromise
- 0169 3. 6.3.5.3 Attestation Certificate Hierarchy
- 0170 7. 7 Relying Party Operations
- 0171 1. 7.1 Registering a new credential
- 0172 2. 7.2 Verifying an authentication assertion
- 0173 8. 8 Defined Attestation Statement Formats
- 0174 1. 8.1 Attestation Statement Format Identifiers
- 0175 2. 8.2 Packed Attestation Statement Format
- 0176 1. 8.2.1 Packed attestation statement certificate
- 0177 requirements
- 0178 3. 8.3 TPM Attestation Statement Format
- 0179 1. 8.3.1 TPM attestation statement certificate requirements
- 0180 4. 8.4 Android Key Attestation Statement Format
- 0181 5. 8.5 Android SafetyNet Attestation Statement Format
- 0182 6. 8.6 FIDO U2F Attestation Statement Format
- 0183 9. 9 WebAuthn Extensions
- 0184 1. 9.1 Extension Identifiers
- 0185 2. 9.2 Defining extensions
- 0186 3. 9.3 Extending request parameters
- 0187 4. 9.4 Client extension processing
- 0188 5. 9.5 Authenticator extension processing
- 0189 6. 9.6 Example Extension
- 0190 10. 10 Defined Extensions
- 0191 1. 10.1 FIDO AppId Extension (appid)
- 0192 2. 10.2 Simple Transaction Authorization Extension (txAuthSimple)
- 0193 3. 10.3 Generic Transaction Authorization Extension
- 0194 (txAuthGeneric)
- 0195 4. 10.4 Authenticator Selection Extension (authnSel)
- 0196 5. 10.5 Supported Extensions Extension (exts)
- 0197 6. 10.6 User Verification Index Extension (uvi)
- 0198 7. 10.7 Location Extension (loc)
- 0199 8. 10.8 User Verification Method Extension (uvm)
- 0200 11. 11 IANA Considerations
- 0201 1. 11.1 WebAuthn Attestation Statement Format Identifier
- 0202 Registrations
- 0203 2. 11.2 WebAuthn Extension Identifier Registrations
- 0204 3. 11.3 COSE Algorithm Registrations
- 0205 12. 12 Sample scenarios
- 0206 1. 12.1 Registration
- 0207 2. 12.2 Registration Specifically with Platform Authenticator
- 0208 3. 12.3 Authentication
- 0209 4. 12.4 Decommissioning

- 0140 6. 5.6 Authentication Extensions (typedef
- 0141 AuthenticationExtensions)
- 0142 7. 5.7 Supporting Data Structures
- 0143 1. 5.7.1 Client data used in WebAuthn signatures (dictionary
- 0144 CollectedClientData)
- 0145 2. 5.7.2 Credential Type enumeration (enum
- 0146 PublicKeyCredentialType)
- 0147 3. 5.7.3 Credential Descriptor (dictionary
- 0148 PublicKeyCredentialDescriptor)
- 0149 4. 5.7.4 Authenticator Transport enumeration (enum
- 0150 AuthenticatorTransport)
- 0151 5. 5.7.5 Cryptographic Algorithm Identifier (typedef
- 0152 COSEAlgorithmIdentifier)
- 0153 6. 6 WebAuthn Authenticator model
- 0154 1. 6.1 Authenticator data
- 0155 1. 6.1.1 Signature Counter Considerations
- 0156 2. 6.2 Authenticator operations
- 0157 1. 6.2.1 The authenticatorMakeCredential operation
- 0158 2. 6.2.2 The authenticatorGetAssertion operation
- 0159 3. 6.2.3 The authenticatorCancel operation
- 0160 3. 6.3 Attestation
- 0161 1. 6.3.1 Attested credential data
- 0162 2. 6.3.2 Attestation Statement Formats
- 0163 3. 6.3.3 Attestation Types
- 0164 4. 6.3.4 Generating an Attestation Object
- 0165 5. 6.3.5 Security Considerations
- 0166 1. 6.3.5.1 Privacy
- 0167 2. 6.3.5.2 Attestation Certificate and Attestation
- 0168 Certificate CA Compromise
- 0169 3. 6.3.5.3 Attestation Certificate Hierarchy
- 0170 7. 7 Relying Party Operations
- 0171 1. 7.1 Registering a new credential
- 0172 2. 7.2 Verifying an authentication assertion
- 0173 8. 8 Defined Attestation Statement Formats
- 0174 1. 8.1 Attestation Statement Format Identifiers
- 0175 2. 8.2 Packed Attestation Statement Format
- 0176 1. 8.2.1 Packed attestation statement certificate
- 0177 requirements
- 0178 3. 8.3 TPM Attestation Statement Format
- 0179 1. 8.3.1 TPM attestation statement certificate requirements
- 0180 4. 8.4 Android Key Attestation Statement Format
- 0181 5. 8.5 Android SafetyNet Attestation Statement Format
- 0182 6. 8.6 FIDO U2F Attestation Statement Format
- 0183 9. 9 WebAuthn Extensions
- 0184 1. 9.1 Extension Identifiers
- 0185 2. 9.2 Defining extensions
- 0186 3. 9.3 Extending request parameters
- 0187 4. 9.4 Client extension processing
- 0188 5. 9.5 Authenticator extension processing
- 0189 6. 9.6 Example Extension
- 0190 10. 10 Defined Extensions
- 0191 1. 10.1 FIDO AppId Extension (appid)
- 0192 2. 10.2 Simple Transaction Authorization Extension (txAuthSimple)
- 0193 3. 10.3 Generic Transaction Authorization Extension
- 0194 (txAuthGeneric)
- 0195 4. 10.4 Authenticator Selection Extension (authnSel)
- 0196 5. 10.5 Supported Extensions Extension (exts)
- 0197 6. 10.6 User Verification Index Extension (uvi)
- 0198 7. 10.7 Location Extension (loc)
- 0199 8. 10.8 User Verification Method Extension (uvm)
- 0200 11. 11 IANA Considerations
- 0201 1. 11.1 WebAuthn Attestation Statement Format Identifier
- 0202 Registrations
- 0203 2. 11.2 WebAuthn Extension Identifier Registrations
- 0204 3. 11.3 COSE Algorithm Registrations
- 0205 12. 12 Sample scenarios
- 0206 1. 12.1 Registration
- 0207 2. 12.2 Registration Specifically with Platform Authenticator
- 0208 3. 12.3 Authentication
- 0209 4. 12.4 Decommissioning

0210 13. 13 Acknowledgements
0211 14. Index
0212 1. Terms defined by this specification
0213 2. Terms defined by reference
0214 15. References
0215 1. Normative References
0216 2. Informative References
0217 16. IDL Index

0218
0219 1. Introduction
0220
0221 This section is not normative.
0222
0223 This specification defines an API enabling the creation and use of
0224 strong, attested, scoped, public key-based credentials by web
0225 applications, for the purpose of strongly authenticating users. A
0226 public key credential is created and stored by an authenticator at the
0227 behest of a Relying Party, subject to user consent. Subsequently, the
0228 public key credential can only be accessed by origins belonging to that
0229 Relying Party. This scoping is enforced jointly by conforming User
0230 Agents and authenticators. Additionally, privacy across Relying Parties
0231 is maintained; Relying Parties are not able to detect any properties,
0232 or even the existence, of credentials scoped to other Relying Parties.
0233
0234 Relying Parties employ the Web Authentication API during two distinct,
0235 but related, ceremonies involving a user. The first is Registration,
0236 where a public key credential is created on an authenticator, and
0237 associated by a Relying Party with the present user's account (the
0238 account may already exist or may be created at this time). The second
0239 is Authentication, where the Relying Party is presented with an
0240 Authentication Assertion proving the presence and consent of the user
0241 who registered the public key credential. Functionally, the Web
0242 Authentication API comprises a PublicKeyCredential which extends the
0243 Credential Management API [CREDENTIAL-MANAGEMENT-1], and infrastructure
0244 which allows those credentials to be used with
0245 navigator.credentials.create() and navigator.credentials.get(). The
0246 former is used during Registration, and the latter during
0247 Authentication.
0248
0249 Broadly, compliant authenticators protect public key credentials, and
0250 interact with user agents to implement the Web Authentication API. Some
0251 authenticators may run on the same computing device (e.g., smart phone,
0252 tablet, desktop PC) as the user agent is running on. For instance, such
0253 an authenticator might consist of a Trusted Execution Environment (TEE)
0254 applet, a Trusted Platform Module (TPM), or a Secure Element (SE)
0255 integrated into the computing device in conjunction with some means for
0256 user verification, along with appropriate platform software to mediate
0257 access to these components' functionality. Other authenticators may
0258 operate autonomously from the computing device running the user agent,
0259 and be accessed over a transport such as Universal Serial Bus (USB),
0260 Bluetooth Low Energy (BLE) or Near Field Communications (NFC).
0261
0262 1.1. Use Cases
0263
0264 The below use case scenarios illustrate use of two very different types
0265 of authenticators, as well as outline further scenarios. Additional
0266 scenarios, including sample code, are given later in 12 Sample
0267 scenarios.
0268
0269 1.1.1. Registration
0270
0271 * On a phone:
0272 + User navigates to example.com in a browser and signs in to an
0273 existing account using whatever method they have been using
0274 (possibly a legacy method such as a password), or creates a
0275 new account.
0276 + The phone prompts, "Do you want to register this device with
0277 example.com?"
0278 + User agrees.

0210 13. 13 Acknowledgements
0211 14. Index
0212 1. Terms defined by this specification
0213 2. Terms defined by reference
0214 15. References
0215 1. Normative References
0216 2. Informative References
0217 16. IDL Index
0218 17. Issues Index

0219 1. Introduction
0220
0221 This section is not normative.
0222
0223 This specification defines an API enabling the creation and use of
0224 strong, attested, scoped, public key-based credentials by web
0225 applications, for the purpose of strongly authenticating users. A
0226 public key credential is created and stored by an authenticator at the
0227 behest of a Relying Party, subject to user consent. Subsequently, the
0228 public key credential can only be accessed by origins belonging to that
0229 Relying Party. This scoping is enforced jointly by conforming User
0230 Agents and authenticators. Additionally, privacy across Relying Parties
0231 is maintained; Relying Parties are not able to detect any properties,
0232 or even the existence, of credentials scoped to other Relying Parties.
0233
0234 Relying Parties employ the Web Authentication API during two distinct,
0235 but related, ceremonies involving a user. The first is Registration,
0236 where a public key credential is created on an authenticator, and
0237 associated by a Relying Party with the present user's account (the
0238 account may already exist or may be created at this time). The second
0239 is Authentication, where the Relying Party is presented with an
0240 Authentication Assertion proving the presence and consent of the user
0241 who registered the public key credential. Functionally, the Web
0242 Authentication API comprises a PublicKeyCredential which extends the
0243 Credential Management API [CREDENTIAL-MANAGEMENT-1], and infrastructure
0244 which allows those credentials to be used with
0245 navigator.credentials.create() and navigator.credentials.get(). The
0246 former is used during Registration, and the latter during
0247 Authentication.
0248
0249 Broadly, compliant authenticators protect public key credentials, and
0250 interact with user agents to implement the Web Authentication API. Some
0251 authenticators may run on the same computing device (e.g., smart phone,
0252 tablet, desktop PC) as the user agent is running on. For instance, such
0253 an authenticator might consist of a Trusted Execution Environment (TEE)
0254 applet, a Trusted Platform Module (TPM), or a Secure Element (SE)
0255 integrated into the computing device in conjunction with some means for
0256 user verification, along with appropriate platform software to mediate
0257 access to these components' functionality. Other authenticators may
0258 operate autonomously from the computing device running the user agent,
0259 and be accessed over a transport such as Universal Serial Bus (USB),
0260 Bluetooth Low Energy (BLE) or Near Field Communications (NFC).
0261
0262 1.1. Use Cases
0263
0264 The below use case scenarios illustrate use of two very different types
0265 of authenticators, as well as outline further scenarios. Additional
0266 scenarios, including sample code, are given later in 12 Sample
0267 scenarios.
0268
0269 1.1.1. Registration
0270
0271 * On a phone:
0272 + User navigates to example.com in a browser and signs in to an
0273 existing account using whatever method they have been using
0274 (possibly a legacy method such as a password), or creates a
0275 new account.
0276 + The phone prompts, "Do you want to register this device with
0277 example.com?"
0278 + User agrees.

0275 + The phone prompts the user for a previously configured
0280 authorization gesture (PIN, biometric, etc.); the user
0281 provides this.
0282 + Website shows message, "Registration complete."
0283
0284 **1.1.2. Authentication**
0285
0286 * On a laptop or desktop:
0287 + User navigates to example.com in a browser, sees an option to
0288 "Sign in with your phone."
0289 + User chooses this option and gets a message from the browser,
0290 "Please complete this action on your phone."
0291 * Next, on their phone:
0292 + User sees a discrete prompt or notification, "Sign in to
0293 example.com."
0294 + User selects this prompt / notification.
0295 + User is shown a list of their example.com identities, e.g.,
0296 "Sign in as Alice / Sign in as Bob."
0297 + User picks an identify, is prompted for an authorization
0298 gesture (PIN, biometric, etc.) and provides this.
0299 * Now, back on the laptop:
0300 + Web page shows that the selected user is signed-in, and
0301 navigates to the signed-in page.
0302
0303 **1.1.3. Other use cases and configurations**
0304
0305 A variety of additional use cases and configurations are also possible,
0306 including (but not limited to):
0307 * A user navigates to example.com on their laptop, is guided through
0308 a flow to create and register a credential on their phone.
0309 * A user obtains an discrete, roaming authenticator, such as a "fob"
0310 with USB or USB+NFC/BLE connectivity options, loads example.com in
0311 their browser on a laptop or phone, and is guided through a flow to
0312 create and register a credential on the fob.
0313 * A Relying Party prompts the user for their authorization gesture in
0314 order to authorize a single transaction, such as a payment or other
0315 financial transaction.
0316
0317 **2. Conformance**
0318
0319 This specification defines three conformance classes. Each of these
0320 classes is specified so that conforming members of the class are secure
0321 against non-conforming or hostile members of the other classes.
0322
0323 **2.1. User Agents**
0324
0325 A User Agent MUST behave as described by 5 Web Authentication API in
0326 order to be considered conformant. Conforming User Agents MAY implement
0327 algorithms given in this specification in any way desired, so long as
0328 the end result is indistinguishable from the result that would be
0329 obtained by the specification's algorithms.
0330
0331 A conforming User Agent MUST also be a conforming implementation of the
0332 IDL fragments of this specification, as described in the "Web IDL"
0333 specification. [WebIDL-1]
0334
0335 **2.2. Authenticators**
0336
0337 An authenticator MUST provide the operations defined by 6 WebAuthn
0338 Authenticator model, and those operations MUST behave as described
0339 there. This is a set of functional and security requirements for an
0340 authenticator to be usable by a Conforming User Agent.
0341
0342 As described in 1.1 Use Cases, an authenticator may be implemented in
0343 the operating system underlying the User Agent, or in external
0344 hardware, or a combination of both.
0345
0346 **2.3. Relying Parties**
0347
0348 A Relying Party MUST behave as described in 7 Relying Party Operations

0280 + The phone prompts the user for a previously configured
0281 authorization gesture (PIN, biometric, etc.); the user
0282 provides this.
0283 + Website shows message, "Registration complete."
0284
0285 **1.1.2. Authentication**
0286
0287 * On a laptop or desktop:
0288 + User navigates to example.com in a browser, sees an option to
0289 "Sign in with your phone."
0290 + User chooses this option and gets a message from the browser,
0291 "Please complete this action on your phone."
0292 * Next, on their phone:
0293 + User sees a discrete prompt or notification, "Sign in to
0294 example.com."
0295 + User selects this prompt / notification.
0296 + User is shown a list of their example.com identities, e.g.,
0297 "Sign in as Alice / Sign in as Bob."
0298 + User picks an identify, is prompted for an authorization
0299 gesture (PIN, biometric, etc.) and provides this.
0300 * Now, back on the laptop:
0301 + Web page shows that the selected user is signed-in, and
0302 navigates to the signed-in page.
0303
0304 **1.1.3. Other use cases and configurations**
0305
0306 A variety of additional use cases and configurations are also possible,
0307 including (but not limited to):
0308 * A user navigates to example.com on their laptop, is guided through
0309 a flow to create and register a credential on their phone.
0310 * A user obtains an discrete, roaming authenticator, such as a "fob"
0311 with USB or USB+NFC/BLE connectivity options, loads example.com in
0312 their browser on a laptop or phone, and is guided through a flow to
0313 create and register a credential on the fob.
0314 * A Relying Party prompts the user for their authorization gesture in
0315 order to authorize a single transaction, such as a payment or other
0316 financial transaction.
0317
0318 **2. Conformance**
0319
0320 This specification defines three conformance classes. Each of these
0321 classes is specified so that conforming members of the class are secure
0322 against non-conforming or hostile members of the other classes.
0323
0324 **2.1. User Agents**
0325
0326 A User Agent MUST behave as described by 5 Web Authentication API in
0327 order to be considered conformant. Conforming User Agents MAY implement
0328 algorithms given in this specification in any way desired, so long as
0329 the end result is indistinguishable from the result that would be
0330 obtained by the specification's algorithms.
0331
0332 A conforming User Agent MUST also be a conforming implementation of the
0333 IDL fragments of this specification, as described in the "Web IDL"
0334 specification. [WebIDL-1]
0335
0336 **2.2. Authenticators**
0337
0338 An authenticator MUST provide the operations defined by 6 WebAuthn
0339 Authenticator model, and those operations MUST behave as described
0340 there. This is a set of functional and security requirements for an
0341 authenticator to be usable by a Conforming User Agent.
0342
0343 As described in 1.1 Use Cases, an authenticator may be implemented in
0344 the operating system underlying the User Agent, or in external
0345 hardware, or a combination of both.
0346
0347 **2.3. Relying Parties**
0348
0349 A Relying Party MUST behave as described in 7 Relying Party Operations

034E to get the security benefits offered by this specification.
0350
0351 **3. Dependencies**
0352
0353 This specification relies on several other underlying specifications,
0354 listed below and in Terms defined by reference.
035E
035E **Base64url encoding**
0357 The term Base64url Encoding refers to the base64 encoding using
035E the URL- and filename-safe character set defined in Section 5 of
035E [RFC4648], with all trailing '=' characters omitted (as
036C permitted by Section 3.2) and without the inclusion of any line
0361 breaks, whitespace, or other additional characters.
0362
0363 **CBOR**
0364 A number of structures in this specification, including
036E attestation statements and extensions, are encoded using the
036E Compact Binary Object Representation (CBOR) [RFC7049].
0367
036E **CDDL**
036E This specification describes the syntax of all CBOR-encoded data
037C using the CBOR Data Definition Language (CDDL) [CDDL].
0371
0372 **COSE**
0373 CBOR Object Signing and Encryption (COSE) [RFC8152]. The IANA
0374 COSE Algorithms registry established by this specification is
037E also used.
037E
0377 **Credential Management**
037E The API described in this document is an extension of the
037E Credential concept defined in [CREDENTIAL-MANAGEMENT-1].
038C
0381 **DOM**
038E DOMException and the DOMException values used in this
038E specification are defined in [DOM4].
0384
038E **ECMAScript**
038E %ArrayBuffer% is defined in [ECMAScript].
0387
038E **HTML**
038E The concepts of relevant settings object, origin, opaque origin,
039C and is a registrable domain suffix of or is equal to are defined
0391 in [HTML52].
0392
0393 **Web IDL**
0394 Many of the interface definitions and all of the IDL in this
039E specification depend on [WebIDL-1]. This updated version of the
039E Web IDL standard adds support for Promises, which are now the
0397 preferred mechanism for asynchronous interaction in all new web
039E APIs.
039E
040C The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
0401 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
0402 document are to be interpreted as described in [RFC2119].
0403
0404 **4. Terminology**
040E
040E **Assertion**
0407 See Authentication Assertion.
040E
040E **Attestation**
041C Generally, attestation is a statement serving to bear witness,
0411 confirm, or authenticate. In the WebAuthn context, attestation
0412 is employed to attest to the provenance of an authenticator and
0413 the data it emits; including, for example: credential IDs,
0414 credential key pairs, signature counters, etc. An attestation
041E statement is conveyed in an attestation object during
041E registration. See also 6.3 Attestation and Figure 3.
0417
041E **Attestation Certificate**

0350 to get the security benefits offered by this specification.
0351
0352 **3. Dependencies**
0353
0354 This specification relies on several other underlying specifications,
035E listed below and in Terms defined by reference.
035E
035E **Base64url encoding**
0357 The term Base64url Encoding refers to the base64 encoding using
035E the URL- and filename-safe character set defined in Section 5 of
035E [RFC4648], with all trailing '=' characters omitted (as
036C permitted by Section 3.2) and without the inclusion of any line
0361 breaks, whitespace, or other additional characters.
0362
0363 **CBOR**
0364 A number of structures in this specification, including
036E attestation statements and extensions, are encoded using the
036E Compact Binary Object Representation (CBOR) [RFC7049].
0367
036E **CDDL**
036E This specification describes the syntax of all CBOR-encoded data
037C using the CBOR Data Definition Language (CDDL) [CDDL].
0371
0372 **COSE**
0373 CBOR Object Signing and Encryption (COSE) [RFC8152]. The IANA
0374 COSE Algorithms registry established by this specification is
037E also used.
037E
0377 **Credential Management**
037E The API described in this document is an extension of the
037E Credential concept defined in [CREDENTIAL-MANAGEMENT-1].
038C
0381 **DOM**
038E DOMException and the DOMException values used in this
038E specification are defined in [DOM4].
0384
038E **ECMAScript**
038E %ArrayBuffer% is defined in [ECMAScript].
0387
038E **HTML**
038E The concepts of relevant settings object, origin, opaque origin,
039C and is a registrable domain suffix of or is equal to are defined
0391 in [HTML52].
0392
0393 **Web IDL**
0394 Many of the interface definitions and all of the IDL in this
039E specification depend on [WebIDL-1]. This updated version of the
039E Web IDL standard adds support for Promises, which are now the
0397 preferred mechanism for asynchronous interaction in all new web
039E APIs.
039E
040C The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
0401 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
0402 document are to be interpreted as described in [RFC2119].
0403
0404 **4. Terminology**
040E
040E **Assertion**
0407 See Authentication Assertion.
040E
040E **Attestation**
041C Generally, attestation is a statement serving to bear witness,
0411 confirm, or authenticate. In the WebAuthn context, attestation
0412 is employed to attest to the provenance of an authenticator and
0413 the data it emits; including, for example: credential IDs,
0414 credential key pairs, signature counters, etc. An attestation
041E statement is conveyed in an attestation object during
041E registration. See also 6.3 Attestation and Figure 3.
0417
041E **Attestation Certificate**

0419 A X.509 Certificate for the attestation key pair used by an
0420 authenticator to attest to its manufacture and capabilities. At
0421 registration time, the authenticator uses the attestation
0422 private key to sign the Relying Party-specific credential public
0423 key (and additional data) that it generates and returns via the
0424 authenticatorMakeCredential operation. Relying Parties use the
0425 attestation public key conveyed in the attestation certificate
0426 to verify the attestation signature. Note that in the case of
0427 self attestation, the authenticator has no distinct attestation
0428 key pair nor attestation certificate, see self attestation for
0429 details.

0430
0431 **Authentication**
0432 The ceremony where a user, and the user's computing device(s)
0433 (containing at least one authenticator) work in concert to
0434 cryptographically prove to an Relying Party that the user
0435 controls the credential private key associated with a
0436 previously-registered public key credential (see Registration).
0437 Note that this typically includes employing a test of user
0438 presence or user verification.

0439
0440 **Authentication Assertion**
0441 The cryptographically signed AuthenticatorAssertionResponse
0442 object returned by an authenticator as the result of a
0443 authenticatorGetAssertion operation.

0444
0445 **Authenticator**
0446 A cryptographic entity used by a WebAuthn Client to (i) generate
0447 a public key credential and register it with a Relying Party,
0448 and (ii) authenticate by potentially verifying the user, and
0449 then cryptographically signing and returning, in the form of an
0450 Authentication Assertion, a challenge and other data presented
0451 by a Relying Party (in concert with the WebAuthn Client).

0452
0453 **Authorization Gesture**
0454 An authorization gesture is a physical interaction performed by
0455 a user with an authenticator as part of a ceremony, such as
0456 registration or authentication. By making such an authorization
0457 gesture, a user provides consent for (i.e., authorizes) a
0458 ceremony to proceed. This may involve user verification if the
0459 employed authenticator is capable, or it may involve a simple
0460 test of user presence.

0461
0462 **Biometric Recognition**
0463 The automated recognition of individuals based on their
0464 biological and behavioral characteristics
0465 [ISOBiometricVocabulary].

0466
0467 **Ceremony**
0468 The concept of a ceremony [Ceremony] is an extension of the
0469 concept of a network protocol, with human nodes alongside
0470 computer nodes and with communication links that include user
0471 interface(s), human-to-human communication, and transfers of
0472 physical objects that carry data. What is out-of-band to a
0473 protocol is in-band to a ceremony. In this specification,
0474 Registration and Authentication are ceremonies, and an
0475 authorization gesture is often a component of those ceremonies.

0476
0477 **Client**
0478 See Conforming User Agent.

0479
0480 **Client-Side**
0481 This refers in general to the combination of the user's platform
0482 device, user agent, authenticators, and everything gluing it all
0483 together.

0484
0485 **Client-side-resident Credential Private Key**
0486 A Client-side-resident Credential Private Key is stored either
0487 on the client platform, or in some cases on the authenticator
0488 itself, e.g., in the case of a discrete first-factor roaming

0420 A X.509 Certificate for the attestation key pair used by an
0421 authenticator to attest to its manufacture and capabilities. At
0422 registration time, the authenticator uses the attestation
0423 private key to sign the Relying Party-specific credential public
0424 key (and additional data) that it generates and returns via the
0425 authenticatorMakeCredential operation. Relying Parties use the
0426 attestation public key conveyed in the attestation certificate
0427 to verify the attestation signature. Note that in the case of
0428 self attestation, the authenticator has no distinct attestation
0429 key pair nor attestation certificate, see self attestation for
0430 details.

0431
0432 **Authentication**
0433 The ceremony where a user, and the user's computing device(s)
0434 (containing at least one authenticator) work in concert to
0435 cryptographically prove to an Relying Party that the user
0436 controls the credential private key associated with a
0437 previously-registered public key credential (see Registration).
0438 Note that this typically includes employing a test of user
0439 presence or user verification.

0440
0441 **Authentication Assertion**
0442 The cryptographically signed AuthenticatorAssertionResponse
0443 object returned by an authenticator as the result of a
0444 authenticatorGetAssertion operation.

0445
0446 **Authenticator**
0447 A cryptographic entity used by a WebAuthn Client to (i) generate
0448 a public key credential and register it with a Relying Party,
0449 and (ii) authenticate by potentially verifying the user, and
0450 then cryptographically signing and returning, in the form of an
0451 Authentication Assertion, a challenge and other data presented
0452 by a Relying Party (in concert with the WebAuthn Client).

0453
0454 **Authorization Gesture**
0455 An authorization gesture is a physical interaction performed by
0456 a user with an authenticator as part of a ceremony, such as
0457 registration or authentication. By making such an authorization
0458 gesture, a user provides consent for (i.e., authorizes) a
0459 ceremony to proceed. This may involve user verification if the
0460 employed authenticator is capable, or it may involve a simple
0461 test of user presence.

0462
0463 **Biometric Recognition**
0464 The automated recognition of individuals based on their
0465 biological and behavioral characteristics
0466 [ISOBiometricVocabulary].

0467
0468 **Ceremony**
0469 The concept of a ceremony [Ceremony] is an extension of the
0470 concept of a network protocol, with human nodes alongside
0471 computer nodes and with communication links that include user
0472 interface(s), human-to-human communication, and transfers of
0473 physical objects that carry data. What is out-of-band to a
0474 protocol is in-band to a ceremony. In this specification,
0475 Registration and Authentication are ceremonies, and an
0476 authorization gesture is often a component of those ceremonies.

0477
0478 **Client**
0479 See Conforming User Agent.

0480
0481 **Client-Side**
0482 This refers in general to the combination of the user's platform
0483 device, user agent, authenticators, and everything gluing it all
0484 together.

0485
0486 **Client-side-resident Credential Private Key**
0487 A Client-side-resident Credential Private Key is stored either
0488 on the client platform, or in some cases on the authenticator
0489 itself, e.g., in the case of a discrete first-factor roaming

048E authenticator. Such client-side credential private key storage
 049C has the property that the authenticator is able to select the
 0491 credential private key given only an RP ID, possibly with user
 0492 assistance (e.g., by providing the user a pick list of
 0493 credentials associated with the RP ID). By definition, the
 0494 private key is always exclusively controlled by the
 049E Authenticator. In the case of a Client-side-resident Credential
 049E Private Key, the Authenticator might offload storage of wrapped
 0497 key material to the client platform, but the client platform is
 049E not expected to offload the key storage to remote entities (e.g.
 049E RP Server).
 050C
Conforming User Agent
 0501 A user agent implementing, in conjunction with the underlying
 0502 platform, the Web Authentication API and algorithms given in
 0503 this specification, and handling communication between
 0504 authenticators and Relying Parties.
 0505
Credential Public Key
 0507 The public key portion of an Relying Party-specific credential
 0508 key pair, generated by an authenticator and returned to an
 0509 Relying Party at registration time (see also public key
 051C credential). The private key portion of the credential key pair
 0511 is known as the credential private key. Note that in the case of
 0512 self attestation, the credential key pair is also used as the
 0513 attestation key pair, see self attestation for details.
 0514
Rate Limiting
 051E The process (also known as throttling) by which an authenticator
 0517 implements controls against brute force attacks by limiting the
 051E number of consecutive failed authentication attempts within a
 051E given period of time. If the limit is reached, the authenticator
 052C should impose a delay that increases exponentially with each
 0521 successive attempt, or disable the current authentication
 0522 modality and offer a different authentication factor if
 0522 available. Rate limiting is often implemented as an aspect of
 0524 user verification.
 052E
Registration
 0527 The ceremony where a user, a Relying Party, and the user's
 052E computing device(s) (containing at least one authenticator) work
 052E in concert to create a public key credential and associate it
 053C with the user's Relying Party account. Note that this typically
 0531 includes employing a test of user presence or user verification.
 0532
Relying Party
 0534 The entity whose web application utilizes the Web Authentication
 053E API to register and authenticate users. See Registration and
 053E Authentication, respectively.
 053E
 Note: While the term Relying Party is used in other contexts
 053E (e.g., X.509 and OAuth), an entity acting as a Relying Party in
 054C one context is not necessarily a Relying Party in other
 0542 contexts.
Relying Party Identifier
 0544
RP ID
 054E A valid domain string that identifies the Relying Party on whose
 0547 behalf a given registration or authentication ceremony is being
 054E performed. A public key credential can only be used for
 054E authentication with the same entity (as identified by RP ID) it
 055C was registered with. By default, the RP ID for a WebAuthn
 0551 operation is set to the caller's origin's effective domain. This
 0552 default MAY be overridden by the caller, as long as the
 0553 caller-specified RP ID value is a registrable domain suffix of
 0554 or is equal to the caller's origin's effective domain. See also
 055E 5.1.3 Create a new credential - PublicKeyCredential's
 055E [[Create]](options) method and 5.1.4 Use an existing credential
 0557 to make an assertion.
 055E

049C authenticator. Such client-side credential private key storage
 0491 has the property that the authenticator is able to select the
 0492 credential private key given only an RP ID, possibly with user
 0493 assistance (e.g., by providing the user a pick list of
 0494 credentials associated with the RP ID). By definition, the
 049E private key is always exclusively controlled by the
 049E Authenticator. In the case of a Client-side-resident Credential
 0497 Private Key, the Authenticator might offload storage of wrapped
 049E key material to the client platform, but the client platform is
 049E not expected to offload the key storage to remote entities (e.g.
 049E RP Server).
 050C
Conforming User Agent
 0502 A user agent implementing, in conjunction with the underlying
 0503 platform, the Web Authentication API and algorithms given in
 0504 this specification, and handling communication between
 050E authenticators and Relying Parties.
 050E
Credential Public Key
 050E The public key portion of an Relying Party-specific credential
 0509 key pair, generated by an authenticator and returned to an
 051C Relying Party at registration time (see also public key
 0511 credential). The private key portion of the credential key pair
 0512 is known as the credential private key. Note that in the case of
 0513 self attestation, the credential key pair is also used as the
 0514 attestation key pair, see self attestation for details.
 051E
Rate Limiting
 051E The process (also known as throttling) by which an authenticator
 051E implements controls against brute force attacks by limiting the
 052C number of consecutive failed authentication attempts within a
 0521 given period of time. If the limit is reached, the authenticator
 0522 should impose a delay that increases exponentially with each
 0522 successive attempt, or disable the current authentication
 0524 modality and offer a different authentication factor if
 052E available. Rate limiting is often implemented as an aspect of
 052E user verification.
Registration
 052E The ceremony where a user, a Relying Party, and the user's
 052E computing device(s) (containing at least one authenticator) work
 053C in concert to create a public key credential and associate it
 053E with the user's Relying Party account. Note that this typically
 053E includes employing a test of user presence or user verification.
Relying Party
 053E The entity whose web application utilizes the Web Authentication
 053E API to register and authenticate users. See Registration and
 053E Authentication, respectively.
 Note: While the term Relying Party is used in other contexts
 054C (e.g., X.509 and OAuth), an entity acting as a Relying Party in
 0542 one context is not necessarily a Relying Party in other
 054C contexts.
Relying Party Identifier
 054E
RP ID
 054E A valid domain string that identifies the Relying Party on whose
 0547 behalf a given registration or authentication ceremony is being
 054E performed. A public key credential can only be used for
 055C authentication with the same entity (as identified by RP ID) it
 0551 was registered with. By default, the RP ID for a WebAuthn
 0552 operation is set to the caller's origin's effective domain. This
 0553 default MAY be overridden by the caller, as long as the
 0554 caller-specified RP ID value is a registrable domain suffix of
 055E or is equal to the caller's origin's effective domain. See also
 055E 5.1.3 Create a new credential - PublicKeyCredential's
 055E [[Create]](options) method and 5.1.4 Use an existing credential
 0557 to make an assertion.
 055E

055E Note: A Public key credential's scope is for a Relying Party's
056C origin, with the following restrictions and relaxations:
0561

- 0562 + The scheme is always https (i.e., a restriction), and,
- 0563 + the host may be equal to the Relying Party's origin's
- 0564 effective domain, or it may be equal to a registrable domain
- 0565 suffix of the Relying Party's origin's effective domain (i.e.,
- 0566 an available relaxation), and,
- 0567 + all (TCP) ports on that host (i.e., a relaxation).

0568
0569 This is done in order to match the behavior of pervasively
057C deployed ambient credentials (e.g., cookies, [RFC6265]). Please
0571 note that this is a greater relaxation of "same-origin"
0572 restrictions than what document.domain's setter provides.
0573

0574 **Public Key Credential**
0575 Generically, a credential is data one entity presents to another
0576 in order to authenticate the former to the latter [RFC4949]. A
0577 WebAuthn public key credential is a { identifier, type } pair
0578 identifying authentication information established by the
0579 authenticator and the Relying Party, together, at registration
058C time. The authentication information consists of an asymmetric
0581 key pair, where the public key portion is returned to the
0582 Relying Party, who then stores it in conjunction with the
0583 present user's account. The authenticator maps the private key
0584 portion to the Relying Party's RP ID and stores it.
0585 Subsequently, only that Relying Party, as identified by its RP
0586 ID, is able to employ the public key credential in
0587 authentication ceremonies, via the get() method. The Relying
0588 Party uses its stored copy of the credential public key to
0589 verify the resultant authentication assertion.
059C

0591 **Test of User Presence**
0592 A test of user presence is a simple form of authorization
0593 gesture and technical process where a user interacts with an
0594 authenticator by (typically) simply touching it (other
0595 modalities may also exist), yielding a boolean result. Note that
0596 this does not constitute user verification because a user
0597 presence test, by definition, is not capable of biometric
0598 recognition, nor does it involve the presentation of a shared
0599 secret such as a password or PIN.
060C

0601 **User Consent**
0602 User consent means the user agrees with what they are being
0603 asked, i.e., it encompasses reading and understanding prompts.
0604 An authorization gesture is a ceremony component often employed
0605 to indicate user consent.
0606

0607 **User Handle**
0608 The user handle is specified by a Relying Party and is a unique
0609 identifier for a user account with that Relying Party. A user
061C handle is an opaque byte sequence with a maximum size of 64
0611 bytes.
0612

0613 The user handle is not meant to be displayed to the user, but is
0614 used by the Relying Party to control the number of credentials -
0615 an authenticator will never contain more than one credential for
0616 a given Relying Party under the same user handle.
0617

0618 **User Verification**
0619 The technical process by which an authenticator locally
062C authorizes the invocation of the authenticatorMakeCredential and
0621 authenticatorGetAssertion operations. User verification may be
0622 instigated through various authorization gesture modalities; for
0623 example, through a touch plus pin code, password entry, or
0624 biometric recognition (e.g., presenting a fingerprint)
0625 [ISOBiometricVocabulary]. The intent is to be able to
0626 distinguish individual users. Note that invocation of the
0627 authenticatorMakeCredential and authenticatorGetAssertion
0628 operations implies use of key material managed by the

056C Note: A Public key credential's scope is for a Relying Party's
0561 origin, with the following restrictions and relaxations:
0562

- 0563 + The scheme is always https (i.e., a restriction), and,
- 0564 + the host may be equal to the Relying Party's origin's
- 0565 effective domain, or it may be equal to a registrable domain
- 0566 suffix of the Relying Party's origin's effective domain (i.e.,
- 0567 an available relaxation), and,
- 0568 + all (TCP) ports on that host (i.e., a relaxation).

0569
057C This is done in order to match the behavior of pervasively
0571 deployed ambient credentials (e.g., cookies, [RFC6265]). Please
0572 note that this is a greater relaxation of "same-origin"
0573 restrictions than what document.domain's setter provides.
0574

0575 **Public Key Credential**
0576 Generically, a credential is data one entity presents to another
0577 in order to authenticate the former to the latter [RFC4949]. A
0578 WebAuthn public key credential is a { identifier, type } pair
0579 identifying authentication information established by the
058C authenticator and the Relying Party, together, at registration
0581 time. The authentication information consists of an asymmetric
0582 key pair, where the public key portion is returned to the
0583 Relying Party, who then stores it in conjunction with the
0584 present user's account. The authenticator maps the private key
0585 portion to the Relying Party's RP ID and stores it.
0586 Subsequently, only that Relying Party, as identified by its RP
0587 ID, is able to employ the public key credential in
0588 authentication ceremonies, via the get() method. The Relying
0589 Party uses its stored copy of the credential public key to
059C verify the resultant authentication assertion.
0591

0592 **Test of User Presence**
0593 A test of user presence is a simple form of authorization
0594 gesture and technical process where a user interacts with an
0595 authenticator by (typically) simply touching it (other
0596 modalities may also exist), yielding a boolean result. Note that
0597 this does not constitute user verification because a user
0598 presence test, by definition, is not capable of biometric
0599 recognition, nor does it involve the presentation of a shared
060C secret such as a password or PIN.
0601

0602 **User Consent**
0603 User consent means the user agrees with what they are being
0604 asked, i.e., it encompasses reading and understanding prompts.
0605 An authorization gesture is a ceremony component often employed
0606 to indicate user consent.
0607

0608 **User Handle**
0609 The user handle is specified by a Relying Party and is a unique
061C identifier for a user account with that Relying Party. A user
0611 handle is an opaque byte sequence with a maximum size of 64
0612 bytes.
0613

0614 The user handle is not meant to be displayed to the user, but is
0615 used by the Relying Party to control the number of credentials -
0616 an authenticator will never contain more than one credential for
0617 a given Relying Party under the same user handle.
0618

0619 **User Verification**
062C The technical process by which an authenticator locally
0621 authorizes the invocation of the authenticatorMakeCredential and
0622 authenticatorGetAssertion operations. User verification may be
0623 instigated through various authorization gesture modalities; for
0624 example, through a touch plus pin code, password entry, or
0625 biometric recognition (e.g., presenting a fingerprint)
0626 [ISOBiometricVocabulary]. The intent is to be able to
0627 distinguish individual users. Note that invocation of the
0628 authenticatorMakeCredential and authenticatorGetAssertion
0629 operations implies use of key material managed by the

0625 authenticator. Note that for security, user verification and use
0630 of credential private keys must occur within a single logical
0631 security boundary defining the authenticator.

0632
0633 **User Present**

0634 UP

0635 Upon successful completion of a user presence test, the user is
0636 said to be "present".

0637
0638 **User Verified**

0639 UV

0640 Upon successful completion of a user verification process, the
0641 user is said to be "verified".

0642
0643 **WebAuthn Client**

0644 Also referred to herein as simply a client. See also Conforming
0645 User Agent.

0646
0647 **5. Web Authentication API**

0648 This section normatively specifies the API for creating and using
0649 public key credentials. The basic idea is that the credentials belong
0650 to the user and are managed by an authenticator, with which the Relying
0651 Party interacts through the client (consisting of the browser and
0652 underlying OS platform). Scripts can (with the user's consent) request
0653 the browser to create a new credential for future use by the Relying
0654 Party. Scripts can also request the user's permission to perform
0655 authentication operations with an existing credential. All such
0656 operations are performed in the authenticator and are mediated by the
0657 browser and/or platform on the user's behalf. At no point does the
0658 script get access to the credentials themselves; it only gets
0659 information about the credentials in the form of objects.

0660 In addition to the above script interface, the authenticator may
0661 implement (or come with client software that implements) a user
0662 interface for management. Such an interface may be used, for example,
0663 to reset the authenticator to a clean state or to inspect the current
0664 state of the authenticator. In other words, such an interface is
0665 similar to the user interfaces provided by browsers for managing user
0666 state such as history, saved passwords and cookies. Authenticator
0667 management actions such as credential deletion are considered to be the
0668 responsibility of such a user interface and are deliberately omitted
0669 from the API exposed to scripts.

0670 The security properties of this API are provided by the client and the
0671 authenticator working together. The authenticator, which holds and
0672 manages credentials, ensures that all operations are scoped to a
0673 particular origin, and cannot be replayed against a different origin,
0674 by incorporating the origin in its responses. Specifically, as defined
0675 in 6.2 Authenticator operations, the full origin of the requester is
0676 included, and signed over, in the attestation object produced when a
0677 new credential is created as well as in all assertions produced by
0678 WebAuthn credentials.

0679 Additionally, to maintain user privacy and prevent malicious Relying
0680 Parties from probing for the presence of public key credentials
0681 belonging to other Relying Parties, each credential is also associated
0682 with a Relying Party Identifier, or RP ID. This RP ID is provided by
0683 the client to the authenticator for all operations, and the
0684 authenticator ensures that credentials created by a Relying Party can
0685 only be used in operations requested by the same RP ID. Separating the
0686 origin from the RP ID in this way allows the API to be used in cases
0687 where a single Relying Party maintains multiple origins.

0688 The client facilitates these security measures by providing the Relying
0689 Party's origin and RP ID to the authenticator for each operation. Since
0690 this is an integral part of the WebAuthn security model, user agents
0691 only expose this API to callers in secure contexts.

0692 The Web Authentication API is defined by the union of the Web IDL

0630 authenticator. Note that for security, user verification and use
0631 of credential private keys must occur within a single logical
0632 security boundary defining the authenticator.

0633
0634 **User Present**

0635 UP

0636 Upon successful completion of a user presence test, the user is
0637 said to be "present".

0638
0639 **User Verified**

0640 UV

0641 Upon successful completion of a user verification process, the
0642 user is said to be "verified".

0643
0644 **WebAuthn Client**

0645 Also referred to herein as simply a client. See also Conforming
0646 User Agent.

0647
0648 **5. Web Authentication API**

0649 This section normatively specifies the API for creating and using
0650 public key credentials. The basic idea is that the credentials belong
0651 to the user and are managed by an authenticator, with which the Relying
0652 Party interacts through the client (consisting of the browser and
0653 underlying OS platform). Scripts can (with the user's consent) request
0654 the browser to create a new credential for future use by the Relying
0655 Party. Scripts can also request the user's permission to perform
0656 authentication operations with an existing credential. All such
0657 operations are performed in the authenticator and are mediated by the
0658 browser and/or platform on the user's behalf. At no point does the
0659 script get access to the credentials themselves; it only gets
0660 information about the credentials in the form of objects.

0661 In addition to the above script interface, the authenticator may
0662 implement (or come with client software that implements) a user
0663 interface for management. Such an interface may be used, for example,
0664 to reset the authenticator to a clean state or to inspect the current
0665 state of the authenticator. In other words, such an interface is
0666 similar to the user interfaces provided by browsers for managing user
0667 state such as history, saved passwords and cookies. Authenticator
0668 management actions such as credential deletion are considered to be the
0669 responsibility of such a user interface and are deliberately omitted
0670 from the API exposed to scripts.

0671 The security properties of this API are provided by the client and the
0672 authenticator working together. The authenticator, which holds and
0673 manages credentials, ensures that all operations are scoped to a
0674 particular origin, and cannot be replayed against a different origin,
0675 by incorporating the origin in its responses. Specifically, as defined
0676 in 6.2 Authenticator operations, the full origin of the requester is
0677 included, and signed over, in the attestation object produced when a
0678 new credential is created as well as in all assertions produced by
0679 WebAuthn credentials.

0680 Additionally, to maintain user privacy and prevent malicious Relying
0681 Parties from probing for the presence of public key credentials
0682 belonging to other Relying Parties, each credential is also associated
0683 with a Relying Party Identifier, or RP ID. This RP ID is provided by
0684 the client to the authenticator for all operations, and the
0685 authenticator ensures that credentials created by a Relying Party can
0686 only be used in operations requested by the same RP ID. Separating the
0687 origin from the RP ID in this way allows the API to be used in cases
0688 where a single Relying Party maintains multiple origins.

0689 The client facilitates these security measures by providing the Relying
0690 Party's origin and RP ID to the authenticator for each operation. Since
0691 this is an integral part of the WebAuthn security model, user agents
0692 only expose this API to callers in secure contexts.

0693 The Web Authentication API is defined by the union of the Web IDL

069E fragments presented in the following sections. A combined IDL listing
070C is given in the IDL Index.
0701
0702 5.1. PublicKeyCredential Interface
0703
0704 The PublicKeyCredential interface inherits from Credential
0705 [CREDENTIAL-MANAGEMENT-1], and contains the attributes that are
0706 returned to the caller when a new credential is created, or a new
0707 assertion is requested.
0708 [SecureContext, Exposed=Window]
0709 interface PublicKeyCredential : Credential {
0710 [SameObject] readonly attribute ArrayBuffer rawId;
0711 [SameObject] readonly attribute AuthenticatorResponse response;
0712 [SameObject] readonly attribute AuthenticationExtensions clientExtensionResu
0713 lts;
0714 };
0715
0716 id
0717 This attribute is inherited from Credential, though
0718 PublicKeyCredential overrides Credential's getter, instead
0719 returning the base64url encoding of the data contained in the
0720 object's [[identifier]] internal slot.
0721
0722 rawId
0723 This attribute returns the ArrayBuffer contained in the
0724 [[identifier]] internal slot.
0725
0726 response, of type AuthenticatorResponse, readonly
0727 This attribute contains the authenticator's response to the
0728 client's request to either create a public key credential, or
0729 generate an authentication assertion. If the PublicKeyCredential
0730 is created in response to create(), this attribute's value will
0731 be an AuthenticatorAttestationResponse, otherwise, the
0732 PublicKeyCredential was created in response to get(), and this
0733 attribute's value will be an AuthenticatorAssertionResponse.
0734
0735 clientExtensionResults, of type AuthenticationExtensions, readonly
0736 This attribute contains a map containing extension identifier ->
0737 client extension output entries produced by the extension's
0738 client extension processing.
0739
0740 [[type]]
0741 The PublicKeyCredential interface object's [[type]] internal
0742 slot's value is the string "public-key".
0743
0744 Note: This is reflected via the type attribute getter inherited
0745 from Credential.
0746
0747 [[discovery]]
0748 The PublicKeyCredential interface object's [[discovery]]
0749 internal slot's value is "remote".
0750
0751 [[identifier]]
0752 This internal slot contains an identifier for the credential,
0753 chosen by the platform with help from the authenticator. This
0754 identifier is used to look up credentials for use, and is
0755 therefore expected to be globally unique with high probability
0756 across all credentials of the same type, across all
0757 authenticators. This API does not constrain the format or length
0758 of this identifier, except that it must be sufficient for the
0759 platform to uniquely select a key. For example, an authenticator
0760 without on-board storage may create identifiers containing a
0761 credential private key wrapped with a symmetric key that is
0762 burned into the authenticator.
0763
0764 PublicKeyCredential's interface object inherits Credential's
0765 implementation of [[CollectFromCredentialStore]](options) and
0766 [[Store]](credential), and defines its own implementation of
0767 [[DiscoverFromExternalSource]](options) and [[Create]](options).
0768

070C fragments presented in the following sections. A combined IDL listing
0701 is given in the IDL Index.
0702
0703 5.1. PublicKeyCredential Interface
0704
0705 The PublicKeyCredential interface inherits from Credential
0706 [CREDENTIAL-MANAGEMENT-1], and contains the attributes that are
0707 returned to the caller when a new credential is created, or a new
0708 assertion is requested.
0709 [SecureContext, Exposed=Window]
0710 interface PublicKeyCredential : Credential {
0711 [SameObject] readonly attribute ArrayBuffer rawId;
0712 [SameObject] readonly attribute AuthenticatorResponse response;
0713 [SameObject] readonly attribute AuthenticationExtensions clientExtensionResu
0714 lts;
0715 };
0716
0717 id
0718 This attribute is inherited from Credential, though
0719 PublicKeyCredential overrides Credential's getter, instead
0720 returning the base64url encoding of the data contained in the
0721 object's [[identifier]] internal slot.
0722
0723 rawId
0724 This attribute returns the ArrayBuffer contained in the
0725 [[identifier]] internal slot.
0726
0727 response, of type AuthenticatorResponse, readonly
0728 This attribute contains the authenticator's response to the
0729 client's request to either create a public key credential, or
0730 generate an authentication assertion. If the PublicKeyCredential
0731 is created in response to create(), this attribute's value will
0732 be an AuthenticatorAttestationResponse, otherwise, the
0733 PublicKeyCredential was created in response to get(), and this
0734 attribute's value will be an AuthenticatorAssertionResponse.
0735
0736 clientExtensionResults, of type AuthenticationExtensions, readonly
0737 This attribute contains a map containing extension identifier ->
0738 client extension output entries produced by the extension's
0739 client extension processing.
0740
0741 [[type]]
0742 The PublicKeyCredential interface object's [[type]] internal
0743 slot's value is the string "public-key".
0744
0745 Note: This is reflected via the type attribute getter inherited
0746 from Credential.
0747
0748 [[discovery]]
0749 The PublicKeyCredential interface object's [[discovery]]
0750 internal slot's value is "remote".
0751
0752 [[identifier]]
0753 This internal slot contains an identifier for the credential,
0754 chosen by the platform with help from the authenticator. This
0755 identifier is used to look up credentials for use, and is
0756 therefore expected to be globally unique with high probability
0757 across all credentials of the same type, across all
0758 authenticators. This API does not constrain the format or length
0759 of this identifier, except that it must be sufficient for the
0760 platform to uniquely select a key. For example, an authenticator
0761 without on-board storage may create identifiers containing a
0762 credential private key wrapped with a symmetric key that is
0763 burned into the authenticator.
0764
0765 PublicKeyCredential's interface object inherits Credential's
0766 implementation of [[CollectFromCredentialStore]](options) and
0767 [[Store]](credential), and defines its own implementation of
0768 [[DiscoverFromExternalSource]](options) and [[Create]](options).
0769

0765 5.1.1. CredentialCreationOptions Extension
0770
0771 To support registration via navigator.credentials.create(), this
0772 document extends the CredentialCreationOptions dictionary as follows:
0773 partial dictionary CredentialCreationOptions {
0774 MakePublicKeyCredentialOptions publicKey;
0775 };
0776
0777 5.1.2. CredentialRequestOptions Extension
0778
0779 To support obtaining assertions via navigator.credentials.get(), this
0780 document extends the CredentialRequestOptions dictionary as follows:
0781 partial dictionary CredentialRequestOptions {
0782 PublicKeyCredentialRequestOptions publicKey;
0783 };
0784
0785 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](options)
0786 method
0787
0788 PublicKeyCredential's interface object's implementation of the
0789 [[Create]](options) method allows scripts to call
0790 navigator.credentials.create() to request the creation of a new
0791 credential key pair and PublicKeyCredential, managed by an
0792 authenticator. On success, the returned promise will be resolved with a
0793 PublicKeyCredential containing an AuthenticatorAttestationResponse
0794 object.
0795
0796 Note: This algorithm is synchronous; the Promise resolution/rejection
0797 is handled by navigator.credentials.create().
0798
0799 This method accepts a single argument:
0800
0801 options
0802 This argument is a CredentialCreationOptions object whose
0803 options.publicKey member contains a
0804 MakePublicKeyCredentialOptions object specifying the desired
0805 attributes of the to-be-created public key credential.
0806
0807 When this method is invoked, the user agent MUST execute the following
0808 algorithm:
0809 1. Assert: options.publicKey is present.
0810 2. Let options be the value of options.publicKey.
0811 3. If any of the name member of options.rp, the name member of
0812 options.user, the displayName member of options.user, or the id
0813 member of options.user are not present, return a TypeError simple
0814 exception.
0815 4. If the timeout member of options is present, check if its value
0816 lies within a reasonable range as defined by the platform and if
0817 not, correct it to the closest value lying within that range. Set
0818 adjustedTimeout to this adjusted value. If the timeout member of
0819 options is not present, then set adjustedTimeout to a
0820 platform-specific default.
0821 5. Let global be the PublicKeyCredential's interface object's
0822 environment settings object's global object.
0823 6. Let callerOrigin be the origin specified by this
0824 PublicKeyCredential interface object's relevant settings object. If
0825 callerOrigin is an opaque origin, return a DOMException whose name
0826 is "NotAllowedError", and terminate this algorithm.
0827 7. Let effectiveDomain be the callerOrigin's effective domain. If
0828 effective domain is not a valid domain, then return a DOMException
0829 whose name is "SecurityError" and terminate this algorithm.
0830 Note: An effective domain may resolve to a host, which can be
0831 represented in various manners, such as domain, ipv4 address, ipv6
0832 address, opaque host, or empty host. Only the domain format of host
0833 is allowed here.
0834 8. Let rpId be effectiveDomain.
0835 9. If options.rp.id is present:
0836 1. If options.rp.id is not a registrable domain suffix of and is
0837 not equal to effectiveDomain, return a DOMException whose name
0838 is "SecurityError", and terminate this algorithm.
0839

0770 5.1.1. CredentialCreationOptions Extension
0771
0772 To support registration via navigator.credentials.create(), this
0773 document extends the CredentialCreationOptions dictionary as follows:
0774 partial dictionary CredentialCreationOptions {
0775 MakePublicKeyCredentialOptions publicKey;
0776 };
0777
0778 5.1.2. CredentialRequestOptions Extension
0779
0780 To support obtaining assertions via navigator.credentials.get(), this
0781 document extends the CredentialRequestOptions dictionary as follows:
0782 partial dictionary CredentialRequestOptions {
0783 PublicKeyCredentialRequestOptions publicKey;
0784 };
0785
0786 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](options)
0787 method
0788
0789 PublicKeyCredential's interface object's implementation of the
0790 [[Create]](options) method allows scripts to call
0791 navigator.credentials.create() to request the creation of a new
0792 credential key pair and PublicKeyCredential, managed by an
0793 authenticator. On success, the returned promise will be resolved with a
0794 PublicKeyCredential containing an AuthenticatorAttestationResponse
0795 object.
0796
0797 Note: This algorithm is synchronous; the Promise resolution/rejection
0798 is handled by navigator.credentials.create().
0799
0800 This method accepts a single argument:
0801
0802 options
0803 This argument is a CredentialCreationOptions object whose
0804 options.publicKey member contains a
0805 MakePublicKeyCredentialOptions object specifying the desired
0806 attributes of the to-be-created public key credential.
0807
0808 When this method is invoked, the user agent MUST execute the following
0809 algorithm:
0810 1. Assert: options.publicKey is present.
0811 2. Let options be the value of options.publicKey.
0812 3. If any of the name member of options.rp, the name member of
0813 options.user, the displayName member of options.user, or the id
0814 member of options.user are not present, return a TypeError simple
0815 exception.
0816 4. If the timeout member of options is present, check if its value
0817 lies within a reasonable range as defined by the platform and if
0818 not, correct it to the closest value lying within that range. Set
0819 adjustedTimeout to this adjusted value. If the timeout member of
0820 options is not present, then set adjustedTimeout to a
0821 platform-specific default.
0822 5. Let global be the PublicKeyCredential's interface object's
0823 environment settings object's global object.
0824 6. Let callerOrigin be the origin specified by this
0825 PublicKeyCredential interface object's relevant settings object. If
0826 callerOrigin is an opaque origin, return a DOMException whose name
0827 is "NotAllowedError", and terminate this algorithm.
0828 7. Let effectiveDomain be the callerOrigin's effective domain. If
0829 effective domain is not a valid domain, then return a DOMException
0830 whose name is "SecurityError" and terminate this algorithm.
0831 Note: An effective domain may resolve to a host, which can be
0832 represented in various manners, such as domain, ipv4 address, ipv6
0833 address, opaque host, or empty host. Only the domain format of host
0834 is allowed here.
0835 8. If options.rp.id
0836 is present
0837 If options.rp.id is not a registrable domain suffix of and
0838 is not equal to effectiveDomain, return a DOMException
0839

0835 2. Set `rpId` to `options.rp.id`.
 0840 Note: `rpId` represents the caller's RP ID. The RP ID defaults
 0841 to being the caller's origin's effective domain unless the
 caller has explicitly set `options.rp.id` when calling `create()`.

0842
 0843 10. Let `credTypesAndPubKeyAlgs` be a new list whose items are pairs of
 0844 `PublicKeyCredentialType` and a `COSEAlgorithmIdentifier`.
 0845 11. For each current of `options.pubKeyCredParams`:
 0846 1. If `current.type` does not contain a `PublicKeyCredentialType`
 0847 supported by this implementation, then continue.
 0848 2. Let `alg` be `current.alg`.
 0849 3. Append the pair of `current.type` and `alg` to
 0850 `credTypesAndPubKeyAlgs`.
 0851 12. If `credTypesAndPubKeyAlgs` is empty and `options.pubKeyCredParams` is
 0852 not empty, cancel the timer started in step 2, return a
 0853 `DOMException` whose name is "NotSupportedError", and terminate this
 0854 algorithm.
 0855 13. Let `clientExtensions` be a new map and let `authenticatorExtensions`
 0856 be a new map.
 0857 14. If the `extensions` member of `options` is present, then for each
 0858 `extensionId` -> `clientExtensionInput` of `options.extensions`:
 0859 1. If `extensionId` is not supported by this client platform or is
 0860 not a registration extension, then continue.
 0861 2. Set `clientExtensions[extensionId]` to `clientExtensionInput`.
 0862 3. If `extensionId` is not an authenticator extension, then
 0863 continue.
 0864 4. Let `authenticatorExtensionInput` be the (CBOR) result of
 0865 running `extensionId`'s client extension processing algorithm on
 0866 `clientExtensionInput`. If the algorithm returned an error,
 0867 continue.
 0868 5. Set `authenticatorExtensions[extensionId]` to the base64url
 0869 encoding of `authenticatorExtensionInput`.
 0870 15. Let `collectedClientData` be a new `CollectedClientData` instance whose
 0871 fields are:
 0872
 0873 `challenge`
 0874 The base64url encoding of `options.challenge`.
 0875
 0876 `origin`
 0877 The serialization of `callerOrigin`.
 0878
 0879 `hashAlgorithm`
 0880 The recognized algorithm name of the hash algorithm
 0881 selected by the client for generating the hash of the
 0882 serialized client data.
 0883
 0884 `tokenBindingId`
 0885 The Token Binding ID associated with `callerOrigin`, if one
 0886 is available.
 0887
 0888 `clientExtensions`
 0889 `clientExtensions`
 0890
 0891 `authenticatorExtensions`
 0892 `authenticatorExtensions`
 0893

0894 16. Let `clientDataJSON` be the JSON-serialized client data constructed
 0895 from `collectedClientData`.
 0896 17. Let `clientDataHash` be the hash of the serialized client data
 0897 represented by `clientDataJSON`.
 0898 18. Let `currentlyAvailableAuthenticators` be a new ordered set
 0899 consisting of all authenticators currently available on this
 0900 platform.
 0901 19. Let `selectedAuthenticators` be a new ordered set.
 0902 20. If `currentlyAvailableAuthenticators` is empty, return a `DOMException`
 0903 whose name is "NotFoundError", and terminate this algorithm.

0840 whose name is "SecurityError", and terminate this
 0841 algorithm.
 0842
 0843 Is not present
 0844 Set `options.rp.id` to `effectiveDomain`.
 0845
 0846 Note: `options.rp.id` represents the caller's RP ID. The RP ID
 0847 defaults to being the caller's origin's effective domain unless the
 0848 caller has explicitly set `options.rp.id` when calling `create()`.
 0849 9. Let `credTypesAndPubKeyAlgs` be a new list whose items are pairs of
 0850 `PublicKeyCredentialType` and a `COSEAlgorithmIdentifier`.
 0851 10. For each current of `options.pubKeyCredParams`:
 0852 1. If `current.type` does not contain a `PublicKeyCredentialType`
 0853 supported by this implementation, then continue.
 0854 2. Let `alg` be `current.alg`.
 0855 3. Append the pair of `current.type` and `alg` to
 0856 `credTypesAndPubKeyAlgs`.
 0857 11. If `credTypesAndPubKeyAlgs` is empty and `options.pubKeyCredParams` is
 0858 not empty, return a `DOMException` whose name is "NotSupportedError",
 0859 and terminate this algorithm.
 0860 12. Let `clientExtensions` be a new map and let `authenticatorExtensions`
 0861 be a new map.
 0862 13. If the `extensions` member of `options` is present, then for each
 0863 `extensionId` -> `clientExtensionInput` of `options.extensions`:
 0864 1. If `extensionId` is not supported by this client platform or is
 0865 not a registration extension, then continue.
 0866 2. Set `clientExtensions[extensionId]` to `clientExtensionInput`.
 0867 3. If `extensionId` is not an authenticator extension, then
 0868 continue.
 0869 4. Let `authenticatorExtensionInput` be the (CBOR) result of
 0870 running `extensionId`'s client extension processing algorithm on
 0871 `clientExtensionInput`. If the algorithm returned an error,
 0872 continue.
 0873 5. Set `authenticatorExtensions[extensionId]` to the base64url
 0874 encoding of `authenticatorExtensionInput`.
 0875 14. Let `collectedClientData` be a new `CollectedClientData` instance whose
 0876 fields are:
 0877
 0878 `challenge`
 0879 The base64url encoding of `options.challenge`.
 0880
 0881 `origin`
 0882 The serialization of `callerOrigin`.
 0883
 0884 `hashAlgorithm`
 0885 The recognized algorithm name of the hash algorithm
 0886 selected by the client for generating the hash of the
 0887 serialized client data.
 0888
 0889 `tokenBindingId`
 0890 The Token Binding ID associated with `callerOrigin`, if one
 0891 is available.
 0892
 0893 `clientExtensions`
 0894 `clientExtensions`
 0895
 0896 `authenticatorExtensions`
 0897 `authenticatorExtensions`
 0898

0899 15. Let `clientDataJSON` be the JSON-serialized client data constructed
 0900 from `collectedClientData`.
 0901 16. Let `clientDataHash` be the hash of the serialized client data
 0902 represented by `clientDataJSON`.
 0903 17. Let `currentlyAvailableAuthenticators` be a new ordered set
 0904 consisting of all authenticators currently available on this
 0905 platform.
 0906 18. Let `selectedAuthenticators` be a new ordered set.
 0907 19. If `currentlyAvailableAuthenticators` is empty, return a `DOMException`
 0908 whose name is "NotFoundError", and terminate this algorithm.

0904 21. If options.authenticatorSelection is present, iterate through
0905 currentlyAvailableAuthenticators and do the following for each
0906 authenticator:
0907 1. If authenticatorAttachment is present and its value is not
0908 equal to authenticator's attachment modality, continue.
0909 2. If requireResidentKey is set to true and the authenticator is
0910 not capable of storing a Client-Side-Resident Credential
0911 Private Key, continue.
0912 3. If requireUserVerification is set to true and the
0913 authenticator is not capable of performing user verification,
0914 continue.
0915 4. Append authenticator to selectedAuthenticators.
0916 22. If selectedAuthenticators is empty, return a DOMException whose
0917 name is "ConstraintError", and terminate this algorithm.
0918 23. Let issuedRequests be a new ordered set.
0919 24. For each authenticator in currentlyAvailableAuthenticators:
0920 1. Let excludeCredentialDescriptorList be a new list.
0921 2. For each credential descriptor C in
0922 options.excludeCredentials:
0923 1. If C.transports is not empty, and authenticator is
0924 connected over a transport not mentioned in C.transports,
0925 the client MAY continue.
0926 2. Otherwise, Append C to excludeCredentialDescriptorList.
0927 3. In parallel, invoke the authenticatorMakeCredential operation
0928 on authenticator with rpId, clientDataHash, options.rp,
0929 options.user,
0930 options.authenticatorSelection.requireResidentKey,
0931 credTypesAndPubKeyAlgs, excludeCredentialDescriptorList, and
0932 authenticatorExtensions as parameters.
0933 4. Append authenticator to issuedRequests.
0934 25. Start a timer for adjustedTimeout milliseconds. Then execute the
0935 following steps in parallel. The task source for these tasks is the
0936 dom manipulation task source.
0937 26. While issuedRequests is not empty, perform the following actions
0938 depending upon the adjustedTimeout timer and responses from the
0939 authenticators:
0940
0941 If the adjustedTimeout timer expires,
0942 For each authenticator in issuedRequests invoke the
0943 authenticatorCancel operation on authenticator and remove
0944 authenticator from issuedRequests.
0945
0946 If any authenticator returns a status indicating that the user
0947 cancelled the operation,
0948
0949 1. Remove authenticator from issuedRequests.
0950 2. For each remaining authenticator in issuedRequests invoke
0951 the authenticatorCancel operation on authenticator and
0952 remove it from issuedRequests.
0953
0954 If any authenticator returns an error status,
0955 Remove authenticator from issuedRequests.
0956
0957 If any authenticator indicates success,
0958
0959 1. Remove authenticator from issuedRequests.
0960 2. Let attestationObject be a new ArrayBuffer, created using
0961 global's %ArrayBuffer%, containing the bytes of the value
0962 returned from the successful authenticatorMakeCredential
0963 operation (which is attObj, as defined in 6.3.4
0964 Generating an Attestation Object).
0965 3. Let id be
0966 attestationObject.authData.attestedCredentialData.credent
0967 ialId.
0968 4. Let value be a new PublicKeyCredential object associated
0969 with global whose fields are:

0909 20. If options.authenticatorSelection is present, iterate through
0910 currentlyAvailableAuthenticators and do the following for each
0911 authenticator:
0912 1. If authenticatorAttachment is present and its value is not
0913 equal to authenticator's attachment modality, continue.
0914 2. If requireResidentKey is set to true and the authenticator is
0915 not capable of storing a Client-Side-Resident Credential
0916 Private Key, continue.
0917 3. If requireUserVerification is set to true and the
0918 authenticator is not capable of performing user verification,
0919 continue.
0920 4. Append authenticator to selectedAuthenticators.
0921 21. If selectedAuthenticators is empty, return a DOMException whose
0922 name is "ConstraintError", and terminate this algorithm.
0923 22. Let issuedRequests be a new ordered set.
0924 23. For each authenticator in currentlyAvailableAuthenticators:
0925 1. Let excludeCredentialDescriptorList be a new list.
0926 2. For each credential descriptor C in
0927 options.excludeCredentials:
0928 1. If C.transports is not empty, and authenticator is
0929 connected over a transport not mentioned in C.transports,
0930 the client MAY continue.
0931 2. Otherwise, Append C to excludeCredentialDescriptorList.
0932 3. Invoke the authenticatorMakeCredential operation on
0933 authenticator with clientDataHash, options.rp, options.user,
0934 options.authenticatorSelection.requireResidentKey,
0935 options.authenticatorSelection.requireUserVerification,
0936 credTypesAndPubKeyAlgs, excludeCredentialDescriptorList, and
0937 authenticatorExtensions as parameters.
0938 4. Append authenticator to issuedRequests.
0939 24. Start a timer for adjustedTimeout milliseconds.
0940 25. While issuedRequests is not empty, perform the following actions
0941 depending upon the adjustedTimeout timer and responses from the
0942 authenticators:
0943
0944 If the adjustedTimeout timer expires,
0945 For each authenticator in issuedRequests invoke the
0946 authenticatorCancel operation on authenticator and remove
0947 authenticator from issuedRequests.
0948
0949 If any authenticator returns a status indicating that the user
0950 cancelled the operation,
0951
0952 1. Remove authenticator from issuedRequests.
0953 2. For each remaining authenticator in issuedRequests invoke
0954 the authenticatorCancel operation on authenticator and
0955 remove it from issuedRequests.
0956
0957 If any authenticator returns an error status,
0958 Remove authenticator from issuedRequests.
0959
0960 If any authenticator indicates success,
0961
0962 1. Remove authenticator from issuedRequests.
0963 2. Let credentialCreationData be a struct whose items are:
0964
0965 attestationObjectResult
0966 whose value is the bytes returned from the
0967 successful authenticatorMakeCredential
0968 operation.
0969
0970 Note: this value is attObj, as defined in
0971 6.3.4 Generating an Attestation Object.
0972
0973 clientDataJSONResult
0974 whose value is the bytes of clientDataJSON.
0975

0970
0971
0972 [[identifier]]
0973 id
0974
0975 response
0976 A new AuthenticatorAttestationResponse object
 associated with global whose fields are:

0977
0978 clientDataJSON
0979 A new ArrayBuffer, created using
0980 global's %ArrayBuffer%, containing the
0981 bytes of clientDataJSON.

0982
0983 attestationObject
0984 attestationObject

0985
0986 clientExtensionResults
0987 A new AuthenticationExtensions object
0988 containing the extension identifier -> client
0989 extension output entries created by running
0990 each extension's client extension processing
0991 algorithm to create the client extension
0992 outputs, for each client extension in
0993 clientDataJSON.clientExtensions.

0994
0995 5. For each remaining authenticator in issuedRequests invoke

0996
0997 the authenticatorCancel operation on authenticator and
0998 remove it from issuedRequests.
0999 6. Return value and terminate this algorithm.

1000
1001 27. Return a DOMException whose name is "NotAllowedError".

0976 extensionOutputsMap
0977 whose value is an ordered map with keys of
0978 type extension identifier and values of type
0979 client extension output. extensionOutputsMap's
0980 entries are created by running each
0981 extension's client extension processing
0982 algorithm to create the client extension
0983 outputs, for each client extension in
0984 clientDataJSON.clientExtensions.

0985
0986 3. Let constructCredentialAlg be an algorithm that takes a
0987 global object global, and whose steps are:
0988 1. Let attestationObject be a new ArrayBuffer, created
0989 using global's %ArrayBuffer%, containing the bytes
0990 of credentialCreationData.attestationObjectResult's
0991 value.
0992 2. Let id be
0993 attestationObject.authData.attestedCredentialData.cre
0994 dentialId.
0995 3. Let pubKeyCred be a new PublicKeyCredential object
0996 associated with global whose fields are:

0997 [[identifier]]
0998 id
0999
1000 response
1001 A new AuthenticatorAttestationResponse
1002 object associated with global whose
1003 fields are:
1004
1005 clientDataJSON
1006 A new ArrayBuffer, created using
1007 global's %ArrayBuffer%, containing
1008 the bytes of
1009 credentialCreationData.clientDataJ
1010 SONResult.
1011
1012 attestationObject
1013 attestationObject

1014
1015 clientExtensionResults
1016 A new AuthenticationExtensions object
1017 associated with global containing
1018 credentialCreationData.extensionOutputsM
1019 ap's value.

1020
1021 Note:
1022 credentialCreationData.extensionOutputsM
1023 ap is an ordered map whose keys are all
1024 of type extension identifiers and values
1025 are all of type client extension output.
1026 Thus the latter is implicitly a record
1027 type, which is the
1028 AuthenticationExtensions's type.

1029
1030 4. Return pubKeyCred.
1031 4. For each remaining authenticator in issuedRequests invoke
1032 the authenticatorCancel operation on authenticator and
1033 remove it from issuedRequests.
1034 5. Return constructCredentialAlg and terminate this
1035 algorithm.

1036
1037 Do we need to explicitly return both
1038 constructCredentialAlg and credentialCreationData here?
1039
1040
1041 26. Return a DOMException whose name is "NotAllowedError".
1042

1002 During the above process, the user agent SHOULD show some UI to the
 1003 user to guide them in the process of selecting and authorizing an
 1004 authenticator.
 1005
 1006 5.1.4. Use an existing credential to make an assertion
 1007
 1008 Relying Parties call navigator.credentials.get({publicKey:..., ...}) to
 1009 discover and use an existing public key credential, with the user's
 1010 consent. The script optionally specifies some criteria to indicate what
 1011 credential sources are acceptable to it. The user agent and/or platform
 1012 locates credential sources matching the specified criteria, and guides
 1013 the user to pick one that the script will be allowed to use. The user
 1014 may choose to decline the entire interaction even if a credential
 1015 source is present, for example to maintain privacy. If the user picks a
 1016 credential source, the user agent then uses 6.2.2 The
 1017 authenticatorGetAssertion operation to sign a Relying Party-provided
 1018 challenge and other collected data into an assertion, which is used as
 1019 a credential.
 1020
 1021 The get() implementation [CREDENTIAL-MANAGEMENT-1] calls
 1022 PublicKeyCredential.[[CollectFromCredentialStore]]() to collect any
 1023 credentials that should be available without user mediation (roughly,
 1024 this specification's authorization gesture), and if it doesn't find
 1025 exactly one of those, it calls
 1026 PublicKeyCredential.[[DiscoverFromExternalSource]]() to have the user
 1027 select a credential source.
 1028
 1029 Since this specification requires an authorization gesture to create
 1030 any credentials,
 1031 PublicKeyCredential.[[CollectFromCredentialStore]](options) inherits
 1032 the default behavior of Credential.[[CollectFromCredentialStore]](), of
 1033 returning an empty set.
 1034
 1035 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](options)
 1036 method
 1037
 1038 When the PublicKeyCredential.[[DiscoverFromExternalSource]](options)
 1039 method is invoked, the user agent MUST:
 1040 1. Assert: options.publicKey is present.
 1041 2. Let options be the value of options.publicKey.
 1042 3. If the timeout member of options is present, check if its value
 1043 lies within a reasonable range as defined by the platform and if
 1044 not, correct it to the closest value lying within that range. Set
 1045 adjustedTimeout to this adjusted value. If the timeout member of
 1046 options is not present, then set adjustedTimeout to a
 1047 platform-specific default.
 1048 4. Let global be the PublicKeyCredential's interface object's
 1049 **environment settings object's** global object.
 1050 5. Let callerOrigin be the origin specified by this
 1051 PublicKeyCredential interface object's relevant settings object. If
 1052 callerOrigin is an opaque origin, return a DOMException whose name
 1053 is "NotAllowedError", and terminate this algorithm.
 1054 6. Let effectiveDomain be the callerOrigin's effective domain. If
 1055 effective domain is not a valid domain, then return a DOMException
 1056 whose name is "SecurityError" and terminate this algorithm.
 1057 Note: An effective domain may resolve to a host, which can be
 1058 represented in various manners, such as domain, ipv4 address, ipv6
 1059 address, opaque host, or empty host. Only the domain format of host
 1060 is allowed here.
 1061 7. If options.rpld is not present, then set rpld to effectiveDomain.
 1062 Otherwise:
 1063 1. If options.rpld is not a registrable domain suffix of and is
 1064 not equal to effectiveDomain, return a DOMException whose name
 1065 is "SecurityError", and terminate this algorithm.
 1066 2. Set rpld to options.rpld.
 1067 Note: rpld represents the caller's RP ID. The RP ID defaults
 1068 to being the caller's origin's effective domain unless the
 1069 caller has explicitly set options.rpld when calling get().
 1070 8. Let clientExtensions be a new map and let authenticatorExtensions
 1071 be a new map.

1043 During the above process, the user agent SHOULD show some UI to the
 1044 user to guide them in the process of selecting and authorizing an
 1045 authenticator.
 1046
 1047 5.1.4. Use an existing credential to make an assertion
 1048
 1049 Relying Parties call navigator.credentials.get({publicKey:..., ...}) to
 1050 discover and use an existing public key credential, with the user's
 1051 consent. The script optionally specifies some criteria to indicate what
 1052 credential sources are acceptable to it. The user agent and/or platform
 1053 locates credential sources matching the specified criteria, and guides
 1054 the user to pick one that the script will be allowed to use. The user
 1055 may choose to decline the entire interaction even if a credential
 1056 source is present, for example to maintain privacy. If the user picks a
 1057 credential source, the user agent then uses 6.2.2 The
 1058 authenticatorGetAssertion operation to sign a Relying Party-provided
 1059 challenge and other collected data into an assertion, which is used as
 1060 a credential.
 1061
 1062 The get() implementation [CREDENTIAL-MANAGEMENT-1] calls
 1063 PublicKeyCredential.[[CollectFromCredentialStore]]() to collect any
 1064 credentials that should be available without user mediation (roughly,
 1065 this specification's authorization gesture), and if it doesn't find
 1066 exactly one of those, it calls
 1067 PublicKeyCredential.[[DiscoverFromExternalSource]]() to have the user
 1068 select a credential source.
 1069
 1070 Since this specification requires an authorization gesture to create
 1071 any credentials,
 1072 PublicKeyCredential.[[CollectFromCredentialStore]](options) inherits
 1073 the default behavior of Credential.[[CollectFromCredentialStore]](), of
 1074 returning an empty set.
 1075
 1076 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](options)
 1077 method
 1078
 1079 When the PublicKeyCredential.[[DiscoverFromExternalSource]](options)
 1080 method is invoked, the user agent MUST:
 1081 1. Assert: options.publicKey is present.
 1082 2. Let options be the value of options.publicKey.
 1083 3. If the timeout member of options is present, check if its value
 1084 lies within a reasonable range as defined by the platform and if
 1085 not, correct it to the closest value lying within that range. Set
 1086 adjustedTimeout to this adjusted value. If the timeout member of
 1087 options is not present, then set adjustedTimeout to a
 1088 platform-specific default.
 1089 4. Let global be the PublicKeyCredential's interface object's **relevant**
 1090 global object.
 1091 5. Let callerOrigin be the origin specified by this
 1092 PublicKeyCredential interface object's relevant settings object. If
 1093 callerOrigin is an opaque origin, return a DOMException whose name
 1094 is "NotAllowedError", and terminate this algorithm.
 1095 6. Let effectiveDomain be the callerOrigin's effective domain. If
 1096 effective domain is not a valid domain, then return a DOMException
 1097 whose name is "SecurityError" and terminate this algorithm.
 1098 Note: An effective domain may resolve to a host, which can be
 1099 represented in various manners, such as domain, ipv4 address, ipv6
 1100 address, opaque host, or empty host. Only the domain format of host
 1101 is allowed here.
 1102 7. If options.rpld is not present, then set rpld to effectiveDomain.
 1103 Otherwise:
 1104 1. If options.rpld is not a registrable domain suffix of and is
 1105 not equal to effectiveDomain, return a DOMException whose name
 1106 is "SecurityError", and terminate this algorithm.
 1107 2. Set rpld to options.rpld.
 1108 Note: rpld represents the caller's RP ID. The RP ID defaults
 1109 to being the caller's origin's effective domain unless the
 1110 caller has explicitly set options.rpld when calling get().
 1111 8. Let clientExtensions be a new map and let authenticatorExtensions
 1112 be a new map.

1072 9. If the extensions member of options is present, then for each
1073 extensionId -> clientExtensionInput of options.extensions:
1074 1. If extensionId is not supported by this client platform or is
1075 not an authentication extension, then continue.
1076 2. Set clientExtensions[extensionId] to clientExtensionInput.
1077 3. If extensionId is not an authenticator extension, then
1078 continue.
1079 4. Let authenticatorExtensionInput be the (CBOR) result of
1080 running extensionId's client extension processing algorithm on
1081 clientExtensionInput. If the algorithm returned an error,
1082 continue.
1083 5. Set authenticatorExtensions[extensionId] to the base64url
1084 encoding of authenticatorExtensionInput.
1085 10. Let collectedClientData be a new CollectedClientData instance whose
1086 fields are:
1087 challenge
1088 The base64url encoding of options.challenge
1089 origin
1090 The serialization of callerOrigin.
1091 hashAlgorithm
1092 The recognized algorithm name of the hash algorithm
1093 selected by the client for generating the hash of the
1094 serialized client data
1095 tokenBindingId
1096 The Token Binding ID associated with callerOrigin, if one
1097 is available.
1098 clientExtensions
1099 clientExtensions
1100 authenticatorExtensions
1101 authenticatorExtensions
1102 11. Let clientDataJSON be the JSON-serialized client data constructed
1103 from collectedClientData.
1104 12. Let clientDataHash be the hash of the serialized client data
1105 represented by clientDataJSON.
1106 13. Let issuedRequests be a new ordered set.
1107 14. If there are no authenticators currently available on this
1108 platform, return a DOMException whose name is "NotFoundError", and
1109 terminate this algorithm.
1110 15. Let authenticator be a platform-specific handle whose value
1111 identifies an authenticator.
1112 16. For each authenticator currently available on this platform,
1113 perform the following steps:
1114 1. Let allowCredentialDescriptorList be a new list.
1115 2. If options.allowCredentials is not empty, execute a
1116 platform-specific procedure to determine which, if any, public
1117 key credentials described by options.allowCredentials are
1118 bound to this authenticator, by matching with rpId,
1119 options.allowCredentials.id, and
1120 options.allowCredentials.type. Set
1121 allowCredentialDescriptorList to this filtered list.
1122 3. If allowCredentialDescriptorList
1123 is not empty
1124 1. Let distinctTransports be a new ordered set.
1125 2. For each credential descriptor C in
1126 allowCredentialDescriptorList, append each value, if
1127 any, of C.transports to distinctTransports.
1128 Note: This will aggregate only distinct values of

1113 9. If the extensions member of options is present, then for each
1114 extensionId -> clientExtensionInput of options.extensions:
1115 1. If extensionId is not supported by this client platform or is
1116 not an authentication extension, then continue.
1117 2. Set clientExtensions[extensionId] to clientExtensionInput.
1118 3. If extensionId is not an authenticator extension, then
1119 continue.
1120 4. Let authenticatorExtensionInput be the (CBOR) result of
1121 running extensionId's client extension processing algorithm on
1122 clientExtensionInput. If the algorithm returned an error,
1123 continue.
1124 5. Set authenticatorExtensions[extensionId] to the base64url
1125 encoding of authenticatorExtensionInput.
1126 10. Let collectedClientData be a new CollectedClientData instance whose
1127 fields are:
1128 challenge
1129 The base64url encoding of options.challenge
1130 origin
1131 The serialization of callerOrigin.
1132 hashAlgorithm
1133 The recognized algorithm name of the hash algorithm
1134 selected by the client for generating the hash of the
1135 serialized client data
1136 tokenBindingId
1137 The Token Binding ID associated with callerOrigin, if one
1138 is available.
1139 clientExtensions
1140 clientExtensions
1141 authenticatorExtensions
1142 authenticatorExtensions
1143 11. Let clientDataJSON be the JSON-serialized client data constructed
1144 from collectedClientData.
1145 12. Let clientDataHash be the hash of the serialized client data
1146 represented by clientDataJSON.
1147 13. Let issuedRequests be a new ordered set.
1148 14. If there are no authenticators currently available on this
1149 platform, return a DOMException whose name is "NotFoundError", and
1150 terminate this algorithm.
1151 15. Let authenticator be a platform-specific handle whose value
1152 identifies an authenticator.
1153 16. For each authenticator currently available on this platform,
1154 perform the following steps:
1155 1. Let allowCredentialDescriptorList be a new list.
1156 2. If options.allowCredentials is not empty, execute a
1157 platform-specific procedure to determine which, if any, public
1158 key credentials described by options.allowCredentials are
1159 bound to this authenticator, by matching with rpId,
1160 options.allowCredentials.id, and
1161 options.allowCredentials.type. Set
1162 allowCredentialDescriptorList to this filtered list.
1163 3. If allowCredentialDescriptorList
1164 is not empty
1165 1. Let distinctTransports be a new ordered set.
1166 2. If allowCredentialDescriptorList has exactly one
1167 value, let savedCredentialId be a new ArrayBuffer,
1168 created using global's %ArrayBuffer%, and containing
1169 the bytes of allowCredentialDescriptorList[0].id.
1170 3. For each credential descriptor C in
1171 allowCredentialDescriptorList, append each value, if
1172 any, of C.transports to distinctTransports.
1173 Note: This will aggregate only distinct values of

1138 transports (for this authenticator) in
 1139 distinctTransports due to the properties of ordered
 1140 sets.
 1141 **3. If distinctTransports**
 1142
 1143 is not empty
 1144 The client selects one transport value
 1145 from distinctTransports, possibly
 1146 incorporating local configuration
 1147 knowledge of the appropriate transport
 1148 to use with authenticator in making its
 1149 selection.
 1150
 1151 Then, using transport, invoke in
 1152 parallel the authenticatorGetAssertion
 1153 operation on authenticator, with rpId,
 1154 clientDataHash,
 1155 allowCredentialDescriptorList, and
 1156 authenticatorExtensions as parameters.
 1157
 1158 is empty
 1159 Using local configuration knowledge of
 1160 the appropriate transport to use with
 1161 authenticator, invoke in parallel the
 1162 authenticatorGetAssertion operation on
 1163 authenticator with rpId, clientDataHash,
 1164 allowCredentialDescriptorList, and
 1165 clientExtensions as parameters.
 1166
 1167 is empty
 1168 Using local configuration knowledge of the
 1169 appropriate transport to use with authenticator,
 1170 invoke in parallel the authenticatorGetAssertion
 1171 operation on authenticator with rpId,
 1172 clientDataHash, and clientExtensions as parameters.
 1173
 1174 Note: In this case, the Relying Party did not supply
 1175 a list of acceptable credential descriptors. Thus
 1176 the authenticator is being asked to exercise any
 1177 credential it may possess that is bound to the
 1178 Relying Party, as identified by rpId.
 1179
 1180 4. Append authenticator to issuedRequests.
 1181 17. Start a timer for adjustedTimeout milliseconds. Then execute the
 1182 following steps in parallel. The task source for these tasks is the
 1183 dom manipulation task source.
 1184 **18. While issuedRequests is not empty, perform the following actions**
 1185 **depending upon the adjustedTimeout timer and responses from the**
 1186 **authenticators:**
 1187
 1188 If the adjustedTimeout timer expires,
 1189 For each authenticator in issuedRequests invoke the
 1190 authenticatorCancel operation on authenticator and **remove**
 1191 authenticator from issuedRequests.
 1192
 1193 If any authenticator returns a status indicating that the **user**
 1194 cancelled the operation,
 1195
 1196 1. Remove authenticator from issuedRequests.
 1197 2. For each remaining authenticator in issuedRequests **invoke**
 1198 **the authenticatorCancel operation on authenticator and**
 1199 **remove it from issuedRequests.**
 1200
 1201 If any authenticator returns an error status,
 1202 Remove authenticator from issuedRequests.
 1203
 1204 If any authenticator indicates success,
 1205
 1206 1. Remove authenticator from issuedRequests.
 1207 2. Let value be a new PublicKeyCredential associated **with**

1183 transports (for this authenticator) in
 1184 distinctTransports due to the properties of ordered
 1185 sets.
 1186 **4. If distinctTransports**
 1187
 1188 is not empty
 1189 The client selects one transport value
 1190 from distinctTransports, possibly
 1191 incorporating local configuration
 1192 knowledge of the appropriate transport
 1193 to use with authenticator in making its
 1194 selection.
 1195
 1196 Then, using transport, invoke in
 1197 parallel the authenticatorGetAssertion
 1198 operation on authenticator, with rpId,
 1199 clientDataHash,
 1200 allowCredentialDescriptorList, and
 1201 authenticatorExtensions as parameters.
 1202
 1203 is empty
 1204 Using local configuration knowledge of
 1205 the appropriate transport to use with
 1206 authenticator, invoke in parallel the
 1207 authenticatorGetAssertion operation on
 1208 authenticator with rpId, clientDataHash,
 1209 allowCredentialDescriptorList, and
 1210 clientExtensions as parameters.
 1211
 1212 is empty
 1213 Using local configuration knowledge of the
 1214 appropriate transport to use with authenticator,
 1215 invoke in parallel the authenticatorGetAssertion
 1216 operation on authenticator with rpId,
 1217 clientDataHash, and clientExtensions as parameters.
 1218
 1219 Note: In this case, the Relying Party did not supply
 1220 a list of acceptable credential descriptors. Thus
 1221 the authenticator is being asked to exercise any
 1222 credential it may possess that is bound to the
 1223 Relying Party, as identified by rpId.
 1224
 1225 4. Append authenticator to issuedRequests.
 1226 17. Start a timer for adjustedTimeout milliseconds. Then execute the
 1227 following steps in parallel. The task source for these tasks is the
 1228 dom manipulation task source.
 1229 **1. While issuedRequests is not empty, perform the following**
 1230 **actions depending upon the adjustedTimeout timer and responses**
 1231 **from the authenticators:**
 1232
 1233 If the adjustedTimeout timer expires,
 1234 For each authenticator in issuedRequests invoke the
 1235 authenticatorCancel operation on authenticator and
 1236 **remove** authenticator from issuedRequests.
 1237
 1238 If any authenticator returns a status indicating that the
 1239 **user** cancelled the operation,
 1240
 1241 1. Remove authenticator from issuedRequests.
 1242 2. For each remaining authenticator in issuedRequests
 1243 **invoke the authenticatorCancel operation on**
 1244 **authenticator and remove it from issuedRequests.**
 1245
 1246 If any authenticator returns an error status,
 1247 Remove authenticator from issuedRequests.
 1248
 1249 If any authenticator indicates success,
 1250
 1251 1. Remove authenticator from issuedRequests.
 1252 2. Let value be a new PublicKeyCredential associated

1208 global whose fields are:
 1209
 1210 [[identifier]]
 1211 A new ArrayBuffer, created using global's
 1212 %ArrayBuffer%, containing the bytes of the
 1213 credential ID returned from the successful
 1214 authenticatorGetAssertion operation, as
 1215 defined in 6.2.2 The

1216 authenticatorGetAssertion operation.
 1217
 1218 response
 1219 A new AuthenticatorAssertionResponse object
 1220 associated with global whose fields are:

1221 clientDataJSON
 1222 A new ArrayBuffer, created using
 1223 global's %ArrayBuffer%, containing the
 1224 bytes of clientDataJSON
 1225
 1226
 1227 authenticatorData
 1228 A new ArrayBuffer, created using
 1229 global's %ArrayBuffer%, containing the
 1230 bytes of the returned authenticatorData

1231 signature
 1232 A new ArrayBuffer, created using
 1233 global's %ArrayBuffer%, containing the
 1234 bytes of the returned signature
 1235

1236 userHandle
 1237 A new ArrayBuffer, created using
 1238 global's %ArrayBuffer%, containing the
 1239 user handle returned from the successful
 1240 authenticatorGetAssertion operation, as
 1241 defined in 6.2.2 The
 1242 authenticatorGetAssertion operation.
 1243

1244 clientExtensionResults
 1245 A new AuthenticationExtensions object
 1246 containing the extension identifier -> client
 1247 extension output entries created by running
 1248 each extension's client extension processing
 1249 algorithm to create the client extension
 1250 outputs, for each client extension in
 1251

1252 clientDataJSON.clientExtensions.
 1253
 1254 3. For each remaining authenticator in issuedRequests invoke
 1255 the authenticatorCancel operation on authenticator and
 1256 remove it from issuedRequests.
 1257 4. Return value and terminate this algorithm.
 1258
 1259 19. Return a DOMException whose name is "NotAllowedError".
 1260
 1261 During the above process, the user agent SHOULD show some UI to the
 1262 user to guide them in the process of selecting and authorizing an
 1263 authenticator with which to complete the operation.
 1264
 1265 5.1.5. Store an existing credential - PublicKeyCredential's
 1266 [[Store]](credential) method
 1267

1253 with global whose fields are:
 1254
 1255 [[identifier]]
 1256 Create a new ArrayBuffer, using global's
 1257 %ArrayBuffer%. If savedCredentialId
 1258 exists, set the value of the new
 1259 ArrayBuffer to be the bytes of
 1260 savedCredentialId. Otherwise, set the
 1261 value of the new ArrayBuffer to be the
 1262 bytes of the credential ID returned from
 1263 the successful authenticatorGetAssertion
 1264 operation, as defined in 6.2.2 The
 1265 authenticatorGetAssertion operation.
 1266
 1267 response
 1268 A new AuthenticatorAssertionResponse
 1269 object associated with global whose
 1270 fields are:

1271 clientDataJSON
 1272 A new ArrayBuffer, created using
 1273 global's %ArrayBuffer%, containing
 1274 the bytes of clientDataJSON.
 1275
 1276
 1277 authenticatorData
 1278 A new ArrayBuffer, created using
 1279 global's %ArrayBuffer%, containing
 1280 the bytes of the returned
 1281 authenticatorData.
 1282
 1283 signature
 1284 A new ArrayBuffer, created using
 1285 global's %ArrayBuffer%, containing
 1286 the bytes of the returned
 1287 signature.
 1288
 1289 userHandle
 1290 A new ArrayBuffer, created using
 1291 global's %ArrayBuffer%, containing
 1292 the user handle returned from the
 1293 successful
 1294 authenticatorGetAssertion
 1295 operation, as defined in 6.2.2
 1296 The authenticatorGetAssertion
 1297 operation.
 1298
 1299 clientExtensionResults
 1300 A new AuthenticationExtensions object
 1301 containing the extension identifier ->
 1302 client extension output entries created
 1303 by running each extension's client
 1304 extension processing algorithm to create
 1305 the client extension outputs, for each
 1306 client extension in
 1307 clientDataJSON.clientExtensions.
 1308
 1309 3. For each remaining authenticator in issuedRequests
 1310 invoke the authenticatorCancel operation on
 1311 authenticator and remove it from issuedRequests.
 1312 4. Return value and terminate this algorithm.
 1313
 1314 18. Return a DOMException whose name is "NotAllowedError".
 1315
 1316 During the above process, the user agent SHOULD show some UI to the
 1317 user to guide them in the process of selecting and authorizing an
 1318 authenticator with which to complete the operation.
 1319
 1320 5.1.5. Store an existing credential - PublicKeyCredential's
 1321 [[Store]](credential) method
 1322

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

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

This method accepts a single argument:

`credential`
This argument is a `PublicKeyCredential` object.

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

- Return a `DOMException` whose name is "NotSupportedError", and terminate this algorithm

5.1.6. Platform Authenticator Availability - `PublicKeyCredential`'s `isPlatformAuthenticatorAvailable()` method

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

- * Whether the user is running in private or incognito mode.
- * Whether the user has configured the client to not create such credentials.
- * Whether the user has previously expressed an unwillingness to create a new credential for this Relying Party, either through configuration or by declining a user interface prompt.
- * The user's explicitly stated intentions, determined through user interaction.

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

This method has no arguments and returns a boolean value.

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

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

5.2. Authenticator Responses (interface `AuthenticatorResponse`)

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

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

`clientDataJSON`, of type `ArrayBuffer`, readonly
This attribute contains a JSON serialization of the client data passed to the authenticator by the client in its call to either

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

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

This method accepts a single argument:

`credential`
This argument is a `PublicKeyCredential` object.

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

- Return a `DOMException` whose name is "NotSupportedError", and terminate this algorithm

5.1.6. Platform Authenticator Availability - `PublicKeyCredential`'s `isPlatformAuthenticatorAvailable()` method

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

- * Whether the user is running in private or incognito mode.
- * Whether the user has configured the client to not create such credentials.
- * Whether the user has previously expressed an unwillingness to create a new credential for this Relying Party, either through configuration or by declining a user interface prompt.
- * The user's explicitly stated intentions, determined through user interaction.

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

This method has no arguments and returns a boolean value.

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

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

5.2. Authenticator Responses (interface `AuthenticatorResponse`)

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

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

`clientDataJSON`, of type `ArrayBuffer`, readonly
This attribute contains a JSON serialization of the client data passed to the authenticator by the client in its call to either

```

133E create() or get().
133F
134C 5.2.1. Information about Public Key Credential (interface
134D AuthenticatorAttestationResponse)
134E
134F The AuthenticatorAttestationResponse interface represents the
1350 authenticator's response to a client's request for the creation of a
1351 new public key credential. It contains information about the new
1352 credential that can be used to identify it for later use, and metadata
1353 that can be used by the Relying Party to assess the characteristics of
1354 the credential during registration.
1355 [SecureContext, Exposed=Window]
1356 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
1357   [SameObject] readonly attribute ArrayBuffer attestationObject;
1358 };
1359
1360 clientDataJSON
1361 This attribute, inherited from AuthenticatorResponse, contains
1362 the JSON-serialized client data (see 6.3 Attestation) passed to
1363 the authenticator by the client in order to generate this
1364 credential. The exact JSON serialization must be preserved, as
1365 the hash of the serialized client data has been computed over
1366 it.
1367
1368 attestationObject, of type ArrayBuffer, readonly
1369 This attribute contains an attestation object, which is opaque
1370 to, and cryptographically protected against tampering by, the
1371 client. The attestation object contains both authenticator data
1372 and an attestation statement. The former contains the AAGUID, a
1373 unique credential ID, and the credential public key. The
1374 contents of the attestation statement are determined by the
1375 attestation statement format used by the authenticator. It also
1376 contains any additional information that the Relying Party's
1377 server requires to validate the attestation statement, as well
1378 as to decode and validate the authenticator data along with the
1379 JSON-serialized client data. For more details, see 6.3
1380 Attestation, 6.3.4 Generating an Attestation Object, and Figure
1381 3.
1382
1383 5.2.2. Web Authentication Assertion (interface
1384 AuthenticatorAssertionResponse)
1385
1386 The AuthenticatorAssertionResponse interface represents an
1387 authenticator's response to a client's request for generation of a new
1388 authentication assertion given the Relying Party's challenge and
1389 optional list of credentials it is aware of. This response contains a
1390 cryptographic signature proving possession of the credential private
1391 key, and optionally evidence of user consent to a specific transaction.
1392 [SecureContext, Exposed=Window]
1393 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
1394   [SameObject] readonly attribute ArrayBuffer authenticatorData;
1395   [SameObject] readonly attribute ArrayBuffer signature;
1396   [SameObject] readonly attribute ArrayBuffer userHandle;
1397 };
1398
1399 clientDataJSON
1400 This attribute, inherited from AuthenticatorResponse, contains
1401 the JSON-serialized client data (see 5.7.1 Client data used in
1402 WebAuthn signatures (dictionary CollectedClientData)) passed to
1403 the authenticator by the client in order to generate this
1404 assertion. The exact JSON serialization must be preserved, as
1405 the hash of the serialized client data has been computed over
1406 it.
1407
1408 authenticatorData, of type ArrayBuffer, readonly
1409 This attribute contains the authenticator data returned by the
1410 authenticator. See 6.1 Authenticator data.
1411
1412 signature, of type ArrayBuffer, readonly
1413 This attribute contains the raw signature returned from the

```

```

139E create() or get().
139F
140C 5.2.1. Information about Public Key Credential (interface
140D AuthenticatorAttestationResponse)
140E
140F The AuthenticatorAttestationResponse interface represents the
1410 authenticator's response to a client's request for the creation of a
1411 new public key credential. It contains information about the new
1412 credential that can be used to identify it for later use, and metadata
1413 that can be used by the Relying Party to assess the characteristics of
1414 the credential during registration.
1415 [SecureContext, Exposed=Window]
1416 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
1417   [SameObject] readonly attribute ArrayBuffer attestationObject;
1418 };
1419
1420 clientDataJSON
1421 This attribute, inherited from AuthenticatorResponse, contains
1422 the JSON-serialized client data (see 6.3 Attestation) passed to
1423 the authenticator by the client in order to generate this
1424 credential. The exact JSON serialization must be preserved, as
1425 the hash of the serialized client data has been computed over
1426 it.
1427
1428 attestationObject, of type ArrayBuffer, readonly
1429 This attribute contains an attestation object, which is opaque
1430 to, and cryptographically protected against tampering by, the
1431 client. The attestation object contains both authenticator data
1432 and an attestation statement. The former contains the AAGUID, a
1433 unique credential ID, and the credential public key. The
1434 contents of the attestation statement are determined by the
1435 attestation statement format used by the authenticator. It also
1436 contains any additional information that the Relying Party's
1437 server requires to validate the attestation statement, as well
1438 as to decode and validate the authenticator data along with the
1439 JSON-serialized client data. For more details, see 6.3
1440 Attestation, 6.3.4 Generating an Attestation Object, and Figure
1441 3.
1442
1443 5.2.2. Web Authentication Assertion (interface
1444 AuthenticatorAssertionResponse)
1445
1446 The AuthenticatorAssertionResponse interface represents an
1447 authenticator's response to a client's request for generation of a new
1448 authentication assertion given the Relying Party's challenge and
1449 optional list of credentials it is aware of. This response contains a
1450 cryptographic signature proving possession of the credential private
1451 key, and optionally evidence of user consent to a specific transaction.
1452 [SecureContext, Exposed=Window]
1453 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
1454   [SameObject] readonly attribute ArrayBuffer authenticatorData;
1455   [SameObject] readonly attribute ArrayBuffer signature;
1456   [SameObject] readonly attribute ArrayBuffer userHandle;
1457 };
1458
1459 clientDataJSON
1460 This attribute, inherited from AuthenticatorResponse, contains
1461 the JSON-serialized client data (see 5.7.1 Client data used in
1462 WebAuthn signatures (dictionary CollectedClientData)) passed to
1463 the authenticator by the client in order to generate this
1464 assertion. The exact JSON serialization must be preserved, as
1465 the hash of the serialized client data has been computed over
1466 it.
1467
1468 authenticatorData, of type ArrayBuffer, readonly
1469 This attribute contains the authenticator data returned by the
1470 authenticator. See 6.1 Authenticator data.
1471
1472 signature, of type ArrayBuffer, readonly
1473 This attribute contains the raw signature returned from the

```

140E authenticator. See 6.2.2 The authenticatorGetAssertion
 140F operation.
 141C
 1411 userHandle, of type ArrayBuffer, readonly
 1412 This attribute contains the user handle returned from the
 1413 authenticator. See 6.2.2 The authenticatorGetAssertion
 1414 operation.
 141E
 1416 5.3. Parameters for Credential Generation (dictionary
 1417 PublicKeyCredentialParameters)
 141E
 1419 dictionary PublicKeyCredentialParameters {
 1420 required PublicKeyCredentialType type;
 1421 required COSEAlgorithmIdentifier alg;
 1422 };
 1423
 1424 This dictionary is used to supply additional parameters when creating a
 1425 new credential.
 142E
 1427 The type member specifies the type of credential to be created.
 142E
 1429 The alg member specifies the cryptographic signature algorithm with
 1430 which the newly generated credential will be used, and thus also the
 1431 type of asymmetric key pair to be generated, e.g., RSA or Elliptic
 1432 Curve.
 1433
 1434 Note: we use "alg" as the latter member name, rather than spelling-out
 1435 "algorithm", because it will be serialized into a message to the
 1436 authenticator, which may be sent over a low-bandwidth link.
 1437
 143E 5.4. Options for Credential Creation (dictionary
 143F MakePublicKeyCredentialOptions)
 144C
 1441 dictionary MakePublicKeyCredentialOptions {
 1442 required PublicKeyCredentialRpEntity rp;
 1443 required PublicKeyCredentialUserEntity user;
 1444
 1445 required BufferSource challenge;
 1446 required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
 1447
 1448 unsigned long timeout;
 1449 sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
 1450 AuthenticatorSelectionCriteria authenticatorSelection;
 1451 AuthenticationExtensions extensions;
 1452 };
 1453
 1454 rp, of type PublicKeyCredentialRpEntity
 1455 This member contains data about the Relying Party responsible
 1456 for the request.
 1457
 1458 Its value's name member is required, and contains the friendly
 1459 name of the Relying Party (e.g. "Acme Corporation", "Widgets,
 1460 Inc.", or "Awesome Site").
 1461
 1462 Its value's id member specifies the relying party identifier
 1463 with which the credential should be associated. If omitted, its
 1464 value will be the CredentialsContainer object's relevant
 1465 settings object's origin's effective domain.
 1466
 1467 user, of type PublicKeyCredentialUserEntity
 1468 This member contains data about the user account for which the
 1469 Relying Party is requesting attestation.
 147C
 1471 Its value's name member is required, and contains a name for the
 1472 user account (e.g., "john.p.smith@example.com" or
 1473 "+14255551234").
 1474
 1475 Its value's displayName member is required, and contains a
 1476 friendly name for the user account (e.g., "John P. Smith").
 1477

1463 authenticator. See 6.2.2 The authenticatorGetAssertion
 1464 operation.
 146E
 146F userHandle, of type ArrayBuffer, readonly
 1467 This attribute contains the user handle returned from the
 1468 authenticator. See 6.2.2 The authenticatorGetAssertion
 1469 operation.
 147C
 1471 5.3. Parameters for Credential Generation (dictionary
 1472 PublicKeyCredentialParameters)
 147E
 1474 dictionary PublicKeyCredentialParameters {
 1475 required PublicKeyCredentialType type;
 1476 required COSEAlgorithmIdentifier alg;
 1477 };
 147E
 1479 This dictionary is used to supply additional parameters when creating a
 1480 new credential.
 1481
 1482 The type member specifies the type of credential to be created.
 148E
 1484 The alg member specifies the cryptographic signature algorithm with
 1485 which the newly generated credential will be used, and thus also the
 1486 type of asymmetric key pair to be generated, e.g., RSA or Elliptic
 1487 Curve.
 148E
 1489 Note: we use "alg" as the latter member name, rather than spelling-out
 1490 "algorithm", because it will be serialized into a message to the
 1491 authenticator, which may be sent over a low-bandwidth link.
 1492
 1493 5.4. Options for Credential Creation (dictionary
 1494 MakePublicKeyCredentialOptions)
 149E
 1496 dictionary MakePublicKeyCredentialOptions {
 1497 required PublicKeyCredentialRpEntity rp;
 1498 required PublicKeyCredentialUserEntity user;
 1499
 1500 required BufferSource challenge;
 1501 required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
 1502
 1503 unsigned long timeout;
 1504 sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
 1505 AuthenticatorSelectionCriteria authenticatorSelection;
 1506 AuthenticationExtensions extensions;
 1507 };
 150E
 1509 rp, of type PublicKeyCredentialRpEntity
 1510 This member contains data about the Relying Party responsible
 1511 for the request.
 1512
 1513 Its value's name member is required, and contains the friendly
 1514 name of the Relying Party (e.g. "Acme Corporation", "Widgets,
 1515 Inc.", or "Awesome Site").
 151E
 1517 Its value's id member specifies the relying party identifier
 1518 with which the credential should be associated. If omitted, its
 1519 value will be the CredentialsContainer object's relevant
 1520 settings object's origin's effective domain.
 1521
 1522 user, of type PublicKeyCredentialUserEntity
 1523 This member contains data about the user account for which the
 1524 Relying Party is requesting attestation.
 152E
 152F Its value's name member is required, and contains a name for the
 1527 user account (e.g., "john.p.smith@example.com" or
 1528 "+14255551234").
 152E
 1529 Its value's displayName member is required, and contains a
 1530 friendly name for the user account (e.g., "John P. Smith").
 1531
 1532

147E Its value's id member is required and contains the user handle
147F for the account, specified by the Relying Party.
148C

1481 challenge, of type BufferSource
1482 This member contains a challenge intended to be used for
1483 generating the newly created credential's attestation object.
1484

148E pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>
148E This member contains information about the desired properties of
1487 the credential to be created. The sequence is ordered from most
148E preferred to least preferred. The platform makes a best-effort
148E to create the most preferred credential that it can.
149C

1491 timeout, of type unsigned long
1492 This member specifies a time, in milliseconds, that the caller
1493 is willing to wait for the call to complete. This is treated as
1494 a hint, and may be overridden by the platform.
149E

149E excludeCredentials, of type sequence<PublicKeyCredentialDescriptor>,
1497 defaulting to None
149E This member is intended for use by Relying Parties that wish to
149E limit the creation of multiple credentials for the same account
150C on a single authenticator. The platform is requested to return
1501 an error if the new credential would be created on an
1502 authenticator that also contains one of the credentials
1503 enumerated in this parameter.
1504

150E authenticatorSelection, of type AuthenticatorSelectionCriteria
150E This member is intended for use by Relying Parties that wish to
1507 select the appropriate authenticators to participate in the
150E create() or get() operation.
150E

151C extensions, of type AuthenticationExtensions
1511 This member contains additional parameters requesting additional
1512 processing by the client and authenticator. For example, the
1513 caller may request that only authenticators with certain
1514 capabilities be used to create the credential, or that particular
1515 information be returned in the attestation object. Some
1516 extensions are defined in 9 WebAuthn Extensions; consult the
1517 IANA "WebAuthn Extension Identifier" registry established by
151E [WebAuthn-Registries] for an up-to-date list of registered
151E WebAuthn Extensions.
152C

1521 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)
1522

1523 The PublicKeyCredentialEntity dictionary describes a user account, or a
1524 Relying Party, with which a public key credential is associated.
152E dictionary PublicKeyCredentialEntity {
152E DOMString name;
1527 USVString icon;
152E };

153C name, of type DOMString
1531 A human-friendly identifier for the entity. For example, this
1532 could be a company name for a Relying Party, or a user's name.
1533 This identifier is intended for display.
1534

153E icon, of type USVString
153E A serialized URL which resolves to an image associated with the
1537 entity. For example, this could be a user's avatar or a Relying
153E Party's logo. This URL MUST be an a priori authenticated URL.
153E

154C 5.4.2. RP Parameters for Credential Generation (dictionary
1541 PublicKeyCredentialRpEntity)
1542

1543 The PublicKeyCredentialRpEntity dictionary is used to supply additional
1544 Relying Party attributes when creating a new credential.
154E dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
154E DOMString id;
1547 };

153E Its value's id member is required and contains the user handle
153A for the account, specified by the Relying Party.
153E

153E challenge, of type BufferSource
1537 This member contains a challenge intended to be used for
153E generating the newly created credential's attestation object.
153E

154C pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>
1541 This member contains information about the desired properties of
1542 the credential to be created. The sequence is ordered from most
1543 preferred to least preferred. The platform makes a best-effort
1544 to create the most preferred credential that it can.
154E

154E timeout, of type unsigned long
1547 This member specifies a time, in milliseconds, that the caller
154E is willing to wait for the call to complete. This is treated as
154E a hint, and may be overridden by the platform.
155C

1551 excludeCredentials, of type sequence<PublicKeyCredentialDescriptor>,
1552 defaulting to None
1553 This member is intended for use by Relying Parties that wish to
1554 limit the creation of multiple credentials for the same account
155E on a single authenticator. The platform is requested to return
155E an error if the new credential would be created on an
1557 authenticator that also contains one of the credentials
155E enumerated in this parameter.
155E

156C authenticatorSelection, of type AuthenticatorSelectionCriteria
1561 This member is intended for use by Relying Parties that wish to
1562 select the appropriate authenticators to participate in the
156E create() or get() operation.
156E

156E extensions, of type AuthenticationExtensions
156E This member contains additional parameters requesting additional
1567 processing by the client and authenticator. For example, the
156E caller may request that only authenticators with certain
156E capabilities be used to create the credential, or that particular
157C information be returned in the attestation object. Some
1571 extensions are defined in 9 WebAuthn Extensions; consult the
1572 IANA "WebAuthn Extension Identifier" registry established by
1573 [WebAuthn-Registries] for an up-to-date list of registered
1574 WebAuthn Extensions.
157E

157E 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)
1577

157E The PublicKeyCredentialEntity dictionary describes a user account, or a
157E Relying Party, with which a public key credential is associated.
158C dictionary PublicKeyCredentialEntity {
1581 DOMString name;
1582 USVString icon;
158E };

158E name, of type DOMString
158E A human-friendly identifier for the entity. For example, this
1587 could be a company name for a Relying Party, or a user's name.
158E This identifier is intended for display.
158E

159C icon, of type USVString
1591 A serialized URL which resolves to an image associated with the
1592 entity. For example, this could be a user's avatar or a Relying
159E Party's logo. This URL MUST be an a priori authenticated URL.
159E

159E 5.4.2. RP Parameters for Credential Generation (dictionary
159E PublicKeyCredentialRpEntity)
1597

159E The PublicKeyCredentialRpEntity dictionary is used to supply additional
159E Relying Party attributes when creating a new credential.
160C dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
1601 DOMString id;
1602 };

```

1548 id, of type DOMString
1549     A unique identifier for the Relying Party entity, which sets the
1550     RP ID.
1551
1552 5.4.3. User Account Parameters for Credential Generation (dictionary
1553 PublicKeyCredentialUserEntity)
1554
1555     The PublicKeyCredentialUserEntity dictionary is used to supply
1556     additional user account attributes when creating a new credential.
1557     dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
1558     BufferSource id;
1559     DOMString displayName;
1560     };
1561
1562 id, of type BufferSource
1563     The user handle of the user account entity.
1564
1565 displayName, of type DOMString
1566     A friendly name for the user account (e.g., "John P. Smith").
1567
1568 5.4.4. Authenticator Selection Criteria (dictionary
1569 AuthenticatorSelectionCriteria)
1570
1571     Relying Parties may use the AuthenticatorSelectionCriteria dictionary
1572     to specify their requirements regarding authenticator attributes.
1573     dictionary AuthenticatorSelectionCriteria {
1574     AuthenticatorAttachment authenticatorAttachment;
1575     boolean requireResidentKey = false;
1576     boolean requireUserVerification = false;
1577     };
1578
1579 authenticatorAttachment, of type AuthenticatorAttachment
1580     If this member is present, eligible authenticators are filtered
1581     to only authenticators attached with the specified 5.4.5
1582     Authenticator Attachment enumeration (enum
1583     AuthenticatorAttachment).
1584
1585 requireResidentKey, of type boolean, defaulting to false
1586     This member describes the Relying Parties' requirements
1587     regarding availability of the Client-side-resident Credential
1588     Private Key. If the parameter is set to true, the authenticator
1589     MUST create a Client-side-resident Credential Private Key when
1590     creating a public key credential.
1591
1592 requireUserVerification, of type boolean, defaulting to false
1593     This member describes the Relying Parties' requirements
1594     regarding the authenticator being capable of performing user
1595     verification. If the parameter is set to true, the authenticator
1596     MUST perform user verification when performing the create()
1597     operation and future 5.1.4 Use an existing credential to make
1598     an assertion operations when it is requested to verify the
1599     credential.
1600
1601 5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)
1602
1603 enum AuthenticatorAttachment {
1604     "platform", // Platform attachment
1605     "cross-platform" // Cross-platform attachment
1606 };
1607
1608     Clients may communicate with authenticators using a variety of
1609     mechanisms. For example, a client may use a platform-specific API to
1610     communicate with an authenticator which is physically bound to a
1611     platform. On the other hand, a client may use a variety of standardized
1612     cross-platform transport protocols such as Bluetooth (see 5.7.4
1613     Authenticator Transport enumeration (enum AuthenticatorTransport)) to
1614     discover and communicate with cross-platform attached authenticators.
1615     Therefore, we use AuthenticatorAttachment to describe an
1616     authenticator's attachment modality. We define authenticators that are
1617

```

```

1603 id, of type DOMString
1604     A unique identifier for the Relying Party entity, which sets the
1605     RP ID.
1606
1607 5.4.3. User Account Parameters for Credential Generation (dictionary
1608 PublicKeyCredentialUserEntity)
1609
1610     The PublicKeyCredentialUserEntity dictionary is used to supply
1611     additional user account attributes when creating a new credential.
1612     dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
1613     BufferSource id;
1614     DOMString displayName;
1615     };
1616
1617 id, of type BufferSource
1618     The user handle of the user account entity.
1619
1620 displayName, of type DOMString
1621     A friendly name for the user account (e.g., "John P. Smith").
1622
1623 5.4.4. Authenticator Selection Criteria (dictionary
1624 AuthenticatorSelectionCriteria)
1625
1626     Relying Parties may use the AuthenticatorSelectionCriteria dictionary
1627     to specify their requirements regarding authenticator attributes.
1628     dictionary AuthenticatorSelectionCriteria {
1629     AuthenticatorAttachment authenticatorAttachment;
1630     boolean requireResidentKey = false;
1631     boolean requireUserVerification = false;
1632     };
1633
1634 authenticatorAttachment, of type AuthenticatorAttachment
1635     If this member is present, eligible authenticators are filtered
1636     to only authenticators attached with the specified 5.4.5
1637     Authenticator Attachment enumeration (enum
1638     AuthenticatorAttachment).
1639
1640 requireResidentKey, of type boolean, defaulting to false
1641     This member describes the Relying Parties' requirements
1642     regarding availability of the Client-side-resident Credential
1643     Private Key. If the parameter is set to true, the authenticator
1644     MUST create a Client-side-resident Credential Private Key when
1645     creating a public key credential.
1646
1647 requireUserVerification, of type boolean, defaulting to false
1648     This member describes the Relying Parties' requirements
1649     regarding the authenticator being capable of performing user
1650     verification. If the parameter is set to true, the authenticator
1651     MUST perform user verification when performing the create()
1652     operation and future 5.1.4 Use an existing credential to make
1653     an assertion operations when it is requested to verify the
1654     credential.
1655
1656 5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)
1657
1658 enum AuthenticatorAttachment {
1659     "platform", // Platform attachment
1660     "cross-platform" // Cross-platform attachment
1661 };
1662
1663     Clients may communicate with authenticators using a variety of
1664     mechanisms. For example, a client may use a platform-specific API to
1665     communicate with an authenticator which is physically bound to a
1666     platform. On the other hand, a client may use a variety of standardized
1667     cross-platform transport protocols such as Bluetooth (see 5.7.4
1668     Authenticator Transport enumeration (enum AuthenticatorTransport)) to
1669     discover and communicate with cross-platform attached authenticators.
1670     Therefore, we use AuthenticatorAttachment to describe an
1671     authenticator's attachment modality. We define authenticators that are
1672

```


1618 part of the client's platform as having a platform attachment, and
 1619 refer to them as platform authenticators. While those that are
 1620 reachable via cross-platform transport protocols are defined as having
 1621 cross-platform attachment, and refer to them as roaming authenticators.
 1622 * platform attachment - the respective authenticator is attached
 1623 using platform-specific transports. Usually, authenticators of this
 1624 class are non-removable from the platform.
 1625 * cross-platform attachment - the respective authenticator is
 1626 attached using cross-platform transports. Authenticators of this
 1627 class are removable from, and can "roam" among, client platforms.
 1628

1629 This distinction is important because there are use-cases where only
 1630 platform authenticators are acceptable to a Relying Party, and
 1631 conversely ones where only roaming authenticators are employed. As a
 1632 concrete example of the former, a credential on a platform
 1633 authenticator may be used by Relying Parties to quickly and
 1634 conveniently reauthenticate the user with a minimum of friction, e.g.,
 1635 the user will not have to dig around in their pocket for their key fob
 1636 or phone. As a concrete example of the latter, when the user is
 1637 accessing the Relying Party from a given client for the first time,
 1638 they may be required to use a roaming authenticator which was
 1639 originally registered with the Relying Party using a different client.
 1640

1641 5.5. Options for Assertion Generation (dictionary
 1642 PublicKeyCredentialRequestOptions)

1643 The PublicKeyCredentialRequestOptions dictionary supplies get() with
 1644 the data it needs to generate an assertion. Its challenge member must
 1645 be present, while its other members are optional.
 1646 dictionary PublicKeyCredentialRequestOptions {
 1647 required BufferSource challenge;
 1648 unsigned long timeout;
 1649 USVString rpld;
 1650 sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
 1651 AuthenticationExtensions extensions;
 1652 };

1653 challenge, of type BufferSource
 1654 This member represents a challenge that the selected
 1655 authenticator signs, along with other data, when producing an
 1656 authentication assertion.
 1657

1658 timeout, of type unsigned long
 1659 This optional member specifies a time, in milliseconds, that the
 1660 caller is willing to wait for the call to complete. The value is
 1661 treated as a hint, and may be overridden by the platform.
 1662

1663 rpld, of type USVString
 1664 This optional member specifies the relying party identifier
 1665 claimed by the caller. If omitted, its value will be the
 1666 CredentialsContainer object's relevant settings object's
 1667 origin's effective domain.
 1668

1669 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,
 1670 defaulting to None
 1671 This optional member contains a list of
 1672 PublicKeyCredentialDescriptor object representing public key
 1673 credentials acceptable to the caller, in decending order of the
 1674 caller's preference (the first item in the list is the most
 1675 preferred credential, and so on down the list).
 1676

1677 extensions, of type AuthenticationExtensions
 1678 This optional member contains additional parameters requesting
 1679 additional processing by the client and authenticator. For
 1680 example, if transaction confirmation is sought from the user,
 1681 then the prompt string might be included as an extension.
 1682

1683 5.6. Authentication Extensions (typedef AuthenticationExtensions)

1684 typedef record<DOMString, any> AuthenticationExtensions;

1673 part of the client's platform as having a platform attachment, and
 1674 refer to them as platform authenticators. While those that are
 1675 reachable via cross-platform transport protocols are defined as having
 1676 cross-platform attachment, and refer to them as roaming authenticators.
 1677 * platform attachment - the respective authenticator is attached
 1678 using platform-specific transports. Usually, authenticators of this
 1679 class are non-removable from the platform.
 1680 * cross-platform attachment - the respective authenticator is
 1681 attached using cross-platform transports. Authenticators of this
 1682 class are removable from, and can "roam" among, client platforms.
 1683

1684 This distinction is important because there are use-cases where only
 1685 platform authenticators are acceptable to a Relying Party, and
 1686 conversely ones where only roaming authenticators are employed. As a
 1687 concrete example of the former, a credential on a platform
 1688 authenticator may be used by Relying Parties to quickly and
 1689 conveniently reauthenticate the user with a minimum of friction, e.g.,
 1690 the user will not have to dig around in their pocket for their key fob
 1691 or phone. As a concrete example of the latter, when the user is
 1692 accessing the Relying Party from a given client for the first time,
 1693 they may be required to use a roaming authenticator which was
 1694 originally registered with the Relying Party using a different client.
 1695

1696 5.5. Options for Assertion Generation (dictionary
 1697 PublicKeyCredentialRequestOptions)

1698 The PublicKeyCredentialRequestOptions dictionary supplies get() with
 1699 the data it needs to generate an assertion. Its challenge member must
 1700 be present, while its other members are optional.
 1701 dictionary PublicKeyCredentialRequestOptions {
 1702 required BufferSource challenge;
 1703 unsigned long timeout;
 1704 USVString rpld;
 1705 sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
 1706 AuthenticationExtensions extensions;
 1707 };

1708 challenge, of type BufferSource
 1709 This member represents a challenge that the selected
 1710 authenticator signs, along with other data, when producing an
 1711 authentication assertion.
 1712

1713 timeout, of type unsigned long
 1714 This optional member specifies a time, in milliseconds, that the
 1715 caller is willing to wait for the call to complete. The value is
 1716 treated as a hint, and may be overridden by the platform.
 1717

1718 rpld, of type USVString
 1719 This optional member specifies the relying party identifier
 1720 claimed by the caller. If omitted, its value will be the
 1721 CredentialsContainer object's relevant settings object's
 1722 origin's effective domain.
 1723

1724 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,
 1725 defaulting to None
 1726 This optional member contains a list of
 1727 PublicKeyCredentialDescriptor object representing public key
 1728 credentials acceptable to the caller, in decending order of the
 1729 caller's preference (the first item in the list is the most
 1730 preferred credential, and so on down the list).
 1731

1732 extensions, of type AuthenticationExtensions
 1733 This optional member contains additional parameters requesting
 1734 additional processing by the client and authenticator. For
 1735 example, if transaction confirmation is sought from the user,
 1736 then the prompt string might be included as an extension.
 1737

1738 5.6. Authentication Extensions (typedef AuthenticationExtensions)

1739 typedef record<DOMString, any> AuthenticationExtensions;

1688 This is a dictionary containing zero or more WebAuthn extensions, as
 1689 defined in 9 WebAuthn Extensions. An AuthenticationExtensions instance
 1690 can contain either client extensions or authenticator extensions,
 1691 depending upon context.
 1692
 1693
 1694 5.7. Supporting Data Structures
 1695
 1696 The public key credential type uses certain data structures that are
 1697 specified in supporting specifications. These are as follows.
 1698
 1699 5.7.1. Client data used in WebAuthn signatures (dictionary
 1700 CollectedClientData)
 1701
 1702 The client data represents the contextual bindings of both the Relying
 1703 Party and the client platform. It is a key-value mapping with
 1704 string-valued keys. Values may be any type that has a valid encoding in
 1705 JSON. Its structure is defined by the following Web IDL.
 1706 dictionary CollectedClientData {
 1707 required DOMString challenge;
 1708 required DOMString origin;
 1709 required DOMString hashAlgorithm;
 1710 DOMString tokenBindingId;
 1711 AuthenticationExtensions clientExtensions;
 1712 AuthenticationExtensions authenticatorExtensions;
 1713 };
 1714
 1715 The challenge member contains the base64url encoding of the challenge
 1716 provided by the RP.
 1717
 1718 The origin member contains the fully qualified origin of the requester,
 1719 as provided to the authenticator by the client, in the syntax defined
 1720 by [RFC6454].
 1721
 1722 The hashAlgorithm member is a recognized algorithm name that supports
 1723 the "digest" operation, which specifies the algorithm used to compute
 1724 the hash of the serialized client data. This algorithm is chosen by the
 1725 client at its sole discretion.
 1726
 1727 The tokenBindingId member contains the base64url encoding of the Token
 1728 Binding ID that this client uses for the Token Binding protocol when
 1729 communicating with the Relying Party. This can be omitted if no Token
 1730 Binding has been negotiated between the client and the Relying Party.
 1731
 1732 The optional clientExtensions and authenticatorExtensions members
 1733 contain additional parameters generated by processing the extensions
 1734 passed in by the Relying Party. WebAuthn extensions are detailed in
 1735 Section 9 WebAuthn Extensions.
 1736
 1737 This structure is used by the client to compute the following
 1738 quantities:
 1739
 1740 JSON-serialized client data
 1741 This is the UTF-8 encoding of the result of calling the initial
 1742 value of JSON.stringify on a CollectedClientData dictionary.
 1743
 1744 Hash of the serialized client data
 1745 This is the hash (computed using hashAlgorithm) of the
 1746 JSON-serialized client data, as constructed by the client.
 1747
 1748 5.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
 1749
 1750 enum PublicKeyCredentialType {
 1751 "public-key"
 1752 };
 1753
 1754 This enumeration defines the valid credential types. It is an extension
 1755 point; values may be added to it in the future, as more credential
 1756 types are defined. The values of this enumeration are used for
 1757 versioning the Authentication Assertion and attestation structures

1743 This is a dictionary containing zero or more WebAuthn extensions, as
 1744 defined in 9 WebAuthn Extensions. An AuthenticationExtensions instance
 1745 can contain either client extensions or authenticator extensions,
 1746 depending upon context.
 1747
 1748
 1749 5.7. Supporting Data Structures
 1750
 1751 The public key credential type uses certain data structures that are
 1752 specified in supporting specifications. These are as follows.
 1753
 1754 5.7.1. Client data used in WebAuthn signatures (dictionary
 1755 CollectedClientData)
 1756
 1757 The client data represents the contextual bindings of both the Relying
 1758 Party and the client platform. It is a key-value mapping with
 1759 string-valued keys. Values may be any type that has a valid encoding in
 1760 JSON. Its structure is defined by the following Web IDL.
 1761 dictionary CollectedClientData {
 1762 required DOMString challenge;
 1763 required DOMString origin;
 1764 required DOMString hashAlgorithm;
 1765 DOMString tokenBindingId;
 1766 AuthenticationExtensions clientExtensions;
 1767 AuthenticationExtensions authenticatorExtensions;
 1768 };
 1769
 1770 The challenge member contains the base64url encoding of the challenge
 1771 provided by the RP.
 1772
 1773 The origin member contains the fully qualified origin of the requester,
 1774 as provided to the authenticator by the client, in the syntax defined
 1775 by [RFC6454].
 1776
 1777 The hashAlgorithm member is a recognized algorithm name that supports
 1778 the "digest" operation, which specifies the algorithm used to compute
 1779 the hash of the serialized client data. This algorithm is chosen by the
 1780 client at its sole discretion.
 1781
 1782 The tokenBindingId member contains the base64url encoding of the Token
 1783 Binding ID that this client uses for the Token Binding protocol when
 1784 communicating with the Relying Party. This can be omitted if no Token
 1785 Binding has been negotiated between the client and the Relying Party.
 1786
 1787 The optional clientExtensions and authenticatorExtensions members
 1788 contain additional parameters generated by processing the extensions
 1789 passed in by the Relying Party. WebAuthn extensions are detailed in
 1790 Section 9 WebAuthn Extensions.
 1791
 1792 This structure is used by the client to compute the following
 1793 quantities:
 1794
 1795 JSON-serialized client data
 1796 This is the UTF-8 encoding of the result of calling the initial
 1797 value of JSON.stringify on a CollectedClientData dictionary.
 1798
 1799 Hash of the serialized client data
 1800 This is the hash (computed using hashAlgorithm) of the
 1801 JSON-serialized client data, as constructed by the client.
 1802
 1803 5.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
 1804
 1805 enum PublicKeyCredentialType {
 1806 "public-key"
 1807 };
 1808
 1809 This enumeration defines the valid credential types. It is an extension
 1810 point; values may be added to it in the future, as more credential
 1811 types are defined. The values of this enumeration are used for
 1812 versioning the Authentication Assertion and attestation structures

175E according to the type of the authenticator.
175F
176C Currently one credential type is defined, namely "public-key".
176D
176E 5.7.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)
176F
176G dictionary PublicKeyCredentialDescriptor {
176H required PublicKeyCredentialType type;
176I required BufferSource id;
176J sequence<AuthenticatorTransport> transports;
176K };
176L
177C This dictionary contains the attributes that are specified by a caller
177D when referring to a credential as an input parameter to the create() or
177E get() methods. It mirrors the fields of the PublicKeyCredential object
177F returned by the latter methods.
177G The type member contains the type of the credential the caller is
177H referring to.
177I
177J The id member contains the identifier of the credential that the caller
177K is referring to.
177L
178C 5.7.4. Authenticator Transport enumeration (enum AuthenticatorTransport)
178D
178E enum AuthenticatorTransport {
178F "usb",
178G "nfc",
178H "ble"
178I };
178J
178K Authenticators may communicate with Clients using a variety of
178L transports. This enumeration defines a hint as to how Clients might
178M communicate with a particular Authenticator in order to obtain an
178N assertion for a specific credential. Note that these hints represent
178O the Relying Party's best belief as to how an Authenticator may be
178P reached. A Relying Party may obtain a list of transports hints from
178Q some attestation statement formats or via some out-of-band mechanism;
178R it is outside the scope of this specification to define that mechanism.
178S * usb - the respective Authenticator may be contacted over USB.
178T * nfc - the respective Authenticator may be contacted over Near Field
178U Communication (NFC).
178V * ble - the respective Authenticator may be contacted over Bluetooth
178W Smart (Bluetooth Low Energy / BLE).
178X
179C 5.7.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)
179D
179E typedef long COSEAlgorithmIdentifier;
179F
179G A COSEAlgorithmIdentifier's value is a number identifying a
179H cryptographic algorithm. The algorithm identifiers SHOULD be values
179I registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],
179J for instance, -7 for "ES256" and -257 for "RS256".
179K
180C 6. WebAuthn Authenticator model
180D
180E The API defined in this specification implies a specific abstract
180F functional model for an authenticator. This section describes the
180G authenticator model.
180H
180I Client platforms may implement and expose this abstract model in any
180J way desired. However, the behavior of the client's Web Authentication
180K API implementation, when operating on the authenticators supported by
180L that platform, MUST be indistinguishable from the behavior specified in
180M 5 Web Authentication API.
180N
180O For authenticators, this model defines the logical operations that they
180P must support, and the data formats that they expose to the client and
180Q the Relying Party. However, it does not define the details of how
180R authenticators communicate with the client platform, unless they are
180S required for interoperability with Relying Parties. For instance, this

181E according to the type of the authenticator.
181F
181G Currently one credential type is defined, namely "public-key".
181H
181I 5.7.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)
181J
181K dictionary PublicKeyCredentialDescriptor {
181L required PublicKeyCredentialType type;
181M required BufferSource id;
181N sequence<AuthenticatorTransport> transports;
181O };
181P
181Q This dictionary contains the attributes that are specified by a caller
181R when referring to a credential as an input parameter to the create() or
181S get() methods. It mirrors the fields of the PublicKeyCredential object
181T returned by the latter methods.
181U The type member contains the type of the credential the caller is
181V referring to.
181W
181X The id member contains the identifier of the credential that the caller
181Y is referring to.
181Z
182C 5.7.4. Authenticator Transport enumeration (enum AuthenticatorTransport)
182D
182E enum AuthenticatorTransport {
182F "usb",
182G "nfc",
182H "ble"
182I };
182J
182K Authenticators may communicate with Clients using a variety of
182L transports. This enumeration defines a hint as to how Clients might
182M communicate with a particular Authenticator in order to obtain an
182N assertion for a specific credential. Note that these hints represent
182O the Relying Party's best belief as to how an Authenticator may be
182P reached. A Relying Party may obtain a list of transports hints from
182Q some attestation statement formats or via some out-of-band mechanism;
182R it is outside the scope of this specification to define that mechanism.
182S * usb - the respective Authenticator may be contacted over USB.
182T * nfc - the respective Authenticator may be contacted over Near Field
182U Communication (NFC).
182V * ble - the respective Authenticator may be contacted over Bluetooth
182W Smart (Bluetooth Low Energy / BLE).
182X
182Y 5.7.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)
182Z
182A typedef long COSEAlgorithmIdentifier;
182B
182C A COSEAlgorithmIdentifier's value is a number identifying a
182D cryptographic algorithm. The algorithm identifiers SHOULD be values
182E registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],
182F for instance, -7 for "ES256" and -257 for "RS256".
182G
182H 6. WebAuthn Authenticator model
182I
182J The API defined in this specification implies a specific abstract
182K functional model for an authenticator. This section describes the
182L authenticator model.
182M
182N Client platforms may implement and expose this abstract model in any
182O way desired. However, the behavior of the client's Web Authentication
182P API implementation, when operating on the authenticators supported by
182Q that platform, MUST be indistinguishable from the behavior specified in
182R 5 Web Authentication API.
182S
182T For authenticators, this model defines the logical operations that they
182U must support, and the data formats that they expose to the client and
182V the Relying Party. However, it does not define the details of how
182W authenticators communicate with the client platform, unless they are
182X required for interoperability with Relying Parties. For instance, this

1828 abstract model does not define protocols for connecting authenticators
1829 to clients over transports such as USB or NFC. Similarly, this abstract
1830 model does not define specific error codes or methods of returning
1831 them; however, it does define error behavior in terms of the needs of
1832 the client. Therefore, specific error codes are mentioned as a means of
1833 showing which error conditions must be distinguishable (or not) from
1834 each other in order to enable a compliant and secure client
1835 implementation.
1836

1837 In this abstract model, the authenticator provides key management and
1838 cryptographic signatures. It may be embedded in the WebAuthn client, or
1839 housed in a separate device entirely. The authenticator may itself
1840 contain a cryptographic module which operates at a higher security
1841 level than the rest of the authenticator. This is particularly
1842 important for authenticators that are embedded in the WebAuthn client,
1843 as in those cases this cryptographic module (which may, for example, be
1844 a TPM) could be considered more trustworthy than the rest of the
1845 authenticator.
1846

1847 Each authenticator stores some number of public key credentials. Each
1848 public key credential has an identifier which is unique (or extremely
1849 unlikely to be duplicated) among all public key credentials. Each
1850 credential is also associated with a Relying Party, whose identity is
1851 represented by a Relying Party Identifier (RP ID).
1852

1853 Each authenticator has an AAGUID, which is a 128-bit identifier that
1854 indicates the type (e.g. make and model) of the authenticator. The
1855 AAGUID MUST be chosen by the manufacturer to be identical across all
1856 substantially identical authenticators made by that manufacturer, and
1857 different (with probability $1-2^{-128}$ or greater) from the AAGUIDs of
1858 all other types of authenticators. The RP MAY use the AAGUID to infer
1859 certain properties of the authenticator, such as certification level
1860 and strength of key protection, using information from other sources.
1861

1862 The primary function of the authenticator is to provide WebAuthn
1863 signatures, which are bound to various contextual data. These data are
1864 observed, and added at different levels of the stack as a signature
1865 request passes from the server to the authenticator. In verifying a
1866 signature, the server checks these bindings against expected values.
1867 These contextual bindings are divided in two: Those added by the RP or
1868 the client, referred to as client data; and those added by the
1869 authenticator, referred to as the authenticator data. The authenticator
1870 signs over the client data, but is otherwise not interested in its
1871 contents. To save bandwidth and processing requirements on the
1872 authenticator, the client hashes the client data and sends only the
1873 result to the authenticator. The authenticator signs over the
1874 combination of the hash of the serialized client data, and its own
1875 authenticator data.
1876

- 1877 The goals of this design can be summarized as follows.
1878 * The scheme for generating signatures should accommodate cases where
1879 the link between the client platform and authenticator is very
1880 limited, in bandwidth and/or latency. Examples include Bluetooth
1881 Low Energy and Near-Field Communication.
1882 * The data processed by the authenticator should be small and easy to
1883 interpret in low-level code. In particular, authenticators should
1884 not have to parse high-level encodings such as JSON.
1885 * Both the client platform and the authenticator should have the
1886 flexibility to add contextual bindings as needed.
1887 * The design aims to reuse as much as possible of existing encoding
1888 formats in order to aid adoption and implementation.
1889

1890 Authenticators produce cryptographic signatures for two distinct
1891 purposes:
1892 1. An attestation signature is produced when a new public key
1893 credential is created via an authenticatorMakeCredential operation.
1894 An attestation signature provides cryptographic proof of certain
1895 properties of the the authenticator and the credential. For
1896 instance, an attestation signature asserts the authenticator type
1897 (as denoted by its AAGUID) and the credential public key. The

1883 abstract model does not define protocols for connecting authenticators
1884 to clients over transports such as USB or NFC. Similarly, this abstract
1885 model does not define specific error codes or methods of returning
1886 them; however, it does define error behavior in terms of the needs of
1887 the client. Therefore, specific error codes are mentioned as a means of
1888 showing which error conditions must be distinguishable (or not) from
1889 each other in order to enable a compliant and secure client
1890 implementation.
1891

1892 In this abstract model, the authenticator provides key management and
1893 cryptographic signatures. It may be embedded in the WebAuthn client, or
1894 housed in a separate device entirely. The authenticator may itself
1895 contain a cryptographic module which operates at a higher security
1896 level than the rest of the authenticator. This is particularly
1897 important for authenticators that are embedded in the WebAuthn client,
1898 as in those cases this cryptographic module (which may, for example, be
1899 a TPM) could be considered more trustworthy than the rest of the
1900 authenticator.
1901

1902 Each authenticator stores some number of public key credentials. Each
1903 public key credential has an identifier which is unique (or extremely
1904 unlikely to be duplicated) among all public key credentials. Each
1905 credential is also associated with a Relying Party, whose identity is
1906 represented by a Relying Party Identifier (RP ID).
1907

1908 Each authenticator has an AAGUID, which is a 128-bit identifier that
1909 indicates the type (e.g. make and model) of the authenticator. The
1910 AAGUID MUST be chosen by the manufacturer to be identical across all
1911 substantially identical authenticators made by that manufacturer, and
1912 different (with probability $1-2^{-128}$ or greater) from the AAGUIDs of
1913 all other types of authenticators. The RP MAY use the AAGUID to infer
1914 certain properties of the authenticator, such as certification level
1915 and strength of key protection, using information from other sources.
1916

1917 The primary function of the authenticator is to provide WebAuthn
1918 signatures, which are bound to various contextual data. These data are
1919 observed, and added at different levels of the stack as a signature
1920 request passes from the server to the authenticator. In verifying a
1921 signature, the server checks these bindings against expected values.
1922 These contextual bindings are divided in two: Those added by the RP or
1923 the client, referred to as client data; and those added by the
1924 authenticator, referred to as the authenticator data. The authenticator
1925 signs over the client data, but is otherwise not interested in its
1926 contents. To save bandwidth and processing requirements on the
1927 authenticator, the client hashes the client data and sends only the
1928 result to the authenticator. The authenticator signs over the
1929 combination of the hash of the serialized client data, and its own
1930 authenticator data.
1931

- 1932 The goals of this design can be summarized as follows.
1933 * The scheme for generating signatures should accommodate cases where
1934 the link between the client platform and authenticator is very
1935 limited, in bandwidth and/or latency. Examples include Bluetooth
1936 Low Energy and Near-Field Communication.
1937 * The data processed by the authenticator should be small and easy to
1938 interpret in low-level code. In particular, authenticators should
1939 not have to parse high-level encodings such as JSON.
1940 * Both the client platform and the authenticator should have the
1941 flexibility to add contextual bindings as needed.
1942 * The design aims to reuse as much as possible of existing encoding
1943 formats in order to aid adoption and implementation.
1944

1945 Authenticators produce cryptographic signatures for two distinct
1946 purposes:
1947 1. An attestation signature is produced when a new public key
1948 credential is created via an authenticatorMakeCredential operation.
1949 An attestation signature provides cryptographic proof of certain
1950 properties of the the authenticator and the credential. For
1951 instance, an attestation signature asserts the authenticator type
1952 (as denoted by its AAGUID) and the credential public key. The

1898 attestation signature is signed by an attestation private key,
 1899 which is chosen depending on the type of attestation desired. For
 1900 more details on attestation, see 6.3 Attestation.
 1901 2. An assertion signature is produced when the
 1902 authenticatorGetAssertion method is invoked. It represents an
 1903 assertion by the authenticator that the user has consented to a
 1904 specific transaction, such as logging in, or completing a purchase.
 1905 Thus, an assertion signature asserts that the authenticator
 1906 possessing a particular credential private key has established, to
 1907 the best of its ability, that the user requesting this transaction
 1908 is the same user who consented to creating that particular public
 1909 key credential. It also asserts additional information, termed
 1910 client data, that may be useful to the caller, such as the means by
 1911 which user consent was provided, and the prompt shown to the user
 1912 by the authenticator. The assertion signature format is illustrated
 1913 in Figure 2, below.

1914
 1915 The formats of these signatures, as well as the procedures for
 1916 generating them, are specified below.

1917
 1918 6.1. Authenticator data

1919 The authenticator data structure encodes contextual bindings made by
 1920 the authenticator. These bindings are controlled by the authenticator
 1921 itself, and derive their trust from the Relying Party's assessment of
 1922 the security properties of the authenticator. In one extreme case, the
 1923 authenticator may be embedded in the client, and its bindings may be no
 1924 more trustworthy than the client data. At the other extreme, the
 1925 authenticator may be a discrete entity with high-security hardware and
 1926 software, connected to the client over a secure channel. In both cases,
 1927 the Relying Party receives the authenticator data in the same format,
 1928 and uses its knowledge of the authenticator to make trust decisions.

1929 The authenticator data has a compact but extensible encoding. This is
 1930 desired since authenticators can be devices with limited capabilities
 1931 and low power requirements, with much simpler software stacks than the
 1932 client platform components.

1933 The authenticator data structure is a byte array of 37 bytes or more,
 1934 as follows.

1935	Name	Length (in bytes)	Description
1936	rpldHash	32	SHA-256 hash of the RP ID associated with the credential.
1937	flags	1	Flags (bit 0 is the least significant bit):
1938	* Bit 0:	User Present (UP) result.	
1939	+ 1	means the user is present.	
1940	+ 0	means the user is not present.	
1941	* Bit 1:	Reserved for future use (RFU1).	
1942	* Bit 2:	User Verified (UV) result.	
1943	+ 1	means the user is verified.	
1944	+ 0	means the user is not verified.	
1945	* Bits 3-5:	Reserved for future use (RFU2).	
1946	* Bit 6:	Attested credential data included (AT).	
1947	+ 1	Indicates whether the authenticator added attested credential data.	
1948	* Bit 7:	Extension data included (ED).	
1949	+ 1	Indicates if the authenticator data has extensions.	

1950 signCount 4 Signature counter, 32-bit unsigned big-endian integer.
 1951 attestedCredentialData variable (if present) attested credential data
 1952 (if present). See 6.3.1 Attested credential data for details. Its
 1953 length depends on the length of the credential ID and credential public
 1954 key being attested.
 1955 extensions variable (if present) Extension-defined authenticator data.
 1956 This is a CBOR [RFC7049] map with extension identifiers as keys, and
 1957 authenticator extension outputs as values. See 9 WebAuthn Extensions
 1958 for details.

1959 NOTE: The names in the Name column in the above table are only for
 1960 reference within this document, and are not present in the actual
 1961 representation of the authenticator data.

1953 attestation signature is signed by an attestation private key,
 1954 which is chosen depending on the type of attestation desired. For
 1955 more details on attestation, see 6.3 Attestation.
 1956 2. An assertion signature is produced when the
 1957 authenticatorGetAssertion method is invoked. It represents an
 1958 assertion by the authenticator that the user has consented to a
 1959 specific transaction, such as logging in, or completing a purchase.
 1960 Thus, an assertion signature asserts that the authenticator
 1961 possessing a particular credential private key has established, to
 1962 the best of its ability, that the user requesting this transaction
 1963 is the same user who consented to creating that particular public
 1964 key credential. It also asserts additional information, termed
 1965 client data, that may be useful to the caller, such as the means by
 1966 which user consent was provided, and the prompt shown to the user
 1967 by the authenticator. The assertion signature format is illustrated
 1968 in Figure 2, below.

1969
 1970 The formats of these signatures, as well as the procedures for
 1971 generating them, are specified below.

1972
 1973 6.1. Authenticator data

1974 The authenticator data structure encodes contextual bindings made by
 1975 the authenticator. These bindings are controlled by the authenticator
 1976 itself, and derive their trust from the Relying Party's assessment of
 1977 the security properties of the authenticator. In one extreme case, the
 1978 authenticator may be embedded in the client, and its bindings may be no
 1979 more trustworthy than the client data. At the other extreme, the
 1980 authenticator may be a discrete entity with high-security hardware and
 1981 software, connected to the client over a secure channel. In both cases,
 1982 the Relying Party receives the authenticator data in the same format,
 1983 and uses its knowledge of the authenticator to make trust decisions.

1984 The authenticator data has a compact but extensible encoding. This is
 1985 desired since authenticators can be devices with limited capabilities
 1986 and low power requirements, with much simpler software stacks than the
 1987 client platform components.

1988 The authenticator data structure is a byte array of 37 bytes or more,
 1989 as follows.

1990	Name	Length (in bytes)	Description
1991	rpldHash	32	SHA-256 hash of the RP ID associated with the credential.
1992	flags	1	Flags (bit 0 is the least significant bit):
1993	* Bit 0:	User Present (UP) result.	
1994	+ 1	means the user is present.	
1995	+ 0	means the user is not present.	
1996	* Bit 1:	Reserved for future use (RFU1).	
1997	* Bit 2:	User Verified (UV) result.	
1998	+ 1	means the user is verified.	
1999	+ 0	means the user is not verified.	
2000	* Bits 3-5:	Reserved for future use (RFU2).	
2001	* Bit 6:	Attested credential data included (AT).	
2002	+ 1	Indicates whether the authenticator added attested credential data.	
2003	* Bit 7:	Extension data included (ED).	
2004	+ 1	Indicates if the authenticator data has extensions.	

2005 signCount 4 Signature counter, 32-bit unsigned big-endian integer.
 2006 attestedCredentialData variable (if present) attested credential data
 2007 (if present). See 6.3.1 Attested credential data for details. Its
 2008 length depends on the length of the credential ID and credential public
 2009 key being attested.
 2010 extensions variable (if present) Extension-defined authenticator data.
 2011 This is a CBOR [RFC7049] map with extension identifiers as keys, and
 2012 authenticator extension outputs as values. See 9 WebAuthn Extensions
 2013 for details.

2014 NOTE: The names in the Name column in the above table are only for
 2015 reference within this document, and are not present in the actual
 2016 representation of the authenticator data.

196E The RP ID is originally received from the client when the credential is
196F created, and again when an assertion is generated. However, it differs
1970 from other client data in some important ways. First, unlike the client
1971 data, the RP ID of a credential does not change between operations but
1972 instead remains the same for the lifetime of that credential. Secondly,
1973 it is validated by the authenticator during the
1974 authenticatorGetAssertion operation, by verifying that the RP ID
1975 associated with the requested credential exactly matches the RP ID
1976 supplied by the client, and that the RP ID is a registrable domain
1977 suffix of or is equal to the effective domain of the RP's origin's
1978 effective domain.
1979

1980 The UP flag SHALL be set if and only if the authenticator detected a
1981 user through an authenticator specific gesture. The RFU bits SHALL be
1982 set to zero.
1983

1984 For attestation signatures, the authenticator MUST set the AT flag and
1985 include the attestedCredentialData. For authentication signatures, the
1986 AT flag MUST NOT be set and the attestedCredentialData MUST NOT be
1987 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 attested
1998 credential data (which is only present if the AT flag is set) describes
2000 its own length. If the ED flag is set, then the total length is 37
2001 bytes plus the length of the attested credential data, plus the length
2002 of the CBOR map that follows.
2003

2004 6.1.1. Signature Counter Considerations
2005

2006 Authenticators MUST implement a signature counter feature. The
2007 signature counter is incremented for each successful
2008 authenticatorGetAssertion operation by some positive value, and its
2009 value is returned to the Relying Party within the authenticator data.
2010 The signature counter's purpose is to aid Relying Parties in detecting
2011 cloned authenticators. Clone detection is more important for
2012 authenticators with limited protection measures.
2013

2014 An Relying Party stores the signature counter of the most recent
2015 authenticatorGetAssertion operation. Upon a new
2016 authenticatorGetAssertion operation, the Relying Party compares the
2017 stored signature counter value with the new signCount value returned in
2018 the assertion's authenticator data. If this new signCount value is less
2019 than or equal to the stored value, a cloned authenticator may exist, or
2020 the authenticator may be malfunctioning.
2021

2022 Detecting a signature counter mismatch does not indicate whether the
2023 current operation was performed by a cloned authenticator or the
2024 original authenticator. Relying Parties should address this situation
2025 appropriately relative to their individual situations, i.e., their risk
2026 tolerance.
2027

2028 Authenticators:
2029 * should implement per-RP ID signature counters. This prevents the
2030 signature counter value from being shared between Relying Parties
2031 and being possibly employed as a correlation handle for the user.
2032 Authenticators may implement a global signature counter, i.e., on a
2033 per-authenticator basis, but this is less privacy-friendly for
2034 users.
2035 * should ensure that the signature counter value does not
2036 accidentally decrease (e.g., due to hardware failures).
2037

202E The RP ID is originally received from the client when the credential is
202F created, and again when an assertion is generated. However, it differs
2030 from other client data in some important ways. First, unlike the client
2031 data, the RP ID of a credential does not change between operations but
2032 instead remains the same for the lifetime of that credential. Secondly,
2033 it is validated by the authenticator during the
2034 authenticatorGetAssertion operation, by verifying that the RP ID
2035 associated with the requested credential exactly matches the RP ID
2036 supplied by the client, and that the RP ID is a registrable domain
2037 suffix of or is equal to the effective domain of the RP's origin's
2038 effective domain.
2039

203E The UP flag SHALL be set if and only if the authenticator detected a
203F user through an authenticator specific gesture. The RFU bits SHALL be
2040 set to zero.
2041

2042 For attestation signatures, the authenticator MUST set the AT flag and
2043 include the attestedCredentialData. For authentication signatures, the
2044 AT flag MUST NOT be set and the attestedCredentialData MUST NOT be
2045 included.
2046

2047 If the authenticator does not include any extension data, it MUST set
2048 the ED flag to zero, and to one if extension data is included.
2049

204E The figure below shows a visual representation of the authenticator
204F data structure.
2050 [fido-signature-formats-figure1.svg] Authenticator data layout.
2051

2052 Note that the authenticator data describes its own length: If the AT
2053 and ED flags are not set, it is always 37 bytes long. The attested
2054 credential data (which is only present if the AT flag is set) describes
2055 its own length. If the ED flag is set, then the total length is 37
2056 bytes plus the length of the attested credential data, plus the length
2057 of the CBOR map that follows.
2058

2059 6.1.1. Signature Counter Considerations
2060

2061 Authenticators MUST implement a signature counter feature. The
2062 signature counter is incremented for each successful
2063 authenticatorGetAssertion operation by some positive value, and its
2064 value is returned to the Relying Party within the authenticator data.
2065 The signature counter's purpose is to aid Relying Parties in detecting
2066 cloned authenticators. Clone detection is more important for
2067 authenticators with limited protection measures.
2068

2069 An Relying Party stores the signature counter of the most recent
2070 authenticatorGetAssertion operation. Upon a new
2071 authenticatorGetAssertion operation, the Relying Party compares the
2072 stored signature counter value with the new signCount value returned in
2073 the assertion's authenticator data. If this new signCount value is less
2074 than or equal to the stored value, a cloned authenticator may exist, or
2075 the authenticator may be malfunctioning.
2076

2077 Detecting a signature counter mismatch does not indicate whether the
2078 current operation was performed by a cloned authenticator or the
2079 original authenticator. Relying Parties should address this situation
2080 appropriately relative to their individual situations, i.e., their risk
2081 tolerance.
2082

2083 Authenticators:
2084 * should implement per-RP ID signature counters. This prevents the
2085 signature counter value from being shared between Relying Parties
2086 and being possibly employed as a correlation handle for the user.
2087 Authenticators may implement a global signature counter, i.e., on a
2088 per-authenticator basis, but this is less privacy-friendly for
2089 users.
2090 * should ensure that the signature counter value does not
2091 accidentally decrease (e.g., due to hardware failures).
2092

203E 6.2. Authenticator operations
 203F
 204C A client must connect to an authenticator in order to invoke any of the
 2041 operations of that authenticator. This connection defines an
 2042 authenticator session. An authenticator must maintain isolation between
 2043 sessions. It may do this by only allowing one session to exist at any
 2044 particular time, or by providing more complicated session management.
 2045
 2046 The following operations can be invoked by the client in an
 2047 authenticator session.
 2048
 2049 6.2.1. The authenticatorMakeCredential operation
 2050
 2051 This operation must be invoked in an authenticator session which has no
 2052 other operations in progress. It takes the following input parameters:
 2053
 2054 **rpld**
 2055 The caller's RP ID, as determined by the user agent and the
 2056 client.
 2057
 2058 **hash**
 2059 The hash of the serialized client data, provided by the client.
 2060
 2061 **rpEntity**
 2062 The Relying Party's PublicKeyCredentialRpEntity.
 2063
 2064 **userEntity**
 2065 The user account's PublicKeyCredentialUserEntity, containing the
 2066 user handle given by the Relying Party.
 2067
 2068
 2069 **credTypesAndPubKeyAlgs**
 2070 A sequence of pairs of PublicKeyCredentialType and public key
 2071 algorithms (COSEAlgorithmIdentifier) requested by the Relying
 2072 Party. This sequence is ordered from most preferred to least
 2073 preferred. The platform makes a best-effort to create the most
 2074 preferred credential that it can.
 2075
 2076 **excludeCredentialDescriptorList**
 2077 An optional list of PublicKeyCredentialDescriptor objects
 2078 provided by the Relying Party with the intention that, if any of
 2079 these are known to the authenticator, it should not create a new
 2080 credential. excludeCredentialDescriptorList contains a list of
 2081 known credentials.
 2082
 2083 **requireResidentKey**
 2084 options.authenticatorSelection.requireResidentKey.
 2085
 2086 **requireUserVerification**
 2087 options.authenticatorSelection.requireUserVerification
 2088
 2089 **extensions**
 2090 A map from extension identifiers to their authenticator
 2091 extension inputs, created by the client based on the extensions
 2092 requested by the Relying Party, if any.
 2093
 2094 When this operation is invoked, the authenticator must perform the
 2095 following procedure:
 2096 1. Check if all the supplied parameters are syntactically well-formed
 2097 and of the correct length. If not, return an error code equivalent
 2098 to "UnknownError" and terminate the operation.
 2099 2. Check if at least one of the specified combinations of
 2099 PublicKeyCredentialType and cryptographic parameters in
 2100 credTypesAndPubKeyAlgs is supported. If not, return an error code
 2101 equivalent to "NotSupportedError" and terminate the operation.

2093 6.2. Authenticator operations
 2094
 2095 A client must connect to an authenticator in order to invoke any of the
 2096 operations of that authenticator. This connection defines an
 2097 authenticator session. An authenticator must maintain isolation between
 2098 sessions. It may do this by only allowing one session to exist at any
 2099 particular time, or by providing more complicated session management.
 2100
 2101 The following operations can be invoked by the client in an
 2102 authenticator session.
 2103
 2104 6.2.1. The authenticatorMakeCredential operation
 2105
 2106 This operation must be invoked in an authenticator session which has no
 2107 other operations in progress. It takes the following input parameters:
 2108
 2109
 2110 **hash**
 2111 The hash of the serialized client data, provided by the client.
 2112
 2113 **rpEntity**
 2114 The Relying Party's PublicKeyCredentialRpEntity.
 2115
 2116 **userEntity**
 2117 The user account's PublicKeyCredentialUserEntity, containing the
 2118 user handle given by the Relying Party.
 2119
 2120 **requireResidentKey**
 2121 options.authenticatorSelection.requireResidentKey.
 2122
 2123 **requireUserVerification**
 2124 options.authenticatorSelection.requireUserVerification.
 2125
 2126 **credTypesAndPubKeyAlgs**
 2127 A sequence of pairs of PublicKeyCredentialType and public key
 2128 algorithms (COSEAlgorithmIdentifier) requested by the Relying
 2129 Party. This sequence is ordered from most preferred to least
 2130 preferred. The platform makes a best-effort to create the most
 2131 preferred credential that it can.
 2132
 2133 **excludeCredentialDescriptorList**
 2134 An optional list of PublicKeyCredentialDescriptor objects
 2135 provided by the Relying Party with the intention that, if any of
 2136 these are known to the authenticator, it should not create a new
 2137 credential. excludeCredentialDescriptorList contains a list of
 2138 known credentials.
 2139
 2140
 2141 **extensions**
 2142 A map from extension identifiers to their authenticator
 2143 extension inputs, created by the client based on the extensions
 2144 requested by the Relying Party, if any.
 2145
 2146 When this operation is invoked, the authenticator must perform the
 2147 following procedure:
 2148 1. Check if all the supplied parameters are syntactically well-formed
 2149 and of the correct length. If not, return an error code equivalent
 2150 to "UnknownError" and terminate the operation.
 2151 2. Check if at least one of the specified combinations of
 2152 PublicKeyCredentialType and cryptographic parameters in
 2153 credTypesAndPubKeyAlgs is supported. If not, return an error code
 2154 equivalent to "NotSupportedError" and terminate the operation.

2102 3. Check if a credential matching an item of
2103 excludeCredentialDescriptorList is present on this authenticator.
2104 If so, return an error code equivalent to "NotAllowedError" and
2105 terminate the operation.
2106 4. If requireResidentKey is true and the authenticator cannot store a
2107 Client-side-resident Credential Private Key, return an error code
2108 equivalent to "ConstraintError" and terminate the operation.
2109 5. If requireUserVerification is true and the authenticator cannot
2110 perform user verification, return an error code equivalent to
2111 "ConstraintError" and terminate the operation.
2112 6. Prompt the user for consent to create a new credential. The prompt
2113 for obtaining this consent is shown by the authenticator if it has
2114 its own output capability, or by the user agent otherwise. If the
2115 user denies consent, return an error code equivalent to
2116 "NotAllowedError" and terminate the operation. The Authenticator
2117 and user agent MAY skip this prompt if the Authenticator is a
2118 platform authenticator and excludeCredentialDescriptorList is
2119 empty.
2120 7. Once user consent has been obtained, generate a new credential
2121 object:
2122 1. Let (publicKey,privateKey) be a new set of cryptographic keys
2123 using the combination of PublicKeyCredentialType and
2124 cryptographic parameters represented by the first item in
2125 credTypesAndPubKeyAlgs that is supported by this
2126 authenticator.
2127 2. Let credentialId be a new identifier for this credential that
2128 is globally unique with high probability across all
2129 credentials with the same type across all authenticators.
2130 3. Let userHandle be userEntity.id.
2131 4. Associate the credentialId and privateKey with rpId and
2132 userHandle.
2133 5. Delete any older credentials with the same rpId and userHandle
2134 that are stored locally by the authenticator.
2135 8. If any error occurred while creating the new credential object,
2136 return an error code equivalent to "UnknownError" and terminate the
2137 operation.
2138 9. Let processedExtensions be the result of authenticator extension
2139 processing for each supported extension identifier/input pair in
2140 extensions.
2141 10. If the authenticator supports:
2142
2143 a per-RP ID signature counter
2144 allocate the counter, associate it with the RP ID, and
2145 initialize the counter value as zero.
2146
2147 a global signature counter
2148 Use the global signature counter's actual value when
2149 generating authenticator data.
2150
2151 a per credential signature counter
2152 allocate the counter, associate it with the new
2153 credential, and initialize the counter value as zero.
2154
2155 11. Let attestedCredentialData be the attested credential data byte
2156 array including the credentialId and publicKey.
2157 12. Let authenticatorData be the byte array specified in 6.1
2158 Authenticator data, including attestedCredentialData as the
2159 attestedCredentialData and processedExtensions, if any, as the
2160 extensions.
2161 13. Return the attestation object for the new credential created by the
2162 procedure specified in 6.3.4 Generating an Attestation Object
2163 using an authenticator-chosen attestation statement format,
2164 authenticatorData, and hash. For more details on attestation, see
2165 6.3 Attestation.
2166
2167 On successful completion of this operation, the authenticator returns
2168 the attestation object to the client.
2169
2170 6.2.2. The authenticatorGetAssertion operation
2171

2153 3. Check if a credential matching an item of
2154 excludeCredentialDescriptorList is present on this authenticator.
2155 If so, return an error code equivalent to "NotAllowedError" and
2156 terminate the operation.
2157 4. If requireResidentKey is true and the authenticator cannot store a
2158 Client-side-resident Credential Private Key, return an error code
2159 equivalent to "ConstraintError" and terminate the operation.
2160 5. If requireUserVerification is true and the authenticator cannot
2161 perform user verification, return an error code equivalent to
2162 "ConstraintError" and terminate the operation.
2163 6. Prompt the user for consent to create a new credential. The prompt
2164 for obtaining this consent is shown by the authenticator if it has
2165 its own output capability, or by the user agent otherwise. If the
2166 user denies consent, return an error code equivalent to
2167 "NotAllowedError" and terminate the operation. The Authenticator
2168 and user agent MAY skip this prompt if the Authenticator is a
2169 platform authenticator and excludeCredentialDescriptorList is
2170 empty.
2171 7. Once user consent has been obtained, generate a new credential
2172 object:
2173 1. Let (publicKey,privateKey) be a new set of cryptographic keys
2174 using the combination of PublicKeyCredentialType and
2175 cryptographic parameters represented by the first item in
2176 credTypesAndPubKeyAlgs that is supported by this
2177 authenticator.
2178 2. Let credentialId be a new identifier for this credential that
2179 is globally unique with high probability across all
2180 credentials with the same type across all authenticators.
2181 3. Let userHandle be userEntity.id.
2182 4. Associate the credentialId and privateKey with rpId and
2183 userHandle.
2184 5. Delete any older credentials with the same rpId and userHandle
2185 that are stored locally by the authenticator.
2186 8. If any error occurred while creating the new credential object,
2187 return an error code equivalent to "UnknownError" and terminate the
2188 operation.
2189 9. Let processedExtensions be the result of authenticator extension
2190 processing for each supported extension identifier/input pair in
2191 extensions.
2192 10. If the authenticator supports:
2193
2194 a per-RP ID signature counter
2195 allocate the counter, associate it with the RP ID, and
2196 initialize the counter value as zero.
2197
2198 a global signature counter
2199 Use the global signature counter's actual value when
2200 generating authenticator data.
2201
2202 a per credential signature counter
2203 allocate the counter, associate it with the new
2204 credential, and initialize the counter value as zero.
2205
2206 11. Let attestedCredentialData be the attested credential data byte
2207 array including the credentialId and publicKey.
2208 12. Let authenticatorData be the byte array specified in 6.1
2209 Authenticator data, including attestedCredentialData as the
2210 attestedCredentialData and processedExtensions, if any, as the
2211 extensions.
2212 13. Return the attestation object for the new credential created by the
2213 procedure specified in 6.3.4 Generating an Attestation Object
2214 using an authenticator-chosen attestation statement format,
2215 authenticatorData, and hash. For more details on attestation, see
2216 6.3 Attestation.
2217
2218 On successful completion of this operation, the authenticator returns
2219 the attestation object to the client.
2220
2221 6.2.2. The authenticatorGetAssertion operation
2222

2172 This operation must be invoked in an authenticator session which has no
 2173 other operations in progress. It takes the following input parameters:
 2174
 2175 rpId
 2176 The caller's RP ID, as determined by the user agent and the
 2177 client.
 2178
 2179 hash
 2180 The hash of the serialized client data, provided by the client.
 2181
 2182 allowCredentialDescriptorList
 2183 An optional list of PublicKeyCredentialDescriptors describing
 2184 credentials acceptable to the Relying Party (possibly filtered
 2185 by the client), if any.
 2186
 2187 extensions
 2188 A map from extension identifiers to their authenticator
 2189 extension inputs, created by the client based on the extensions
 2190 requested by the Relying Party, if any.
 2191
 2192 When this method is invoked, the authenticator must perform the
 2193 following procedure:
 2194 1. Check if all the supplied parameters are syntactically well-formed
 2195 and of the correct length. If not, return an error code equivalent
 2196 to "UnknownError" and terminate the operation.
 2197 2. If allowCredentialDescriptorList was not supplied, set it to a list
 2198 of all credentials stored for rpId (as determined by an exact match
 2199 of rpId).
 2200 3. Remove any items from allowCredentialDescriptorList that are not
 2201 present on this authenticator.
 2202 4. If allowCredentialDescriptorList is now empty, return an error code
 2203 equivalent to "NotAllowedError" and terminate the operation.
 2204 5. Prompt the user to select a credential selectedCredential from
 2205 allowCredentialDescriptorList. Obtain user consent for using
 2206 selectedCredential. The prompt for obtaining this consent may be
 2207 shown by the authenticator if it has its own output capability, or
 2208 by the user agent otherwise.
 2209 6. Let processedExtensions be the result of authenticator extension
 2210 processing for each supported extension identifier/input pair in
 2211 extensions.
 2212 7. Increment the RP ID-associated signature counter or the global
 2213 signature counter value, depending on which approach is implemented
 2214 by the authenticator, by some positive value.
 2215 8. Let authenticatorData be the byte array specified in 6.1
 2216 Authenticator data including processedExtensions, if any, as the
 2217 extensions and excluding attestedCredentialData.
 2218 9. Let signature be the assertion signature of the concatenation
 2219 authenticatorData || hash using the private key of
 2220 selectedCredential as shown in Figure 2, below. A simple,
 2221 undelimited concatenation is safe to use here because the
 2222 authenticator data describes its own length. The hash of the
 2223 serialized client data (which potentially has a variable length) is
 2224 always the last element.
 2225 [fido-signature-formats-figure2.svg] Generating an assertion
 2226 signature.
 2227 10. If any error occurred while generating the assertion signature,
 2228 return an error code equivalent to "UnknownError" and terminate the
 2229 operation.
 2230 11. Return to the user agent:
 2231 + selectedCredential's credential ID

- 2232 + authenticatorData
- 2233 + signature
- 2234 + The user handle associated with selectedCredential.

2235 If the authenticator cannot find any credential corresponding to the

2236

2223 This operation must be invoked in an authenticator session which has no
 2224 other operations in progress. It takes the following input parameters:
 2225
 2226 rpId
 2227 The caller's RP ID, as determined by the user agent and the
 2228 client.
 2229
 2230 hash
 2231 The hash of the serialized client data, provided by the client.
 2232
 2233 allowCredentialDescriptorList
 2234 An optional list of PublicKeyCredentialDescriptors describing
 2235 credentials acceptable to the Relying Party (possibly filtered
 2236 by the client), if any.
 2237
 2238 extensions
 2239 A map from extension identifiers to their authenticator
 2240 extension inputs, created by the client based on the extensions
 2241 requested by the Relying Party, if any.
 2242
 2243 When this method is invoked, the authenticator must perform the
 2244 following procedure:
 2245 1. Check if all the supplied parameters are syntactically well-formed
 2246 and of the correct length. If not, return an error code equivalent
 2247 to "UnknownError" and terminate the operation.
 2248 2. If allowCredentialDescriptorList was not supplied, set it to a list
 2249 of all credentials stored for rpId (as determined by an exact match
 2250 of rpId).
 2251 3. Remove any items from allowCredentialDescriptorList that are not
 2252 present on this authenticator.
 2253 4. If allowCredentialDescriptorList is now empty, return an error code
 2254 equivalent to "NotAllowedError" and terminate the operation.
 2255 5. Prompt the user to select a credential selectedCredential from
 2256 allowCredentialDescriptorList. Obtain user consent for using
 2257 selectedCredential. The prompt for obtaining this consent may be
 2258 shown by the authenticator if it has its own output capability, or
 2259 by the user agent otherwise.
 2260 6. Let processedExtensions be the result of authenticator extension
 2261 processing for each supported extension identifier/input pair in
 2262 extensions.
 2263 7. Increment the RP ID-associated signature counter or the global
 2264 signature counter value, depending on which approach is implemented
 2265 by the authenticator, by some positive value.
 2266 8. Let authenticatorData be the byte array specified in 6.1
 2267 Authenticator data including processedExtensions, if any, as the
 2268 extensions and excluding attestedCredentialData.
 2269 9. Let signature be the assertion signature of the concatenation
 2270 authenticatorData || hash using the private key of
 2271 selectedCredential as shown in Figure 2, below. A simple,
 2272 undelimited concatenation is safe to use here because the
 2273 authenticator data describes its own length. The hash of the
 2274 serialized client data (which potentially has a variable length) is
 2275 always the last element.
 2276 [fido-signature-formats-figure2.svg] Generating an assertion
 2277 signature.
 2278 10. If any error occurred while generating the assertion signature,
 2279 return an error code equivalent to "UnknownError" and terminate the
 2280 operation.
 2281 11. Return to the user agent:
 2282 + selectedCredential's credential ID, if either a list of
 2283 credentials of length 2 or greater was supplied by the client,
 2284 or no such list was supplied.
 2285 Note: If the client supplies a list of exactly one credential
 2286 and it was successfully employed, then its credential ID is
 2287 not returned since the client already knows it.
 2288 + authenticatorData
 2289 + signature
 2290 + The user handle associated with selectedCredential.

- 2282 + selectedCredential's credential ID, if either a list of
- 2283 credentials of length 2 or greater was supplied by the client,
- 2284 or no such list was supplied.
- 2285 Note: If the client supplies a list of exactly one credential
- 2286 and it was successfully employed, then its credential ID is
- 2287 not returned since the client already knows it.
- 2288 + authenticatorData
- 2289 + signature
- 2290 + The user handle associated with selectedCredential.

2291 If the authenticator cannot find any credential corresponding to the

2292

2237 specified Relying Party that matches the specified criteria, it
 2238 terminates the operation and returns an error.
 2239
 2240 If the user refuses consent, the authenticator returns an appropriate
 2241 error status to the client.
 2242
 2243 6.2.3. The authenticatorCancel operation
 2244
 2245 This operation takes no input parameters and returns no result.
 2246
 2247 When this operation is invoked by the client in an authenticator
 2248 session, it has the effect of terminating any
 2249 authenticatorMakeCredential or authenticatorGetAssertion operation
 2250 currently in progress in that authenticator session. The authenticator
 2251 stops prompting for, or accepting, any user input related to
 2252 authorizing the canceled operation. The client ignores any further
 2253 responses from the authenticator for the canceled operation.
 2254
 2255 This operation is ignored if it is invoked in an authenticator session
 2256 which does not have an authenticatorMakeCredential or
 2257 authenticatorGetAssertion operation currently in progress.
 2258
 2259 6.3. Attestation
 2260
 2261 Authenticators must also provide some form of attestation. The basic
 2262 requirement is that the authenticator can produce, for each credential
 2263 public key, an attestation statement verifiable by the Relying Party.
 2264 Typically, this attestation statement contains a signature by an
 2265 attestation private key over the attested credential public key and a
 2266 challenge, as well as a certificate or similar data providing
 2267 provenance information for the attestation public key, enabling the
 2268 Relying Party to make a trust decision. However, if an attestation key
 2269 pair is not available, then the authenticator MUST perform self
 2270 attestation of the credential public key with the corresponding
 2271 credential private key. All this information is returned by
 2272 authenticators any time a new public key credential is generated, in
 2273 the overall form of an attestation object. The relationship of the
 2274 attestation object with authenticator data (containing attested
 2275 credential data) and the attestation statement is illustrated in figure
 2276 3, below.
 2277 [fido-attestation-structures.svg] Attestation object layout
 2278 illustrating the included authenticator data (containing attested
 2279 credential data) and the attestation statement.
 2280 This figure illustrates only the packed attestation statement format.
 2281 Several additional attestation statement formats are defined in 8
 2282 Defined Attestation Statement Formats.
 2283
 2284 An important component of the attestation object is the attestation
 2285 statement. This is a specific type of signed data object, containing
 2286 statements about a public key credential itself and the authenticator
 2287 that created it. It contains an attestation signature created using the
 2288 key of the attesting authority (except for the case of self
 2289 attestation, when it is created using the credential private key). In
 2290 order to correctly interpret an attestation statement, a Relying Party
 2291 needs to understand these two aspects of attestation:
 2292 1. The attestation statement format is the manner in which the
 2293 signature is represented and the various contextual bindings are
 2294 incorporated into the attestation statement by the authenticator.
 2295 In other words, this defines the syntax of the statement. Various
 2296 existing devices and platforms (such as TPMs and the Android OS)
 2297 have previously defined attestation statement formats. This
 2298 specification supports a variety of such formats in an extensible
 2299 way, as defined in 6.3.2 Attestation Statement Formats.
 2300 2. The attestation type defines the semantics of attestation
 2301 statements and their underlying trust models. Specifically, it
 2302 defines how a Relying Party establishes trust in a particular
 2303 attestation statement, after verifying that it is cryptographically
 2304 valid. This specification supports a number of attestation types,
 2305 as described in 6.3.3 Attestation Types.
 2306

2293 specified Relying Party that matches the specified criteria, it
 2294 terminates the operation and returns an error.
 2295
 2296 If the user refuses consent, the authenticator returns an appropriate
 2297 error status to the client.
 2298
 2299 6.2.3. The authenticatorCancel operation
 2300
 2301 This operation takes no input parameters and returns no result.
 2302
 2303 When this operation is invoked by the client in an authenticator
 2304 session, it has the effect of terminating any
 2305 authenticatorMakeCredential or authenticatorGetAssertion operation
 2306 currently in progress in that authenticator session. The authenticator
 2307 stops prompting for, or accepting, any user input related to
 2308 authorizing the canceled operation. The client ignores any further
 2309 responses from the authenticator for the canceled operation.
 2310
 2311 This operation is ignored if it is invoked in an authenticator session
 2312 which does not have an authenticatorMakeCredential or
 2313 authenticatorGetAssertion operation currently in progress.
 2314
 2315 6.3. Attestation
 2316
 2317 Authenticators must also provide some form of attestation. The basic
 2318 requirement is that the authenticator can produce, for each credential
 2319 public key, an attestation statement verifiable by the Relying Party.
 2320 Typically, this attestation statement contains a signature by an
 2321 attestation private key over the attested credential public key and a
 2322 challenge, as well as a certificate or similar data providing
 2323 provenance information for the attestation public key, enabling the
 2324 Relying Party to make a trust decision. However, if an attestation key
 2325 pair is not available, then the authenticator MUST perform self
 2326 attestation of the credential public key with the corresponding
 2327 credential private key. All this information is returned by
 2328 authenticators any time a new public key credential is generated, in
 2329 the overall form of an attestation object. The relationship of the
 2330 attestation object with authenticator data (containing attested
 2331 credential data) and the attestation statement is illustrated in figure
 2332 3, below.
 2333 [fido-attestation-structures.svg] Attestation object layout
 2334 illustrating the included authenticator data (containing attested
 2335 credential data) and the attestation statement.
 2336 This figure illustrates only the packed attestation statement format.
 2337 Several additional attestation statement formats are defined in 8
 2338 Defined Attestation Statement Formats.
 2339
 2340 An important component of the attestation object is the attestation
 2341 statement. This is a specific type of signed data object, containing
 2342 statements about a public key credential itself and the authenticator
 2343 that created it. It contains an attestation signature created using the
 2344 key of the attesting authority (except for the case of self
 2345 attestation, when it is created using the credential private key). In
 2346 order to correctly interpret an attestation statement, a Relying Party
 2347 needs to understand these two aspects of attestation:
 2348 1. The attestation statement format is the manner in which the
 2349 signature is represented and the various contextual bindings are
 2350 incorporated into the attestation statement by the authenticator.
 2351 In other words, this defines the syntax of the statement. Various
 2352 existing devices and platforms (such as TPMs and the Android OS)
 2353 have previously defined attestation statement formats. This
 2354 specification supports a variety of such formats in an extensible
 2355 way, as defined in 6.3.2 Attestation Statement Formats.
 2356 2. The attestation type defines the semantics of attestation
 2357 statements and their underlying trust models. Specifically, it
 2358 defines how a Relying Party establishes trust in a particular
 2359 attestation statement, after verifying that it is cryptographically
 2360 valid. This specification supports a number of attestation types,
 2361 as described in 6.3.3 Attestation Types.
 2362

2307 In general, there is no simple mapping between attestation statement
2308 formats and attestation types. For example, the "packed" attestation
2309 statement format defined in 8.2 Packed Attestation Statement Format
2310 can be used in conjunction with all attestation types, while other
2311 formats and types have more limited applicability.

2312 The privacy, security and operational characteristics of attestation
2313 depend on:

- 2314 * The attestation type, which determines the trust model,
- 2315 * The attestation statement format, which may constrain the strength
2316 of the attestation by limiting what can be expressed in an
2317 attestation statement, and
- 2318 * The characteristics of the individual authenticator, such as its
2319 construction, whether part or all of it runs in a secure operating
2320 environment, and so on.

2321 It is expected that most authenticators will support a small number of
2322 attestation types and attestation statement formats, while Relying
2323 Parties will decide what attestation types are acceptable to them by
2324 policy. Relying Parties will also need to understand the
2325 characteristics of the authenticators that they trust, based on
2326 information they have about these authenticators. For example, the FIDO
2327 Metadata Service [FIDOMetadataService] provides one way to access such
2328 information.

2329 6.3.1. Attested credential data

2330 Attested credential data is a variable-length byte array added to the
2331 authenticator data when generating an attestation object for a given
2332 credential. It has the following format:
2333 Name Length (in bytes) Description
2334 aaguid 16 The AAGUID of the authenticator.
2335 credentialIdLength 2 Byte length L of Credential ID
2336 credentialId L Credential ID
2337 credentialPublicKey variable The credential public key encoded in
2338 COSE_Key format, as defined in Section 7 of [RFC8152]. The encoded
2339 credential public key MUST contain the "alg" parameter and MUST NOT
2340 contain any other optional parameters. The "alg" parameter MUST contain
2341 a COSEAlgorithmIdentifier value.

2342 NOTE: The names in the Name column in the above table are only for
2343 reference within this document, and are not present in the actual
2344 representation of the attested credential data.

2345 6.3.2. Attestation Statement Formats

2346 As described above, an attestation statement format is a data format
2347 which represents a cryptographic signature by an authenticator over a
2348 set of contextual bindings. Each attestation statement format MUST be
2349 defined using the following template:

- 2350 * Attestation statement format identifier:
- 2351 * Supported attestation types:
- 2352 * Syntax: The syntax of an attestation statement produced in this
2353 format, defined using [CDDL] for the extension point \$attStmtFormat
2354 defined in 6.3.4 Generating an Attestation Object.
- 2355 * Signing procedure: The signing procedure for computing an
2356 attestation statement in this format given the public key
2357 credential to be attested, the authenticator data structure
2358 containing the authenticator data for the attestation, and the hash
2359 of the serialized client data.
- 2360 * Verification procedures: The procedure for verifying an attestation
2361 statement, which takes as inputs the authenticator data structure
2362 containing the authenticator data claimed to have been used for the
2363 attestation and the hash of the serialized client data, and returns
2364 either:
2365 + An error indicating that the attestation is invalid, or
2366 + The attestation type, and the trust path of the attestation.
2367 This trust path is either empty (in case of self attestation),
2368 an identifier of a ECDAA-Issuer public key (in the case of
2369 ECDAA), or a set of X.509 certificates.

2363 In general, there is no simple mapping between attestation statement
2364 formats and attestation types. For example, the "packed" attestation
2365 statement format defined in 8.2 Packed Attestation Statement Format
2366 can be used in conjunction with all attestation types, while other
2367 formats and types have more limited applicability.

2368 The privacy, security and operational characteristics of attestation
2369 depend on:

- 2370 * The attestation type, which determines the trust model,
- 2371 * The attestation statement format, which may constrain the strength
2372 of the attestation by limiting what can be expressed in an
2373 attestation statement, and
- 2374 * The characteristics of the individual authenticator, such as its
2375 construction, whether part or all of it runs in a secure operating
2376 environment, and so on.

2377 It is expected that most authenticators will support a small number of
2378 attestation types and attestation statement formats, while Relying
2379 Parties will decide what attestation types are acceptable to them by
2380 policy. Relying Parties will also need to understand the
2381 characteristics of the authenticators that they trust, based on
2382 information they have about these authenticators. For example, the FIDO
2383 Metadata Service [FIDOMetadataService] provides one way to access such
2384 information.

2385 6.3.1. Attested credential data

2386 Attested credential data is a variable-length byte array added to the
2387 authenticator data when generating an attestation object for a given
2388 credential. It has the following format:
2389 Name Length (in bytes) Description
2390 aaguid 16 The AAGUID of the authenticator.
2391 credentialIdLength 2 Byte length L of Credential ID
2392 credentialId L Credential ID
2393 credentialPublicKey variable The credential public key encoded in
2394 COSE_Key format, as defined in Section 7 of [RFC8152]. The encoded
2395 credential public key MUST contain the "alg" parameter and MUST NOT
2396 contain any other optional parameters. The "alg" parameter MUST contain
2397 a COSEAlgorithmIdentifier value.

2398 NOTE: The names in the Name column in the above table are only for
2399 reference within this document, and are not present in the actual
2400 representation of the attested credential data.

2401 6.3.2. Attestation Statement Formats

2402 As described above, an attestation statement format is a data format
2403 which represents a cryptographic signature by an authenticator over a
2404 set of contextual bindings. Each attestation statement format MUST be
2405 defined using the following template:

- 2406 * Attestation statement format identifier:
- 2407 * Supported attestation types:
- 2408 * Syntax: The syntax of an attestation statement produced in this
2409 format, defined using [CDDL] for the extension point \$attStmtFormat
2410 defined in 6.3.4 Generating an Attestation Object.
- 2411 * Signing procedure: The signing procedure for computing an
2412 attestation statement in this format given the public key
2413 credential to be attested, the authenticator data structure
2414 containing the authenticator data for the attestation, and the hash
2415 of the serialized client data.
- 2416 * Verification procedures: The procedure for verifying an attestation
2417 statement, which takes as inputs the authenticator data structure
2418 containing the authenticator data claimed to have been used for the
2419 attestation and the hash of the serialized client data, and returns
2420 either:
2421 + An error indicating that the attestation is invalid, or
2422 + The attestation type, and the trust path of the attestation.
2423 This trust path is either empty (in case of self attestation),
2424 an identifier of a ECDAA-Issuer public key (in the case of
2425 ECDAA), or a set of X.509 certificates.

2377
2378 The initial list of specified attestation statement formats is in 8
2379 Defined Attestation Statement Formats.

2380 6.3.3. Attestation Types

2381
2382
2383 WebAuthn supports multiple attestation types:

2384 Basic Attestation

2385 In the case of basic attestation [UAFProtocol], the
2386 authenticator's attestation key pair is specific to an
2387 authenticator model. Thus, authenticators of the same model
2388 often share the same attestation key pair. See 6.3.5.1 Privacy
2389 for further information.

2390 Self Attestation

2391 In the case of self attestation, also known as surrogate basic
2392 attestation [UAFProtocol], the Authenticator does not have any
2393 specific attestation key. Instead it uses the authentication key
2394 itself to create the attestation signature. Authenticators
2395 without meaningful protection measures for an attestation
2396 private key typically use this attestation type.

2397 Privacy CA

2398 In this case, the Authenticator owns an authenticator-specific
2399 (endorsement) key. This key is used to securely communicate with
2400 a trusted third party, the Privacy CA. The Authenticator can
2401 generate multiple attestation key pairs and asks the Privacy CA
2402 to issue an attestation certificate for it. Using this approach,
2403 the Authenticator can limit the exposure of the endorsement key
2404 (which is a global correlation handle) to Privacy CA(s).
2405 Attestation keys can be requested for each public key credential
2406 individually.

2407
2408 Note: This concept typically leads to multiple attestation
2409 certificates. The attestation certificate requested most
2410 recently is called "active".

2411 Elliptic Curve based Direct Anonymous Attestation (ECDAA)

2412 In this case, the Authenticator receives direct anonymous
2413 attestation (DAA) credentials from a single DAA-Issuer. These
2414 DAA credentials are used along with blinding to sign the
2415 attested credential data. The concept of blinding avoids the DAA
2416 credentials being misused as global correlation handle. WebAuthn
2417 supports DAA using elliptic curve cryptography and bilinear
2418 pairings, called ECDAA (see [FIDOEcdaaAlgorithm]) in this
2419 specification. Consequently we denote the DAA-Issuer as
2420 ECDAA-Issuer (see [FIDOEcdaaAlgorithm]).

2421 6.3.4. Generating an Attestation Object

2422 To generate an attestation object (see: Figure 3) given:

2423
2424 attestationFormat
2425 An attestation statement format.

2426
2427 authData
2428 A byte array containing authenticator data.

2429
2430 hash
2431 The hash of the serialized client data.

2432
2433 the authenticator MUST:

- 2434 1. Let attStmt be the result of running attestationFormat's signing
2435 procedure given authData and hash.
- 2436 2. Let fmt be attestationFormat's attestation statement format
2437 identifier
- 2438 3. Return the attestation object as a CBOR map with the following
2439 syntax, filled in with variables initialized by this algorithm:
2440 attObj = {

2433
2434 The initial list of specified attestation statement formats is in 8
2435 Defined Attestation Statement Formats.

2436 6.3.3. Attestation Types

2437
2438
2439 WebAuthn supports multiple attestation types:

2440 Basic Attestation

2441 In the case of basic attestation [UAFProtocol], the
2442 authenticator's attestation key pair is specific to an
2443 authenticator model. Thus, authenticators of the same model
2444 often share the same attestation key pair. See 6.3.5.1 Privacy
2445 for further information.

2446 Self Attestation

2447 In the case of self attestation, also known as surrogate basic
2448 attestation [UAFProtocol], the Authenticator does not have any
2449 specific attestation key. Instead it uses the authentication key
2450 itself to create the attestation signature. Authenticators
2451 without meaningful protection measures for an attestation
2452 private key typically use this attestation type.

2453 Privacy CA

2454 In this case, the Authenticator owns an authenticator-specific
2455 (endorsement) key. This key is used to securely communicate with
2456 a trusted third party, the Privacy CA. The Authenticator can
2457 generate multiple attestation key pairs and asks the Privacy CA
2458 to issue an attestation certificate for it. Using this approach,
2459 the Authenticator can limit the exposure of the endorsement key
2460 (which is a global correlation handle) to Privacy CA(s).
2461 Attestation keys can be requested for each public key credential
2462 individually.

2463
2464 Note: This concept typically leads to multiple attestation
2465 certificates. The attestation certificate requested most
2466 recently is called "active".

2467 Elliptic Curve based Direct Anonymous Attestation (ECDAA)

2468 In this case, the Authenticator receives direct anonymous
2469 attestation (DAA) credentials from a single DAA-Issuer. These
2470 DAA credentials are used along with blinding to sign the
2471 attested credential data. The concept of blinding avoids the DAA
2472 credentials being misused as global correlation handle. WebAuthn
2473 supports DAA using elliptic curve cryptography and bilinear
2474 pairings, called ECDAA (see [FIDOEcdaaAlgorithm]) in this
2475 specification. Consequently we denote the DAA-Issuer as
2476 ECDAA-Issuer (see [FIDOEcdaaAlgorithm]).

2477 6.3.4. Generating an Attestation Object

2478 To generate an attestation object (see: Figure 3) given:

2479
2480 attestationFormat
2481 An attestation statement format.

2482
2483 authData
2484 A byte array containing authenticator data.

2485
2486 hash
2487 The hash of the serialized client data.

2488
2489 the authenticator MUST:

- 2490 1. Let attStmt be the result of running attestationFormat's signing
2491 procedure given authData and hash.
- 2492 2. Let fmt be attestationFormat's attestation statement format
2493 identifier
- 2494 3. Return the attestation object as a CBOR map with the following
2495 syntax, filled in with variables initialized by this algorithm:
2496 attObj = {

```

2447     authData: bytes,
2448     $$attStmtType
2449   }
2450
2451   attStmtTemplate = (
2452     fmt: text,
2453     attStmt: { * tstr => any } ; Map is filled in by each
2454 concrete attStmtType
2455 )
2456
2457 ; Every attestation statement format must have the above fields
2458 attStmtTemplate .within $$attStmtType
2459
2460 6.3.5. Security Considerations
2461
2462 6.3.5.1. Privacy
2463
2464 Attestation keys may be used to track users or link various online
2465 identities of the same user together. This may be mitigated in several
2466 ways, including:
2467 * A WebAuthn authenticator manufacturer may choose to ship all of
2468 their devices with the same (or a fixed number of) attestation
2469 key(s) (called Basic Attestation). This will anonymize the user at
2470 the risk of not being able to revoke a particular attestation key
2471 should its WebAuthn Authenticator be compromised.
2472 * A WebAuthn Authenticator may be capable of dynamically generating
2473 different attestation keys (and requesting related certificates)
2474 per origin (following the Privacy CA approach). For example, a
2475 WebAuthn Authenticator can ship with a master attestation key (and
2476 certificate), and combined with a cloud operated privacy CA, can
2477 dynamically generate per origin attestation keys and attestation
2478 certificates.
2479 * A WebAuthn Authenticator can implement Elliptic Curve based direct
2480 anonymous attestation (see [FIDOEcdaaAlgorithm]). Using this
2481 scheme, the authenticator generates a blinded attestation
2482 signature. This allows the Relying Party to verify the signature
2483 using the ECDAAs-Issuer public key, but the attestation signature
2484 does not serve as a global correlation handle.
2485
2486 6.3.5.2. Attestation Certificate and Attestation Certificate CA Compromise
2487
2488 When an intermediate CA or a root CA used for issuing attestation
2489 certificates is compromised, WebAuthn authenticator attestation keys
2490 are still safe although their certificates can no longer be trusted. A
2491 WebAuthn Authenticator manufacturer that has recorded the public
2492 attestation keys for their devices can issue new attestation
2493 certificates for these keys from a new intermediate CA or from a new
2494 root CA. If the root CA changes, the Relying Parties must update their
2495 trusted root certificates accordingly.
2496
2497 A WebAuthn Authenticator attestation certificate must be revoked by the
2498 issuing CA if its key has been compromised. A WebAuthn Authenticator
2499 manufacturer may need to ship a firmware update and inject new
2500 attestation keys and certificates into already manufactured WebAuthn
2501 Authenticators, if the exposure was due to a firmware flaw. (The
2502 process by which this happens is out of scope for this specification.)
2503 If the WebAuthn Authenticator manufacturer does not have this
2504 capability, then it may not be possible for Relying Parties to trust
2505 any further attestation statements from the affected WebAuthn
2506 Authenticators.
2507
2508 If attestation certificate validation fails due to a revoked
2509 intermediate attestation CA certificate, and the Relying Party's policy
2510 requires rejecting the registration/authentication request in these
2511 situations, then it is recommended that the Relying Party also
2512 un-registers (or marks with a trust level equivalent to "self
2513 attestation") public key credentials that were registered after the CA
2514 compromise date using an attestation certificate chaining up to the
2515 same intermediate CA. It is thus recommended that Relying Parties
2516 remember intermediate attestation CA certificates during Authenticator

```

```

2503     authData: bytes,
2504     $$attStmtType
2505   }
2506
2507   attStmtTemplate = (
2508     fmt: text,
2509     attStmt: { * tstr => any } ; Map is filled in by each
2510 concrete attStmtType
2511 )
2512
2513 ; Every attestation statement format must have the above fields
2514 attStmtTemplate .within $$attStmtType
2515
2516 6.3.5. Security Considerations
2517
2518 6.3.5.1. Privacy
2519
2520 Attestation keys may be used to track users or link various online
2521 identities of the same user together. This may be mitigated in several
2522 ways, including:
2523 * A WebAuthn authenticator manufacturer may choose to ship all of
2524 their devices with the same (or a fixed number of) attestation
2525 key(s) (called Basic Attestation). This will anonymize the user at
2526 the risk of not being able to revoke a particular attestation key
2527 should its WebAuthn Authenticator be compromised.
2528 * A WebAuthn Authenticator may be capable of dynamically generating
2529 different attestation keys (and requesting related certificates)
2530 per origin (following the Privacy CA approach). For example, a
2531 WebAuthn Authenticator can ship with a master attestation key (and
2532 certificate), and combined with a cloud operated privacy CA, can
2533 dynamically generate per origin attestation keys and attestation
2534 certificates.
2535 * A WebAuthn Authenticator can implement Elliptic Curve based direct
2536 anonymous attestation (see [FIDOEcdaaAlgorithm]). Using this
2537 scheme, the authenticator generates a blinded attestation
2538 signature. This allows the Relying Party to verify the signature
2539 using the ECDAAs-Issuer public key, but the attestation signature
2540 does not serve as a global correlation handle.
2541
2542 6.3.5.2. Attestation Certificate and Attestation Certificate CA Compromise
2543
2544 When an intermediate CA or a root CA used for issuing attestation
2545 certificates is compromised, WebAuthn authenticator attestation keys
2546 are still safe although their certificates can no longer be trusted. A
2547 WebAuthn Authenticator manufacturer that has recorded the public
2548 attestation keys for their devices can issue new attestation
2549 certificates for these keys from a new intermediate CA or from a new
2550 root CA. If the root CA changes, the Relying Parties must update their
2551 trusted root certificates accordingly.
2552
2553 A WebAuthn Authenticator attestation certificate must be revoked by the
2554 issuing CA if its key has been compromised. A WebAuthn Authenticator
2555 manufacturer may need to ship a firmware update and inject new
2556 attestation keys and certificates into already manufactured WebAuthn
2557 Authenticators, if the exposure was due to a firmware flaw. (The
2558 process by which this happens is out of scope for this specification.)
2559 If the WebAuthn Authenticator manufacturer does not have this
2560 capability, then it may not be possible for Relying Parties to trust
2561 any further attestation statements from the affected WebAuthn
2562 Authenticators.
2563
2564 If attestation certificate validation fails due to a revoked
2565 intermediate attestation CA certificate, and the Relying Party's policy
2566 requires rejecting the registration/authentication request in these
2567 situations, then it is recommended that the Relying Party also
2568 un-registers (or marks with a trust level equivalent to "self
2569 attestation") public key credentials that were registered after the CA
2570 compromise date using an attestation certificate chaining up to the
2571 same intermediate CA. It is thus recommended that Relying Parties
2572 remember intermediate attestation CA certificates during Authenticator

```

2517 registration in order to un-register related public key credentials if
2518 the registration was performed after revocation of such certificates.
2519
2520 If an ECDAA attestation key has been compromised, it can be added to
2521 the RogueList (i.e., the list of revoked authenticators) maintained by
2522 the related ECDAA-Issuer. The Relying Party should verify whether an
2523 authenticator belongs to the RogueList when performing ECDAA-Verify
2524 (see section 3.6 in [FIDOEcdaaAlgorithm]). For example, the FIDO
2525 Metadata Service [FIDOMetadataService] provides one way to access such
2526 information.

6.3.5.3. Attestation Certificate Hierarchy

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

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

7. Relying Party Operations

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

7.1. Registering a new credential

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

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

2573 registration in order to un-register related public key credentials if
2574 the registration was performed after revocation of such certificates.
2575

2576 If an ECDAA attestation key has been compromised, it can be added to
2577 the RogueList (i.e., the list of revoked authenticators) maintained by
2578 the related ECDAA-Issuer. The Relying Party should verify whether an
2579 authenticator belongs to the RogueList when performing ECDAA-Verify
2580 (see section 3.6 in [FIDOEcdaaAlgorithm]). For example, the FIDO
2581 Metadata Service [FIDOMetadataService] provides one way to access such
2582 information.

6.3.5.3. Attestation Certificate Hierarchy

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

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

7. Relying Party Operations

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

7.1. Registering a new credential

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

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

anchors (attestation root certificates or ECDAAs-Issuer public keys) for that attestation type and attestation statement format fmt, from a trusted source or from policy. For example, the FIDO Metadata Service [FIDOMetadataService] provides one way to obtain such information, using the aaguid in the attestedCredentialData in authData.

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

- + If self attestation was used, check if self attestation is acceptable under Relying Party policy.
- + If ECDAAs were used, verify that the identifier of the ECDAAs-Issuer public key used is included in the set of acceptable trust anchors obtained in step 11.
- + Otherwise, use the X.509 certificates returned by the verification procedure to verify that the attestation public key correctly chains up to an acceptable root certificate.

13. If the attestation statement attStmt verified successfully and is found to be trustworthy, then register the new credential with the account that was denoted in the options.user passed to create(), by associating it with the credentialId and credentialPublicKey in the attestedCredentialData in authData, as appropriate for the Relying Party's system.

14. If the attestation statement attStmt successfully verified but is not trustworthy per step 12 above, the Relying Party SHOULD fail the registration ceremony.

NOTE: However, if permitted by policy, the Relying Party MAY register the credential ID and credential public key but treat the credential as one with self attestation (see 6.3.3 Attestation Types). If doing so, the Relying Party is asserting there is no cryptographic proof that the public key credential has been generated by a particular authenticator model. See [FIDOsecRef] and [UAFProtocol] for a more detailed discussion.

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

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

7.2. Verifying an authentication assertion

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

1. Using credential's id attribute (or the corresponding rawId, if base64url encoding is inappropriate for your use case), look up the corresponding credential public key.
2. Let cData, aData and sig denote the value of credential's response's clientDataJSON, authenticatorData, and signature respectively.
3. Perform JSON deserialization on cData to extract the client data C used for the signature.
4. Verify that the challenge member of C matches the challenge that was sent to the authenticator in the PublicKeyCredentialRequestOptions passed to the get() call.
5. Verify that the origin member of C matches the Relying Party's origin.
6. Verify that the tokenBindingId member of C (if present) matches the Token Binding ID for the TLS connection over which the signature was obtained.
7. Verify that the clientExtensions member of C is a subset of the

anchors (attestation root certificates or ECDAAs-Issuer public keys) for that attestation type and attestation statement format fmt, from a trusted source or from policy. For example, the FIDO Metadata Service [FIDOMetadataService] provides one way to obtain such information, using the aaguid in the attestedCredentialData in authData.

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

- + If self attestation was used, check if self attestation is acceptable under Relying Party policy.
- + If ECDAAs were used, verify that the identifier of the ECDAAs-Issuer public key used is included in the set of acceptable trust anchors obtained in step 11.
- + Otherwise, use the X.509 certificates returned by the verification procedure to verify that the attestation public key correctly chains up to an acceptable root certificate.

13. If the attestation statement attStmt verified successfully and is found to be trustworthy, then register the new credential with the account that was denoted in the options.user passed to create(), by associating it with the credentialId and credentialPublicKey in the attestedCredentialData in authData, as appropriate for the Relying Party's system.

14. If the attestation statement attStmt successfully verified but is not trustworthy per step 12 above, the Relying Party SHOULD fail the registration ceremony.

NOTE: However, if permitted by policy, the Relying Party MAY register the credential ID and credential public key but treat the credential as one with self attestation (see 6.3.3 Attestation Types). If doing so, the Relying Party is asserting there is no cryptographic proof that the public key credential has been generated by a particular authenticator model. See [FIDOsecRef] and [UAFProtocol] for a more detailed discussion.

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

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

7.2. Verifying an authentication assertion

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

1. Using credential's id attribute (or the corresponding rawId, if base64url encoding is inappropriate for your use case), look up the corresponding credential public key.
2. Let cData, aData and sig denote the value of credential's response's clientDataJSON, authenticatorData, and signature respectively.
3. Perform JSON deserialization on cData to extract the client data C used for the signature.
4. Verify that the challenge member of C matches the challenge that was sent to the authenticator in the PublicKeyCredentialRequestOptions passed to the get() call.
5. Verify that the origin member of C matches the Relying Party's origin.
6. Verify that the tokenBindingId member of C (if present) matches the Token Binding ID for the TLS connection over which the signature was obtained.
7. Verify that the clientExtensions member of C is a subset of the

2657 extensions requested by the Relying Party and that the
2658 authenticatorExtensions in C is also a subset of the extensions
2659 requested by the Relying Party.
2660 8. Verify that the rpIdHash in aData is the SHA-256 hash of the RP ID
2661 expected by the Relying Party.
2662 9. Let hash be the result of computing a hash over the cData using the
2663 algorithm represented by the hashAlgorithm member of C.
2664 10. Using the credential public key looked up in step 1, verify that
2665 sig is a valid signature over the binary concatenation of aData and
2666 hash.
2667 11. If the signature counter value adata.signCount is nonzero or the
2668 value stored in conjunction with credential's id attribute is
2669 nonzero, then run the following substep:
2670 + If the signature counter value adata.signCount is
2671
2672 greater than the signature counter value stored in
2673 conjunction with credential's id attribute.
2674 Update the stored signature counter value,
2675 associated with credential's id attribute, to be the
2676 value of adata.signCount.
2677
2678 less than or equal to the signature counter value stored in
2679 conjunction with credential's id attribute.
2680 This is an signal that the authenticator may be
2681 cloned, i.e. at least two copies of the credential
2682 private key may exist and are being used in
2683 parallel. Relying Parties should incorporate this
2684 information into their risk scoring. Whether the
2685 Relying Party updates the stored signature counter
2686 value in this case, or not, or fails the
2687 authentication ceremony or not, is Relying
2688 Party-specific.
2689
2690 12. If all the above steps are successful, continue with the
2691 authentication ceremony as appropriate. Otherwise, fail the
2692 authentication ceremony.
2693
2694 8. Defined Attestation Statement Formats
2695
2696 WebAuthn supports pluggable attestation statement formats. This section
2697 defines an initial set of such formats.
2698
2699 8.1. Attestation Statement Format Identifiers
2700
2701 Attestation statement formats are identified by a string, called a
2702 attestation statement format identifier, chosen by the author of the
2703 attestation statement format.
2704
2705 Attestation statement format identifiers SHOULD be registered per
2706 [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)".
2707 All registered attestation statement format identifiers are unique
2708 amongst themselves as a matter of course.
2709
2710 Unregistered attestation statement format identifiers SHOULD use
2711 lowercase reverse domain-name naming, using a domain name registered by
2712 the developer, in order to assure uniqueness of the identifier. All
2713 attestation statement format identifiers MUST be a maximum of 32 octets
2714 in length and MUST consist only of printable USASCII characters,
2715 excluding backslash and doublequote, i.e., VCHAR as defined in
2716 [RFC5234] but without %x22 and %x5c.
2717
2718 Note: This means attestation statement format identifiers based on
2719 domain names MUST incorporate only LDH Labels [RFC5890].
2720
2721 Implementations MUST match WebAuthn attestation statement format
2722 identifiers in a case-sensitive fashion.
2723
2724 Attestation statement formats that may exist in multiple versions
2725 SHOULD include a version in their identifier. In effect, different
2726 versions are thus treated as different formats, e.g., packed2 as a new

2713 extensions requested by the Relying Party and that the
2714 authenticatorExtensions in C is also a subset of the extensions
2715 requested by the Relying Party.
2716 8. Verify that the rpIdHash in aData is the SHA-256 hash of the RP ID
2717 expected by the Relying Party.
2718 9. Let hash be the result of computing a hash over the cData using the
2719 algorithm represented by the hashAlgorithm member of C.
2720 10. Using the credential public key looked up in step 1, verify that
2721 sig is a valid signature over the binary concatenation of aData and
2722 hash.
2723 11. If the signature counter value adata.signCount is nonzero or the
2724 value stored in conjunction with credential's id attribute is
2725 nonzero, then run the following substep:
2726 + If the signature counter value adata.signCount is
2727
2728 greater than the signature counter value stored in
2729 conjunction with credential's id attribute.
2730 Update the stored signature counter value,
2731 associated with credential's id attribute, to be the
2732 value of adata.signCount.
2733
2734 less than or equal to the signature counter value stored in
2735 conjunction with credential's id attribute.
2736 This is an signal that the authenticator may be
2737 cloned, i.e. at least two copies of the credential
2738 private key may exist and are being used in
2739 parallel. Relying Parties should incorporate this
2740 information into their risk scoring. Whether the
2741 Relying Party updates the stored signature counter
2742 value in this case, or not, or fails the
2743 authentication ceremony or not, is Relying
2744 Party-specific.
2745
2746 12. If all the above steps are successful, continue with the
2747 authentication ceremony as appropriate. Otherwise, fail the
2748 authentication ceremony.
2749
2750 8. Defined Attestation Statement Formats
2751
2752 WebAuthn supports pluggable attestation statement formats. This section
2753 defines an initial set of such formats.
2754
2755 8.1. Attestation Statement Format Identifiers
2756
2757 Attestation statement formats are identified by a string, called a
2758 attestation statement format identifier, chosen by the author of the
2759 attestation statement format.
2760
2761 Attestation statement format identifiers SHOULD be registered per
2762 [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)".
2763 All registered attestation statement format identifiers are unique
2764 amongst themselves as a matter of course.
2765
2766 Unregistered attestation statement format identifiers SHOULD use
2767 lowercase reverse domain-name naming, using a domain name registered by
2768 the developer, in order to assure uniqueness of the identifier. All
2769 attestation statement format identifiers MUST be a maximum of 32 octets
2770 in length and MUST consist only of printable USASCII characters,
2771 excluding backslash and doublequote, i.e., VCHAR as defined in
2772 [RFC5234] but without %x22 and %x5c.
2773
2774 Note: This means attestation statement format identifiers based on
2775 domain names MUST incorporate only LDH Labels [RFC5890].
2776
2777 Implementations MUST match WebAuthn attestation statement format
2778 identifiers in a case-sensitive fashion.
2779
2780 Attestation statement formats that may exist in multiple versions
2781 SHOULD include a version in their identifier. In effect, different
2782 versions are thus treated as different formats, e.g., packed2 as a new

2727 version of the packed attestation statement format.
 2728
 2729 The following sections present a set of currently-defined and
 2730 registered attestation statement formats and their identifiers. The
 2731 up-to-date list of registered WebAuthn Extensions is maintained in the
 2732 IANA "WebAuthn Attestation Statement Format Identifier" registry
 2733 established by [WebAuthn-Registries].
 2734

2735 8.2. Packed Attestation Statement Format

2736
 2737 This is a WebAuthn optimized attestation statement format. It uses a
 2738 very compact but still extensible encoding method. It is implementable
 2739 by authenticators with limited resources (e.g., secure elements).
 2740

2741 Attestation statement format identifier
 2742 packed

2743

2744 Attestation types supported
 2745 All

2746

2747 Syntax
 2748 The syntax of a Packed Attestation statement is defined by the
 2749 following CDDL:
 2750

```

2751 $$attStmtType //= (
2752     fmt: "packed",
2753     attStmt: packedStmtFormat
2754 )
2755
2756 packedStmtFormat = {
2757     alg: COSEAlgorithmIdentifier,
2758     sig: bytes,
2759     x5c: [ attestnCert: bytes, * (caCert: bytes) ]
2760 } //
2761     alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261
2762 for ED512)
2763     sig: bytes,
2764     ecdaaKeyld: bytes
2765 }
  
```

2766

2767 The semantics of the fields are as follows:

2768

2769 alg
 2770 A COSEAlgorithmIdentifier containing the identifier of the
 2771 algorithm used to generate the attestation signature.

2772

2773 sig
 2774 A byte string containing the attestation signature.

2775

2776 x5c
 2777 The elements of this array contain the attestation
 2778 certificate and its certificate chain, each encoded in
 2779 X.509 format. The attestation certificate must be the
 2780 first element in the array.

2781

2782 ecdaaKeyld
 2783 The identifier of the ECDAA-Issuer public key. This is the
 2784 BigNumberToB encoding of the component "c" of the
 2785 ECDAA-Issuer public key as defined section 3.3, step 3.5
 2786 in [FIDOecdaaAlgorithm].
 2787

2788

2789 Signing procedure
 2790 The signing procedure for this attestation statement format is
 2791 similar to the procedure for generating assertion signatures.
 2792

- 2793 1. Let authenticatorData denote the authenticator data for the
 2794 attestation, and let clientDataHash denote the hash of the
 2795 serialized client data.
- 2796 2. If Basic or Privacy CA attestation is in use, the

2783 version of the packed attestation statement format.
 2784
 2785 The following sections present a set of currently-defined and
 2786 registered attestation statement formats and their identifiers. The
 2787 up-to-date list of registered WebAuthn Extensions is maintained in the
 2788 IANA "WebAuthn Attestation Statement Format Identifier" registry
 2789 established by [WebAuthn-Registries].
 2790

2791 8.2. Packed Attestation Statement Format

2792
 2793 This is a WebAuthn optimized attestation statement format. It uses a
 2794 very compact but still extensible encoding method. It is implementable
 2795 by authenticators with limited resources (e.g., secure elements).
 2796

2797 Attestation statement format identifier
 2798 packed

2799

2800 Attestation types supported
 2801 All

2802

2803 Syntax
 2804 The syntax of a Packed Attestation statement is defined by the
 2805 following CDDL:
 2806

```

2807 $$attStmtType //= (
2808     fmt: "packed",
2809     attStmt: packedStmtFormat
2810 )
2811
2812 packedStmtFormat = {
2813     alg: COSEAlgorithmIdentifier,
2814     sig: bytes,
2815     x5c: [ attestnCert: bytes, * (caCert: bytes) ]
2816 } //
2817     alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261
2818 for ED512)
2819     sig: bytes,
2820     ecdaaKeyld: bytes
2821 }
  
```

2822

2823 The semantics of the fields are as follows:

2824

2825 alg
 2826 A COSEAlgorithmIdentifier containing the identifier of the
 2827 algorithm used to generate the attestation signature.

2828

2829 sig
 2830 A byte string containing the attestation signature.

2831

2832 x5c
 2833 The elements of this array contain the attestation
 2834 certificate and its certificate chain, each encoded in
 2835 X.509 format. The attestation certificate must be the
 2836 first element in the array.

2837

2838 ecdaaKeyld
 2839 The identifier of the ECDAA-Issuer public key. This is the
 2840 BigNumberToB encoding of the component "c" of the
 2841 ECDAA-Issuer public key as defined section 3.3, step 3.5
 2842 in [FIDOecdaaAlgorithm].
 2843

2844

2845 Signing procedure
 2846 The signing procedure for this attestation statement format is
 2847 similar to the procedure for generating assertion signatures.
 2848

- 2849 1. Let authenticatorData denote the authenticator data for the
 2850 attestation, and let clientDataHash denote the hash of the
 2851 serialized client data.
- 2852 2. If Basic or Privacy CA attestation is in use, the

2797 authenticator produces the sig by concatenating
 2798 authenticatorData and clientDataHash, and signing the result
 2799 using an attestation private key selected through an
 2800 authenticator-specific mechanism. It sets x5c to the
 2801 certificate chain of the attestation public key and alg to the
 2802 algorithm of the attestation private key.
 2803 3. If ECDAAs is in use, the authenticator produces sig by
 2804 concatenating authenticatorData and clientDataHash, and
 2805 signing the result using ECDAAs-Sign (see section 3.5 of
 2806 [FIDOEcdaaAlgorithm]) with a ECDAAs-Issuer public key selected
 2807 through an authenticator-specific mechanism (see
 2808 [FIDOEcdaaAlgorithm]). It sets alg to the algorithm of the
 2809 ECDAAs-Issuer public key and ecdaaKeyId to the identifier of
 2810 the ECDAAs-Issuer public key (see above).
 2811 4. If self attestation is in use, the authenticator produces sig
 2812 by concatenating authenticatorData and clientDataHash, and
 2813 signing the result using the credential private key. It sets
 2814 alg to the algorithm of the credential private key, and omits
 2815 the other fields.

2816 Verification procedure

2817 The verification procedure is as follows:

- 2818 1. Perform CBOR decoding on the given attestation
 2819 statementattStmt structure to obtain the attestation
 2820 certificate array x5c, and the signature value sig. If a
 2821 decoding error occurs, terminate this algorithm and return an
 2822 appropriate error.
- 2823 2. Let authenticatorData denote the authenticator data claimed to
 2824 have been used for the attestation, and let clientDataHash
 2825 denote the hash of the serialized client data.
- 2826 3. If x5c is present, this indicates that the attestation type is
 2827 not ECDAAs. In this case:
 2828 o Verify that sig is a valid signature over the
 2829 concatenation of authenticatorData and clientDataHash
 2830 using the attestation public key in x5c with the
 2831 algorithm specified in alg.
 2832 o Verify that x5c meets the requirements in 8.2.1 Packed
 2833 attestation statement certificate requirements.
 2834 o If x5c contains an extension with OID 1.3.6.1.4.1.45724.1
 2835 1.4 (id-fido-gen-ce-aaguid) verify that the value of this
 2836 extension matches the aaguid in authenticatorData.
 2837 o If successful, return attestation type Basic and trust
 2838 path x5c.
- 2839 4. If ecdaaKeyId is present, then the attestation type is ECDAAs.
 2840 In this case:
 2841 o Verify that sig is a valid signature over the
 2842 concatenation of authenticatorData and clientDataHash
 2843 using ECDAAs-Verify with ECDAAs-Issuer public key
 2844 identified by ecdaaKeyId (see [FIDOEcdaaAlgorithm]).
 2845 o If successful, return attestation type ECDAAs and trust
 2846 path ecdaaKeyId.
- 2847 5. If neither x5c nor ecdaaKeyId is present, self attestation is
 2848 in use.
 2849 o Validate that alg matches the algorithm of the
 2850 credentialPublicKey in authenticatorData.
 2851 o Verify that sig is a valid signature over the
 2852 concatenation of authenticatorData and clientDataHash
 2853 using the credential public key with alg.
 2854 o If successful, return attestation type Self and empty
 2855 trust path.

2856 8.2.1. Packed attestation statement certificate requirements

2857 The attestation certificate MUST have the following fields/extensions:

- 2858 * Version must be set to 3.
- 2859 * Subject field MUST be set to:

2860 Subject-C
 2861 Country where the Authenticator vendor is incorporated

2862

2853 authenticator produces the sig by concatenating
 2854 authenticatorData and clientDataHash, and signing the result
 2855 using an attestation private key selected through an
 2856 authenticator-specific mechanism. It sets x5c to the
 2857 certificate chain of the attestation public key and alg to the
 2858 algorithm of the attestation private key.
 2859 3. If ECDAAs is in use, the authenticator produces sig by
 2860 concatenating authenticatorData and clientDataHash, and
 2861 signing the result using ECDAAs-Sign (see section 3.5 of
 2862 [FIDOEcdaaAlgorithm]) with a ECDAAs-Issuer public key selected
 2863 through an authenticator-specific mechanism (see
 2864 [FIDOEcdaaAlgorithm]). It sets alg to the algorithm of the
 2865 ECDAAs-Issuer public key and ecdaaKeyId to the identifier of
 2866 the ECDAAs-Issuer public key (see above).
 2867 4. If self attestation is in use, the authenticator produces sig
 2868 by concatenating authenticatorData and clientDataHash, and
 2869 signing the result using the credential private key. It sets
 2870 alg to the algorithm of the credential private key, and omits
 2871 the other fields.

2872 Verification procedure

2873 The verification procedure is as follows:

- 2874 1. Perform CBOR decoding on the given attestation
 2875 statementattStmt structure to obtain the attestation
 2876 certificate array x5c, and the signature value sig. If a
 2877 decoding error occurs, terminate this algorithm and return an
 2878 appropriate error.
- 2879 2. Let authenticatorData denote the authenticator data claimed to
 2880 have been used for the attestation, and let clientDataHash
 2881 denote the hash of the serialized client data.
- 2882 3. If x5c is present, this indicates that the attestation type is
 2883 not ECDAAs. In this case:
 2884 o Verify that sig is a valid signature over the
 2885 concatenation of authenticatorData and clientDataHash
 2886 using the attestation public key in x5c with the
 2887 algorithm specified in alg.
 2888 o Verify that x5c meets the requirements in 8.2.1 Packed
 2889 attestation statement certificate requirements.
 2890 o If x5c contains an extension with OID 1.3.6.1.4.1.45724.1
 2891 1.4 (id-fido-gen-ce-aaguid) verify that the value of this
 2892 extension matches the aaguid in authenticatorData.
 2893 o If successful, return attestation type Basic and trust
 2894 path x5c.
- 2895 4. If ecdaaKeyId is present, then the attestation type is ECDAAs.
 2896 In this case:
 2897 o Verify that sig is a valid signature over the
 2898 concatenation of authenticatorData and clientDataHash
 2899 using ECDAAs-Verify with ECDAAs-Issuer public key
 2900 identified by ecdaaKeyId (see [FIDOEcdaaAlgorithm]).
 2901 o If successful, return attestation type ECDAAs and trust
 2902 path ecdaaKeyId.
- 2903 5. If neither x5c nor ecdaaKeyId is present, self attestation is
 2904 in use.
 2905 o Validate that alg matches the algorithm of the
 2906 credentialPublicKey in authenticatorData.
 2907 o Verify that sig is a valid signature over the
 2908 concatenation of authenticatorData and clientDataHash
 2909 using the credential public key with alg.
 2910 o If successful, return attestation type Self and empty
 2911 trust path.

2912 8.2.1. Packed attestation statement certificate requirements

2913 The attestation certificate MUST have the following fields/extensions:

- 2914 * Version must be set to 3.
- 2915 * Subject field MUST be set to:

2916 Subject-C
 2917 Country where the Authenticator vendor is incorporated

2922

2867 Subject-O
 2868 Legal name of the Authenticator vendor
 2869
 2870 Subject-OU
 2871 Authenticator Attestation
 2872
 2873 Subject-CN
 2874 No stipulation.
 2875
 2876
 2877 * If the related attestation root certificate is used for multiple
 2878 authenticator models, the Extension OID 1 3 6 1 4 1 45724 1 1 4
 2879 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as
 2880 value.
 2881 * The Basic Constraints extension MUST have the CA component set to
 2882 false
 2883 * An Authority Information Access (AIA) extension with entry
 2884 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
 2885 both optional as the status of many attestation certificates is
 2886 available through authenticator metadata services. See, for
 2887 example, the FIDO Metadata Service [FIDOMetadataService].
 2888

2889 8.3. TPM Attestation Statement Format

2890 This attestation statement format is generally used by authenticators
 2891 that use a Trusted Platform Module as their cryptographic engine.
 2892
 2893 Attestation statement format identifier
 2894 tpm
 2895
 2896 Attestation types supported
 2897 Privacy CA, ECDA
 2898
 2899 Syntax
 2900 The syntax of a TPM Attestation statement is as follows:
 2901
 2902
 2903 \$\$attStmtType // = (
 2904 fmt: "tpm",
 2905 attStmt: tpmStmtFormat
 2906)
 2907
 2908 tpmStmtFormat = {
 2909 ver: "2.0",
 2910 (
 2911 alg: COSEAlgorithmIdentifier,
 2912 x5c: [aikCert: bytes, * (caCert: bytes)]
 2913) //
 2914 (
 2915 alg: COSEAlgorithmIdentifier, (-260 for ED256 / -26
 2916 1 for ED512)
 2917 ecdaaKeyId: bytes
 2918),
 2919 sig: bytes,
 2920 certInfo: bytes,
 2921 pubArea: bytes
 2922 }
 2923
 2924 The semantics of the above fields are as follows:
 2925
 2926 ver
 2927 The version of the TPM specification to which the
 2928 signature conforms.
 2929
 2930 alg
 2931 A COSEAlgorithmIdentifier containing the identifier of the
 2932 algorithm used to generate the attestation signature.
 2933
 2934 x5c
 2935 The AIK certificate used for the attestation and its
 2936 certificate chain, in X.509 encoding.

2923 Subject-O
 2924 Legal name of the Authenticator vendor
 2925
 2926 Subject-OU
 2927 Authenticator Attestation
 2928
 2929 Subject-CN
 2930 No stipulation.
 2931
 2932
 2933 * If the related attestation root certificate is used for multiple
 2934 authenticator models, the Extension OID 1 3 6 1 4 1 45724 1 1 4
 2935 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as
 2936 value.
 2937 * The Basic Constraints extension MUST have the CA component set to
 2938 false
 2939 * An Authority Information Access (AIA) extension with entry
 2940 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
 2941 both optional as the status of many attestation certificates is
 2942 available through authenticator metadata services. See, for
 2943 example, the FIDO Metadata Service [FIDOMetadataService].
 2944

2945 8.3. TPM Attestation Statement Format

2946 This attestation statement format is generally used by authenticators
 2947 that use a Trusted Platform Module as their cryptographic engine.
 2948
 2949 Attestation statement format identifier
 2950 tpm
 2951
 2952 Attestation types supported
 2953 Privacy CA, ECDA
 2954
 2955 Syntax
 2956 The syntax of a TPM Attestation statement is as follows:
 2957
 2958
 2959 \$\$attStmtType // = (
 2960 fmt: "tpm",
 2961 attStmt: tpmStmtFormat
 2962)
 2963
 2964 tpmStmtFormat = {
 2965 ver: "2.0",
 2966 (
 2967 alg: COSEAlgorithmIdentifier,
 2968 x5c: [aikCert: bytes, * (caCert: bytes)]
 2969) //
 2970 (
 2971 alg: COSEAlgorithmIdentifier, (-260 for ED256 / -26
 2972 1 for ED512)
 2973 ecdaaKeyId: bytes
 2974),
 2975 sig: bytes,
 2976 certInfo: bytes,
 2977 pubArea: bytes
 2978 }
 2979
 2980 The semantics of the above fields are as follows:
 2981
 2982 ver
 2983 The version of the TPM specification to which the
 2984 signature conforms.
 2985
 2986 alg
 2987 A COSEAlgorithmIdentifier containing the identifier of the
 2988 algorithm used to generate the attestation signature.
 2989
 2990 x5c
 2991 The AIK certificate used for the attestation and its
 2992 certificate chain, in X.509 encoding.

2937
 2938
 2939 ecdaaKeyId
 2940 The identifier of the ECDAAs-issuer public key. This is the
 2941 BigNumberToB encoding of the component "c" as defined
 2942 section 3.3, step 3.5 in [FIDOEcdaaAlgorithm].
 2943
 2944 sig
 2945 The attestation signature, in the form of a TPMT_SIGNATURE
 2946 structure as specified in [TPMv2-Part2] section 11.3.4.
 2947
 2948 certInfo
 2949 The TPMS_ATTEST structure over which the above signature
 2950 was computed, as specified in [TPMv2-Part2] section
 2951 10.12.8.
 2952
 2953 pubArea
 2954 The TPMT_PUBLIC structure (see [TPMv2-Part2] section
 2955 12.2.4) used by the TPM to represent the credential public
 2956 key.
 2957
 2958 Signing procedure
 2959 Let authenticatorData denote the authenticator data for the
 2960 attestation, and let clientDataHash denote the hash of the
 2961 serialized client data.
 2962
 2963 Concatenate authenticatorData and clientDataHash to form
 2964 attToBeSigned.
 2965
 2966 Generate a signature using the procedure specified in
 2967 [TPMv2-Part3] Section 18.2, using the attestation private key
 2968 and setting the qualifyingData parameter to attToBeSigned.
 2969
 2970 Set the pubArea field to the public area of the credential
 2971 public key, the certInfo field to the output parameter of the
 2972 same name, and the sig field to the signature obtained from the
 2973 above procedure.
 2974
 2975 Verification procedure
 2976 Verify that the given attestation statement is valid CBOR
 2977 conforming to the syntax defined above.
 2978
 2979 Let authenticatorData denote the authenticator data claimed to
 2980 have been used for the attestation, and let clientDataHash
 2981 denote the hash of the serialized client data.
 2982
 2983 Verify that the public key specified by the parameters and
 2984 unique fields of pubArea is identical to the credentialPublicKey
 2985 in the attestedCredentialData in authenticatorData.
 2986
 2987 Concatenate authenticatorData and clientDataHash to form
 2988 attToBeSigned.
 2989
 2990 Validate that certInfo is valid:
 2991
 2992 + Verify that magic is set to TPM_GENERATED_VALUE.
 2993 + Verify that type is set to TPM_ST_ATTEST_CERTIFY.
 2994 + Verify that extraData is set to attToBeSigned.
 2995 + Verify that attested contains a TPMS_CERTIFY_INFO structure,
 2996 whose name field contains a valid Name for pubArea, as
 2997 computed using the algorithm in the nameAlg field of pubArea
 2998 using the procedure specified in [TPMv2-Part1] section 16.
 2999
 3000 If x5c is present, this indicates that the attestation type is
 3001 not ECDAAs. In this case:
 3002
 3003 + Verify the sig is a valid signature over certInfo using the
 3004 attestation public key in x5c with the algorithm specified in
 3005 alg.
 3006 + Verify that x5c meets the requirements in 8.3.1 TPM
 attestation statement certificate requirements.

2993
 2994
 2995 ecdaaKeyId
 2996 The identifier of the ECDAAs-issuer public key. This is the
 2997 BigNumberToB encoding of the component "c" as defined
 2998 section 3.3, step 3.5 in [FIDOEcdaaAlgorithm].
 2999
 3000 sig
 3001 The attestation signature, in the form of a TPMT_SIGNATURE
 3002 structure as specified in [TPMv2-Part2] section 11.3.4.
 3003
 3004 certInfo
 3005 The TPMS_ATTEST structure over which the above signature
 3006 was computed, as specified in [TPMv2-Part2] section
 3007 10.12.8.
 3008
 3009 pubArea
 3010 The TPMT_PUBLIC structure (see [TPMv2-Part2] section
 3011 12.2.4) used by the TPM to represent the credential public
 3012 key.
 3013
 3014 Signing procedure
 3015 Let authenticatorData denote the authenticator data for the
 3016 attestation, and let clientDataHash denote the hash of the
 3017 serialized client data.
 3018
 3019 Concatenate authenticatorData and clientDataHash to form
 3020 attToBeSigned.
 3021
 3022 Generate a signature using the procedure specified in
 3023 [TPMv2-Part3] Section 18.2, using the attestation private key
 3024 and setting the qualifyingData parameter to attToBeSigned.
 3025
 3026 Set the pubArea field to the public area of the credential
 3027 public key, the certInfo field to the output parameter of the
 3028 same name, and the sig field to the signature obtained from the
 3029 above procedure.
 3030
 3031 Verification procedure
 3032 Verify that the given attestation statement is valid CBOR
 3033 conforming to the syntax defined above.
 3034
 3035 Let authenticatorData denote the authenticator data claimed to
 3036 have been used for the attestation, and let clientDataHash
 3037 denote the hash of the serialized client data.
 3038
 3039 Verify that the public key specified by the parameters and
 3040 unique fields of pubArea is identical to the credentialPublicKey
 3041 in the attestedCredentialData in authenticatorData.
 3042
 3043 Concatenate authenticatorData and clientDataHash to form
 3044 attToBeSigned.
 3045
 3046 Validate that certInfo is valid:
 3047
 3048 + Verify that magic is set to TPM_GENERATED_VALUE.
 3049 + Verify that type is set to TPM_ST_ATTEST_CERTIFY.
 3050 + Verify that extraData is set to attToBeSigned.
 3051 + Verify that attested contains a TPMS_CERTIFY_INFO structure,
 3052 whose name field contains a valid Name for pubArea, as
 3053 computed using the algorithm in the nameAlg field of pubArea
 3054 using the procedure specified in [TPMv2-Part1] section 16.
 3055
 3056 If x5c is present, this indicates that the attestation type is
 3057 not ECDAAs. In this case:
 3058
 3059 + Verify the sig is a valid signature over certInfo using the
 3060 attestation public key in x5c with the algorithm specified in
 3061 alg.
 3062 + Verify that x5c meets the requirements in 8.3.1 TPM
 attestation statement certificate requirements.

3007 + If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4
 3008 (id-fido-gen-ce-aaguid) verify that the value of this
 3009 extension matches the aaguid in authenticatorData.
 3010 + If successful, return attestation type Privacy CA and trust
 3011 path x5c.
 3012
 3013 If ecdaaKeyId is present, then the attestation type is ECDA.
 3014
 3015 + Perform ECDA-Verify on sig to verify that it is a valid
 3016 signature over certInfo (see [FIDOEcdaaAlgorithm]).
 3017 + If successful, return attestation type ECDA and the
 3018 identifier of the ECDA-Issuer public key ecdaaKeyId.
 3019
 3020 **8.3.1. TPM attestation statement certificate requirements**
 3021
 3022 TPM attestation certificate MUST have the following fields/extensions:
 3023 * Version must be set to 3.
 3024 * Subject field MUST be set to empty.
 3025 * The Subject Alternative Name extension must be set as defined in
 3026 [TPMv2-EK-Profile] section 3.2.9.
 3027 * The Extended Key Usage extension MUST contain the
 3028 "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8)
 3029 tcg-kp-AIKCertificate(3)" OID.
 3030 * The Basic Constraints extension MUST have the CA component set to
 3031 false.
 3032 * An Authority Information Access (AIA) extension with entry
 3033 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
 3034 both optional as the status of many attestation certificates is
 3035 available through metadata services. See, for example, the FIDO
 3036 Metadata Service [FIDOMetadataService].
 3037
 3038 **8.4. Android Key Attestation Statement Format**
 3039
 3040 When the authenticator in question is a platform-provided Authenticator
 3041 on the Android "N" or later platform, the attestation statement is
 3042 based on the Android key attestation. In these cases, the attestation
 3043 statement is produced by a component running in a secure operating
 3044 environment, but the authenticator data for the attestation is produced
 3045 outside this environment. The Relying Party is expected to check that
 3046 the authenticator data claimed to have been used for the attestation is
 3047 consistent with the fields of the attestation certificate's extension
 3048 data.
 3049
 3050 Attestation statement format identifier
 3051 android-key
 3052
 3053 Attestation types supported
 3054 Basic
 3055
 3056 Syntax
 3057 An Android key attestation statement consists simply of the
 3058 Android attestation statement, which is a series of DER encoded
 3059 X.509 certificates. See the Android developer documentation. Its
 3060 syntax is defined as follows:
 3061
 3062 \$\$attStmtType ::= (
 3063 fmt: "android-key",
 3064 attStmt: androidStmtFormat
 3065)
 3066
 3067 androidStmtFormat = {
 3068 alg: COSEAlgorithmIdentifier,
 3069 sig: bytes,
 3070 x5c: [credCert: bytes, * (caCert: bytes)]
 3071 }
 3072
 3073
 3074 Signing procedure
 3075 Let authenticatorData denote the authenticator data for the
 3076 attestation, and let clientDataHash denote the hash of the

3063 + If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4
 3064 (id-fido-gen-ce-aaguid) verify that the value of this
 3065 extension matches the aaguid in authenticatorData.
 3066 + If successful, return attestation type Privacy CA and trust
 3067 path x5c.
 3068
 3069 If ecdaaKeyId is present, then the attestation type is ECDA.
 3070
 3071 + Perform ECDA-Verify on sig to verify that it is a valid
 3072 signature over certInfo (see [FIDOEcdaaAlgorithm]).
 3073 + If successful, return attestation type ECDA and the
 3074 identifier of the ECDA-Issuer public key ecdaaKeyId.
 3075
 3076 **8.3.1. TPM attestation statement certificate requirements**
 3077
 3078 TPM attestation certificate MUST have the following fields/extensions:
 3079 * Version must be set to 3.
 3080 * Subject field MUST be set to empty.
 3081 * The Subject Alternative Name extension must be set as defined in
 3082 [TPMv2-EK-Profile] section 3.2.9.
 3083 * The Extended Key Usage extension MUST contain the
 3084 "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8)
 3085 tcg-kp-AIKCertificate(3)" OID.
 3086 * The Basic Constraints extension MUST have the CA component set to
 3087 false.
 3088 * An Authority Information Access (AIA) extension with entry
 3089 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
 3090 both optional as the status of many attestation certificates is
 3091 available through metadata services. See, for example, the FIDO
 3092 Metadata Service [FIDOMetadataService].
 3093
 3094 **8.4. Android Key Attestation Statement Format**
 3095
 3096 When the authenticator in question is a platform-provided Authenticator
 3097 on the Android "N" or later platform, the attestation statement is
 3098 based on the Android key attestation. In these cases, the attestation
 3099 statement is produced by a component running in a secure operating
 3100 environment, but the authenticator data for the attestation is produced
 3101 outside this environment. The Relying Party is expected to check that
 3102 the authenticator data claimed to have been used for the attestation is
 3103 consistent with the fields of the attestation certificate's extension
 3104 data.
 3105
 3106 Attestation statement format identifier
 3107 android-key
 3108
 3109 Attestation types supported
 3110 Basic
 3111
 3112 Syntax
 3113 An Android key attestation statement consists simply of the
 3114 Android attestation statement, which is a series of DER encoded
 3115 X.509 certificates. See the Android developer documentation. Its
 3116 syntax is defined as follows:
 3117
 3118 \$\$attStmtType ::= (
 3119 fmt: "android-key",
 3120 attStmt: androidStmtFormat
 3121)
 3122
 3123 androidStmtFormat = {
 3124 alg: COSEAlgorithmIdentifier,
 3125 sig: bytes,
 3126 x5c: [credCert: bytes, * (caCert: bytes)]
 3127 }
 3128
 3129
 3130 Signing procedure
 3131 Let authenticatorData denote the authenticator data for the
 3132 attestation, and let clientDataHash denote the hash of the

3077 serialized client data.
 3078
 3079 Request an Android Key Attestation by calling
 3080 "keyStore.getCertificateChain(myKeyUUID)") providing
 3081 clientDataHash as the challenge value (e.g., by using
 3082 setAttestationChallenge). Set x5c to the returned value.
 3083
 3084 The authenticator produces sig by concatenating
 3085 authenticatorData and clientDataHash, and signing the result
 3086 using the credential private key. It sets alg to the algorithm
 3087 of the signature format.
 3088
 3089 Verification procedure
 3090 Verification is performed as follows:
 3091
 3092 + Let authenticatorData denote the authenticator data claimed to
 3093 have been used for the attestation, and let clientDataHash
 3094 denote the hash of the serialized client data.
 3095 + Verify that the public key in the first certificate in the
 3096 series of certificates represented by the signature matches
 3097 the credentialPublicKey in the attestedCredentialData in
 3098 authenticatorData.
 3099 + Verify that in the attestation certificate extension data:
 3100 o The value of the attestationChallenge field is identical
 3101 to the concatenation of authenticatorData and
 3102 clientDataHash.
 3103 o The AuthorizationList.allApplications field is not
 3104 present, since PublicKeyCredentials must be bound to the
 3105 RP ID.
 3106 o The value in the AuthorizationList.origin field is equal
 3107 to KM_TAG_GENERATED.
 3108 o The value in the AuthorizationList.purpose field is equal
 3109 to KM_PURPOSE_SIGN.
 3110 + If successful, return attestation type Basic with the trust
 3111 path set to the entire attestation statement.
 3112
 3113 8.5. Android SafetyNet Attestation Statement Format
 3114
 3115 When the authenticator in question is a platform-provided Authenticator
 3116 on certain Android platforms, the attestation statement is based on the
 3117 SafetyNet API. In this case the authenticator data is completely
 3118 controlled by the caller of the SafetyNet API (typically an application
 3119 running on the Android platform) and the attestation statement only
 3120 provides some statements about the health of the platform and the
 3121 identity of the calling application.
 3122
 3123 Attestation statement format identifier
 3124 android-safetynet
 3125
 3126 Attestation types supported
 3127 Basic
 3128
 3129 Syntax
 3130 The syntax of an Android Attestation statement is defined as
 3131 follows:
 3132
 3133 \$\$attStmtType ::= (
 3134 fmt: "android-safetynet",
 3135 attStmt: safetynetStmtFormat
 3136)
 3137
 3138 safetynetStmtFormat = {
 3139 ver: text,
 3140 response: bytes
 3141 }
 3142
 3143 The semantics of the above fields are as follows:
 3144
 3145 ver
 3146 The version number of Google Play Services responsible for

3133 serialized client data.
 3134
 3135 Request an Android Key Attestation by calling
 3136 "keyStore.getCertificateChain(myKeyUUID)") providing
 3137 clientDataHash as the challenge value (e.g., by using
 3138 setAttestationChallenge). Set x5c to the returned value.
 3139
 3140 The authenticator produces sig by concatenating
 3141 authenticatorData and clientDataHash, and signing the result
 3142 using the credential private key. It sets alg to the algorithm
 3143 of the signature format.
 3144
 3145 Verification procedure
 3146 Verification is performed as follows:
 3147
 3148 + Let authenticatorData denote the authenticator data claimed to
 3149 have been used for the attestation, and let clientDataHash
 3150 denote the hash of the serialized client data.
 3151 + Verify that the public key in the first certificate in the
 3152 series of certificates represented by the signature matches
 3153 the credentialPublicKey in the attestedCredentialData in
 3154 authenticatorData.
 3155 + Verify that in the attestation certificate extension data:
 3156 o The value of the attestationChallenge field is identical
 3157 to the concatenation of authenticatorData and
 3158 clientDataHash.
 3159 o The AuthorizationList.allApplications field is not
 3160 present, since PublicKeyCredentials must be bound to the
 3161 RP ID.
 3162 o The value in the AuthorizationList.origin field is equal
 3163 to KM_TAG_GENERATED.
 3164 o The value in the AuthorizationList.purpose field is equal
 3165 to KM_PURPOSE_SIGN.
 3166 + If successful, return attestation type Basic with the trust
 3167 path set to the entire attestation statement.
 3168
 3169 8.5. Android SafetyNet Attestation Statement Format
 3170
 3171 When the authenticator in question is a platform-provided Authenticator
 3172 on certain Android platforms, the attestation statement is based on the
 3173 SafetyNet API. In this case the authenticator data is completely
 3174 controlled by the caller of the SafetyNet API (typically an application
 3175 running on the Android platform) and the attestation statement only
 3176 provides some statements about the health of the platform and the
 3177 identity of the calling application.
 3178
 3179 Attestation statement format identifier
 3180 android-safetynet
 3181
 3182 Attestation types supported
 3183 Basic
 3184
 3185 Syntax
 3186 The syntax of an Android Attestation statement is defined as
 3187 follows:
 3188
 3189 \$\$attStmtType ::= (
 3190 fmt: "android-safetynet",
 3191 attStmt: safetynetStmtFormat
 3192)
 3193
 3194 safetynetStmtFormat = {
 3195 ver: text,
 3196 response: bytes
 3197 }
 3198
 3199 The semantics of the above fields are as follows:
 3200
 3201 ver
 3202 The version number of Google Play Services responsible for

3147 providing the SafetyNet API.
3148
3149 response
3150 The UTF-8 encoded result of the getJwsResult() call of the
3151 SafetyNet API. This value is a JWS [RFC7515] object (see
3152 SafetyNet online documentation) in Compact Serialization.
3153
3154 Signing procedure
3155 Let authenticatorData denote the authenticator data for the
3156 attestation, and let clientDataHash denote the hash of the
3157 serialized client data.
3158
3159 Concatenate authenticatorData and clientDataHash to form
3160 attToBeSigned.
3161
3162 Request a SafetyNet attestation, providing attToBeSigned as the
3163 nonce value. Set response to the result, and ver to the version
3164 of Google Play Services running in the authenticator.
3165
3166 Verification procedure
3167 Verification is performed as follows:
3168
3169 + Verify that the given attestation statement is valid CBOR
3170 conforming to the syntax defined above.
3171 + Verify that response is a valid SafetyNet response of version
3172 ver.
3173 + Verify that the nonce in the response is identical to the
3174 concatenation of the authenticatorData and clientDataHash.
3175 + Verify that the attestation certificate is issued to the
3176 hostname "attest.android.com" (see SafetyNet online
3177 documentation).
3178 + Verify that the ctsProfileMatch attribute in the payload of
3179 response is true.
3180 + If successful, return attestation type Basic with the trust
3181 path set to the above attestation certificate.
3182
3183 8.6. FIDO U2F Attestation Statement Format
3184
3185 This attestation statement format is used with FIDO U2F authenticators
3186 using the formats defined in [FIDO-U2F-Message-Formats].
3187
3188 Attestation statement format identifier
3189 fido-u2f
3190
3191 Attestation types supported
3192 Basic, self attestation
3193
3194 Syntax
3195 The syntax of a FIDO U2F attestation statement is defined as
3196 follows:
3197
3198 \$\$attStmtType ::= (
3199 fmt: "fido-u2f",
3200 attStmt: u2fStmtFormat
3201)
3202
3203 u2fStmtFormat = {
3204 x5c: [attestnCert: bytes, * (caCert: bytes)],
3205 sig: bytes
3206 }
3207
3208 The semantics of the above fields are as follows:
3209
3210 x5c
3211 The elements of this array contain the attestation
3212 certificate and its certificate chain, each encoded in
3213 X.509 format. The attestation certificate must be the
3214 first element in the array.
3215
3216 sig

3203 providing the SafetyNet API.
3204
3205 response
3206 The UTF-8 encoded result of the getJwsResult() call of the
3207 SafetyNet API. This value is a JWS [RFC7515] object (see
3208 SafetyNet online documentation) in Compact Serialization.
3209
3210 Signing procedure
3211 Let authenticatorData denote the authenticator data for the
3212 attestation, and let clientDataHash denote the hash of the
3213 serialized client data.
3214
3215 Concatenate authenticatorData and clientDataHash to form
3216 attToBeSigned.
3217
3218 Request a SafetyNet attestation, providing attToBeSigned as the
3219 nonce value. Set response to the result, and ver to the version
3220 of Google Play Services running in the authenticator.
3221
3222 Verification procedure
3223 Verification is performed as follows:
3224
3225 + Verify that the given attestation statement is valid CBOR
3226 conforming to the syntax defined above.
3227 + Verify that response is a valid SafetyNet response of version
3228 ver.
3229 + Verify that the nonce in the response is identical to the
3230 concatenation of the authenticatorData and clientDataHash.
3231 + Verify that the attestation certificate is issued to the
3232 hostname "attest.android.com" (see SafetyNet online
3233 documentation).
3234 + Verify that the ctsProfileMatch attribute in the payload of
3235 response is true.
3236 + If successful, return attestation type Basic with the trust
3237 path set to the above attestation certificate.
3238
3239 8.6. FIDO U2F Attestation Statement Format
3240
3241 This attestation statement format is used with FIDO U2F authenticators
3242 using the formats defined in [FIDO-U2F-Message-Formats].
3243
3244 Attestation statement format identifier
3245 fido-u2f
3246
3247 Attestation types supported
3248 Basic, self attestation
3249
3250 Syntax
3251 The syntax of a FIDO U2F attestation statement is defined as
3252 follows:
3253
3254 \$\$attStmtType ::= (
3255 fmt: "fido-u2f",
3256 attStmt: u2fStmtFormat
3257)
3258
3259 u2fStmtFormat = {
3260 x5c: [attestnCert: bytes, * (caCert: bytes)],
3261 sig: bytes
3262 }
3263
3264 The semantics of the above fields are as follows:
3265
3266 x5c
3267 The elements of this array contain the attestation
3268 certificate and its certificate chain, each encoded in
3269 X.509 format. The attestation certificate must be the
3270 first element in the array.
3271
3272 sig

3217 The attestation signature. The signature was calculated
3218 over the (raw) U2F registration response message
3219 [FIDO-U2F-Message-Formats] received by the platform from
3220 the authenticator.
3221

3222 **Signing procedure**

3223 If the credential public key of the given credential is not of
3224 algorithm -7 ("ES256"), stop and return an error. Otherwise, let
3225 authenticatorData denote the authenticator data for the
3226 attestation, and let clientDataHash denote the hash of the
3227 serialized client data.
3228

3229 If clientDataHash is 256 bits long, set tbsHash to this value.
3230 Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
3231

3232 Generate a Registration Response Message as specified in
3233 [FIDO-U2F-Message-Formats] section 4.3, with the application
3234 parameter set to the SHA-256 hash of the RP ID associated with
3235 the given credential, the challenge parameter set to tbsHash,
3236 and the key handle parameter set to the credential ID of the
3237 given credential. Set the raw signature part of this
3238 Registration Response Message (i.e., without the user public
3239 key, key handle, and attestation certificates) as sig and set
3240 the attestation certificates of the attestation public key as
3241 x5c.
3242

3243 **Verification procedure**

3244 Verification is performed as follows:

- 3245 1. Verify that the given attestation statement is valid CBOR
3246 conforming to the syntax defined above.
- 3247 2. Perform CBOR decoding on the given attestation
3248 statementattStmt structure to obtain the attestation
3249 certificate array x5c, and the signature value sig. If a
3250 decoding error occurs, terminate this algorithm and return an
3251 appropriate error.
- 3252 3. Let attCert be value of the first element of x5c. Let
3253 certificate public key be the public key conveyed by attCert.
3254 If certificate public key is not an Elliptic Curve (EC) public
3255 key over the P-256 curve, terminate this algorithm and return
3256 an appropriate error.
- 3257 4. Let authenticatorData denote the given authenticator data
3258 claimed to have been used for the attestation, and let
3259 clientDataHash denote the given hash of the serialized client
3260 data.
- 3261 5. Extract the claimed rpIdHash from authenticatorData, and the
3262 claimed credentialId and credentialPublicKey from
3263 authenticatorData.attestedCredentialData.
- 3264 6. If clientDataHash is 256 bits long, set tbsHash to this value.
3265 Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
- 3266 7. Convert the COSE_KEY formatted credentialPublicKey (see
3267 Section 7 of [RFC8152]) to CTAP1/U2F public Key format
3268 [FIDO-CTAP].
 - 3269 o Let publicKeyU2F represent the result of the conversion
3270 operation and set its first byte to 0x04. Note: This
3271 signifies uncompressed ECC key format.
 - 3272 o Extract the value corresponding to the "-2" key
3273 (representing x coordinate) from credentialPublicKey,
3274 confirm its size to be of 32 bytes and concatenate it
3275 with publicKeyU2F. If size differs or "-2" key is not
3276 found, terminate this algorithm and return an appropriate
3277 error.
 - 3278 o Extract the value corresponding to the "-3" key
3279 (representing y coordinate) from credentialPublicKey,
3280 confirm its size to be of 32 bytes and concatenate it
3281 with publicKeyU2F. If size differs or "-3" key is not
3282 found, terminate this algorithm and return an appropriate
3283 error.
- 3284 8. Let verificationData be the concatenation of (0x00 || rpIdHash
3285 || tbsHash || credentialId || publicKeyU2F) (see Section 4.3
3286

3273 The attestation signature. The signature was calculated
3274 over the (raw) U2F registration response message
3275 [FIDO-U2F-Message-Formats] received by the platform from
3276 the authenticator.
3277

3278 **Signing procedure**

3279 If the credential public key of the given credential is not of
3280 algorithm -7 ("ES256"), stop and return an error. Otherwise, let
3281 authenticatorData denote the authenticator data for the
3282 attestation, and let clientDataHash denote the hash of the
3283 serialized client data.
3284

3285 If clientDataHash is 256 bits long, set tbsHash to this value.
3286 Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
3287

3288 Generate a Registration Response Message as specified in
3289 [FIDO-U2F-Message-Formats] section 4.3, with the application
3290 parameter set to the SHA-256 hash of the RP ID associated with
3291 the given credential, the challenge parameter set to tbsHash,
3292 and the key handle parameter set to the credential ID of the
3293 given credential. Set the raw signature part of this
3294 Registration Response Message (i.e., without the user public
3295 key, key handle, and attestation certificates) as sig and set
3296 the attestation certificates of the attestation public key as
3297 x5c.
3298

3299 **Verification procedure**

3300 Verification is performed as follows:

- 3301 1. Verify that the given attestation statement is valid CBOR
3302 conforming to the syntax defined above.
- 3303 2. Perform CBOR decoding on the given attestation
3304 statementattStmt structure to obtain the attestation
3305 certificate array x5c, and the signature value sig. If a
3306 decoding error occurs, terminate this algorithm and return an
3307 appropriate error.
- 3308 3. Let attCert be value of the first element of x5c. Let
3309 certificate public key be the public key conveyed by attCert.
3310 If certificate public key is not an Elliptic Curve (EC) public
3311 key over the P-256 curve, terminate this algorithm and return
3312 an appropriate error.
- 3313 4. Let authenticatorData denote the given authenticator data
3314 claimed to have been used for the attestation, and let
3315 clientDataHash denote the given hash of the serialized client
3316 data.
- 3317 5. Extract the claimed rpIdHash from authenticatorData, and the
3318 claimed credentialId and credentialPublicKey from
3319 authenticatorData.attestedCredentialData.
- 3320 6. If clientDataHash is 256 bits long, set tbsHash to this value.
3321 Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
- 3322 7. Convert the COSE_KEY formatted credentialPublicKey (see
3323 Section 7 of [RFC8152]) to CTAP1/U2F public Key format
3324 [FIDO-CTAP].
 - 3325 o Let publicKeyU2F represent the result of the conversion
3326 operation and set its first byte to 0x04. Note: This
3327 signifies uncompressed ECC key format.
 - 3328 o Extract the value corresponding to the "-2" key
3329 (representing x coordinate) from credentialPublicKey,
3330 confirm its size to be of 32 bytes and concatenate it
3331 with publicKeyU2F. If size differs or "-2" key is not
3332 found, terminate this algorithm and return an appropriate
3333 error.
 - 3334 o Extract the value corresponding to the "-3" key
3335 (representing y coordinate) from credentialPublicKey,
3336 confirm its size to be of 32 bytes and concatenate it
3337 with publicKeyU2F. If size differs or "-3" key is not
3338 found, terminate this algorithm and return an appropriate
3339 error.
- 3340 8. Let verificationData be the concatenation of (0x00 || rpIdHash
3341 || tbsHash || credentialId || publicKeyU2F) (see Section 4.3
3342

of [FIDO-U2F-Message-Formats]).

9. Verify the sig using verificationData and certificate public key per [SEC1].

10. If successful, return attestation type Basic with the trust path set to x5c.

9. WebAuthn Extensions

The mechanism for generating public key credentials, as well as requesting and generating Authentication assertions, as defined in 5 Web Authentication API, can be extended to suit particular use cases. Each case is addressed by defining a registration extension and/or an authentication extension.

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

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

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

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

- * authenticatorMakeCredential extension request parameters and response values for registration extensions.
- * authenticatorGetAssertion extension request parameters and response values for authentication extensions.
- * Authenticator extension processing for registration extensions and authentication extensions.

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

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

of [FIDO-U2F-Message-Formats]).

9. Verify the sig using verificationData and certificate public key per [SEC1].

10. If successful, return attestation type Basic with the trust path set to x5c.

9. WebAuthn Extensions

The mechanism for generating public key credentials, as well as requesting and generating Authentication assertions, as defined in 5 Web Authentication API, can be extended to suit particular use cases. Each case is addressed by defining a registration extension and/or an authentication extension.

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

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

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

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

- * authenticatorMakeCredential extension request parameters and response values for registration extensions.
- * authenticatorGetAssertion extension request parameters and response values for authentication extensions.
- * Authenticator extension processing for registration extensions and authentication extensions.

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

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

3357 Clients wishing to support the widest possible range of extensions may
3358 choose to pass through any extensions that they do not recognize to
3359 authenticators, generating the authenticator extension input by simply
3360 encoding the client extension input in CBOR. All WebAuthn extensions
3361 MUST be defined in such a way that this implementation choice does not
3362 endanger the user's security or privacy. For instance, if an extension
3363 requires client processing, it could be defined in a manner that
3364 ensures such a naive pass-through will produce a semantically invalid
3365 authenticator extension input value, resulting in the extension being
3366 ignored by the authenticator. Since all extensions are optional, this
3367 will not cause a functional failure in the API operation. Likewise,
3368 clients can choose to produce a client extension output value for an
3369 extension that it does not understand by encoding the authenticator
3370 extension output value into JSON, provided that the CBOR output uses
3371 only types present in JSON.

3372 The IANA "WebAuthn Extension Identifier" registry established by
3373 [WebAuthn-Registries] should be consulted for an up-to-date list of
3374 registered WebAuthn Extensions.
3375

3376 9.1. Extension Identifiers

3377 Extensions are identified by a string, called an extension identifier,
3378 chosen by the extension author.

3379 Extension identifiers SHOULD be registered per [WebAuthn-Registries]
3380 "Registries for Web Authentication (WebAuthn)". All registered
3381 extension identifiers are unique amongst themselves as a matter of
3382 course.

3383 Unregistered extension identifiers should aim to be globally unique,
3384 e.g., by including the defining entity such as myCompany_extension.
3385

3386 All extension identifiers MUST be a maximum of 32 octets in length and
3387 MUST consist only of printable USASCII characters, excluding backslash
3388 and doublequote, i.e., VCHAR as defined in [RFC5234] but without %x22
3389 and %x5c. Implementations MUST match WebAuthn extension identifiers in
3390 a case-sensitive fashion.

3391 Extensions that may exist in multiple versions should take care to
3392 include a version in their identifier. In effect, different versions
3393 are thus treated as different extensions, e.g., myCompany_extension_01
3394

3395 10 Defined Extensions defines an initial set of extensions and their
3396 identifiers. See the IANA "WebAuthn Extension Identifier" registry
3397 established by [WebAuthn-Registries] for an up-to-date list of
3398 registered WebAuthn Extension Identifiers.
3399

3400 9.2. Defining extensions

3401 A definition of an extension must specify an extension identifier, a
3402 client extension input argument to be sent via the get() or create()
3403 call, the client extension processing rules, and a client extension
3404 output value. If the extension communicates with the authenticator
3405 (meaning it is an authenticator extension), it must also specify the
3406 CBOR authenticator extension input argument sent via the
3407 authenticatorGetAssertion or authenticatorMakeCredential call, the
3408 authenticator extension processing rules, and the CBOR authenticator
3409 extension output value.

3410 Any client extension that is processed by the client MUST return a
3411 client extension output value so that the Relying Party knows that the
3412 extension was honored by the client. Similarly, any extension that
3413 requires authenticator processing MUST return an authenticator
3414 extension output to let the Relying Party know that the extension was
3415 honored by the authenticator. If an extension does not otherwise
3416 require any result values, it SHOULD be defined as returning a JSON
3417 Boolean client extension output result, set to true to signify that the
3418 extension was understood and processed. Likewise, any authenticator
3419 extension that does not otherwise require any result values MUST return

3420

3413 Clients wishing to support the widest possible range of extensions may
3414 choose to pass through any extensions that they do not recognize to
3415 authenticators, generating the authenticator extension input by simply
3416 encoding the client extension input in CBOR. All WebAuthn extensions
3417 MUST be defined in such a way that this implementation choice does not
3418 endanger the user's security or privacy. For instance, if an extension
3419 requires client processing, it could be defined in a manner that
3420 ensures such a naive pass-through will produce a semantically invalid
3421 authenticator extension input value, resulting in the extension being
3422 ignored by the authenticator. Since all extensions are optional, this
3423 will not cause a functional failure in the API operation. Likewise,
3424 clients can choose to produce a client extension output value for an
3425 extension that it does not understand by encoding the authenticator
3426 extension output value into JSON, provided that the CBOR output uses
3427 only types present in JSON.

3428 The IANA "WebAuthn Extension Identifier" registry established by
3429 [WebAuthn-Registries] should be consulted for an up-to-date list of
3430 registered WebAuthn Extensions.
3431

3432 9.1. Extension Identifiers

3433 Extensions are identified by a string, called an extension identifier,
3434 chosen by the extension author.

3435 Extension identifiers SHOULD be registered per [WebAuthn-Registries]
3436 "Registries for Web Authentication (WebAuthn)". All registered
3437 extension identifiers are unique amongst themselves as a matter of
3438 course.

3439 Unregistered extension identifiers should aim to be globally unique,
3440 e.g., by including the defining entity such as myCompany_extension.
3441

3442 All extension identifiers MUST be a maximum of 32 octets in length and
3443 MUST consist only of printable USASCII characters, excluding backslash
3444 and doublequote, i.e., VCHAR as defined in [RFC5234] but without %x22
3445 and %x5c. Implementations MUST match WebAuthn extension identifiers in
3446 a case-sensitive fashion.

3447 Extensions that may exist in multiple versions should take care to
3448 include a version in their identifier. In effect, different versions
3449 are thus treated as different extensions, e.g., myCompany_extension_01
3450

3451 10 Defined Extensions defines an initial set of extensions and their
3452 identifiers. See the IANA "WebAuthn Extension Identifier" registry
3453 established by [WebAuthn-Registries] for an up-to-date list of
3454 registered WebAuthn Extension Identifiers.
3455

3456 9.2. Defining extensions

3457 A definition of an extension must specify an extension identifier, a
3458 client extension input argument to be sent via the get() or create()
3459 call, the client extension processing rules, and a client extension
3460 output value. If the extension communicates with the authenticator
3461 (meaning it is an authenticator extension), it must also specify the
3462 CBOR authenticator extension input argument sent via the
3463 authenticatorGetAssertion or authenticatorMakeCredential call, the
3464 authenticator extension processing rules, and the CBOR authenticator
3465 extension output value.

3466 Any client extension that is processed by the client MUST return a
3467 client extension output value so that the Relying Party knows that the
3468 extension was honored by the client. Similarly, any extension that
3469 requires authenticator processing MUST return an authenticator
3470 extension output to let the Relying Party know that the extension was
3471 honored by the authenticator. If an extension does not otherwise
3472 require any result values, it SHOULD be defined as returning a JSON
3473 Boolean client extension output result, set to true to signify that the
3474 extension was understood and processed. Likewise, any authenticator
3475 extension that does not otherwise require any result values MUST return

3476

3427 a value and SHOULD return a CBOR Boolean authenticator extension output
3428 result, set to true to signify that the extension was understood and
3429 processed.
3430

9.3. Extending request parameters

3431 An extension defines one or two request arguments. The client extension
3432 input, which is a value that can be encoded in JSON, is passed from the
3433 Relying Party to the client in the get() or create() call, while the
3434 CBOR authenticator extension input is passed from the client to the
3435 authenticator for authenticator extensions during the processing of
3436 these calls.
3437

3438 A Relying Party simultaneously requests the use of an extension and
3439 sets its client extension input by including an entry in the extensions
3440 option to the create() or get() call. The entry key is the extension
3441 identifier and the value is the client extension input.

```
3442 var assertionPromise = navigator.credentials.get({  
3443   publicKey: {  
3444     challenge: "...",  
3445     extensions: {  
3446       "webauthnExample_foobar": 42  
3447     }  
3448   }  
3449 });
```

3450 Extension definitions MUST specify the valid values for their client
3451 extension input. Clients SHOULD ignore extensions with an invalid
3452 client extension input. If an extension does not require any parameters
3453 from the Relying Party, it SHOULD be defined as taking a Boolean client
3454 argument, set to true to signify that the extension is requested by the
3455 Relying Party.

3456 Extensions that only affect client processing need not specify
3457 authenticator extension input. Extensions that have authenticator
3458 processing MUST specify the method of computing the authenticator
3459 extension input from the client extension input. For extensions that do
3460 not require input parameters and are defined as taking a Boolean client
3461 extension input value set to true, this method SHOULD consist of
3462 passing an authenticator extension input value of true (CBOR major type
3463 7, value 21).

3464 Note: Extensions should aim to define authenticator arguments that are
3465 as small as possible. Some authenticators communicate over
3466 low-bandwidth links such as Bluetooth Low-Energy or NFC.

9.4. Client extension processing

3467 Extensions may define additional processing requirements on the client
3468 platform during the creation of credentials or the generation of an
3469 assertion. The client extension input for the extension is used an
3470 input to this client processing. Supported client extensions are
3471 recorded as a dictionary in the client data with the key
3472 clientExtensions. For each such extension, the client adds an entry to
3473 this dictionary with the extension identifier as the key, and the
3474 extension's client extension input as the value.

3475 Likewise, the client extension outputs are represented as a dictionary
3476 in the clientExtensionResults with extension identifiers as keys, and
3477 the client extension output value of each extension as the value. Like
3478 the client extension input, the client extension output is a value that
3479 can be encoded in JSON.

3480 Extensions that require authenticator processing MUST define the
3481 process by which the client extension input can be used to determine
3482 the CBOR authenticator extension input and the process by which the
3483 CBOR authenticator extension output can be used to determine the client
3484 extension output.

9.5. Authenticator extension processing

3496

3483 a value and SHOULD return a CBOR Boolean authenticator extension output
3484 result, set to true to signify that the extension was understood and
3485 processed.
3486

9.3. Extending request parameters

3487 An extension defines one or two request arguments. The client extension
3488 input, which is a value that can be encoded in JSON, is passed from the
3489 Relying Party to the client in the get() or create() call, while the
3490 CBOR authenticator extension input is passed from the client to the
3491 authenticator for authenticator extensions during the processing of
3492 these calls.
3493

3494 A Relying Party simultaneously requests the use of an extension and
3495 sets its client extension input by including an entry in the extensions
3496 option to the create() or get() call. The entry key is the extension
3497 identifier and the value is the client extension input.

```
3498 var assertionPromise = navigator.credentials.get({  
3499   publicKey: {  
3500     challenge: "...",  
3501     extensions: {  
3502       "webauthnExample_foobar": 42  
3503     }  
3504   }  
3505 });
```

3506 Extension definitions MUST specify the valid values for their client
3507 extension input. Clients SHOULD ignore extensions with an invalid
3508 client extension input. If an extension does not require any parameters
3509 from the Relying Party, it SHOULD be defined as taking a Boolean client
3510 argument, set to true to signify that the extension is requested by the
3511 Relying Party.

3512 Extensions that only affect client processing need not specify
3513 authenticator extension input. Extensions that have authenticator
3514 processing MUST specify the method of computing the authenticator
3515 extension input from the client extension input. For extensions that do
3516 not require input parameters and are defined as taking a Boolean client
3517 extension input value set to true, this method SHOULD consist of
3518 passing an authenticator extension input value of true (CBOR major type
3519 7, value 21).

3520 Note: Extensions should aim to define authenticator arguments that are
3521 as small as possible. Some authenticators communicate over
3522 low-bandwidth links such as Bluetooth Low-Energy or NFC.

9.4. Client extension processing

3523 Extensions may define additional processing requirements on the client
3524 platform during the creation of credentials or the generation of an
3525 assertion. The client extension input for the extension is used an
3526 input to this client processing. Supported client extensions are
3527 recorded as a dictionary in the client data with the key
3528 clientExtensions. For each such extension, the client adds an entry to
3529 this dictionary with the extension identifier as the key, and the
3530 extension's client extension input as the value.

3531 Likewise, the client extension outputs are represented as a dictionary
3532 in the clientExtensionResults with extension identifiers as keys, and
3533 the client extension output value of each extension as the value. Like
3534 the client extension input, the client extension output is a value that
3535 can be encoded in JSON.

3536 Extensions that require authenticator processing MUST define the
3537 process by which the client extension input can be used to determine
3538 the CBOR authenticator extension input and the process by which the
3539 CBOR authenticator extension output can be used to determine the client
3540 extension output.

9.5. Authenticator extension processing

3552

3497 The CBOR authenticator extension input value of each processed
3498 authenticator extension is included in the extensions data part of the
3499 authenticator request. This part is a CBOR map, with CBOR extension
3500 identifier values as keys, and the CBOR authenticator extension input
3501 value of each extension as the value.
3502
3503

3504 Likewise, the extension output is represented in the authenticator data
3505 as a CBOR map with CBOR extension identifiers as keys, and the CBOR
3506 authenticator extension output value of each extension as the value.
3507

3508 The authenticator extension processing rules are used create the
3509 authenticator extension output from the authenticator extension input,
3510 and possibly also other inputs, for each extension.
3511

3512 9.6. Example Extension

3513 This section is not normative.
3514

3515 To illustrate the requirements above, consider a hypothetical
3516 registration extension and authentication extension "Geo". This
3517 extension, if supported, enables a geolocation location to be returned
3518 from the authenticator or client to the Relying Party.
3519

3520 The extension identifier is chosen as webauthnExample_geo. The client
3521 extension input is the constant value true, since the extension does
3522 not require the Relying Party to pass any particular information to the
3523 client, other than that it requests the use of the extension. The
3524 Relying Party sets this value in its request for an assertion:
3525

```
3526 var assertionPromise =  
3527   navigator.credentials.get({  
3528     publicKey: {  
3529       challenge: "SGFuIFNvbG8gc2hvdCBmaXJzdC4",  
3530       allowCredentials: [], /* Empty filter */  
3531       extensions: { 'webauthnExample_geo': true }  
3532     }  
3533   });  
3534
```

3535 The extension also requires the client to set the authenticator
3536 parameter to the fixed value true.
3537

3538 The extension requires the authenticator to specify its geolocation in
3539 the authenticator extension output, if known. The extension e.g.
3540 specifies that the location shall be encoded as a two-element array of
3541 floating point numbers, encoded with CBOR. An authenticator does this
3542 by including it in the authenticator data. As an example, authenticator
3543 data may be as follows (notation taken from [RFC7049]):
3544

```
3544 81 (hex)           -- Flags, ED and UP both set.  
3545 20 05 58 1F       -- Signature counter  
3546 A1                -- CBOR map of one element  
3547   73              -- Key 1: CBOR text string of 19 byt  
3548 es  
3549   77 65 62 61 75 74 68 6E 45 78 61  
3550   6D 70 6C 65 5F 67 65 6F       -- "webauthnExample_geo" [=UTF-8 enc  
3551 oded=] string  
3552   82                -- Value 1: CBOR array of two elemen  
3553 ts  
3554   FA 42 82 1E B3       -- Element 1: Latitude as CBOR encod  
3555 ed float  
3556   FA C1 5F E3 7F       -- Element 2: Longitude as CBOR enco  
3557 ded float  
3558
```

3559 The extension defines the client extension output to be the geolocation
3560 information, if known, as a GeoJSON [GeoJSON] point. The client
3561 constructs the following client data:
3562

```
3563 {  
3564   'extensions': {  
3565     'webauthnExample_geo': {  
3566       'type': 'Point',
```

3553 The CBOR authenticator extension input value of each processed
3554 authenticator extension is included in the extensions data part of the
3555 authenticator request. This part is a CBOR map, with CBOR extension
3556 identifier values as keys, and the CBOR authenticator extension input
3557 value of each extension as the value.
3558
3559

3560 Likewise, the extension output is represented in the authenticator data
3561 as a CBOR map with CBOR extension identifiers as keys, and the CBOR
3562 authenticator extension output value of each extension as the value.
3563

3564 The authenticator extension processing rules are used create the
3565 authenticator extension output from the authenticator extension input,
3566 and possibly also other inputs, for each extension.
3567

3568 9.6. Example Extension

3569 This section is not normative.
3570

3571 To illustrate the requirements above, consider a hypothetical
3572 registration extension and authentication extension "Geo". This
3573 extension, if supported, enables a geolocation location to be returned
3574 from the authenticator or client to the Relying Party.
3575

3576 The extension identifier is chosen as webauthnExample_geo. The client
3577 extension input is the constant value true, since the extension does
3578 not require the Relying Party to pass any particular information to the
3579 client, other than that it requests the use of the extension. The
3580 Relying Party sets this value in its request for an assertion:
3581

```
3582 var assertionPromise =  
3583   navigator.credentials.get({  
3584     publicKey: {  
3585       challenge: "SGFuIFNvbG8gc2hvdCBmaXJzdC4",  
3586       allowCredentials: [], /* Empty filter */  
3587       extensions: { 'webauthnExample_geo': true }  
3588     }  
3589   });  
3590
```

3591 The extension also requires the client to set the authenticator
3592 parameter to the fixed value true.
3593

3594 The extension requires the authenticator to specify its geolocation in
3595 the authenticator extension output, if known. The extension e.g.
3596 specifies that the location shall be encoded as a two-element array of
3597 floating point numbers, encoded with CBOR. An authenticator does this
3598 by including it in the authenticator data. As an example, authenticator
3599 data may be as follows (notation taken from [RFC7049]):
3600

```
3600 81 (hex)           -- Flags, ED and UP both set.  
3601 20 05 58 1F       -- Signature counter  
3602 A1                -- CBOR map of one element  
3603   73              -- Key 1: CBOR text string of 19 byt  
3604 es  
3605   77 65 62 61 75 74 68 6E 45 78 61  
3606   6D 70 6C 65 5F 67 65 6F       -- "webauthnExample_geo" [=UTF-8 enc  
3607 oded=] string  
3608   82                -- Value 1: CBOR array of two elemen  
3609 ts  
3610   FA 42 82 1E B3       -- Element 1: Latitude as CBOR encod  
3611 ed float  
3612   FA C1 5F E3 7F       -- Element 2: Longitude as CBOR enco  
3613 ded float  
3614
```

3615 The extension defines the client extension output to be the geolocation
3616 information, if known, as a GeoJSON [GeoJSON] point. The client
3617 constructs the following client data:
3618

```
3619 {  
3620   'extensions': {  
3621     'webauthnExample_geo': {  
3622       'type': 'Point',
```

```

3567     'coordinates': [65.059962, -13.993041]
3568   }
3569 }
3570 }
3571 }
3572
3573 10. Defined Extensions
3574
3575 This section defines the initial set of extensions to be registered in
3576 the IANA "WebAuthn Extension Identifier" registry established by
3577 [WebAuthn-Registries]. These are recommended for implementation by user
3578 agents targeting broad interoperability.
3579
3580 10.1. FIDO AppId Extension (appid)
3581
3582 This authentication extension allows Relying Parties that have
3583 previously registered a credential using the legacy FIDO JavaScript
3584 APIs to request an assertion. Specifically, this extension allows
3585 Relying Parties to specify an appid [FIDO-APPID] to overwrite the
3586 otherwise computed rpId. This extension is only valid if used during
3587 the get() call; other usage will result in client error.
3588
3589 Extension identifier
3590 appid
3591
3592 Client extension input
3593 A single JSON string specifying a FIDO appid.
3594
3595 Client extension processing
3596 If rpId is present, reject promise with a DOMException whose
3597 name is "NotAllowedError", and terminate this algorithm. Replace
3598 the calculation of rpId in Step 3 of 5.1.4 Use an existing
3599 credential to make an assertion with the following procedure:
3600 The client uses the value of appid to perform the AppId
3601 validation procedure (as defined by [FIDO-APPID]). If valid, the
3602 value of rpId for all client processing should be replaced by
3603 the value of appid.
3604
3605 Client extension output
3606 Returns the JSON value true to indicate to the RP that the
3607 extension was acted upon
3608
3609 Authenticator extension input
3610 None.
3611
3612 Authenticator extension processing
3613 None.
3614
3615 Authenticator extension output
3616 None.
3617
3618 10.2. Simple Transaction Authorization Extension (txAuthSimple)
3619
3620 This registration extension and authentication extension allows for a
3621 simple form of transaction authorization. A Relying Party can specify a
3622 prompt string, intended for display on a trusted device on the
3623 authenticator.
3624
3625 Extension identifier
3626 txAuthSimple
3627
3628 Client extension input
3629 A single JSON string prompt.
3630
3631 Client extension processing
3632 None, except creating the authenticator extension input from the
3633 client extension input.
3634
3635 Client extension output
3636 Returns the authenticator extension output string UTF-8 decoded
    into a JSON string
  
```

```

3623     'coordinates': [65.059962, -13.993041]
3624   }
3625 }
3626 }
3627 }
3628
3629 10. Defined Extensions
3630
3631 This section defines the initial set of extensions to be registered in
3632 the IANA "WebAuthn Extension Identifier" registry established by
3633 [WebAuthn-Registries]. These are recommended for implementation by user
3634 agents targeting broad interoperability.
3635
3636 10.1. FIDO AppId Extension (appid)
3637
3638 This authentication extension allows Relying Parties that have
3639 previously registered a credential using the legacy FIDO JavaScript
3640 APIs to request an assertion. Specifically, this extension allows
3641 Relying Parties to specify an appid [FIDO-APPID] to overwrite the
3642 otherwise computed rpId. This extension is only valid if used during
3643 the get() call; other usage will result in client error.
3644
3645 Extension identifier
3646 appid
3647
3648 Client extension input
3649 A single JSON string specifying a FIDO appid.
3650
3651 Client extension processing
3652 If rpId is present, reject promise with a DOMException whose
3653 name is "NotAllowedError", and terminate this algorithm. Replace
3654 the calculation of rpId in Step 3 of 5.1.4 Use an existing
3655 credential to make an assertion with the following procedure:
3656 The client uses the value of appid to perform the AppId
3657 validation procedure (as defined by [FIDO-APPID]). If valid, the
3658 value of rpId for all client processing should be replaced by
3659 the value of appid.
3660
3661 Client extension output
3662 Returns the JSON value true to indicate to the RP that the
3663 extension was acted upon
3664
3665 Authenticator extension input
3666 None.
3667
3668 Authenticator extension processing
3669 None.
3670
3671 Authenticator extension output
3672 None.
3673
3674 10.2. Simple Transaction Authorization Extension (txAuthSimple)
3675
3676 This registration extension and authentication extension allows for a
3677 simple form of transaction authorization. A Relying Party can specify a
3678 prompt string, intended for display on a trusted device on the
3679 authenticator.
3680
3681 Extension identifier
3682 txAuthSimple
3683
3684 Client extension input
3685 A single JSON string prompt.
3686
3687 Client extension processing
3688 None, except creating the authenticator extension input from the
3689 client extension input.
3690
3691 Client extension output
3692 Returns the authenticator extension output string UTF-8 decoded
    into a JSON string
  
```

3637 Authenticator extension input
 3638 The client extension input encoded as a CBOR text string (major
 3639 type 3).
 3640
 3641 Authenticator extension processing
 3642 The authenticator MUST display the prompt to the user before
 3643 performing either user verification or test of user presence.
 3644 The authenticator may insert line breaks if needed.
 3645
 3646 Authenticator extension output
 3647 A single CBOR string, representing the prompt as displayed
 3648 (including any eventual line breaks).
 3649
 3650 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
 3651
 3652 This registration extension and authentication extension allows images
 3653 to be used as transaction authorization prompts as well. This allows
 3654 authenticators without a font rendering engine to be used and also
 3655 supports a richer visual appearance.
 3656
 3657 Extension identifier
 3658 txAuthGeneric
 3659
 3660 Client extension input
 3661 A CBOR map defined as follows:
 3662
 3663 txAuthGenericArg = {
 3664 contentType: text, ; MIME-Type of the content, e.g.
 3665 "image/png"
 3666 content: bytes
 3667 }
 3668
 3669 Client extension processing
 3670 None, except creating the authenticator extension input from the
 3671 client extension input.
 3672
 3673 Client extension output
 3674 Returns the base64url encoding of the authenticator extension
 3675 output value as a JSON string
 3676
 3677 Authenticator extension input
 3678 The client extension input encoded as a CBOR map.
 3679
 3680 Authenticator extension processing
 3681 The authenticator MUST display the content to the user before
 3682 performing either user verification or test of user presence.
 3683 The authenticator may add other information below the content.
 3684 No changes are allowed to the content itself, i.e., inside
 3685 content boundary box.
 3686
 3687 Authenticator extension output
 3688 The hash value of the content which was displayed. The
 3689 authenticator MUST use the same hash algorithm as it uses for
 3690 the signature itself.
 3691
 3692 10.4. Authenticator Selection Extension (authnSel)
 3693
 3694 This registration extension allows a Relying Party to guide the
 3695 selection of the authenticator that will be leveraged when creating the
 3696 credential. It is intended primarily for Relying Parties that wish to
 3697 tightly control the experience around credential creation.
 3698
 3699 Extension identifier
 3700 authnSel
 3701
 3702 Client extension input
 3703 A sequence of AAGUIDs:
 3704
 3705 typedef sequence<AAGUID> AuthenticatorSelectionList;
 3706

3693 Authenticator extension input
 3694 The client extension input encoded as a CBOR text string (major
 3695 type 3).
 3696
 3697 Authenticator extension processing
 3698 The authenticator MUST display the prompt to the user before
 3699 performing either user verification or test of user presence.
 3700 The authenticator may insert line breaks if needed.
 3701
 3702 Authenticator extension output
 3703 A single CBOR string, representing the prompt as displayed
 3704 (including any eventual line breaks).
 3705
 3706 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
 3707
 3708 This registration extension and authentication extension allows images
 3709 to be used as transaction authorization prompts as well. This allows
 3710 authenticators without a font rendering engine to be used and also
 3711 supports a richer visual appearance.
 3712
 3713 Extension identifier
 3714 txAuthGeneric
 3715
 3716 Client extension input
 3717 A CBOR map defined as follows:
 3718
 3719 txAuthGenericArg = {
 3720 contentType: text, ; MIME-Type of the content, e.g.
 3721 "image/png"
 3722 content: bytes
 3723 }
 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 the base64url encoding of the authenticator extension
 3731 output value as a JSON string
 3732
 3733 Authenticator extension input
 3734 The client extension input encoded as a CBOR map.
 3735
 3736 Authenticator extension processing
 3737 The authenticator MUST display the content to the user before
 3738 performing either user verification or test of user presence.
 3739 The authenticator may add other information below the content.
 3740 No changes are allowed to the content itself, i.e., inside
 3741 content boundary box.
 3742
 3743 Authenticator extension output
 3744 The hash value of the content which was displayed. The
 3745 authenticator MUST use the same hash algorithm as it uses for
 3746 the signature itself.
 3747
 3748 10.4. Authenticator Selection Extension (authnSel)
 3749
 3750 This registration extension allows a Relying Party to guide the
 3751 selection of the authenticator that will be leveraged when creating the
 3752 credential. It is intended primarily for Relying Parties that wish to
 3753 tightly control the experience around credential creation.
 3754
 3755 Extension identifier
 3756 authnSel
 3757
 3758 Client extension input
 3759 A sequence of AAGUIDs:
 3760
 3761 typedef sequence<AAGUID> AuthenticatorSelectionList;
 3762

3707 Each AAGUID corresponds to an authenticator model that is
 3708 acceptable to the Relying Party for this credential creation.
 3709 The list is ordered by decreasing preference.
 3710
 3711 An AAGUID is defined as an array containing the globally unique
 3712 identifier of the authenticator model being sought.
 3713
 3714 typedef BufferSource AAGUID;
 3715
 3716 Client extension processing
 3717 This extension can only be used during create(). If the client
 3718 supports the Authenticator Selection Extension, it MUST use the
 3719 first available authenticator whose AAGUID is present in the
 3720 AuthenticatorSelectionList. If none of the available
 3721 authenticators match a provided AAGUID, the client MUST select
 3722 an authenticator from among the available authenticators to
 3723 generate the credential.
 3724
 3725 Client extension output
 3726 Returns the JSON value true to indicate to the RP that the
 3727 extension was acted upon
 3728
 3729 Authenticator extension input
 3730 None.
 3731
 3732 Authenticator extension processing
 3733 None.
 3734
 3735 Authenticator extension output
 3736 None.
 3737
 3738 10.5. Supported Extensions Extension (exts)
 3739
 3740 This registration extension enables the Relying Party to determine
 3741 which extensions the authenticator supports.
 3742
 3743 Extension identifier
 3744 exts
 3745
 3746 Client extension input
 3747 The Boolean value true to indicate that this extension is
 3748 requested by the Relying Party.
 3749
 3750 Client extension processing
 3751 None, except creating the authenticator extension input from the
 3752 client extension input.
 3753
 3754 Client extension output
 3755 Returns the list of supported extensions as a JSON array of
 3756 extension identifier strings
 3757
 3758 Authenticator extension input
 3759 The Boolean value true, encoded in CBOR (major type 7, value
 3760 21).
 3761
 3762 Authenticator extension processing
 3763 The authenticator sets the authenticator extension output to be
 3764 a list of extensions that the authenticator supports, as defined
 3765 below. This extension can be added to attestation objects.
 3766
 3767 Authenticator extension output
 3768 The SupportedExtensions extension is a list (CBOR array) of
 3769 extension identifier (UTF-8 encoded strings).
 3770
 3771 10.6. User Verification Index Extension (uvi)
 3772
 3773 This registration extension and authentication extension enables use of
 3774 a user verification index.
 3775
 3776

3763 Each AAGUID corresponds to an authenticator model that is
 3764 acceptable to the Relying Party for this credential creation.
 3765 The list is ordered by decreasing preference.
 3766
 3767 An AAGUID is defined as an array containing the globally unique
 3768 identifier of the authenticator model being sought.
 3769
 3770 typedef BufferSource AAGUID;
 3771
 3772 Client extension processing
 3773 This extension can only be used during create(). If the client
 3774 supports the Authenticator Selection Extension, it MUST use the
 3775 first available authenticator whose AAGUID is present in the
 3776 AuthenticatorSelectionList. If none of the available
 3777 authenticators match a provided AAGUID, the client MUST select
 3778 an authenticator from among the available authenticators to
 3779 generate the credential.
 3780
 3781 Client extension output
 3782 Returns the JSON value true to indicate to the RP that the
 3783 extension was acted upon
 3784
 3785 Authenticator extension input
 3786 None.
 3787
 3788 Authenticator extension processing
 3789 None.
 3790
 3791 Authenticator extension output
 3792 None.
 3793
 3794 10.5. Supported Extensions Extension (exts)
 3795
 3796 This registration extension enables the Relying Party to determine
 3797 which extensions the authenticator supports.
 3798
 3799 Extension identifier
 3800 exts
 3801
 3802 Client extension input
 3803 The Boolean value true to indicate that this extension is
 3804 requested by the Relying Party.
 3805
 3806 Client extension processing
 3807 None, except creating the authenticator extension input from the
 3808 client extension input.
 3809
 3810 Client extension output
 3811 Returns the list of supported extensions as a JSON array of
 3812 extension identifier strings
 3813
 3814 Authenticator extension input
 3815 The Boolean value true, encoded in CBOR (major type 7, value
 3816 21).
 3817
 3818 Authenticator extension processing
 3819 The authenticator sets the authenticator extension output to be
 3820 a list of extensions that the authenticator supports, as defined
 3821 below. This extension can be added to attestation objects.
 3822
 3823 Authenticator extension output
 3824 The SupportedExtensions extension is a list (CBOR array) of
 3825 extension identifier (UTF-8 encoded strings).
 3826
 3827 10.6. User Verification Index Extension (uvi)
 3828
 3829 This registration extension and authentication extension enables use of
 3830 a user verification index.
 3831
 3832

3777 Extension identifier
3778 uvi
3779
3780 Client extension input
3781 The Boolean value true to indicate that this extension is
3782 requested by the Relying Party.
3783
3784 Client extension processing
3785 None, except creating the authenticator extension input from the
3786 client extension input.
3787
3788 Client extension output
3789 Returns a JSON string containing the base64url encoding of the
3790 authenticator extension output
3791
3792 Authenticator extension input
3793 The Boolean value true, encoded in CBOR (major type 7, value
3794 21).
3795
3796 Authenticator extension processing
3797 The authenticator sets the authenticator extension output to be
3798 a user verification index indicating the method used by the user
3799 to authorize the operation, as defined below. This extension can
3800 be added to attestation objects and assertions.
3801
3802 Authenticator extension output
3803 The user verification index (UVI) is a value uniquely
3804 identifying a user verification data record. The UVI is encoded
3805 as CBOR byte string (type 0x58). Each UVI value MUST be specific
3806 to the related key (in order to provide unlinkability). It also
3807 must contain sufficient entropy that makes guessing impractical.
3808 UVI values MUST NOT be reused by the Authenticator (for other
3809 biometric data or users).
3810
3811 The UVI data can be used by servers to understand whether an
3812 authentication was authorized by the exact same biometric data
3813 as the initial key generation. This allows the detection and
3814 prevention of "friendly fraud".
3815
3816 As an example, the UVI could be computed as SHA256(KeyID ||
3817 SHA256(rawUVI)), where || represents concatenation, and the
3818 rawUVI reflects (a) the biometric reference data, (b) the
3819 related OS level user ID and (c) an identifier which changes
3820 whenever a factory reset is performed for the device, e.g.
3821 rawUVI = biometricReferenceData || OSLevelUserID ||
3822 FactoryResetCounter.
3823
3824 Servers supporting UVI extensions MUST support a length of up to
3825 32 bytes for the UVI value.
3826
3827 Example for authenticator data containing one UVI extension
3828
3829 ... -- [=RP ID=] hash (32 bytes)
3830 81 -- UP and ED set
3831 00 00 00 01 -- (initial) signature counter
3832 ... -- all public key alg etc.
3833 A1 -- extension: CBOR map of one elemen
3834 t
3835 63 -- Key 1: CBOR text string of 3 byte
3836 s
3837 75 76 69 -- "uvi" [=UTF-8 encoded=] string
3838 58 20 -- Value 1: CBOR byte string with 0x
3839 20 bytes
3840 00 43 B8 E3 BE 27 95 8C -- the UVI value itself
3841 28 D5 74 BF 46 8A 85 CF
3842 46 9A 14 F0 E5 16 69 31
3843 DA 4B CF FF C1 BB 11 32
3844 82
3845
3846 10.7. Location Extension (loc)

3833 Extension identifier
3834 uvi
3835
3836 Client extension input
3837 The Boolean value true to indicate that this extension is
3838 requested by the Relying Party.
3839
3840 Client extension processing
3841 None, except creating the authenticator extension input from the
3842 client extension input.
3843
3844 Client extension output
3845 Returns a JSON string containing the base64url encoding of the
3846 authenticator extension output
3847
3848 Authenticator extension input
3849 The Boolean value true, encoded in CBOR (major type 7, value
3850 21).
3851
3852 Authenticator extension processing
3853 The authenticator sets the authenticator extension output to be
3854 a user verification index indicating the method used by the user
3855 to authorize the operation, as defined below. This extension can
3856 be added to attestation objects and assertions.
3857
3858 Authenticator extension output
3859 The user verification index (UVI) is a value uniquely
3860 identifying a user verification data record. The UVI is encoded
3861 as CBOR byte string (type 0x58). Each UVI value MUST be specific
3862 to the related key (in order to provide unlinkability). It also
3863 must contain sufficient entropy that makes guessing impractical.
3864 UVI values MUST NOT be reused by the Authenticator (for other
3865 biometric data or users).
3866
3867 The UVI data can be used by servers to understand whether an
3868 authentication was authorized by the exact same biometric data
3869 as the initial key generation. This allows the detection and
3870 prevention of "friendly fraud".
3871
3872 As an example, the UVI could be computed as SHA256(KeyID ||
3873 SHA256(rawUVI)), where || represents concatenation, and the
3874 rawUVI reflects (a) the biometric reference data, (b) the
3875 related OS level user ID and (c) an identifier which changes
3876 whenever a factory reset is performed for the device, e.g.
3877 rawUVI = biometricReferenceData || OSLevelUserID ||
3878 FactoryResetCounter.
3879
3880 Servers supporting UVI extensions MUST support a length of up to
3881 32 bytes for the UVI value.
3882
3883 Example for authenticator data containing one UVI extension
3884
3885 ... -- [=RP ID=] hash (32 bytes)
3886 81 -- UP and ED set
3887 00 00 00 01 -- (initial) signature counter
3888 ... -- all public key alg etc.
3889 A1 -- extension: CBOR map of one elemen
3890 t
3891 63 -- Key 1: CBOR text string of 3 byte
3892 s
3893 75 76 69 -- "uvi" [=UTF-8 encoded=] string
3894 58 20 -- Value 1: CBOR byte string with 0x
3895 20 bytes
3896 00 43 B8 E3 BE 27 95 8C -- the UVI value itself
3897 28 D5 74 BF 46 8A 85 CF
3898 46 9A 14 F0 E5 16 69 31
3899 DA 4B CF FF C1 BB 11 32
3900 82
3901
3902 10.7. Location Extension (loc)

3847 The location registration extension and authentication extension
3848 provides the client device's current location to the WebAuthn Relying
3849 Party.
3850
3851 Extension identifier
3852 loc
3853
3854 Client extension input
3855 The Boolean value true to indicate that this extension is
3856 requested by the Relying Party.
3857
3858 Client extension processing
3859 None, except creating the authenticator extension input from the
3860 client extension input.
3861
3862 Client extension output
3863 Returns a JSON object that encodes the location information in
3864 the authenticator extension output as a Coordinates value, as
3865 defined by The W3C Geolocation API Specification.
3866
3867 Authenticator extension input
3868 The Boolean value true, encoded in CBOR (major type 7, value
3869 21).
3870
3871 Authenticator extension processing
3872 If the authenticator does not support the extension, then the
3873 authenticator MUST ignore the extension request. If the
3874 authenticator accepts the extension, then the authenticator
3875 SHOULD only add this extension data to a packed attestation or
3876 assertion.
3877
3878 Authenticator extension output
3879 If the authenticator accepts the extension request, then
3880 authenticator extension output SHOULD provide location data in
3881 the form of a CBOR-encoded map, with the first value being the
3882 extension identifier and the second being an array of returned
3883 values. The array elements SHOULD be derived from (key,value)
3884 pairings for each location attribute that the authenticator
3885 supports. The following is an example of authenticator data
3886 where the returned array is comprised of a {longitude, latitude,
3887 altitude} triplet, following the coordinate representation
3888 defined in The W3C Geolocation API Specification.
3889
3890 ... -- [=RP ID=] hash (32 bytes)
3891 81 -- UP and ED set
3892 00 00 00 01 -- (initial) signature counter
3893 ... -- all public key alg etc.
3894 A1 -- extension: CBOR map of one elemen
3895 t
3896 63 -- Value 1: CBOR text string of 3 by
3897 tes
3898 6C 6F 63 -- "loc" [=UTF-8 encoded=] string
3899 86 -- Value 2: array of 6 elements
3900 68 -- Element 1: CBOR text string of 8 bytes
3901 6C 61 74 69 74 75 64 65 -- "latitude" [=UTF-8 encoded=] stri
3902 ng
3903 FB ... -- Element 2: Latitude as CBOR encoded double-p
3904 recision float
3905 69 -- Element 3: CBOR text string of 9 bytes
3906 6C 6F 6E 67 69 74 75 64 65 -- "longitude" [=UTF-8 encoded=] str
3907 ing
3908 FB ... -- Element 4: Longitude as CBOR encoded double-
3909 precision float
3910 68 -- Element 5: CBOR text string of 8 bytes
3911 61 6C 74 69 74 75 64 65 -- "altitude" [=UTF-8 encoded=] stri
3912 ng
3913 FB ... -- Element 6: Altitude as CBOR encoded double-p
3914 recision float
3915
3916

3903 The location registration extension and authentication extension
3904 provides the client device's current location to the WebAuthn Relying
3905 Party.
3906
3907 Extension identifier
3908 loc
3909
3910 Client extension input
3911 The Boolean value true to indicate that this extension is
3912 requested by the Relying Party.
3913
3914 Client extension processing
3915 None, except creating the authenticator extension input from the
3916 client extension input.
3917
3918 Client extension output
3919 Returns a JSON object that encodes the location information in
3920 the authenticator extension output as a Coordinates value, as
3921 defined by The W3C Geolocation API Specification.
3922
3923 Authenticator extension input
3924 The Boolean value true, encoded in CBOR (major type 7, value
3925 21).
3926
3927 Authenticator extension processing
3928 If the authenticator does not support the extension, then the
3929 authenticator MUST ignore the extension request. If the
3930 authenticator accepts the extension, then the authenticator
3931 SHOULD only add this extension data to a packed attestation or
3932 assertion.
3933
3934 Authenticator extension output
3935 If the authenticator accepts the extension request, then
3936 authenticator extension output SHOULD provide location data in
3937 the form of a CBOR-encoded map, with the first value being the
3938 extension identifier and the second being an array of returned
3939 values. The array elements SHOULD be derived from (key,value)
3940 pairings for each location attribute that the authenticator
3941 supports. The following is an example of authenticator data
3942 where the returned array is comprised of a {longitude, latitude,
3943 altitude} triplet, following the coordinate representation
3944 defined in The W3C Geolocation API Specification.
3945
3946 ... -- [=RP ID=] hash (32 bytes)
3947 81 -- UP and ED set
3948 00 00 00 01 -- (initial) signature counter
3949 ... -- all public key alg etc.
3950 A1 -- extension: CBOR map of one elemen
3951 t
3952 63 -- Value 1: CBOR text string of 3 by
3953 tes
3954 6C 6F 63 -- "loc" [=UTF-8 encoded=] string
3955 86 -- Value 2: array of 6 elements
3956 68 -- Element 1: CBOR text string of 8 bytes
3957 6C 61 74 69 74 75 64 65 -- "latitude" [=UTF-8 encoded=] stri
3958 ng
3959 FB ... -- Element 2: Latitude as CBOR encoded double-p
3960 recision float
3961 69 -- Element 3: CBOR text string of 9 bytes
3962 6C 6F 6E 67 69 74 75 64 65 -- "longitude" [=UTF-8 encoded=] str
3963 ing
3964 FB ... -- Element 4: Longitude as CBOR encoded double-
3965 precision float
3966 68 -- Element 5: CBOR text string of 8 bytes
3967 61 6C 74 69 74 75 64 65 -- "altitude" [=UTF-8 encoded=] stri
3968 ng
3969 FB ... -- Element 6: Altitude as CBOR encoded double-p
3970 recision float
3971
3972

3917 10.8. User Verification Method Extension (uvm)
3918
3919 This registration extension and authentication extension enables use of
3920 a user verification method.
3921
3922 Extension identifier
3923 uvm
3924
3925 Client extension input
3926 The Boolean value true to indicate that this extension is
3927 requested by the WebAuthn Relying Party.
3928
3929 Client extension processing
3930 None, except creating the authenticator extension input from the
3931 client extension input.
3932
3933 Client extension output
3934 Returns a JSON array of 3-element arrays of numbers that encodes
3935 the factors in the authenticator extension output
3936
3937 Authenticator extension input
3938 The Boolean value true, encoded in CBOR (major type 7, value
3939 21).
3940
3941 Authenticator extension processing
3942 The authenticator sets the authenticator extension output to be
3943 a user verification index indicating the method used by the user
3944 to authorize the operation, as defined below. This extension can
3945 be added to attestation objects and assertions.
3946
3947 Authenticator extension output
3948 Authenticators can report up to 3 different user verification
3949 methods (factors) used in a single authentication instance,
3950 using the CBOR syntax defined below:
3951
3952 uvmFormat = [1*3 uvmEntry]
3953 uvmEntry = [
3954 userVerificationMethod: uint .size 4,
3955 keyProtectionType: uint .size 2,
3956 matcherProtectionType: uint .size 2
3957]
3958
3959 The semantics of the fields in each uvmEntry are as follows:
3960
3961 userVerificationMethod
3962 The authentication method/factor used by the authenticator
3963 to verify the user. Available values are defined in
3964 [FIDOReg], "User Verification Methods" section.
3965
3966 keyProtectionType
3967 The method used by the authenticator to protect the FIDO
3968 registration private key material. Available values are
3969 defined in [FIDOReg], "Key Protection Types" section.
3970
3971 matcherProtectionType
3972 The method used by the authenticator to protect the
3973 matcher that performs user verification. Available values
3974 are defined in [FIDOReg], "Matcher Protection Types"
3975 section.
3976
3977 If >3 factors can be used in an authentication instance the
3978 authenticator vendor must select the 3 factors it believes will
3979 be most relevant to the Server to include in the UVM.
3980
3981 Example for authenticator data containing one UVM extension for
3982 a multi-factor authentication instance where 2 factors were
3983 used:
3984
3985 ... -- [=RP ID=] hash (32 bytes)
3986 81 -- UP and ED set

3973 10.8. User Verification Method Extension (uvm)
3974
3975 This registration extension and authentication extension enables use of
3976 a user verification method.
3977
3978 Extension identifier
3979 uvm
3980
3981 Client extension input
3982 The Boolean value true to indicate that this extension is
3983 requested by the WebAuthn Relying Party.
3984
3985 Client extension processing
3986 None, except creating the authenticator extension input from the
3987 client extension input.
3988
3989 Client extension output
3990 Returns a JSON array of 3-element arrays of numbers that encodes
3991 the factors in the authenticator extension output
3992
3993 Authenticator extension input
3994 The Boolean value true, encoded in CBOR (major type 7, value
3995 21).
3996
3997 Authenticator extension processing
3998 The authenticator sets the authenticator extension output to be
3999 a user verification index indicating the method used by the user
4000 to authorize the operation, as defined below. This extension can
4001 be added to attestation objects and assertions.
4002
4003 Authenticator extension output
4004 Authenticators can report up to 3 different user verification
4005 methods (factors) used in a single authentication instance,
4006 using the CBOR syntax defined below:
4007
4008 uvmFormat = [1*3 uvmEntry]
4009 uvmEntry = [
4010 userVerificationMethod: uint .size 4,
4011 keyProtectionType: uint .size 2,
4012 matcherProtectionType: uint .size 2
4013]
4014
4015 The semantics of the fields in each uvmEntry are as follows:
4016
4017 userVerificationMethod
4018 The authentication method/factor used by the authenticator
4019 to verify the user. Available values are defined in
4020 [FIDOReg], "User Verification Methods" section.
4021
4022 keyProtectionType
4023 The method used by the authenticator to protect the FIDO
4024 registration private key material. Available values are
4025 defined in [FIDOReg], "Key Protection Types" section.
4026
4027 matcherProtectionType
4028 The method used by the authenticator to protect the
4029 matcher that performs user verification. Available values
4030 are defined in [FIDOReg], "Matcher Protection Types"
4031 section.
4032
4033 If >3 factors can be used in an authentication instance the
4034 authenticator vendor must select the 3 factors it believes will
4035 be most relevant to the Server to include in the UVM.
4036
4037 Example for authenticator data containing one UVM extension for
4038 a multi-factor authentication instance where 2 factors were
4039 used:
4040
4041 ... -- [=RP ID=] hash (32 bytes)
4042 81 -- UP and ED set

```

3987 00 00 00 01 -- (initial) signature counter
3988 ... -- all public key alg etc.
3989 A1 -- extension: CBOR map of one element
3990 63 -- Key 1: CBOR text string of 3 bytes
3991 75 76 6d -- "uvm" [=UTF-8 encoded=] string
3992 82 -- Value 1: CBOR array of length 2 indicating two factor
3993 usage
3994 83 -- Item 1: CBOR array of length 3
3995 02 -- Subitem 1: CBOR integer for User Verification Method
3996 Fingerprint
3997 04 -- Subitem 2: CBOR short for Key Protection Type TEE
3998 02 -- Subitem 3: CBOR short for Matcher Protection Type TE
3999 E
4000 83 -- Item 2: CBOR array of length 3
4001 04 -- Subitem 1: CBOR integer for User Verification Method
4002 Passcode
4003 01 -- Subitem 2: CBOR short for Key Protection Type Softwa
4004 re
4005 01 -- Subitem 3: CBOR short for Matcher Protection Type So
4006 ftware
4007
4008 11. IANA Considerations
4009
4010 11.1. WebAuthn Attestation Statement Format Identifier Registrations
4011
4012 This section registers the attestation statement formats defined in
4013 Section 8 Defined Attestation Statement Formats in the IANA "WebAuthn
4014 Attestation Statement Format Identifier" registry established by
4015 [WebAuthn-Registries].
4016 * WebAuthn Attestation Statement Format Identifier: packed
4017 * Description: The "packed" attestation statement format is a
4018 WebAuthn-optimized format for attestation. It uses a very compact
4019 but still extensible encoding method. This format is implementable
4020 by authenticators with limited resources (e.g., secure elements).
4021 * Specification Document: Section 8.2 Packed Attestation Statement
4022 Format of this specification
4023 * WebAuthn Attestation Statement Format Identifier: tpm
4024 * Description: The TPM attestation statement format returns an
4025 attestation statement in the same format as the packed attestation
4026 statement format, although the the rawData and signature fields are
4027 computed differently.
4028 * Specification Document: Section 8.3 TPM Attestation Statement
4029 Format of this specification
4030 * WebAuthn Attestation Statement Format Identifier: android-key
4031 * Description: Platform-provided authenticators based on versions
4032 "N", and later, may provide this proprietary "hardware attestation"
4033 statement.
4034 * Specification Document: Section 8.4 Android Key Attestation
4035 Statement Format of this specification
4036 * WebAuthn Attestation Statement Format Identifier: android-safetynet
4037 * Description: Android-based, platform-provided authenticators may
4038 produce an attestation statement based on the Android SafetyNet
4039 API.
4040 * Specification Document: Section 8.5 Android SafetyNet Attestation
4041 Statement Format of this specification
4042 * WebAuthn Attestation Statement Format Identifier: fido-u2f
4043 * Description: Used with FIDO U2F authenticators
4044 * Specification Document: Section 8.6 FIDO U2F Attestation Statement
4045 Format of this specification
4046
4047 11.2. WebAuthn Extension Identifier Registrations
4048
4049 This section registers the extension identifier values defined in
4050 Section 9 WebAuthn Extensions in the IANA "WebAuthn Extension
4051 Identifier" registry established by [WebAuthn-Registries].
4052 * WebAuthn Extension Identifier: appid
4053 * Description: This authentication extension allows Relying Parties
4054 that have previously registered a credential using the legacy FIDO
4055 JavaScript APIs to request an assertion.
4056 * Specification Document: Section 10.1 FIDO AppId Extension (appid)

```

```

4043 00 00 00 01 -- (initial) signature counter
4044 ... -- all public key alg etc.
4045 A1 -- extension: CBOR map of one element
4046 63 -- Key 1: CBOR text string of 3 bytes
4047 75 76 6d -- "uvm" [=UTF-8 encoded=] string
4048 82 -- Value 1: CBOR array of length 2 indicating two factor
4049 usage
4050 83 -- Item 1: CBOR array of length 3
4051 02 -- Subitem 1: CBOR integer for User Verification Method
4052 Fingerprint
4053 04 -- Subitem 2: CBOR short for Key Protection Type TEE
4054 02 -- Subitem 3: CBOR short for Matcher Protection Type TE
4055 E
4056 83 -- Item 2: CBOR array of length 3
4057 04 -- Subitem 1: CBOR integer for User Verification Method
4058 Passcode
4059 01 -- Subitem 2: CBOR short for Key Protection Type Softwa
4060 re
4061 01 -- Subitem 3: CBOR short for Matcher Protection Type So
4062 ftware
4063
4064 11. IANA Considerations
4065
4066 11.1. WebAuthn Attestation Statement Format Identifier Registrations
4067
4068 This section registers the attestation statement formats defined in
4069 Section 8 Defined Attestation Statement Formats in the IANA "WebAuthn
4070 Attestation Statement Format Identifier" registry established by
4071 [WebAuthn-Registries].
4072 * WebAuthn Attestation Statement Format Identifier: packed
4073 * Description: The "packed" attestation statement format is a
4074 WebAuthn-optimized format for attestation. It uses a very compact
4075 but still extensible encoding method. This format is implementable
4076 by authenticators with limited resources (e.g., secure elements).
4077 * Specification Document: Section 8.2 Packed Attestation Statement
4078 Format of this specification
4079 * WebAuthn Attestation Statement Format Identifier: tpm
4080 * Description: The TPM attestation statement format returns an
4081 attestation statement in the same format as the packed attestation
4082 statement format, although the the rawData and signature fields are
4083 computed differently.
4084 * Specification Document: Section 8.3 TPM Attestation Statement
4085 Format of this specification
4086 * WebAuthn Attestation Statement Format Identifier: android-key
4087 * Description: Platform-provided authenticators based on versions
4088 "N", and later, may provide this proprietary "hardware attestation"
4089 statement.
4090 * Specification Document: Section 8.4 Android Key Attestation
4091 Statement Format of this specification
4092 * WebAuthn Attestation Statement Format Identifier: android-safetynet
4093 * Description: Android-based, platform-provided authenticators may
4094 produce an attestation statement based on the Android SafetyNet
4095 API.
4096 * Specification Document: Section 8.5 Android SafetyNet Attestation
4097 Statement Format of this specification
4098 * WebAuthn Attestation Statement Format Identifier: fido-u2f
4099 * Description: Used with FIDO U2F authenticators
4100 * Specification Document: Section 8.6 FIDO U2F Attestation Statement
4101 Format of this specification
4102
4103 11.2. WebAuthn Extension Identifier Registrations
4104
4105 This section registers the extension identifier values defined in
4106 Section 9 WebAuthn Extensions in the IANA "WebAuthn Extension
4107 Identifier" registry established by [WebAuthn-Registries].
4108 * WebAuthn Extension Identifier: appid
4109 * Description: This authentication extension allows Relying Parties
4110 that have previously registered a credential using the legacy FIDO
4111 JavaScript APIs to request an assertion.
4112 * Specification Document: Section 10.1 FIDO AppId Extension (appid)

```

4057 of this specification
4058 * WebAuthn Extension Identifier: txAuthSimple
4059 * Description: This registration extension and authentication
4060 extension allows for a simple form of transaction authorization. A
4061 WebAuthn Relying Party can specify a prompt string, intended for
4062 display on a trusted device on the authenticator
4063 * Specification Document: Section 10.2 Simple Transaction
4064 Authorization Extension (txAuthSimple) of this specification
4065 * WebAuthn Extension Identifier: txAuthGeneric
4066 * Description: This registration extension and authentication
4067 extension allows images to be used as transaction authorization
4068 prompts as well. This allows authenticators without a font
4069 rendering engine to be used and also supports a richer visual
4070 appearance than accomplished with the webauthn.txauth.simple
4071 extension.
4072 * Specification Document: Section 10.3 Generic Transaction
4073 Authorization Extension (txAuthGeneric) of this specification
4074 * WebAuthn Extension Identifier: authnSel
4075 * Description: This registration extension allows a WebAuthn Relying
4076 Party to guide the selection of the authenticator that will be
4077 leveraged when creating the credential. It is intended primarily
4078 for WebAuthn Relying Parties that wish to tightly control the
4079 experience around credential creation.
4080 * Specification Document: Section 10.4 Authenticator Selection
4081 Extension (authnSel) of this specification
4082 * WebAuthn Extension Identifier: exts
4083 * Description: This registration extension enables the Relying Party
4084 to determine which extensions the authenticator supports. The
4085 extension data is a list (CBOR array) of extension identifiers
4086 encoded as UTF-8 Strings. This extension is added automatically by
4087 the authenticator. This extension can be added to attestation
4088 statements.
4089 * Specification Document: Section 10.5 Supported Extensions
4090 Extension (exts) of this specification
4091 * WebAuthn Extension Identifier: uvi
4092 * Description: This registration extension and authentication
4093 extension enables use of a user verification index. The user
4094 verification index is a value uniquely identifying a user
4095 verification data record. The UVI data can be used by servers to
4096 understand whether an authentication was authorized by the exact
4097 same biometric data as the initial key generation. This allows the
4098 detection and prevention of "friendly fraud".
4099 * Specification Document: Section 10.6 User Verification Index
4100 Extension (uvi) of this specification
4101 * WebAuthn Extension Identifier: loc
4102 * Description: The location registration extension and authentication
4103 extension provides the client device's current location to the
4104 WebAuthn relying party, if supported by the client device and
4105 subject to user consent.
4106 * Specification Document: Section 10.7 Location Extension (loc) of
4107 this specification
4108 * WebAuthn Extension Identifier: uvm
4109 * Description: This registration extension and authentication
4110 extension enables use of a user verification method. The user
4111 verification method extension returns to the Webauthn relying party
4112 which user verification methods (factors) were used for the
4113 WebAuthn operation.
4114 * Specification Document: Section 10.8 User Verification Method
4115 Extension (uvm) of this specification

11.3. COSE Algorithm Registrations

4116 This section registers identifiers for RSASSA-PKCS1-v1_5 [RFC8017]
4117 algorithms using SHA-2 hash functions in the IANA COSE Algorithms
4118 registry [IANA-COSE-ALGS-REG].

- 4119 * Name: RS256
- 4120 * Value: -257
- 4121 * Description: RSASSA-PKCS1-v1_5 w/ SHA-256
- 4122 * Reference: Section 8.2 of [RFC8017]
- 4123 * Recommended: No

4113 of this specification
4114 * WebAuthn Extension Identifier: txAuthSimple
4115 * Description: This registration extension and authentication
4116 extension allows for a simple form of transaction authorization. A
4117 WebAuthn Relying Party can specify a prompt string, intended for
4118 display on a trusted device on the authenticator
4119 * Specification Document: Section 10.2 Simple Transaction
4120 Authorization Extension (txAuthSimple) of this specification
4121 * WebAuthn Extension Identifier: txAuthGeneric
4122 * Description: This registration extension and authentication
4123 extension allows images to be used as transaction authorization
4124 prompts as well. This allows authenticators without a font
4125 rendering engine to be used and also supports a richer visual
4126 appearance than accomplished with the webauthn.txauth.simple
4127 extension.
4128 * Specification Document: Section 10.3 Generic Transaction
4129 Authorization Extension (txAuthGeneric) of this specification
4130 * WebAuthn Extension Identifier: authnSel
4131 * Description: This registration extension allows a WebAuthn Relying
4132 Party to guide the selection of the authenticator that will be
4133 leveraged when creating the credential. It is intended primarily
4134 for WebAuthn Relying Parties that wish to tightly control the
4135 experience around credential creation.
4136 * Specification Document: Section 10.4 Authenticator Selection
4137 Extension (authnSel) of this specification
4138 * WebAuthn Extension Identifier: exts
4139 * Description: This registration extension enables the Relying Party
4140 to determine which extensions the authenticator supports. The
4141 extension data is a list (CBOR array) of extension identifiers
4142 encoded as UTF-8 Strings. This extension is added automatically by
4143 the authenticator. This extension can be added to attestation
4144 statements.
4145 * Specification Document: Section 10.5 Supported Extensions
4146 Extension (exts) of this specification
4147 * WebAuthn Extension Identifier: uvi
4148 * Description: This registration extension and authentication
4149 extension enables use of a user verification index. The user
4150 verification index is a value uniquely identifying a user
4151 verification data record. The UVI data can be used by servers to
4152 understand whether an authentication was authorized by the exact
4153 same biometric data as the initial key generation. This allows the
4154 detection and prevention of "friendly fraud".
4155 * Specification Document: Section 10.6 User Verification Index
4156 Extension (uvi) of this specification
4157 * WebAuthn Extension Identifier: loc
4158 * Description: The location registration extension and authentication
4159 extension provides the client device's current location to the
4160 WebAuthn relying party, if supported by the client device and
4161 subject to user consent.
4162 * Specification Document: Section 10.7 Location Extension (loc) of
4163 this specification
4164 * WebAuthn Extension Identifier: uvm
4165 * Description: This registration extension and authentication
4166 extension enables use of a user verification method. The user
4167 verification method extension returns to the Webauthn relying party
4168 which user verification methods (factors) were used for the
4169 WebAuthn operation.
4170 * Specification Document: Section 10.8 User Verification Method
4171 Extension (uvm) of this specification

11.3. COSE Algorithm Registrations

4172 This section registers identifiers for RSASSA-PKCS1-v1_5 [RFC8017]
4173 algorithms using SHA-2 hash functions in the IANA COSE Algorithms
4174 registry [IANA-COSE-ALGS-REG].

- 4175 * Name: RS256
- 4176 * Value: -257
- 4177 * Description: RSASSA-PKCS1-v1_5 w/ SHA-256
- 4178 * Reference: Section 8.2 of [RFC8017]
- 4179 * Recommended: No

4127 * Name: RS384
 4128 * Value: -258
 4129 * Description: RSASSA-PKCS1-v1_5 w/ SHA-384
 4130 * Reference: Section 8.2 of [RFC8017]
 4131 * Recommended: No
 4132 * Name: RS512
 4133 * Value: -259
 4134 * Description: RSASSA-PKCS1-v1_5 w/ SHA-512
 4135 * Reference: Section 8.2 of [RFC8017]
 4136 * Recommended: No
 4137 * Name: ED256
 4138 * Value: -260
 4139 * Description: TPM_ECC_BN_P256 curve w/ SHA-256
 4140 * Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
 4141 * Recommended: Yes
 4142 * Name: ED512
 4143 * Value: -261
 4144 * Description: ECC_BN_ISOP512 curve w/ SHA-512
 4145 * Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
 4146 * Recommended: Yes

12. Sample scenarios

This section is not normative.

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

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

12.1. Registration

This is the first-time flow, in which a new credential is created and registered with the server. In this flow, the Relying Party does not have a preference for platform authenticator or roaming authenticators.

1. The user visits example.com, which serves up a script. At this point, the user may already be logged in using a legacy username and password, or additional authenticator, or other means acceptable to the Relying Party. Or the user may be in the process of creating a new account.
2. The Relying Party script runs the code snippet below.
3. The client platform searches for and locates the authenticator.
4. The client platform connects to the authenticator, performing any pairing actions if necessary.
5. The authenticator shows appropriate UI for the user to select the authenticator on which the new credential will be created, and obtains a biometric or other authorization gesture from the user.
6. The authenticator returns a response to the client platform, which in turn returns a response to the Relying Party script. If the user declined to select an authenticator or provide authorization, an appropriate error is returned.
7. If a new credential was created,
 - + The Relying Party script sends the newly generated credential public key to the server, along with additional information such as attestation regarding the provenance and characteristics of the authenticator.

4183 * Name: RS384
 4184 * Value: -258
 4185 * Description: RSASSA-PKCS1-v1_5 w/ SHA-384
 4186 * Reference: Section 8.2 of [RFC8017]
 4187 * Recommended: No
 4188 * Name: RS512
 4189 * Value: -259
 4190 * Description: RSASSA-PKCS1-v1_5 w/ SHA-512
 4191 * Reference: Section 8.2 of [RFC8017]
 4192 * Recommended: No
 4193 * Name: ED256
 4194 * Value: -260
 4195 * Description: TPM_ECC_BN_P256 curve w/ SHA-256
 4196 * Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
 4197 * Recommended: Yes
 4198 * Name: ED512
 4199 * Value: -261
 4200 * Description: ECC_BN_ISOP512 curve w/ SHA-512
 4201 * Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
 4202 * Recommended: Yes

12. Sample scenarios

This section is not normative.

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

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

12.1. Registration

This is the first-time flow, in which a new credential is created and registered with the server. In this flow, the Relying Party does not have a preference for platform authenticator or roaming authenticators.

1. The user visits example.com, which serves up a script. At this point, the user may already be logged in using a legacy username and password, or additional authenticator, or other means acceptable to the Relying Party. Or the user may be in the process of creating a new account.
2. The Relying Party script runs the code snippet below.
3. The client platform searches for and locates the authenticator.
4. The client platform connects to the authenticator, performing any pairing actions if necessary.
5. The authenticator shows appropriate UI for the user to select the authenticator on which the new credential will be created, and obtains a biometric or other authorization gesture from the user.
6. The authenticator returns a response to the client platform, which in turn returns a response to the Relying Party script. If the user declined to select an authenticator or provide authorization, an appropriate error is returned.
7. If a new credential was created,
 - + The Relying Party script sends the newly generated credential public key to the server, along with additional information such as attestation regarding the provenance and characteristics of the authenticator.

```

4197 + The server stores the credential public key in its database
4198 and associates it with the user as well as with the
4199 characteristics of authentication indicated by attestation,
4200 also storing a friendly name for later use.
4201 + The script may store data such as the credential ID in local
4202 storage, to improve future UX by narrowing the choice of
4203 credential for the user.
4204
4205 The sample code for generating and registering a new key follows:
4206 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4207
4208 var publicKey = {
4209   challenge: Uint8Array.from(window.atob("PGifxAoBwCkWkm4b1Cill5otCphilh6MijdbW
4210 FjomA="), c=>c.charCodeAt(0)),
4211
4212   // Relying Party:
4213   rp: {
4214     name: "Acme"
4215   },
4216
4217   // User:
4218   user: {
4219     id: Uint8Array.from(window.atob("MIIBkzCCATigAwIBAjCCAAMwggE4oAMCAQIwggGTMI
4220 ="), c=>c.charCodeAt(0)),
4221     name: "john.p.smith@example.com",
4222     displayName: "John P. Smith",
4223     icon: "https://pics.acme.com/00/p/aBjjjpqPb.png"
4224   },
4225
4226   // This Relying Party will accept either an ES256 or RS256 credential, but
4227   // prefers an ES256 credential.
4228   pubKeyCredParams: [
4229     {
4230       type: "public-key",
4231       alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
4232     },
4233     {
4234       type: "public-key",
4235       alg: -257 // Value registered by this specification for "RS256"
4236     }
4237   ],
4238
4239   timeout: 60000, // 1 minute
4240   excludeCredentials: [], // No exclude list of PKCredDescriptors
4241   extensions: {"webauthn.location": true} // Include location information
4242               // in attestation
4243 };
4244
4245 // Note: The following call will cause the authenticator to display UI.
4246 navigator.credentials.create({ publicKey })
4247   .then(function (newCredentialInfo) {
4248     // Send new credential info to server for verification and registration.
4249   }).catch(function (err) {
4250     // No acceptable authenticator or user refused consent. Handle appropriately
4251   });
4252
4253
4254 12.2. Registration Specifically with Platform Authenticator
4255
4256 This is flow for when the Relying Party is specifically interested in
4257 creating a public key credential with a platform authenticator.
4258 1. The user visits example.com and clicks on the login button, which
4259   redirects the user to login.example.com.
4260 2. The user enters a username and password to log in. After successful
4261   login, the user is redirected back to example.com.
4262 3. The Relying Party script runs the code snippet below.
4263 4. The user agent asks the user whether they are willing to register
4264   with the Relying Party using an available platform authenticator.
4265 5. If the user is not willing, terminate this flow.
4266 6. The user is shown appropriate UI and guided in creating a

```

```

4253 + The server stores the credential public key in its database
4254 and associates it with the user as well as with the
4255 characteristics of authentication indicated by attestation,
4256 also storing a friendly name for later use.
4257 + The script may store data such as the credential ID in local
4258 storage, to improve future UX by narrowing the choice of
4259 credential for the user.
4260
4261 The sample code for generating and registering a new key follows:
4262 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4263
4264 var publicKey = {
4265   challenge: Uint8Array.from(window.atob("PGifxAoBwCkWkm4b1Cill5otCphilh6MijdbW
4266 FjomA="), c=>c.charCodeAt(0)),
4267
4268   // Relying Party:
4269   rp: {
4270     name: "Acme"
4271   },
4272
4273   // User:
4274   user: {
4275     id: Uint8Array.from(window.atob("MIIBkzCCATigAwIBAjCCAAMwggE4oAMCAQIwggGTMI
4276 ="), c=>c.charCodeAt(0)),
4277     name: "john.p.smith@example.com",
4278     displayName: "John P. Smith",
4279     icon: "https://pics.acme.com/00/p/aBjjjpqPb.png"
4280   },
4281
4282   // This Relying Party will accept either an ES256 or RS256 credential, but
4283   // prefers an ES256 credential.
4284   pubKeyCredParams: [
4285     {
4286       type: "public-key",
4287       alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
4288     },
4289     {
4290       type: "public-key",
4291       alg: -257 // Value registered by this specification for "RS256"
4292     }
4293   ],
4294
4295   timeout: 60000, // 1 minute
4296   excludeCredentials: [], // No exclude list of PKCredDescriptors
4297   extensions: {"webauthn.location": true} // Include location information
4298               // in attestation
4299 };
4300
4301 // Note: The following call will cause the authenticator to display UI.
4302 navigator.credentials.create({ publicKey })
4303   .then(function (newCredentialInfo) {
4304     // Send new credential info to server for verification and registration.
4305   }).catch(function (err) {
4306     // No acceptable authenticator or user refused consent. Handle appropriately
4307   });
4308
4309
4310 12.2. Registration Specifically with Platform Authenticator
4311
4312 This is flow for when the Relying Party is specifically interested in
4313 creating a public key credential with a platform authenticator.
4314 1. The user visits example.com and clicks on the login button, which
4315   redirects the user to login.example.com.
4316 2. The user enters a username and password to log in. After successful
4317   login, the user is redirected back to example.com.
4318 3. The Relying Party script runs the code snippet below.
4319 4. The user agent asks the user whether they are willing to register
4320   with the Relying Party using an available platform authenticator.
4321 5. If the user is not willing, terminate this flow.
4322 6. The user is shown appropriate UI and guided in creating a

```

```

4267 credential using one of the available platform authenticators. Upon
4268 successful credential creation, the RP script conveys the new
4269 credential to the server.
4270 if (!PublicKeyCredential) { /* Platform not capable of the API. Handle error. */
4271 }
4272
4273 PublicKeyCredential.isPlatformAuthenticatorAvailable()
4274 .then(function (userIntent) {
4275
4276     // If the user has affirmed willingness to register with RP using an ava
4277     ilable platform authenticator
4278     if (userIntent) {
4279         var publicKeyOptions = { /* Public key credential creation options.
4280 */};
4281
4282         // Create and register credentials.
4283         return navigator.credentials.create({ "publicKey": publicKeyOptions
4284 });
4285     } else {
4286
4287         // Record that the user does not intend to use a platform authentica
4288         tor
4289         // and default the user to a password-based flow in the future.
4290     }
4291
4292     }).then(function (newCredentialInfo) {
4293         // Send new credential info to server for verification and registration.
4294     }).catch( function(err) {
4295         // Something went wrong. Handle appropriately.
4296     });
4297

```

12.3. Authentication

This is the flow when a user with an already registered credential visits a website and wants to authenticate using the credential.

1. The user visits example.com, which serves up a script.
2. The script asks the client platform for an Authentication Assertion, providing as much information as possible to narrow the choice of acceptable credentials for the user. This may be obtained from the data that was stored locally after registration, or by other means such as prompting the user for a username.
3. The Relying Party script runs one of the code snippets below.
4. The client platform searches for and locates the authenticator.
5. The client platform connects to the authenticator, performing any pairing actions if necessary.
6. The authenticator presents the user with a notification that their attention is required. On opening the notification, the user is shown a friendly selection menu of acceptable credentials using the account information provided when creating the credentials, along with some information on the origin that is requesting these keys.
7. The authenticator obtains a biometric or other authorization gesture from the user.
8. The authenticator returns a response to the client platform, which in turn returns a response to the Relying Party script. If the user declined to select a credential or provide an authorization, an appropriate error is returned.
9. If an assertion was successfully generated and returned,
 - + The script sends the assertion to the server.
 - + The server examines the assertion, extracts the credential ID, looks up the registered credential public key it is database, and verifies the assertion's authentication signature. If valid, it looks up the identity associated with the assertion's credential ID; that identity is now authenticated. If the credential ID is not recognized by the server (e.g., it has been deregistered due to inactivity) then the authentication has failed; each Relying Party will handle this in its own way.
 - + The server now does whatever it would otherwise do upon successful authentication -- return a success page, set authentication cookies, etc.

```

4323 credential using one of the available platform authenticators. Upon
4324 successful credential creation, the RP script conveys the new
4325 credential to the server.
4326 if (!PublicKeyCredential) { /* Platform not capable of the API. Handle error. */
4327 }
4328
4329 PublicKeyCredential.isPlatformAuthenticatorAvailable()
4330 .then(function (userIntent) {
4331
4332     // If the user has affirmed willingness to register with RP using an ava
4333     ilable platform authenticator
4334     if (userIntent) {
4335         var publicKeyOptions = { /* Public key credential creation options.
4336 */};
4337
4338         // Create and register credentials.
4339         return navigator.credentials.create({ "publicKey": publicKeyOptions
4340 });
4341     } else {
4342
4343         // Record that the user does not intend to use a platform authentica
4344         tor
4345         // and default the user to a password-based flow in the future.
4346     }
4347
4348     }).then(function (newCredentialInfo) {
4349         // Send new credential info to server for verification and registration.
4350     }).catch( function(err) {
4351         // Something went wrong. Handle appropriately.
4352     });
4353

```

12.3. Authentication

This is the flow when a user with an already registered credential visits a website and wants to authenticate using the credential.

1. The user visits example.com, which serves up a script.
2. The script asks the client platform for an Authentication Assertion, providing as much information as possible to narrow the choice of acceptable credentials for the user. This may be obtained from the data that was stored locally after registration, or by other means such as prompting the user for a username.
3. The Relying Party script runs one of the code snippets below.
4. The client platform searches for and locates the authenticator.
5. The client platform connects to the authenticator, performing any pairing actions if necessary.
6. The authenticator presents the user with a notification that their attention is required. On opening the notification, the user is shown a friendly selection menu of acceptable credentials using the account information provided when creating the credentials, along with some information on the origin that is requesting these keys.
7. The authenticator obtains a biometric or other authorization gesture from the user.
8. The authenticator returns a response to the client platform, which in turn returns a response to the Relying Party script. If the user declined to select a credential or provide an authorization, an appropriate error is returned.
9. If an assertion was successfully generated and returned,
 - + The script sends the assertion to the server.
 - + The server examines the assertion, extracts the credential ID, looks up the registered credential public key it is database, and verifies the assertion's authentication signature. If valid, it looks up the identity associated with the assertion's credential ID; that identity is now authenticated. If the credential ID is not recognized by the server (e.g., it has been deregistered due to inactivity) then the authentication has failed; each Relying Party will handle this in its own way.
 - + The server now does whatever it would otherwise do upon successful authentication -- return a success page, set authentication cookies, etc.

```

4337
4338 If the Relying Party script does not have any hints available (e.g.,
4339 from locally stored data) to help it narrow the list of credentials,
4340 then the sample code for performing such an authentication might look
4341 like this:
4342 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4343
4344 var options = {
4345     challenge: new TextEncoder().encode("climb a mountain"),
4346     timeout: 60000, // 1 minute
4347     allowCredentials: [{ type: "public-key" }]
4348 };
4349
4350 navigator.credentials.get({ "publicKey": options })
4351     .then(function (assertion) {
4352         // Send assertion to server for verification
4353     }).catch(function (err) {
4354         // No acceptable credential or user refused consent. Handle appropriately.
4355     });
4356
4357 On the other hand, if the Relying Party script has some hints to help
4358 it narrow the list of credentials, then the sample code for performing
4359 such an authentication might look like the following. Note that this
4360 sample also demonstrates how to use the extension for transaction
4361 authorization.
4362 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4363
4364 var encoder = new TextEncoder();
4365 var acceptableCredential1 = {
4366     type: "public-key",
4367     id: encoder.encode("!!!!!!hi there!!!!!!\n")
4368 };
4369 var acceptableCredential2 = {
4370     type: "public-key",
4371     id: encoder.encode("roses are red, violets are blue\n")
4372 };
4373
4374 var options = {
4375     challenge: encoder.encode("climb a mountain"),
4376     timeout: 60000, // 1 minute
4377     allowCredentials: [acceptableCredential1, acceptableCredential2]
4378 ,
4379     extensions: { 'webauthn.txauth.simple':
4380         "Wave your hands in the air like you just don't care" }
4381 };
4382
4383 navigator.credentials.get({ "publicKey": options })
4384     .then(function (assertion) {
4385         // Send assertion to server for verification
4386     }).catch(function (err) {
4387         // No acceptable credential or user refused consent. Handle appropriately.
4388     });
4389
4390 12.4. Decommissioning
4391
4392 The following are possible situations in which decommissioning a
4393 credential might be desired. Note that all of these are handled on the
4394 server side and do not need support from the API specified here.
4395 * Possibility #1 -- user reports the credential as lost.
4396 + User goes to server.example.net, authenticates and follows a
4397 link to report a lost/stolen device.
4398 + Server returns a page showing the list of registered
4399 credentials with friendly names as configured during
4400 registration.
4401 + User selects a credential and the server deletes it from its
4402 database.
4403 + In future, the Relying Party script does not specify this
4404 credential in any list of acceptable credentials, and
4405 assertions signed by this credential are rejected.
4406 * Possibility #2 -- server deregisters the credential due to

```

```

4393
4394 If the Relying Party script does not have any hints available (e.g.,
4395 from locally stored data) to help it narrow the list of credentials,
4396 then the sample code for performing such an authentication might look
4397 like this:
4398 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4399
4400 var options = {
4401     challenge: new TextEncoder().encode("climb a mountain"),
4402     timeout: 60000, // 1 minute
4403     allowCredentials: [{ type: "public-key" }]
4404 };
4405
4406 navigator.credentials.get({ "publicKey": options })
4407     .then(function (assertion) {
4408         // Send assertion to server for verification
4409     }).catch(function (err) {
4410         // No acceptable credential or user refused consent. Handle appropriately.
4411     });
4412
4413 On the other hand, if the Relying Party script has some hints to help
4414 it narrow the list of credentials, then the sample code for performing
4415 such an authentication might look like the following. Note that this
4416 sample also demonstrates how to use the extension for transaction
4417 authorization.
4418 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4419
4420 var encoder = new TextEncoder();
4421 var acceptableCredential1 = {
4422     type: "public-key",
4423     id: encoder.encode("!!!!!!hi there!!!!!!\n")
4424 };
4425 var acceptableCredential2 = {
4426     type: "public-key",
4427     id: encoder.encode("roses are red, violets are blue\n")
4428 };
4429
4430 var options = {
4431     challenge: encoder.encode("climb a mountain"),
4432     timeout: 60000, // 1 minute
4433     allowCredentials: [acceptableCredential1, acceptableCredential2]
4434 ,
4435     extensions: { 'webauthn.txauth.simple':
4436         "Wave your hands in the air like you just don't care" }
4437 };
4438
4439 navigator.credentials.get({ "publicKey": options })
4440     .then(function (assertion) {
4441         // Send assertion to server for verification
4442     }).catch(function (err) {
4443         // No acceptable credential or user refused consent. Handle appropriately.
4444     });
4445
4446 12.4. Decommissioning
4447
4448 The following are possible situations in which decommissioning a
4449 credential might be desired. Note that all of these are handled on the
4450 server side and do not need support from the API specified here.
4451 * Possibility #1 -- user reports the credential as lost.
4452 + User goes to server.example.net, authenticates and follows a
4453 link to report a lost/stolen device.
4454 + Server returns a page showing the list of registered
4455 credentials with friendly names as configured during
4456 registration.
4457 + User selects a credential and the server deletes it from its
4458 database.
4459 + In future, the Relying Party script does not specify this
4460 credential in any list of acceptable credentials, and
4461 assertions signed by this credential are rejected.
4462 * Possibility #2 -- server deregisters the credential due to

```


4407 inactivity.
 4408 + Server deletes credential from its database during maintenance
 4409 activity.
 4410 + In the future, the Relying Party script does not specify this
 4411 credential in any list of acceptable credentials, and
 4412 assertions signed by this credential are rejected.
 4413 * Possibility #3 -- user deletes the credential from the device.
 4414 + User employs a device-specific method (e.g., device settings
 4415 UI) to delete a credential from their device.
 4416 + From this point on, this credential will not appear in any
 4417 selection prompts, and no assertions can be generated with it.
 4418 + Sometime later, the server deregisters this credential due to
 4419 inactivity.
 4420

13. Acknowledgements

4421 We thank the following for their contributions to, and thorough review
 4422 of, this specification: Richard Barnes, Dominic Battr, Domenic
 4423 Denicola, Rahul Ghosh, Brad Hill, Jing Jin, Angelo Liao, Anne van
 4424 Kesteren, Ian Kilpatrick, Giridhar Mandyam, Axel Nennker, Kimberly
 4425 Paulhamus, Adam Powers, Yaron Sheffer, Mike West, Jeffrey Yasskin,
 4426 Boris Zbarsky.
 4427

Index

Terms defined by this specification

- 4432 * aaguid, in 6.3.1
- 4433 * AAGUID, in 10.4
- 4434 * alg, in 5.3
- 4435 * allowCredentials, in 5.5
- 4436 * Assertion, in 4
- 4437 * assertion signature, in 6
- 4438 * attachment modality, in 5.4.5
- 4439 * Attestation, in 4
- 4440 * Attestation Certificate, in 4
- 4441 * attestation key pair, in 4
- 4442 * attestationObject, in 5.2.1
- 4443 * attestation object, in 6.3
- 4444
- 4445 * attestation private key, in 4
- 4446 * attestation public key, in 4
- 4447 * attestation signature, in 6
- 4448 * attestation statement, in 6.3
- 4449 * attestation statement format, in 6.3
- 4450 * attestation statement format identifier, in 8.1
- 4451 * attestation type, in 6.3
- 4452 * Attested credential data, in 6.3.1
- 4453 * attestedCredentialData, in 6.1
- 4454 * authDataExtensions, in 6.1
- 4455 * Authentication, in 4
- 4456 * Authentication Assertion, in 4
- 4457 * authentication extension, in 9
- 4458 * AuthenticationExtensions
- 4459 + definition of, in 5.6
- 4460 + (typedef), in 5.6
- 4461 * Authenticator, in 4
- 4462 * AuthenticatorAssertionResponse, in 5.2.2
- 4463 * AuthenticatorAttachment, in 5.4.5
- 4464 * authenticatorAttachment, in 5.4.4
- 4465 * AuthenticatorAttestationResponse, in 5.2.1
- 4466 * authenticatorCancel, in 6.2.3
- 4467 * authenticator data, in 6.1
- 4468 * authenticatorData, in 5.2.2
- 4469 * authenticator data claimed to have been used for the attestation,
 4470 in 6.3.2
- 4471 * authenticator data for the attestation, in 6.3.2
- 4472 * authenticator extension, in 9
- 4473 * authenticator extension input, in 9.3
- 4474 * authenticator extension output, in 9.5
- 4475

4463 inactivity.
 4464 + Server deletes credential from its database during maintenance
 4465 activity.
 4466 + In the future, the Relying Party script does not specify this
 4467 credential in any list of acceptable credentials, and
 4468 assertions signed by this credential are rejected.
 4469 * Possibility #3 -- user deletes the credential from the device.
 4470 + User employs a device-specific method (e.g., device settings
 4471 UI) to delete a credential from their device.
 4472 + From this point on, this credential will not appear in any
 4473 selection prompts, and no assertions can be generated with it.
 4474 + Sometime later, the server deregisters this credential due to
 4475 inactivity.
 4476

13. Acknowledgements

4477 We thank the following for their contributions to, and thorough review
 4478 of, this specification: Richard Barnes, Dominic Battr, Domenic
 4479 Denicola, Rahul Ghosh, Brad Hill, Jing Jin, Angelo Liao, Anne van
 4480 Kesteren, Ian Kilpatrick, Giridhar Mandyam, Axel Nennker, Kimberly
 4481 Paulhamus, Adam Powers, Yaron Sheffer, Mike West, Jeffrey Yasskin,
 4482 Boris Zbarsky.
 4483

Index

Terms defined by this specification

- 4484 * aaguid, in 6.3.1
- 4485 * AAGUID, in 10.4
- 4486 * alg, in 5.3
- 4487 * allowCredentials, in 5.5
- 4488 * Assertion, in 4
- 4489 * assertion signature, in 6
- 4490 * attachment modality, in 5.4.5
- 4491 * Attestation, in 4
- 4492 * Attestation Certificate, in 4
- 4493 * attestation key pair, in 4
- 4494 * attestationObject, in 5.2.1
- 4495 * attestation object, in 6.3
- 4496 * attestationObjectResult, in 5.1.3
- 4497 * attestation private key, in 4
- 4498 * attestation public key, in 4
- 4499 * attestation signature, in 6
- 4500 * attestation statement, in 6.3
- 4501 * attestation statement format, in 6.3
- 4502 * attestation statement format identifier, in 8.1
- 4503 * attestation type, in 6.3
- 4504 * Attested credential data, in 6.3.1
- 4505 * attestedCredentialData, in 6.1
- 4506 * authDataExtensions, in 6.1
- 4507 * Authentication, in 4
- 4508 * Authentication Assertion, in 4
- 4509 * authentication extension, in 9
- 4510 * AuthenticationExtensions
- 4511 + definition of, in 5.6
- 4512 + (typedef), in 5.6
- 4513 * Authenticator, in 4
- 4514 * AuthenticatorAssertionResponse, in 5.2.2
- 4515 * AuthenticatorAttachment, in 5.4.5
- 4516 * authenticatorAttachment, in 5.4.4
- 4517 * AuthenticatorAttestationResponse, in 5.2.1
- 4518 * authenticatorCancel, in 6.2.3
- 4519 * authenticator data, in 6.1
- 4520 * authenticatorData, in 5.2.2
- 4521 * authenticator data claimed to have been used for the attestation,
 4522 in 6.3.2
- 4523 * authenticator data for the attestation, in 6.3.2
- 4524 * authenticator extension, in 9
- 4525 * authenticator extension input, in 9.3
- 4526 * authenticator extension output, in 9.5
- 4527

4476 * Authenticator extension processing, in 9.5
 4477 * authenticatorExtensions, in 5.7.1
 4478 * authenticatorGetAssertion, in 6.2.2
 4479 * authenticatorMakeCredential, in 6.2.1
 4480 * AuthenticatorResponse, in 5.2
 4481 * authenticatorSelection, in 5.4
 4482 * AuthenticatorSelectionCriteria, in 5.4.4
 4483 * AuthenticatorSelectionList, in 10.4
 4484 * AuthenticatorTransport, in 5.7.4
 4485 * Authorization Gesture, in 4
 4486 * Base64url Encoding, in 3
 4487 * Basic Attestation, in 6.3.3
 4488 * Biometric Recognition, in 4
 4489 * ble, in 5.7.4
 4490 * CBOR, in 3
 4491 * Ceremony, in 4
 4492 * challenge
 4493 + dict-member for MakePublicKeyCredentialOptions, in 5.4
 4494 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
 4495 + dict-member for CollectedClientData, in 5.7.1
 4496 * Client, in 4
 4497 * client data, in 5.7.1
 4498 * clientDataJSON, in 5.2

 4499 * client extension, in 9
 4500 * client extension input, in 9.3
 4501 * client extension output, in 9.4
 4502 * Client extension processing, in 9.4
 4503 * clientExtensionResults, in 5.1
 4504 * clientExtensions, in 5.7.1
 4505 * Client-Side, in 4
 4506 * client-side credential private key storage, in 4
 4507 * Client-side-resident Credential Private Key, in 4
 4508 * CollectedClientData, in 5.7.1
 4509 * [[CollectFromCredentialStore]](options), in 5.1.4
 4510 * Confirming User Agent, in 4
 4511 * COSEAlgorithmIdentifier
 4512 + definition of, in 5.7.5
 4513 + (typedef), in 5.7.5
 4514 * [[Create]](options), in 5.1.3
 4515 * credentialId, in 6.3.1
 4516 * credentialIdLength, in 6.3.1
 4517 * credential key pair, in 4
 4518 * credential private key, in 4
 4519 * credentialPublicKey, in 6.3.1
 4520 * Credential Public Key, in 4
 4521 * "cross-platform", in 5.4.5
 4522 * cross-platform, in 5.4.5
 4523 * cross-platform attached, in 5.4.5
 4524 * cross-platform attachment, in 5.4.5
 4525 * DAA, in 6.3.3
 4526 * [[DiscoverFromExternalSource]](options), in 5.1.4.1
 4527 * [[discovery]], in 5.1
 4528 * displayName, in 5.4.3
 4529 * ECDAA, in 6.3.3
 4530 * ECDAA-Issuer public key, in 8.2
 4531 * Elliptic Curve based Direct Anonymous Attestation, in 6.3.3
 4532 * excludeCredentials, in 5.4
 4533 * extension identifier, in 9.1

 4534 * extensions
 4535 + dict-member for MakePublicKeyCredentialOptions, in 5.4
 4536 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
 4537 * flags, in 6.1
 4538 * hashAlgorithm, in 5.7.1
 4539 * Hash of the serialized client data, in 5.7.1
 4540 * icon, in 5.4.1
 4541 * id
 4542 + dict-member for PublicKeyCredentialRpEntity, in 5.4.2
 4543 + dict-member for PublicKeyCredentialUserEntity, in 5.4.3

4533 * Authenticator extension processing, in 9.5
 4534 * authenticatorExtensions, in 5.7.1
 4535 * authenticatorGetAssertion, in 6.2.2
 4536 * authenticatorMakeCredential, in 6.2.1
 4537 * AuthenticatorResponse, in 5.2
 4538 * authenticatorSelection, in 5.4
 4539 * AuthenticatorSelectionCriteria, in 5.4.4
 4540 * AuthenticatorSelectionList, in 10.4
 4541 * AuthenticatorTransport, in 5.7.4
 4542 * Authorization Gesture, in 4
 4543 * Base64url Encoding, in 3
 4544 * Basic Attestation, in 6.3.3
 4545 * Biometric Recognition, in 4
 4546 * ble, in 5.7.4
 4547 * CBOR, in 3
 4548 * Ceremony, in 4
 4549 * challenge
 4550 + dict-member for MakePublicKeyCredentialOptions, in 5.4
 4551 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
 4552 + dict-member for CollectedClientData, in 5.7.1
 4553 * Client, in 4
 4554 * client data, in 5.7.1
 4555 * clientDataJSON, in 5.2
 4556 * **clientDataJSONResult, in 5.1.3**
 4557 * client extension, in 9
 4558 * client extension input, in 9.3
 4559 * client extension output, in 9.4
 4560 * Client extension processing, in 9.4
 4561 * clientExtensionResults, in 5.1
 4562 * clientExtensions, in 5.7.1
 4563 * Client-Side, in 4
 4564 * client-side credential private key storage, in 4
 4565 * Client-side-resident Credential Private Key, in 4
 4566 * CollectedClientData, in 5.7.1
 4567 * [[CollectFromCredentialStore]](options), in 5.1.4
 4568 * Confirming User Agent, in 4
 4569 * COSEAlgorithmIdentifier
 4570 + definition of, in 5.7.5
 4571 + (typedef), in 5.7.5
 4572 * [[Create]](options), in 5.1.3
 4573 * credentialId, in 6.3.1
 4574 * credentialIdLength, in 6.3.1
 4575 * credential key pair, in 4
 4576 * credential private key, in 4
 4577 * credentialPublicKey, in 6.3.1
 4578 * Credential Public Key, in 4
 4579 * "cross-platform", in 5.4.5
 4580 * cross-platform, in 5.4.5
 4581 * cross-platform attached, in 5.4.5
 4582 * cross-platform attachment, in 5.4.5
 4583 * DAA, in 6.3.3
 4584 * [[DiscoverFromExternalSource]](options), in 5.1.4.1
 4585 * [[discovery]], in 5.1
 4586 * displayName, in 5.4.3
 4587 * ECDAA, in 6.3.3
 4588 * ECDAA-Issuer public key, in 8.2
 4589 * Elliptic Curve based Direct Anonymous Attestation, in 6.3.3
 4590 * excludeCredentials, in 5.4
 4591 * extension identifier, in 9.1
 4592 * **extensionOutputsMap, in 5.1.3**
 4593 * extensions
 4594 + dict-member for MakePublicKeyCredentialOptions, in 5.4
 4595 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
 4596 * flags, in 6.1
 4597 * hashAlgorithm, in 5.7.1
 4598 * Hash of the serialized client data, in 5.7.1
 4599 * icon, in 5.4.1
 4600 * id
 4601 + dict-member for PublicKeyCredentialRpEntity, in 5.4.2
 4602 + dict-member for PublicKeyCredentialUserEntity, in 5.4.3

4544 + dict-member for PublicKeyCredentialDescriptor, in 5.7.3
 4545 * [[identifier]], in 5.1
 4546 * identifier of the ECDAAs-Issuer public key, in 8.2
 4547 * isPlatformAuthenticatorAvailable(), in 5.1.6
 4548 * JSON-serialized client data, in 5.7.1
 4549 * MakePublicKeyCredentialOptions, in 5.4
 4550 * name, in 5.4.1
 4551 * nfc, in 5.7.4
 4552 * origin, in 5.7.1
 4553 * platform, in 5.4.5
 4554 * "platform", in 5.4.5
 4555 * platform attachment, in 5.4.5
 4556 * platform authenticators, in 5.4.5
 4557 * Privacy CA, in 6.3.3
 4558 * pubKeyCredParams, in 5.4
 4559 * publicKey
 4560 + dict-member for CredentialCreationOptions, in 5.1.1
 4561 + dict-member for CredentialRequestOptions, in 5.1.2
 4562 * public-key, in 5.7.2
 4563 * Public Key Credential, in 4
 4564 * PublicKeyCredential, in 5.1
 4565 * PublicKeyCredentialDescriptor, in 5.7.3
 4566 * PublicKeyCredentialEntity, in 5.4.1
 4567 * PublicKeyCredentialParameters, in 5.3
 4568 * PublicKeyCredentialRequestOptions, in 5.5
 4569 * PublicKeyCredentialRpEntity, in 5.4.2
 4570 * PublicKeyCredentialType, in 5.7.2
 4571 * PublicKeyCredentialUserEntity, in 5.4.3
 4572 * Rate Limiting, in 4
 4573 * rawId, in 5.1
 4574 * Registration, in 4
 4575 * registration extension, in 9
 4576 * Relying Party, in 4
 4577 * Relying Party Identifier, in 4
 4578 * requireResidentKey, in 5.4.4
 4579 * requireUserVerification, in 5.4.4
 4580 * response, in 5.1
 4581 * roaming authenticators, in 5.4.5
 4582 * rp, in 5.4
 4583 * rpId, in 5.5
 4584 * RP ID, in 4
 4585 * rpIdHash, in 6.1
 4586 * Self Attestation, in 6.3.3
 4587 * signature, in 5.2.2
 4588 * Signature Counter, in 6.1.1
 4589 * signCount, in 6.1
 4590 * Signing procedure, in 6.3.2
 4591 * [[Store]](credential), in 5.1.5
 4592 * Test of User Presence, in 4
 4593 * timeout
 4594 + dict-member for MakePublicKeyCredentialOptions, in 5.4
 4595 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
 4596 * tokenBindingId, in 5.7.1
 4597 * transports, in 5.7.3
 4598 * [[type]], in 5.1
 4599 * type
 4600 + dict-member for PublicKeyCredentialParameters, in 5.3
 4601 + dict-member for PublicKeyCredentialDescriptor, in 5.7.3
 4602 * UP, in 4
 4603 * usb, in 5.7.4
 4604 * user, in 5.4
 4605 * User Consent, in 4
 4606 * userHandle, in 5.2.2
 4607 * User Handle, in 4
 4608 * User Present, in 4
 4609 * User Verification, in 4
 4610 * User Verified, in 4
 4611 * UV, in 4
 4612 * Verification procedures, in 6.3.2
 4613 * Web Authentication API, in 5

4603 + dict-member for PublicKeyCredentialDescriptor, in 5.7.3
 4604 * [[identifier]], in 5.1
 4605 * identifier of the ECDAAs-Issuer public key, in 8.2
 4606 * isPlatformAuthenticatorAvailable(), in 5.1.6
 4607 * JSON-serialized client data, in 5.7.1
 4608 * MakePublicKeyCredentialOptions, in 5.4
 4609 * name, in 5.4.1
 4610 * nfc, in 5.7.4
 4611 * origin, in 5.7.1
 4612 * platform, in 5.4.5
 4613 * "platform", in 5.4.5
 4614 * platform attachment, in 5.4.5
 4615 * platform authenticators, in 5.4.5
 4616 * Privacy CA, in 6.3.3
 4617 * pubKeyCredParams, in 5.4
 4618 * publicKey
 4619 + dict-member for CredentialCreationOptions, in 5.1.1
 4620 + dict-member for CredentialRequestOptions, in 5.1.2
 4621 * public-key, in 5.7.2
 4622 * Public Key Credential, in 4
 4623 * PublicKeyCredential, in 5.1
 4624 * PublicKeyCredentialDescriptor, in 5.7.3
 4625 * PublicKeyCredentialEntity, in 5.4.1
 4626 * PublicKeyCredentialParameters, in 5.3
 4627 * PublicKeyCredentialRequestOptions, in 5.5
 4628 * PublicKeyCredentialRpEntity, in 5.4.2
 4629 * PublicKeyCredentialType, in 5.7.2
 4630 * PublicKeyCredentialUserEntity, in 5.4.3
 4631 * Rate Limiting, in 4
 4632 * rawId, in 5.1
 4633 * Registration, in 4
 4634 * registration extension, in 9
 4635 * Relying Party, in 4
 4636 * Relying Party Identifier, in 4
 4637 * requireResidentKey, in 5.4.4
 4638 * requireUserVerification, in 5.4.4
 4639 * response, in 5.1
 4640 * roaming authenticators, in 5.4.5
 4641 * rp, in 5.4
 4642 * rpId, in 5.5
 4643 * RP ID, in 4
 4644 * rpIdHash, in 6.1
 4645 * Self Attestation, in 6.3.3
 4646 * signature, in 5.2.2
 4647 * Signature Counter, in 6.1.1
 4648 * signCount, in 6.1
 4649 * Signing procedure, in 6.3.2
 4650 * [[Store]](credential), in 5.1.5
 4651 * Test of User Presence, in 4
 4652 * timeout
 4653 + dict-member for MakePublicKeyCredentialOptions, in 5.4
 4654 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
 4655 * tokenBindingId, in 5.7.1
 4656 * transports, in 5.7.3
 4657 * [[type]], in 5.1
 4658 * type
 4659 + dict-member for PublicKeyCredentialParameters, in 5.3
 4660 + dict-member for PublicKeyCredentialDescriptor, in 5.7.3
 4661 * UP, in 4
 4662 * usb, in 5.7.4
 4663 * user, in 5.4
 4664 * User Consent, in 4
 4665 * userHandle, in 5.2.2
 4666 * User Handle, in 4
 4667 * User Present, in 4
 4668 * User Verification, in 4
 4669 * User Verified, in 4
 4670 * UV, in 4
 4671 * Verification procedures, in 6.3.2
 4672 * Web Authentication API, in 5

```

4614 * WebAuthn Client, in 4
4615
4616 Terms defined by reference
4617
4618 * [CREDENTIAL-MANAGEMENT-1] defines the following terms:
4619   + Credential
4620   + CredentialCreationOptions
4621   + CredentialRequestOptions
4622   + CredentialsContainer
4623   + [[CollectFromCredentialStore]](options)
4624   + [[Store]](credential)
4625   + [[discovery]]
4626   + [[type]]
4627   + create()
4628   + credential
4629   + credential source
4630   + get()
4631   + id
4632   + remote
4633   + store()
4634   + type
4635   + user mediation
4636 * [ECMAScript] defines the following terms:
4637   + %arraybuffer%
4638   + internal slot
4639   + stringify
4640 * [ENCODING] defines the following terms:
4641   + utf-8 encode
4642 * [HTML] defines the following terms:
4643   + ascii serialization of an origin
4644   + dom manipulation task source
4645   + effective domain
4646   + global object
4647   + in parallel
4648   + is a registrable domain suffix of or is equal to
4649   + is not a registrable domain suffix of and is not equal to
4650   + origin
4651   + promise
4652   + relevant settings object
4653   + task
4654   + task source
4655 * [HTML52] defines the following terms:
4656   + document.domain
4657   + opaque origin
4658   + origin
4659 * [INFRA] defines the following terms:
4660   + append (for list)
4661   + append (for set)
4662   + byte sequence
4663   + continue
4664   + for each (for list)
4665   + for each (for map)
4666   + is empty
4667   + is not empty
4668   + item
4669
4670   + list
4671   + map
4672
4673   + ordered set
4674   + remove
4675   + set
4676
4677 * [mixed-content] defines the following terms:
4678   + a priori authenticated url

```

```

4673 * WebAuthn Client, in 4
4674
4675 Terms defined by reference
4676
4677 * [CREDENTIAL-MANAGEMENT-1] defines the following terms:
4678   + Credential
4679   + CredentialCreationOptions
4680   + CredentialRequestOptions
4681   + CredentialsContainer
4682   + [[CollectFromCredentialStore]](options)
4683   + [[Store]](credential)
4684   + [[discovery]]
4685   + [[type]]
4686   + create()
4687   + credential
4688   + credential source
4689   + get()
4690   + id
4691   + remote
4692   + store()
4693   + type
4694   + user mediation
4695 * [ECMAScript] defines the following terms:
4696   + %arraybuffer%
4697   + internal slot
4698   + stringify
4699 * [ENCODING] defines the following terms:
4700   + utf-8 encode
4701 * [HTML] defines the following terms:
4702   + ascii serialization of an origin
4703   + dom manipulation task source
4704   + effective domain
4705   + global object
4706   + in parallel
4707   + is a registrable domain suffix of or is equal to
4708   + is not a registrable domain suffix of and is not equal to
4709   + origin
4710   + promise
4711   + relevant global object
4712   + relevant settings object
4713   + task
4714   + task source
4715 * [HTML52] defines the following terms:
4716   + document.domain
4717   + opaque origin
4718   + origin
4719 * [INFRA] defines the following terms:
4720   + append (for list)
4721   + append (for set)
4722   + byte sequence
4723   + continue
4724   + entry
4725   + for each (for list)
4726   + for each (for map)
4727   + is empty
4728   + is not empty
4729   + item (for list)
4730   + item (for struct)
4731   + key
4732   + list
4733   + map
4734   + ordered map
4735   + ordered set
4736   + remove
4737   + set
4738   + struct
4739   + value
4740   + while
4741 * [mixed-content] defines the following terms:
4742   + a priori authenticated url

```

4676 * [secure-contexts] defines the following terms:
 4677 + secure context
 4678 * [TokenBinding] defines the following terms:
 4679 + token binding
 4680 + token binding id
 4681 * [URL] defines the following terms:
 4682 + domain
 4683 + empty host
 4684 + host
 4685 + ipv4 address
 4686 + ipv6 address
 4687 + opaque host
 4688 + url serializer
 4689 + valid domain
 4690 + valid domain string
 4691 * [WebCryptoAPI] defines the following terms:
 4692 + recognized algorithm name
 4693 * [WebIDL] defines the following terms:
 4694 + ArrayBuffer
 4695 + BufferSource
 4696 + ConstraintError
 4697 + DOMException
 4698 + DOMString
 4699 + Exposed
 4700 + NotAllowedError
 4701 + NotFoundError
 4702 + NotSupportedError
 4703 + Promise
 4704 + SameObject
 4705 + SecureContext
 4706 + SecurityError
 4707 + TypeError
 4708 + USVString
 4709 + UnknownError
 4710 + boolean
 4711 + interface object
 4712 + long
 4713 + present

4714 + simple exception
 4715 + unsigned long

References

Normative References

4721 [CDDL]
 4722 C. Vigano; H. Birkholz. CBOR data definition language (CDDL): a
 4723 notational convention to express CBOR data structures. 21
 4724 September 2016. Internet Draft (work in progress). URL:
 4725 https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl
 4726

4727 [CREDENTIAL-MANAGEMENT-1]
 4728 Mike West. Credential Management Level 1. 4 August 2017. WD.
 4729 URL: https://www.w3.org/TR/credential-management-1/
 4730

4731 [DOM4]
 4732 Anne van Kesteren. DOM Standard. Living Standard. URL:
 4733 https://dom.spec.whatwg.org/
 4734

4735 [ECMAScript]
 4736 ECMAScript Language Specification. URL:
 4737 https://tc39.github.io/ecma262/
 4738

4739 [ENCODING]
 4740 Anne van Kesteren. Encoding Standard. Living Standard. URL:
 4741 https://encoding.spec.whatwg.org/
 4742

4743 [FIDO-CTAP]
 4744 R. Lindemann; et al. FIDO 2.0: Client to Authenticator Protocol.

4743 * [secure-contexts] defines the following terms:
 4744 + secure context
 4745 * [TokenBinding] defines the following terms:
 4746 + token binding
 4747 + token binding id
 4748 * [URL] defines the following terms:
 4749 + domain
 4750 + empty host
 4751 + host
 4752 + ipv4 address
 4753 + ipv6 address
 4754 + opaque host
 4755 + url serializer
 4756 + valid domain
 4757 + valid domain string
 4758 * [WebCryptoAPI] defines the following terms:
 4759 + recognized algorithm name
 4760 * [WebIDL] defines the following terms:
 4761 + ArrayBuffer
 4762 + BufferSource
 4763 + ConstraintError
 4764 + DOMException
 4765 + DOMString
 4766 + Exposed
 4767 + NotAllowedError
 4768 + NotFoundError
 4769 + NotSupportedError
 4770 + Promise
 4771 + SameObject
 4772 + SecureContext
 4773 + SecurityError
 4774 + TypeError
 4775 + USVString
 4776 + UnknownError
 4777 + boolean
 4778 + interface object
 4779 + long
 4780 + present
 4781 + record type
 4782 + simple exception
 4783 + unsigned long

References

Normative References

4791 [CDDL]
 4792 C. Vigano; H. Birkholz. CBOR data definition language (CDDL): a
 4793 notational convention to express CBOR data structures. 21
 4794 September 2016. Internet Draft (work in progress). URL:
 4795 https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl
 4796

4797 [CREDENTIAL-MANAGEMENT-1]
 4798 Mike West. Credential Management Level 1. 4 August 2017. WD.
 4799 URL: https://www.w3.org/TR/credential-management-1/
 4800

4801 [DOM4]
 4802 Anne van Kesteren. DOM Standard. Living Standard. URL:
 4803 https://dom.spec.whatwg.org/
 4804

4805 [ECMAScript]
 4806 ECMAScript Language Specification. URL:
 4807 https://tc39.github.io/ecma262/
 4808

4809 [ENCODING]
 4810 Anne van Kesteren. Encoding Standard. Living Standard. URL:
 4811 https://encoding.spec.whatwg.org/
 4812

4813 [FIDO-CTAP]
 4814 R. Lindemann; et al. FIDO 2.0: Client to Authenticator Protocol.

4745 FIDO Alliance Review Draft. URL:
4746 <https://fidoalliance.org/specs/fido-v2.0-rd-20170927/fido-client>
4747 -to-authenticator-protocol-v2.0-rd-20170927.html
4748
4749 [FIDO-U2F-Message-Formats]
4750 D. Balfanz; J. Ehrensvar; J. Lang. FIDO U2F Raw Message
4751 Formats. FIDO Alliance Implementation Draft. URL:
4752 <https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2>
4753 f-raw-message-formats-v1.1-id-20160915.html
4754
4755 [FIDOEcdaaAlgorithm]
4756 R. Lindemann; et al. FIDO ECDA A Algorithm. FIDO Alliance
4757 Implementation Draft. URL:
4758 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ec>
4759 daa-algorithm-v1.1-id-20170202.html
4760
4761 [FIDOReg]
4762 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO UAF Registry of
4763 Predefined Values. FIDO Alliance Proposed Standard. URL:
4764 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua>
4765 f-reg-v1.0-ps-20141208.html
4766
4767 [HTML]
4768 Anne van Kesteren; et al. HTML Standard. Living Standard. URL:
4769 <https://html.spec.whatwg.org/multipage/>
4770
4771 [HTML52]
4772 Steve Faulkner; et al. HTML 5.2. 8 August 2017. CR. URL:
4773 <https://www.w3.org/TR/html52/>
4774
4775 [IANA-COSE-ALGS-REG]
4776 IANA CBOR Object Signing and Encryption (COSE) Algorithms
4777 Registry. URL:
4778 <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>
4779
4780 [INFRA]
4781 Anne van Kesteren; Domenic Denicola. Infra Standard. Living
4782 Standard. URL: <https://infra.spec.whatwg.org/>
4783
4784 [MIXED-CONTENT]
4785 Mike West. Mixed Content. 2 August 2016. CR. URL:
4786 <https://www.w3.org/TR/mixed-content/>
4787
4788 [RFC2119]
4789 S. Bradner. Key words for use in RFCs to Indicate Requirement
4790 Levels. March 1997. Best Current Practice. URL:
4791 <https://tools.ietf.org/html/rfc2119>
4792
4793 [RFC4648]
4794 S. Josefsson. The Base16, Base32, and Base64 Data Encodings.
4795 October 2006. Proposed Standard. URL:
4796 <https://tools.ietf.org/html/rfc4648>
4797
4798 [RFC5234]
4799 D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax
4800 Specifications: ABNF. January 2008. Internet Standard. URL:
4801 <https://tools.ietf.org/html/rfc5234>
4802
4803 [RFC5890]
4804 J. Klensin. Internationalized Domain Names for Applications
4805 (IDNA): Definitions and Document Framework. August 2010.
4806 Proposed Standard. URL: <https://tools.ietf.org/html/rfc5890>
4807
4808 [RFC7049]
4809 C. Bormann; P. Hoffman. Concise Binary Object Representation
4810 (CBOR). October 2013. Proposed Standard. URL:
4811 <https://tools.ietf.org/html/rfc7049>
4812
4813 [RFC8152]
4814 J. Schaad. CBOR Object Signing and Encryption (COSE). July 2017.

4813 FIDO Alliance Review Draft. URL:
4814 <https://fidoalliance.org/specs/fido-v2.0-rd-20170927/fido-client>
4815 -to-authenticator-protocol-v2.0-rd-20170927.html
4816
4817 [FIDO-U2F-Message-Formats]
4818 D. Balfanz; J. Ehrensvar; J. Lang. FIDO U2F Raw Message
4819 Formats. FIDO Alliance Implementation Draft. URL:
4820 <https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2>
4821 f-raw-message-formats-v1.1-id-20160915.html
4822
4823 [FIDOEcdaaAlgorithm]
4824 R. Lindemann; et al. FIDO ECDA A Algorithm. FIDO Alliance
4825 Implementation Draft. URL:
4826 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ec>
4827 daa-algorithm-v1.1-id-20170202.html
4828
4829 [FIDOReg]
4830 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO UAF Registry of
4831 Predefined Values. FIDO Alliance Proposed Standard. URL:
4832 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua>
4833 f-reg-v1.0-ps-20141208.html
4834
4835 [HTML]
4836 Anne van Kesteren; et al. HTML Standard. Living Standard. URL:
4837 <https://html.spec.whatwg.org/multipage/>
4838
4839 [HTML52]
4840 Steve Faulkner; et al. HTML 5.2. 8 August 2017. CR. URL:
4841 <https://www.w3.org/TR/html52/>
4842
4843 [IANA-COSE-ALGS-REG]
4844 IANA CBOR Object Signing and Encryption (COSE) Algorithms
4845 Registry. URL:
4846 <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>
4847
4848 [INFRA]
4849 Anne van Kesteren; Domenic Denicola. Infra Standard. Living
4850 Standard. URL: <https://infra.spec.whatwg.org/>
4851
4852 [MIXED-CONTENT]
4853 Mike West. Mixed Content. 2 August 2016. CR. URL:
4854 <https://www.w3.org/TR/mixed-content/>
4855
4856 [RFC2119]
4857 S. Bradner. Key words for use in RFCs to Indicate Requirement
4858 Levels. March 1997. Best Current Practice. URL:
4859 <https://tools.ietf.org/html/rfc2119>
4860
4861 [RFC4648]
4862 S. Josefsson. The Base16, Base32, and Base64 Data Encodings.
4863 October 2006. Proposed Standard. URL:
4864 <https://tools.ietf.org/html/rfc4648>
4865
4866 [RFC5234]
4867 D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax
4868 Specifications: ABNF. January 2008. Internet Standard. URL:
4869 <https://tools.ietf.org/html/rfc5234>
4870
4871 [RFC5890]
4872 J. Klensin. Internationalized Domain Names for Applications
4873 (IDNA): Definitions and Document Framework. August 2010.
4874 Proposed Standard. URL: <https://tools.ietf.org/html/rfc5890>
4875
4876 [RFC7049]
4877 C. Bormann; P. Hoffman. Concise Binary Object Representation
4878 (CBOR). October 2013. Proposed Standard. URL:
4879 <https://tools.ietf.org/html/rfc7049>
4880
4881 [RFC8152]
4882 J. Schaad. CBOR Object Signing and Encryption (COSE). July 2017.

4815 Proposed Standard. URL: <https://tools.ietf.org/html/rfc8152>
4816
4817 [SEC1]
4818 SEC1: Elliptic Curve Cryptography, Version 2.0. URL:
4819 <http://www.secg.org/sec1-v2.pdf>
4820
4821 [SECURE-CONTEXTS]
4822 Mike West. Secure Contexts. 15 September 2016. CR. URL:
4823 <https://www.w3.org/TR/secure-contexts/>
4824
4825 [TokenBinding]
4826 A. Popov; et al. The Token Binding Protocol Version 1.0.
4827 February 16, 2017. Internet-Draft. URL:
4828 <https://tools.ietf.org/html/draft-ietf-tokbind-protocol>
4829
4830 [URL]
4831 Anne van Kesteren. URL Standard. Living Standard. URL:
4832 <https://url.spec.whatwg.org/>
4833
4834 [WebAuthn-Registries]
4835 Jeff Hodges; Giridhar Mandyam; Michael B. Jones. Registries for
4836 Web Authentication (WebAuthn). March 2017. Active
4837 Internet-Draft. URL:
4838 <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>
4839
4840 [WebCryptoAPI]
4841 Mark Watson. Web Cryptography API. 26 January 2017. REC. URL:
4842 <https://www.w3.org/TR/WebCryptoAPI/>
4843
4844 [WebIDL]
4845 Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15
4846 December 2016. ED. URL: <https://heycam.github.io/webidl/>
4847
4848 [WebIDL-1]
4849 Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL:
4850 <https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/>
4851
4852 Informative References
4853
4854 [Ceremony]
4855 Carl Ellison. Ceremony Design and Analysis. 2007. URL:
4856 <https://eprint.iacr.org/2007/399.pdf>
4857
4858 [FIDO-APPID]
4859 D. Balfanz; et al. FIDO AppID and Facets. FIDO Alliance Review
4860 Draft. URL:
4861 <https://fidoalliance.org/specs/fido-uaf-v1.1-rd-20161005/fido-appid-and-facets-v1.1-rd-20161005.html>
4862
4863 [FIDOMetadataService]
4864 R. Lindemann; B. Hill; D. Baghdasaryan. FIDO Metadata Service
4865 v1.0. FIDO Alliance Proposed Standard. URL:
4866 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua-f-metadata-service-v1.0-ps-20141208.html>
4867
4868 [FIDOSecRef]
4869 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference.
4870 FIDO Alliance Proposed Standard. URL:
4871 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-security-ref-v1.0-ps-20141208.html>
4872
4873 [GeoJSON]
4874 The GeoJSON Format Specification. URL:
4875 <http://geojson.org/geojson-spec.html>
4876
4877 [ISOBiometricVocabulary]
4878 ISO/IEC JTC1/SC37. Information technology -- Vocabulary --
4879 Biometrics. 15 December 2012. International Standard: ISO/IEC
4880
4881
4882
4883
4884

4883 Proposed Standard. URL: <https://tools.ietf.org/html/rfc8152>
4884
4885 [SEC1]
4886 SEC1: Elliptic Curve Cryptography, Version 2.0. URL:
4887 <http://www.secg.org/sec1-v2.pdf>
4888
4889 [SECURE-CONTEXTS]
4890 Mike West. Secure Contexts. 15 September 2016. CR. URL:
4891 <https://www.w3.org/TR/secure-contexts/>
4892
4893 [TokenBinding]
4894 A. Popov; et al. The Token Binding Protocol Version 1.0.
4895 February 16, 2017. Internet-Draft. URL:
4896 <https://tools.ietf.org/html/draft-ietf-tokbind-protocol>
4897
4898 [URL]
4899 Anne van Kesteren. URL Standard. Living Standard. URL:
4900 <https://url.spec.whatwg.org/>
4901
4902 [WebAuthn-Registries]
4903 Jeff Hodges; Giridhar Mandyam; Michael B. Jones. Registries for
4904 Web Authentication (WebAuthn). March 2017. Active
4905 Internet-Draft. URL:
4906 <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>
4907
4908 [WebCryptoAPI]
4909 Mark Watson. Web Cryptography API. 26 January 2017. REC. URL:
4910 <https://www.w3.org/TR/WebCryptoAPI/>
4911
4912 [WebIDL]
4913 Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15
4914 December 2016. ED. URL: <https://heycam.github.io/webidl/>
4915
4916 [WebIDL-1]
4917 Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL:
4918 <https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/>
4919
4920 Informative References
4921
4922 [Ceremony]
4923 Carl Ellison. Ceremony Design and Analysis. 2007. URL:
4924 <https://eprint.iacr.org/2007/399.pdf>
4925
4926 [FIDO-APPID]
4927 D. Balfanz; et al. FIDO AppID and Facets. FIDO Alliance Review
4928 Draft. URL:
4929 <https://fidoalliance.org/specs/fido-uaf-v1.1-rd-20161005/fido-appid-and-facets-v1.1-rd-20161005.html>
4930
4931 [FIDOMetadataService]
4932 R. Lindemann; B. Hill; D. Baghdasaryan. FIDO Metadata Service
4933 v1.0. FIDO Alliance Proposed Standard. URL:
4934 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua-f-metadata-service-v1.0-ps-20141208.html>
4935
4936 [FIDOSecRef]
4937 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference.
4938 FIDO Alliance Proposed Standard. URL:
4939 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-security-ref-v1.0-ps-20141208.html>
4940
4941 [GeoJSON]
4942 The GeoJSON Format Specification. URL:
4943 <http://geojson.org/geojson-spec.html>
4944
4945 [ISOBiometricVocabulary]
4946 ISO/IEC JTC1/SC37. Information technology -- Vocabulary --
4947 Biometrics. 15 December 2012. International Standard: ISO/IEC
4948
4949
4950
4951
4952

4885 2382-37:2012(E) First Edition. URL:
4886 http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194
4887 _ISOIEC_2382-37_2012.zip
4888
4889 [RFC4949]
4890 R. Shirey. Internet Security Glossary, Version 2. August 2007.
4891 Informational. URL: https://tools.ietf.org/html/rfc4949
4892
4893 [RFC5280]
4894 D. Cooper; et al. Internet X.509 Public Key Infrastructure
4895 Certificate and Certificate Revocation List (CRL) Profile. May
4896 2008. Proposed Standard. URL:
4897 https://tools.ietf.org/html/rfc5280
4898
4899 [RFC6265]
4900 A. Barth. HTTP State Management Mechanism. April 2011. Proposed
4901 Standard. URL: https://tools.ietf.org/html/rfc6265
4902
4903 [RFC6454]
4904 A. Barth. The Web Origin Concept. December 2011. Proposed
4905 Standard. URL: https://tools.ietf.org/html/rfc6454
4906
4907 [RFC7515]
4908 M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May
4909 2015. Proposed Standard. URL:
4910 https://tools.ietf.org/html/rfc7515
4911
4912 [RFC8017]
4913 K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography
4914 Specifications Version 2.2. November 2016. Informational. URL:
4915 https://tools.ietf.org/html/rfc8017
4916
4917 [TPMv2-EK-Profile]
4918 TCG EK Credential Profile for TPM Family 2.0. URL:
4919 http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti
4920 al_Profile_EK_V2.0_R14_published.pdf
4921
4922 [TPMv2-Part1]
4923 Trusted Platform Module Library, Part 1: Architecture. URL:
4924 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4925 2.0-Part-1-Architecture-01.38.pdf
4926
4927 [TPMv2-Part2]
4928 Trusted Platform Module Library, Part 2: Structures. URL:
4929 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4930 2.0-Part-2-Structures-01.38.pdf
4931
4932 [TPMv2-Part3]
4933 Trusted Platform Module Library, Part 3: Commands. URL:
4934 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4935 2.0-Part-3-Commands-01.38.pdf
4936
4937 [UAFProtocol]
4938 R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO
4939 Alliance Proposed Standard. URL:
4940 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua
4941 f-protocol-v1.0-ps-20141208.html
4942
4943 IDL Index
4944
4945 [SecureContext, Exposed=Window]
4946 interface PublicKeyCredential : Credential {
4947 [SameObject] readonly attribute ArrayBuffer rawId;
4948 [SameObject] readonly attribute AuthenticatorResponse response;
4949 [SameObject] readonly attribute AuthenticationExtensions clientExtensionResu
4950 lts;
4951 };
4952
4953 partial dictionary CredentialCreationOptions {
4954 MakePublicKeyCredentialOptions publicKey;

4953 2382-37:2012(E) First Edition. URL:
4954 http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194
4955 _ISOIEC_2382-37_2012.zip
4956
4957 [RFC4949]
4958 R. Shirey. Internet Security Glossary, Version 2. August 2007.
4959 Informational. URL: https://tools.ietf.org/html/rfc4949
4960
4961 [RFC5280]
4962 D. Cooper; et al. Internet X.509 Public Key Infrastructure
4963 Certificate and Certificate Revocation List (CRL) Profile. May
4964 2008. Proposed Standard. URL:
4965 https://tools.ietf.org/html/rfc5280
4966
4967 [RFC6265]
4968 A. Barth. HTTP State Management Mechanism. April 2011. Proposed
4969 Standard. URL: https://tools.ietf.org/html/rfc6265
4970
4971 [RFC6454]
4972 A. Barth. The Web Origin Concept. December 2011. Proposed
4973 Standard. URL: https://tools.ietf.org/html/rfc6454
4974
4975 [RFC7515]
4976 M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May
4977 2015. Proposed Standard. URL:
4978 https://tools.ietf.org/html/rfc7515
4979
4980 [RFC8017]
4981 K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography
4982 Specifications Version 2.2. November 2016. Informational. URL:
4983 https://tools.ietf.org/html/rfc8017
4984
4985 [TPMv2-EK-Profile]
4986 TCG EK Credential Profile for TPM Family 2.0. URL:
4987 http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti
4988 al_Profile_EK_V2.0_R14_published.pdf
4989
4990 [TPMv2-Part1]
4991 Trusted Platform Module Library, Part 1: Architecture. URL:
4992 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4993 2.0-Part-1-Architecture-01.38.pdf
4994
4995 [TPMv2-Part2]
4996 Trusted Platform Module Library, Part 2: Structures. URL:
4997 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4998 2.0-Part-2-Structures-01.38.pdf
4999
5000 [TPMv2-Part3]
5001 Trusted Platform Module Library, Part 3: Commands. URL:
5002 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
5003 2.0-Part-3-Commands-01.38.pdf
5004
5005 [UAFProtocol]
5006 R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO
5007 Alliance Proposed Standard. URL:
5008 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua
5009 f-protocol-v1.0-ps-20141208.html
5010
5011 IDL Index
5012
5013 [SecureContext, Exposed=Window]
5014 interface PublicKeyCredential : Credential {
5015 [SameObject] readonly attribute ArrayBuffer rawId;
5016 [SameObject] readonly attribute AuthenticatorResponse response;
5017 [SameObject] readonly attribute AuthenticationExtensions clientExtensionResu
5018 lts;
5019 };
5020
5021 partial dictionary CredentialCreationOptions {
5022 MakePublicKeyCredentialOptions publicKey;


```

4955 };
4956
4957 partial dictionary CredentialRequestOptions {
4958     PublicKeyCredentialRequestOptions publicKey;
4959 };
4960
4961 partial interface PublicKeyCredential {
4962     static Promise < boolean > isPlatformAuthenticatorAvailable();
4963 };
4964
4965 [SecureContext, Exposed=Window]
4966 interface AuthenticatorResponse {
4967     [SameObject] readonly attribute ArrayBuffer clientDataJSON;
4968 };
4969
4970 [SecureContext, Exposed=Window]
4971 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
4972     [SameObject] readonly attribute ArrayBuffer attestationObject;
4973 };
4974
4975 [SecureContext, Exposed=Window]
4976 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
4977     [SameObject] readonly attribute ArrayBuffer authenticatorData;
4978     [SameObject] readonly attribute ArrayBuffer signature;
4979     [SameObject] readonly attribute ArrayBuffer userHandle;
4980 };
4981
4982 dictionary PublicKeyCredentialParameters {
4983     required PublicKeyCredentialType type;
4984     required COSEAlgorithmIdentifier alg;
4985 };
4986
4987 dictionary MakePublicKeyCredentialOptions {
4988     required PublicKeyCredentialRpEntity rp;
4989     required PublicKeyCredentialUserEntity user;
4990
4991     required BufferSource challenge;
4992     required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
4993
4994     unsigned long timeout;
4995     sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
4996     AuthenticatorSelectionCriteria authenticatorSelection;
4997     AuthenticationExtensions extensions;
4998 };
4999
5000 dictionary PublicKeyCredentialEntity {
5001     DOMString name;
5002     USVString icon;
5003 };
5004
5005 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
5006     DOMString id;
5007 };
5008
5009 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
5010     BufferSource id;
5011     DOMString displayName;
5012 };
5013
5014 dictionary AuthenticatorSelectionCriteria {
5015     AuthenticatorAttachment authenticatorAttachment;
5016     boolean requireResidentKey = false;
5017     boolean requireUserVerification = false;
5018 };
5019
5020 enum AuthenticatorAttachment {
5021     "platform", // Platform attachment
5022     "cross-platform" // Cross-platform attachment
5023 };
5024

```

```

5023 };
5024
5025 partial dictionary CredentialRequestOptions {
5026     PublicKeyCredentialRequestOptions publicKey;
5027 };
5028
5029 partial interface PublicKeyCredential {
5030     static Promise < boolean > isPlatformAuthenticatorAvailable();
5031 };
5032
5033 [SecureContext, Exposed=Window]
5034 interface AuthenticatorResponse {
5035     [SameObject] readonly attribute ArrayBuffer clientDataJSON;
5036 };
5037
5038 [SecureContext, Exposed=Window]
5039 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
5040     [SameObject] readonly attribute ArrayBuffer attestationObject;
5041 };
5042
5043 [SecureContext, Exposed=Window]
5044 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
5045     [SameObject] readonly attribute ArrayBuffer authenticatorData;
5046     [SameObject] readonly attribute ArrayBuffer signature;
5047     [SameObject] readonly attribute ArrayBuffer userHandle;
5048 };
5049
5050 dictionary PublicKeyCredentialParameters {
5051     required PublicKeyCredentialType type;
5052     required COSEAlgorithmIdentifier alg;
5053 };
5054
5055 dictionary MakePublicKeyCredentialOptions {
5056     required PublicKeyCredentialRpEntity rp;
5057     required PublicKeyCredentialUserEntity user;
5058
5059     required BufferSource challenge;
5060     required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
5061
5062     unsigned long timeout;
5063     sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
5064     AuthenticatorSelectionCriteria authenticatorSelection;
5065     AuthenticationExtensions extensions;
5066 };
5067
5068 dictionary PublicKeyCredentialEntity {
5069     DOMString name;
5070     USVString icon;
5071 };
5072
5073 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
5074     DOMString id;
5075 };
5076
5077 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
5078     BufferSource id;
5079     DOMString displayName;
5080 };
5081
5082 dictionary AuthenticatorSelectionCriteria {
5083     AuthenticatorAttachment authenticatorAttachment;
5084     boolean requireResidentKey = false;
5085     boolean requireUserVerification = false;
5086 };
5087
5088 enum AuthenticatorAttachment {
5089     "platform", // Platform attachment
5090     "cross-platform" // Cross-platform attachment
5091 };
5092

```

```

5025 dictionary PublicKeyCredentialRequestOptions {
5026     required BufferSource challenge;
5027     unsigned long timeout;
5028     USVString rpId;
5029     sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
5030     AuthenticationExtensions extensions;
5031 };
5032
5033 typedef record<DOMString, any> AuthenticationExtensions;
5034
5035 dictionary CollectedClientData {
5036     required DOMString challenge;
5037     required DOMString origin;
5038     required DOMString hashAlgorithm;
5039     DOMString tokenBindingId;
5040     AuthenticationExtensions clientExtensions;
5041     AuthenticationExtensions authenticatorExtensions;
5042 };
5043
5044 enum PublicKeyCredentialType {
5045     "public-key"
5046 };
5047
5048 dictionary PublicKeyCredentialDescriptor {
5049     required PublicKeyCredentialType type;
5050     required BufferSource id;
5051     sequence<AuthenticatorTransport> transports;
5052 };
5053
5054 enum AuthenticatorTransport {
5055     "usb",
5056     "nfc",
5057     "ble"
5058 };
5059
5060 typedef long COSEAlgorithmIdentifier;
5061
5062 typedef sequence<AAGUID> AuthenticatorSelectionList;
5063
5064 typedef BufferSource AAGUID;
5065
5066

```

```

5067 #base64url-encodingReferenced in:
5068 * 5.1. Public Key Credential Interface
5069 * 5.1.3. Create a new credential - PublicKeyCredential's
5070 [[Create]](options) method (2)
5071 * 5.1.4.1. PublicKeyCredential's
5072 [[DiscoverFromExternalSource]](options) method (2)
5073 * 7.2. Verifying an authentication assertion
5074
5075 #cborReferenced in:
5076 * 5.1.3. Create a new credential - PublicKeyCredential's
5077 [[Create]](options) method
5078 * 5.1.4.1. PublicKeyCredential's
5079 [[DiscoverFromExternalSource]](options) method
5080 * 6.1. Authenticator data (2)
5081 * 9. WebAuthn Extensions (2) (3)
5082 * 9.2. Defining extensions (2)
5083 * 9.3. Extending request parameters
5084 * 9.4. Client extension processing (2)
5085 * 9.5. Authenticator extension processing (2) (3) (4) (5)
5086
5087 #attestationReferenced in:
5088 * 4. Terminology
5089 * 6. WebAuthn Authenticator model (2)

```

```

5093 dictionary PublicKeyCredentialRequestOptions {
5094     required BufferSource challenge;
5095     unsigned long timeout;
5096     USVString rpId;
5097     sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
5098     AuthenticationExtensions extensions;
5099 };
5100
5101 typedef record<DOMString, any> AuthenticationExtensions;
5102
5103 dictionary CollectedClientData {
5104     required DOMString challenge;
5105     required DOMString origin;
5106     required DOMString hashAlgorithm;
5107     DOMString tokenBindingId;
5108     AuthenticationExtensions clientExtensions;
5109     AuthenticationExtensions authenticatorExtensions;
5110 };
5111
5112 enum PublicKeyCredentialType {
5113     "public-key"
5114 };
5115
5116 dictionary PublicKeyCredentialDescriptor {
5117     required PublicKeyCredentialType type;
5118     required BufferSource id;
5119     sequence<AuthenticatorTransport> transports;
5120 };
5121
5122 enum AuthenticatorTransport {
5123     "usb",
5124     "nfc",
5125     "ble"
5126 };
5127
5128 typedef long COSEAlgorithmIdentifier;
5129
5130 typedef sequence<AAGUID> AuthenticatorSelectionList;
5131
5132 typedef BufferSource AAGUID;
5133
5134

```

[Issues Index](#)

[Do we need to explicitly return both constructCredentialAlg and credentialCreationData here? RET](#)

```

5135 #base64url-encodingReferenced in:
5136 * 5.1. Public Key Credential Interface
5137 * 5.1.3. Create a new credential - PublicKeyCredential's
5138 [[Create]](options) method (2)
5139 * 5.1.4.1. PublicKeyCredential's
5140 [[DiscoverFromExternalSource]](options) method (2)
5141 * 7.2. Verifying an authentication assertion
5142
5143 #cborReferenced in:
5144 * 5.1.3. Create a new credential - PublicKeyCredential's
5145 [[Create]](options) method
5146 * 5.1.4.1. PublicKeyCredential's
5147 [[DiscoverFromExternalSource]](options) method
5148 * 6.1. Authenticator data (2)
5149 * 9. WebAuthn Extensions (2) (3)
5150 * 9.2. Defining extensions (2)
5151 * 9.3. Extending request parameters
5152 * 9.4. Client extension processing (2)
5153 * 9.5. Authenticator extension processing (2) (3) (4) (5)
5154
5155 #attestationReferenced in:
5156 * 4. Terminology
5157 * 6. WebAuthn Authenticator model (2)

```

5090 * 6.3. Attestation (2) (3) (4)
5091 * 11.1. WebAuthn Attestation Statement Format Identifier
5092 Registrations
5093
5094 #attestation-certificateReferenced in:
5095 * 4. Terminology (2)
5096 * 8.3.1. TPM attestation statement certificate requirements
5097
5098 #attestation-key-pairReferenced in:
5099 * 4. Terminology (2)
5100 * 6.3. Attestation
5101
5102 #attestation-private-keyReferenced in:
5103 * 6. WebAuthn Authenticator model
5104 * 6.3. Attestation
5105
5106 #attestation-public-keyReferenced in:
5107 * 6.3. Attestation
5108
5109 #authenticationReferenced in:
5110 * 1. Introduction (2)
5111 * 4. Terminology (2) (3) (4) (5) (6) (7)
5112 * 7.2. Verifying an authentication assertion (2) (3)
5113
5114 #authentication-assertionReferenced in:
5115 * 1. Introduction
5116 * 4. Terminology (2) (3)
5117 * 5.1. PublicKeyCredential Interface
5118 * 5.2.2. Web Authentication Assertion (interface
5119 AuthenticatorAssertionResponse)
5120 * 5.5. Options for Assertion Generation (dictionary
5121 PublicKeyCredentialRequestOptions)
5122 * 9. WebAuthn Extensions
5123
5124 #authenticatorReferenced in:
5125 * 1. Introduction (2) (3) (4)
5126 * 1.1. Use Cases
5127 * 2.2. Authenticators
5128 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
5129 (14) (15)
5130 * 5. Web Authentication API (2) (3)
5131 * 5.1. PublicKeyCredential Interface
5132 * 5.1.3. Create a new credential - PublicKeyCredential's
5133 [[Create]](options) method (2)
5134 * 5.1.4.1. PublicKeyCredential's
5135 [[DiscoverFromExternalSource]](options) method (2) (3)
5136 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
5137 * 5.2.1. Information about Public Key Credential (interface
5138 AuthenticatorAttestationResponse) (2)
5139 * 5.2.2. Web Authentication Assertion (interface
5140 AuthenticatorAssertionResponse)
5141 * 5.4.5. Authenticator Attachment enumeration (enum
5142 AuthenticatorAttachment)
5143 * 5.5. Options for Assertion Generation (dictionary
5144 PublicKeyCredentialRequestOptions)
5145 * 6. WebAuthn Authenticator model (2) (3) (4) (5) (6)
5146 * 6.1. Authenticator data
5147 * 6.2.1. The authenticatorMakeCredential operation (2)
5148 * 6.2.2. The authenticatorGetAssertion operation (2) (3) (4)
5149 * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)
5150 * 6.3.2. Attestation Statement Formats
5151 * 6.3.4. Generating an Attestation Object
5152 * 6.3.5.1. Privacy
5153 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
5154 Compromise
5155 * 7.1. Registering a new credential
5156 * 8.2. Packed Attestation Statement Format
5157 * 8.4. Android Key Attestation Statement Format
5158 * 8.5. Android SafetyNet Attestation Statement Format
5159 * 10.5. Supported Extensions Extension (exts)

5163 * 6.3. Attestation (2) (3) (4)
5164 * 11.1. WebAuthn Attestation Statement Format Identifier
5165 Registrations
5166
5167 #attestation-certificateReferenced in:
5168 * 4. Terminology (2)
5169 * 8.3.1. TPM attestation statement certificate requirements
5170
5171 #attestation-key-pairReferenced in:
5172 * 4. Terminology (2)
5173 * 6.3. Attestation
5174
5175 #attestation-private-keyReferenced in:
5176 * 6. WebAuthn Authenticator model
5177 * 6.3. Attestation
5178
5179 #attestation-public-keyReferenced in:
5180 * 6.3. Attestation
5181
5182 #authenticationReferenced in:
5183 * 1. Introduction (2)
5184 * 4. Terminology (2) (3) (4) (5) (6) (7)
5185 * 7.2. Verifying an authentication assertion (2) (3)
5186
5187 #authentication-assertionReferenced in:
5188 * 1. Introduction
5189 * 4. Terminology (2) (3)
5190 * 5.1. PublicKeyCredential Interface
5191 * 5.2.2. Web Authentication Assertion (interface
5192 AuthenticatorAssertionResponse)
5193 * 5.5. Options for Assertion Generation (dictionary
5194 PublicKeyCredentialRequestOptions)
5195 * 9. WebAuthn Extensions
5196
5197 #authenticatorReferenced in:
5198 * 1. Introduction (2) (3) (4)
5199 * 1.1. Use Cases
5200 * 2.2. Authenticators
5201 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
5202 (14) (15)
5203 * 5. Web Authentication API (2) (3)
5204 * 5.1. PublicKeyCredential Interface
5205 * 5.1.3. Create a new credential - PublicKeyCredential's
5206 [[Create]](options) method (2)
5207 * 5.1.4.1. PublicKeyCredential's
5208 [[DiscoverFromExternalSource]](options) method (2) (3)
5209 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
5210 * 5.2.1. Information about Public Key Credential (interface
5211 AuthenticatorAttestationResponse) (2)
5212 * 5.2.2. Web Authentication Assertion (interface
5213 AuthenticatorAssertionResponse)
5214 * 5.4.5. Authenticator Attachment enumeration (enum
5215 AuthenticatorAttachment)
5216 * 5.5. Options for Assertion Generation (dictionary
5217 PublicKeyCredentialRequestOptions)
5218 * 6. WebAuthn Authenticator model (2) (3) (4) (5) (6)
5219 * 6.1. Authenticator data
5220 * 6.2.1. The authenticatorMakeCredential operation (2)
5221 * 6.2.2. The authenticatorGetAssertion operation (2) (3) (4)
5222 * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)
5223 * 6.3.2. Attestation Statement Formats
5224 * 6.3.4. Generating an Attestation Object
5225 * 6.3.5.1. Privacy
5226 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
5227 Compromise
5228 * 7.1. Registering a new credential
5229 * 8.2. Packed Attestation Statement Format
5230 * 8.4. Android Key Attestation Statement Format
5231 * 8.5. Android SafetyNet Attestation Statement Format
5232 * 10.5. Supported Extensions Extension (exts)

5160 * 10.6. User Verification Index Extension (uvi)
 5161 * 10.7. Location Extension (loc) (2) (3) (4)
 5162 * 10.8. User Verification Method Extension (uvm)
 5163 * 12. Sample scenarios
 5164
 5165 #authorization-gestureReferenced in:
 5166 * 1.1.1. Registration
 5167 * 1.1.2. Authentication
 5168 * 1.1.3. Other use cases and configurations
 5169 * 4. Terminology (2) (3) (4) (5) (6)
 5170 * 5.1.4. Use an existing credential to make an assertion (2)
 5171
 5172 #biometric-recognitionReferenced in:
 5173 * 4. Terminology (2)
 5174
 5175 #ceremonyReferenced in:
 5176 * 1. Introduction
 5177 * 4. Terminology (2) (3) (4) (5) (6) (7)
 5178 * 7.1. Registering a new credential
 5179 * 7.2. Verifying an authentication assertion
 5180
 5181 #clientReferenced in:
 5182 * 4. Terminology
 5183 * 5.1.6. Platform Authenticator Availability - PublicKeyCredential's
 5184 isPlatformAuthenticatorAvailable() method (2) (3) (4)
 5185
 5186 #client-side-resident-credential-private-keyReferenced in:
 5187 * 4. Terminology (2)
 5188 * 5.1.3. Create a new credential - PublicKeyCredential's
 5189 [[Create]](options) method
 5190 * 5.4.4. Authenticator Selection Criteria (dictionary
 5191 AuthenticatorSelectionCriteria) (2)
 5192 * 6.2.1. The authenticatorMakeCredential operation
 5193
 5194 #conforming-user-agentReferenced in:
 5195 * 1. Introduction
 5196 * 2.1. User Agents
 5197 * 2.2. Authenticators
 5198 * 4. Terminology (2)
 5199
 5200 #credential-public-keyReferenced in:
 5201 * 4. Terminology (2) (3)
 5202 * 5.2.1. Information about Public Key Credential (interface
 5203 AuthenticatorAttestationResponse)
 5204 * 6. WebAuthn Authenticator model
 5205 * 6.3. Attestation (2) (3)
 5206 * 6.3.1. Attested credential data (2)
 5207 * 12.1. Registration (2)
 5208
 5209 #credential-key-pairReferenced in:
 5210 * 4. Terminology (2) (3)
 5211 * 5.1.3. Create a new credential - PublicKeyCredential's
 5212 [[Create]](options) method
 5213
 5214 #credential-private-keyReferenced in:
 5215 * 4. Terminology (2) (3) (4)
 5216 * 5.1. PublicKeyCredential Interface
 5217 * 5.2.2. Web Authentication Assertion (interface
 5218 AuthenticatorAssertionResponse)
 5219 * 6. WebAuthn Authenticator model
 5220 * 6.2.2. The authenticatorGetAssertion operation
 5221 * 6.3. Attestation (2)
 5222 * 7.2. Verifying an authentication assertion
 5223
 5224 #registrationReferenced in:
 5225 * 1. Introduction (2)
 5226 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
 5227 * 7.1. Registering a new credential
 5228
 5229 #relying-partyReferenced in:

5233 * 10.6. User Verification Index Extension (uvi)
 5234 * 10.7. Location Extension (loc) (2) (3) (4)
 5235 * 10.8. User Verification Method Extension (uvm)
 5236 * 12. Sample scenarios
 5237
 5238 #authorization-gestureReferenced in:
 5239 * 1.1.1. Registration
 5240 * 1.1.2. Authentication
 5241 * 1.1.3. Other use cases and configurations
 5242 * 4. Terminology (2) (3) (4) (5) (6)
 5243 * 5.1.4. Use an existing credential to make an assertion (2)
 5244
 5245 #biometric-recognitionReferenced in:
 5246 * 4. Terminology (2)
 5247
 5248 #ceremonyReferenced in:
 5249 * 1. Introduction
 5250 * 4. Terminology (2) (3) (4) (5) (6) (7)
 5251 * 7.1. Registering a new credential
 5252 * 7.2. Verifying an authentication assertion
 5253
 5254 #clientReferenced in:
 5255 * 4. Terminology
 5256 * 5.1.6. Platform Authenticator Availability - PublicKeyCredential's
 5257 isPlatformAuthenticatorAvailable() method (2) (3) (4)
 5258
 5259 #client-side-resident-credential-private-keyReferenced in:
 5260 * 4. Terminology (2)
 5261 * 5.1.3. Create a new credential - PublicKeyCredential's
 5262 [[Create]](options) method
 5263 * 5.4.4. Authenticator Selection Criteria (dictionary
 5264 AuthenticatorSelectionCriteria) (2)
 5265 * 6.2.1. The authenticatorMakeCredential operation
 5266
 5267 #conforming-user-agentReferenced in:
 5268 * 1. Introduction
 5269 * 2.1. User Agents
 5270 * 2.2. Authenticators
 5271 * 4. Terminology (2)
 5272
 5273 #credential-public-keyReferenced in:
 5274 * 4. Terminology (2) (3)
 5275 * 5.2.1. Information about Public Key Credential (interface
 5276 AuthenticatorAttestationResponse)
 5277 * 6. WebAuthn Authenticator model
 5278 * 6.3. Attestation (2) (3)
 5279 * 6.3.1. Attested credential data (2)
 5280 * 12.1. Registration (2)
 5281
 5282 #credential-key-pairReferenced in:
 5283 * 4. Terminology (2) (3)
 5284 * 5.1.3. Create a new credential - PublicKeyCredential's
 5285 [[Create]](options) method
 5286
 5287 #credential-private-keyReferenced in:
 5288 * 4. Terminology (2) (3) (4)
 5289 * 5.1. PublicKeyCredential Interface
 5290 * 5.2.2. Web Authentication Assertion (interface
 5291 AuthenticatorAssertionResponse)
 5292 * 6. WebAuthn Authenticator model
 5293 * 6.2.2. The authenticatorGetAssertion operation
 5294 * 6.3. Attestation (2)
 5295 * 7.2. Verifying an authentication assertion
 5296
 5297 #registrationReferenced in:
 5298 * 1. Introduction (2)
 5299 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
 5300 * 7.1. Registering a new credential
 5301
 5302 #relying-partyReferenced in:

5230 * 1. Introduction (2) (3) (4) (5) (6) (7)
 5231 * 1.1.3. Other use cases and configurations
 5232 * 2.3. Relying Parties
 5233 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
 5234 (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26)
 5235 * 5. Web Authentication API (2) (3) (4) (5) (6) (7)
 5236 * 5.1.4. Use an existing credential to make an assertion
 5237 * 5.1.4.1. PublicKeyCredential's
 5238 [[DiscoverFromExternalSource]](options) method (2)
 5239 * 5.1.6. Platform Authenticator Availability - PublicKeyCredential's
 5240 isPlatformAuthenticatorAvailable() method (2) (3)
 5241 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
 5242 * 5.2.1. Information about Public Key Credential (interface
 5243 AuthenticatorAttestationResponse) (2)
 5244 * 5.2.2. Web Authentication Assertion (interface
 5245 AuthenticatorAssertionResponse)
 5246 * 5.4. Options for Credential Creation (dictionary
 5247 MakePublicKeyCredentialOptions) (2) (3) (4) (5) (6)
 5248 * 5.4.1. Public Key Entity Description (dictionary
 5249 PublicKeyCredentialEntity) (2) (3)
 5250 * 5.4.2. RP Parameters for Credential Generation (dictionary
 5251 PublicKeyCredentialRpEntity) (2)
 5252 * 5.4.4. Authenticator Selection Criteria (dictionary
 5253 AuthenticatorSelectionCriteria) (2) (3)
 5254 * 5.4.5. Authenticator Attachment enumeration (enum
 5255 AuthenticatorAttachment) (2) (3) (4)
 5256 * 5.7.1. Client data used in WebAuthn signatures (dictionary
 5257 CollectedClientData) (2) (3) (4)
 5258 * 5.7.4. Authenticator Transport enumeration (enum
 5259 AuthenticatorTransport) (2)
 5260 * 6. WebAuthn Authenticator model (2)
 5261 * 6.1. Authenticator data (2)
 5262 * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6)
 5263 * 6.2.1. The authenticatorMakeCredential operation (2) (3) (4) (5)
 5264 * 6.2.2. The authenticatorGetAssertion operation (2) (3)
 5265 * 6.3. Attestation (2) (3) (4) (5) (6)
 5266 * 6.3.5.1. Privacy
 5267 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
 5268 Compromise (2) (3) (4) (5) (6)
 5269 * 7. Relying Party Operations (2) (3) (4)
 5270 * 7.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)
 5271 (10) (11) (12)
 5272 * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)
 5273 (8)
 5274 * 8.4. Android Key Attestation Statement Format
 5275 * 9. WebAuthn Extensions (2) (3) (4)
 5276 * 9.2. Defining extensions (2)
 5277 * 9.3. Extending request parameters (2) (3) (4)
 5278 * 9.6. Example Extension (2) (3)
 5279 * 10.1. FIDO Appid Extension (appid) (2)
 5280 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
 5281 * 10.4. Authenticator Selection Extension (authnSel) (2) (3)
 5282 * 10.5. Supported Extensions Extension (exts) (2)
 5283 * 10.6. User Verification Index Extension (uvi)
 5284 * 10.7. Location Extension (loc) (2)
 5285 * 11.2. WebAuthn Extension Identifier Registrations (2)
 5286 * 12.1. Registration (2) (3) (4) (5)
 5287 * 12.2. Registration Specifically with Platform Authenticator (2) (3)
 5288 * 12.3. Authentication (2) (3) (4) (5)
 5289 * 12.4. Decommissioning (2)
 5290
 5291 #relying-party-identifierReferenced in:
 5292 * 5. Web Authentication API
 5293 * 5.4. Options for Credential Creation (dictionary
 5294 MakePublicKeyCredentialOptions)
 5295 * 5.5. Options for Assertion Generation (dictionary
 5296 PublicKeyCredentialRequestOptions)
 5297 * 6. WebAuthn Authenticator model
 5298
 5299 #rp-idReferenced in:

5303 * 1. Introduction (2) (3) (4) (5) (6) (7)
 5304 * 1.1.3. Other use cases and configurations
 5305 * 2.3. Relying Parties
 5306 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
 5307 (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26)
 5308 * 5. Web Authentication API (2) (3) (4) (5) (6) (7)
 5309 * 5.1.4. Use an existing credential to make an assertion
 5310 * 5.1.4.1. PublicKeyCredential's
 5311 [[DiscoverFromExternalSource]](options) method (2)
 5312 * 5.1.6. Platform Authenticator Availability - PublicKeyCredential's
 5313 isPlatformAuthenticatorAvailable() method (2) (3)
 5314 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
 5315 * 5.2.1. Information about Public Key Credential (interface
 5316 AuthenticatorAttestationResponse) (2)
 5317 * 5.2.2. Web Authentication Assertion (interface
 5318 AuthenticatorAssertionResponse)
 5319 * 5.4. Options for Credential Creation (dictionary
 5320 MakePublicKeyCredentialOptions) (2) (3) (4) (5) (6)
 5321 * 5.4.1. Public Key Entity Description (dictionary
 5322 PublicKeyCredentialEntity) (2) (3)
 5323 * 5.4.2. RP Parameters for Credential Generation (dictionary
 5324 PublicKeyCredentialRpEntity) (2)
 5325 * 5.4.4. Authenticator Selection Criteria (dictionary
 5326 AuthenticatorSelectionCriteria) (2) (3)
 5327 * 5.4.5. Authenticator Attachment enumeration (enum
 5328 AuthenticatorAttachment) (2) (3) (4)
 5329 * 5.7.1. Client data used in WebAuthn signatures (dictionary
 5330 CollectedClientData) (2) (3) (4)
 5331 * 5.7.4. Authenticator Transport enumeration (enum
 5332 AuthenticatorTransport) (2)
 5333 * 6. WebAuthn Authenticator model (2)
 5334 * 6.1. Authenticator data (2)
 5335 * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6)
 5336 * 6.2.1. The authenticatorMakeCredential operation (2) (3) (4) (5)
 5337 * 6.2.2. The authenticatorGetAssertion operation (2) (3)
 5338 * 6.3. Attestation (2) (3) (4) (5) (6)
 5339 * 6.3.5.1. Privacy
 5340 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
 5341 Compromise (2) (3) (4) (5) (6)
 5342 * 7. Relying Party Operations (2) (3) (4)
 5343 * 7.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)
 5344 (10) (11) (12)
 5345 * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)
 5346 (8)
 5347 * 8.4. Android Key Attestation Statement Format
 5348 * 9. WebAuthn Extensions (2) (3) (4)
 5349 * 9.2. Defining extensions (2)
 5350 * 9.3. Extending request parameters (2) (3) (4)
 5351 * 9.6. Example Extension (2) (3)
 5352 * 10.1. FIDO Appid Extension (appid) (2)
 5353 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
 5354 * 10.4. Authenticator Selection Extension (authnSel) (2) (3)
 5355 * 10.5. Supported Extensions Extension (exts) (2)
 5356 * 10.6. User Verification Index Extension (uvi)
 5357 * 10.7. Location Extension (loc) (2)
 5358 * 11.2. WebAuthn Extension Identifier Registrations (2)
 5359 * 12.1. Registration (2) (3) (4) (5)
 5360 * 12.2. Registration Specifically with Platform Authenticator (2) (3)
 5361 * 12.3. Authentication (2) (3) (4) (5)
 5362 * 12.4. Decommissioning (2)
 5363
 5364 #relying-party-identifierReferenced in:
 5365 * 5. Web Authentication API
 5366 * 5.4. Options for Credential Creation (dictionary
 5367 MakePublicKeyCredentialOptions)
 5368 * 5.5. Options for Assertion Generation (dictionary
 5369 PublicKeyCredentialRequestOptions)
 5370 * 6. WebAuthn Authenticator model
 5371
 5372 #rp-idReferenced in:

5300 * 4. Terminology (2) (3) (4) (5) (6)
5301 * 5. Web Authentication API (2) (3) (4) (5)
5302 * 5.1.3. Create a new credential - PublicKeyCredential's
5303 [[Create]](options) method (2)
5304 * 5.1.4.1. PublicKeyCredential's
5305 [[DiscoverFromExternalSource]](options) method (2)
5306 * 5.4.2. RP Parameters for Credential Generation (dictionary
5307 PublicKeyCredentialRpEntity)
5308 * 6. WebAuthn Authenticator model
5309 * 6.1. Authenticator data (2) (3) (4) (5) (6)
5310 * 6.1.1. Signature Counter Considerations
5311 * 6.2.1. The authenticatorMakeCredential operation (2) (3)
5312 * 6.2.2. The authenticatorGetAssertion operation (2)
5313 * 7.1. Registering a new credential (2)
5314 * 7.2. Verifying an authentication assertion
5315 * 8.4. Android Key Attestation Statement Format
5316 * 8.6. FIDO U2F Attestation Statement Format
5317
5318 #public-key-credentialReferenced in:
5319 * 1. Introduction (2) (3) (4) (5)
5320 * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
5321 * 5. Web Authentication API (2) (3) (4)
5322 * 5.1. PublicKeyCredential Interface
5323 * 5.1.3. Create a new credential - PublicKeyCredential's
5324 [[Create]](options) method
5325 * 5.1.4. Use an existing credential to make an assertion
5326 * 5.1.4.1. PublicKeyCredential's
5327 [[DiscoverFromExternalSource]](options) method
5328 * 5.2.1. Information about Public Key Credential (interface
5329 AuthenticatorAttestationResponse)
5330 * 5.4.1. Public Key Entity Description (dictionary
5331 PublicKeyCredentialEntity)
5332 * 5.4.4. Authenticator Selection Criteria (dictionary
5333 AuthenticatorSelectionCriteria)
5334 * 5.5. Options for Assertion Generation (dictionary
5335 PublicKeyCredentialRequestOptions)
5336 * 5.7. Supporting Data Structures
5337 * 6. WebAuthn Authenticator model (2) (3) (4) (5)
5338 * 6.2.2. The authenticatorGetAssertion operation
5339 * 6.3. Attestation (2)
5340 * 6.3.2. Attestation Statement Formats
5341 * 6.3.3. Attestation Types
5342 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
5343 Compromise (2)
5344 * 7.1. Registering a new credential
5345 * 9. WebAuthn Extensions (2)
5346 * 12. Sample scenarios
5347
5348 #test-of-user-presenceReferenced in:
5349 * 4. Terminology (2) (3) (4) (5) (6)
5350 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
5351 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
5352
5353 #user-consentReferenced in:
5354 * 1. Introduction
5355 * 4. Terminology (2)
5356 * 5.2.2. Web Authentication Assertion (interface
5357 AuthenticatorAssertionResponse)
5358 * 6. WebAuthn Authenticator model (2) (3)
5359 * 6.2.2. The authenticatorGetAssertion operation (2)
5360
5361 #user-handleReferenced in:
5362 * 5.1.4.1. PublicKeyCredential's
5363 [[DiscoverFromExternalSource]](options) method
5364 * 5.2.2. Web Authentication Assertion (interface
5365 AuthenticatorAssertionResponse)
5366 * 5.4. Options for Credential Creation (dictionary
5367 MakePublicKeyCredentialOptions)
5368 * 5.4.3. User Account Parameters for Credential Generation
5369 (dictionary PublicKeyCredentialUserEntity)

5373 * 4. Terminology (2) (3) (4) (5) (6)
5374 * 5. Web Authentication API (2) (3) (4) (5)
5375 * 5.1.3. Create a new credential - PublicKeyCredential's
5376 [[Create]](options) method (2)
5377 * 5.1.4.1. PublicKeyCredential's
5378 [[DiscoverFromExternalSource]](options) method (2)
5379 * 5.4.2. RP Parameters for Credential Generation (dictionary
5380 PublicKeyCredentialRpEntity)
5381 * 6. WebAuthn Authenticator model
5382 * 6.1. Authenticator data (2) (3) (4) (5) (6)
5383 * 6.1.1. Signature Counter Considerations
5384 * 6.2.1. The authenticatorMakeCredential operation (2)
5385 * 6.2.2. The authenticatorGetAssertion operation (2)
5386 * 7.1. Registering a new credential (2)
5387 * 7.2. Verifying an authentication assertion
5388 * 8.4. Android Key Attestation Statement Format
5389 * 8.6. FIDO U2F Attestation Statement Format
5390
5391 #public-key-credentialReferenced in:
5392 * 1. Introduction (2) (3) (4) (5)
5393 * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
5394 * 5. Web Authentication API (2) (3) (4)
5395 * 5.1. PublicKeyCredential Interface
5396 * 5.1.3. Create a new credential - PublicKeyCredential's
5397 [[Create]](options) method
5398 * 5.1.4. Use an existing credential to make an assertion
5399 * 5.1.4.1. PublicKeyCredential's
5400 [[DiscoverFromExternalSource]](options) method
5401 * 5.2.1. Information about Public Key Credential (interface
5402 AuthenticatorAttestationResponse)
5403 * 5.4.1. Public Key Entity Description (dictionary
5404 PublicKeyCredentialEntity)
5405 * 5.4.4. Authenticator Selection Criteria (dictionary
5406 AuthenticatorSelectionCriteria)
5407 * 5.5. Options for Assertion Generation (dictionary
5408 PublicKeyCredentialRequestOptions)
5409 * 5.7. Supporting Data Structures
5410 * 6. WebAuthn Authenticator model (2) (3) (4) (5)
5411 * 6.2.2. The authenticatorGetAssertion operation
5412 * 6.3. Attestation (2)
5413 * 6.3.2. Attestation Statement Formats
5414 * 6.3.3. Attestation Types
5415 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
5416 Compromise (2)
5417 * 7.1. Registering a new credential
5418 * 9. WebAuthn Extensions (2)
5419 * 12. Sample scenarios
5420
5421 #test-of-user-presenceReferenced in:
5422 * 4. Terminology (2) (3) (4) (5) (6)
5423 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
5424 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
5425
5426 #user-consentReferenced in:
5427 * 1. Introduction
5428 * 4. Terminology (2)
5429 * 5.2.2. Web Authentication Assertion (interface
5430 AuthenticatorAssertionResponse)
5431 * 6. WebAuthn Authenticator model (2) (3)
5432 * 6.2.2. The authenticatorGetAssertion operation (2)
5433
5434 #user-handleReferenced in:
5435 * 5.1.4.1. PublicKeyCredential's
5436 [[DiscoverFromExternalSource]](options) method
5437 * 5.2.2. Web Authentication Assertion (interface
5438 AuthenticatorAssertionResponse)
5439 * 5.4. Options for Credential Creation (dictionary
5440 MakePublicKeyCredentialOptions)
5441 * 5.4.3. User Account Parameters for Credential Generation
5442 (dictionary PublicKeyCredentialUserEntity)

5370 * 6.2.1. The authenticatorMakeCredential operation
5371 * 6.2.2. The authenticatorGetAssertion operation
5372
5373 #user-verificationReferenced in:
5374 * 1. Introduction
5375 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
5376 * 5.1.3. Create a new credential - PublicKeyCredential's
5377 [[Create]](options) method
5378 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
5379 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
5380
5381 #concept-user-presentReferenced in:
5382 * 4. Terminology
5383 * 6.1. Authenticator data (2) (3)
5384
5385 #upReferenced in:
5386 * 6.1. Authenticator data
5387
5388 #concept-user-verifiedReferenced in:
5389 * 4. Terminology
5390 * 6.1. Authenticator data (2) (3)
5391
5392 #uvReferenced in:
5393 * 6.1. Authenticator data
5394
5395 #webauthn-clientReferenced in:
5396 * 4. Terminology (2)
5397
5398 #web-authentication-apiReferenced in:
5399 * 1. Introduction (2) (3)
5400 * 4. Terminology (2)
5401
5402 #publickeycredentialReferenced in:
5403 * 1. Introduction
5404 * 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
5405 * 5.1.3. Create a new credential - PublicKeyCredential's
5406 [[Create]](options) method (2) (3) (4) (5) (6)
5407 * 5.1.4.1. PublicKeyCredential's
5408 [[DiscoverFromExternalSource]](options) method (2) (3)
5409 * 5.1.5. Store an existing credential - PublicKeyCredential's
5410 [[Store]](credential) method (2)
5411 * 5.1.6. Platform Authenticator Availability - PublicKeyCredential's
5412 isPlatformAuthenticatorAvailable() method
5413 * 5.7.3. Credential Descriptor (dictionary
5414 PublicKeyCredentialDescriptor)
5415 * 7. Relying Party Operations
5416 * 7.2. Verifying an authentication assertion
5417
5418 #dom-publickeycredential-rawidReferenced in:
5419 * 5.1. PublicKeyCredential Interface
5420 * 7.2. Verifying an authentication assertion
5421
5422 #dom-publickeycredential-responseReferenced in:
5423 * 5.1. PublicKeyCredential Interface
5424 * 5.1.3. Create a new credential - PublicKeyCredential's
5425 [[Create]](options) method
5426 * 5.1.4.1. PublicKeyCredential's
5427 [[DiscoverFromExternalSource]](options) method
5428 * 7.2. Verifying an authentication assertion
5429
5430 #dom-publickeycredential-clientextensionresultsReferenced in:
5431 * 5.1. PublicKeyCredential Interface
5432 * 5.1.3. Create a new credential - PublicKeyCredential's
5433 [[Create]](options) method
5434 * 5.1.4.1. PublicKeyCredential's
5435 [[DiscoverFromExternalSource]](options) method
5436 * 9.4. Client extension processing
5437
5438 #dom-publickeycredential-identifier-slotReferenced in:
5439 * 5.1. PublicKeyCredential Interface (2)

5443 * 6.2.1. The authenticatorMakeCredential operation
5444 * 6.2.2. The authenticatorGetAssertion operation
5445
5446 #user-verificationReferenced in:
5447 * 1. Introduction
5448 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
5449 * 5.1.3. Create a new credential - PublicKeyCredential's
5450 [[Create]](options) method
5451 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
5452 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
5453
5454 #concept-user-presentReferenced in:
5455 * 4. Terminology
5456 * 6.1. Authenticator data (2) (3)
5457
5458 #upReferenced in:
5459 * 6.1. Authenticator data
5460
5461 #concept-user-verifiedReferenced in:
5462 * 4. Terminology
5463 * 6.1. Authenticator data (2) (3)
5464
5465 #uvReferenced in:
5466 * 6.1. Authenticator data
5467
5468 #webauthn-clientReferenced in:
5469 * 4. Terminology (2)
5470
5471 #web-authentication-apiReferenced in:
5472 * 1. Introduction (2) (3)
5473 * 4. Terminology (2)
5474
5475 #publickeycredentialReferenced in:
5476 * 1. Introduction
5477 * 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
5478 * 5.1.3. Create a new credential - PublicKeyCredential's
5479 [[Create]](options) method (2) (3) (4) (5) (6)
5480 * 5.1.4.1. PublicKeyCredential's
5481 [[DiscoverFromExternalSource]](options) method (2) (3)
5482 * 5.1.5. Store an existing credential - PublicKeyCredential's
5483 [[Store]](credential) method (2)
5484 * 5.1.6. Platform Authenticator Availability - PublicKeyCredential's
5485 isPlatformAuthenticatorAvailable() method
5486 * 5.7.3. Credential Descriptor (dictionary
5487 PublicKeyCredentialDescriptor)
5488 * 7. Relying Party Operations
5489 * 7.2. Verifying an authentication assertion
5490
5491 #dom-publickeycredential-rawidReferenced in:
5492 * 5.1. PublicKeyCredential Interface
5493 * 7.2. Verifying an authentication assertion
5494
5495 #dom-publickeycredential-responseReferenced in:
5496 * 5.1. PublicKeyCredential Interface
5497 * 5.1.3. Create a new credential - PublicKeyCredential's
5498 [[Create]](options) method
5499 * 5.1.4.1. PublicKeyCredential's
5500 [[DiscoverFromExternalSource]](options) method
5501 * 7.2. Verifying an authentication assertion
5502
5503 #dom-publickeycredential-clientextensionresultsReferenced in:
5504 * 5.1. PublicKeyCredential Interface
5505 * 5.1.3. Create a new credential - PublicKeyCredential's
5506 [[Create]](options) method
5507 * 5.1.4.1. PublicKeyCredential's
5508 [[DiscoverFromExternalSource]](options) method
5509 * 9.4. Client extension processing
5510
5511 #dom-publickeycredential-identifier-slotReferenced in:
5512 * 5.1. PublicKeyCredential Interface (2)

5440 * 5.1.3. Create a new credential - PublicKeyCredential's
 5441 [[Create]](options) method
 5442 * 5.1.4.1. PublicKeyCredential's
 5443 [[DiscoverFromExternalSource]](options) method
 5444
 5445 #dom-credentialcreationoptions-publickeyReferenced in:
 5446 * 5.1.3. Create a new credential - PublicKeyCredential's
 5447 [[Create]](options) method (2) (3)
 5448
 5449 #dom-credentialrequestoptions-publickeyReferenced in:
 5450 * 5.1.4.1. PublicKeyCredential's
 5451 [[DiscoverFromExternalSource]](options) method (2)
 5452
 5453 #dom-publickeycredential-create-slotReferenced in:
 5454 * 5.1. PublicKeyCredential Interface
 5455
 5456 #dom-publickeycredential-create-options-optionsReferenced in:
 5457 * 7.1. Registering a new credential
 5458

5459 #dom-publickeycredential-collectfromcredentialstore-slotReferenced in:
 5460 * 5.1.4. Use an existing credential to make an assertion
 5461
 5462 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
 5463 * 5.1. PublicKeyCredential Interface
 5464 * 5.1.4. Use an existing credential to make an assertion
 5465
 5466 #authenticatorresponseReferenced in:
 5467 * 5.1. PublicKeyCredential Interface (2)
 5468 * 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)
 5469 * 5.2.1. Information about Public Key Credential (interface
 5470 AuthenticatorAttestationResponse) (2)
 5471 * 5.2.2. Web Authentication Assertion (interface
 5472 AuthenticatorAssertionResponse) (2)
 5473
 5474 #dom-authenticatorresponse-clientdatajsonReferenced in:
 5475 * 5.1.3. Create a new credential - PublicKeyCredential's
 5476 [[Create]](options) method (2)
 5477 * 5.1.4.1. PublicKeyCredential's
 5478 [[DiscoverFromExternalSource]](options) method (2)
 5479 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
 5480 * 5.2.1. Information about Public Key Credential (interface
 5481 AuthenticatorAttestationResponse)
 5482 * 5.2.2. Web Authentication Assertion (interface
 5483 AuthenticatorAssertionResponse)
 5484 * 7.1. Registering a new credential (2)
 5485 * 7.2. Verifying an authentication assertion
 5486
 5487 #authenticatorattestationresponseReferenced in:
 5488 * 5.1. PublicKeyCredential Interface
 5489 * 5.1.3. Create a new credential - PublicKeyCredential's
 5490 [[Create]](options) method (2)
 5491 * 5.2.1. Information about Public Key Credential (interface
 5492 AuthenticatorAttestationResponse) (2)
 5493 * 7. Relying Party Operations
 5494 * 7.1. Registering a new credential (2) (3)
 5495
 5496 #dom-authenticatorattestationresponse-attestationobjectReferenced in:
 5497 * 5.1.3. Create a new credential - PublicKeyCredential's

5513 * 5.1.3. Create a new credential - PublicKeyCredential's
 5514 [[Create]](options) method
 5515 * 5.1.4.1. PublicKeyCredential's
 5516 [[DiscoverFromExternalSource]](options) method
 5517
 5518 #dom-credentialcreationoptions-publickeyReferenced in:
 5519 * 5.1.3. Create a new credential - PublicKeyCredential's
 5520 [[Create]](options) method (2) (3)
 5521
 5522 #dom-credentialrequestoptions-publickeyReferenced in:
 5523 * 5.1.4.1. PublicKeyCredential's
 5524 [[DiscoverFromExternalSource]](options) method (2)
 5525
 5526 #dom-publickeycredential-create-slotReferenced in:
 5527 * 5.1. PublicKeyCredential Interface
 5528
 5529 #dom-publickeycredential-create-options-optionsReferenced in:
 5530 * 7.1. Registering a new credential
 5531
 5532 #attestationobjectresultReferenced in:
 5533 * 5.1.3. Create a new credential - PublicKeyCredential's
 5534 [[Create]](options) method
 5535
 5536 #clientdatajsonresultReferenced in:
 5537 * 5.1.3. Create a new credential - PublicKeyCredential's
 5538 [[Create]](options) method
 5539
 5540 #extensionoutputsmapReferenced in:
 5541 * 5.1.3. Create a new credential - PublicKeyCredential's
 5542 [[Create]](options) method (2) (3)
 5543
 5544 #dom-publickeycredential-collectfromcredentialstore-slotReferenced in:
 5545 * 5.1.4. Use an existing credential to make an assertion
 5546
 5547 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
 5548 * 5.1. PublicKeyCredential Interface
 5549 * 5.1.4. Use an existing credential to make an assertion
 5550
 5551 #authenticatorresponseReferenced in:
 5552 * 5.1. PublicKeyCredential Interface (2)
 5553 * 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)
 5554 * 5.2.1. Information about Public Key Credential (interface
 5555 AuthenticatorAttestationResponse) (2)
 5556 * 5.2.2. Web Authentication Assertion (interface
 5557 AuthenticatorAssertionResponse) (2)
 5558
 5559 #dom-authenticatorresponse-clientdatajsonReferenced in:
 5560 * 5.1.3. Create a new credential - PublicKeyCredential's
 5561 [[Create]](options) method (2)
 5562 * 5.1.4.1. PublicKeyCredential's
 5563 [[DiscoverFromExternalSource]](options) method (2)
 5564 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
 5565 * 5.2.1. Information about Public Key Credential (interface
 5566 AuthenticatorAttestationResponse)
 5567 * 5.2.2. Web Authentication Assertion (interface
 5568 AuthenticatorAssertionResponse)
 5569 * 7.1. Registering a new credential (2)
 5570 * 7.2. Verifying an authentication assertion
 5571
 5572 #authenticatorattestationresponseReferenced in:
 5573 * 5.1. PublicKeyCredential Interface
 5574 * 5.1.3. Create a new credential - PublicKeyCredential's
 5575 [[Create]](options) method (2)
 5576 * 5.2.1. Information about Public Key Credential (interface
 5577 AuthenticatorAttestationResponse) (2)
 5578 * 7. Relying Party Operations
 5579 * 7.1. Registering a new credential (2) (3)
 5580
 5581 #dom-authenticatorattestationresponse-attestationobjectReferenced in:
 5582 * 5.1.3. Create a new credential - PublicKeyCredential's

549f [[Create]](options) method
 549f * 5.2.1. Information about Public Key Credential (interface
 550c AuthenticatorAttestationResponse)
 5501 * 7.1. Registering a new credential
 5502
 5503 #authenticatorassertionresponseReferenced in:
 5504 * 4. Terminology
 5505 * 5.1. PublicKeyCredential Interface
 5506 * 5.1.4.1. PublicKeyCredential's
 5507 [[DiscoverFromExternalSource]](options) method
 5508 * 5.2.2. Web Authentication Assertion (interface
 5509 AuthenticatorAssertionResponse) (2)
 551c * 7. Relying Party Operations
 5511
 5512 #dom-authenticatorassertionresponse-authenticatordataReferenced in:
 5513 * 5.1.4.1. PublicKeyCredential's
 5514 [[DiscoverFromExternalSource]](options) method (2)
 5515 * 5.2.2. Web Authentication Assertion (interface
 5516 AuthenticatorAssertionResponse)
 5517 * 7.2. Verifying an authentication assertion
 5518
 5519 #dom-authenticatorassertionresponse-signatureReferenced in:
 552c * 5.1.4.1. PublicKeyCredential's
 5521 [[DiscoverFromExternalSource]](options) method (2)
 5522 * 5.2.2. Web Authentication Assertion (interface
 5523 AuthenticatorAssertionResponse)
 5524 * 7.2. Verifying an authentication assertion
 5525
 5526 #dom-authenticatorassertionresponse-userhandleReferenced in:
 5527 * 5.1.4.1. PublicKeyCredential's
 5528 [[DiscoverFromExternalSource]](options) method
 5529 * 5.2.2. Web Authentication Assertion (interface
 553c AuthenticatorAssertionResponse)
 5531
 5532 #dictdef-publickeycredentialparametersReferenced in:
 5533 * 5.3. Parameters for Credential Generation (dictionary
 5534 PublicKeyCredentialParameters)
 5535 * 5.4. Options for Credential Creation (dictionary
 5536 MakePublicKeyCredentialOptions) (2)
 5537
 5538 #dom-publickeycredentialparameters-typeReferenced in:
 5539 * 5.1.3. Create a new credential - PublicKeyCredential's
 554c [[Create]](options) method (2)
 5541 * 5.3. Parameters for Credential Generation (dictionary
 5542 PublicKeyCredentialParameters)
 5543
 5544 #dom-publickeycredentialparameters-algReferenced in:
 5545 * 5.1.3. Create a new credential - PublicKeyCredential's
 5546 [[Create]](options) method
 5547 * 5.3. Parameters for Credential Generation (dictionary
 5548 PublicKeyCredentialParameters)
 5549
 555c #dictdef-makepublickeycredentialoptionsReferenced in:
 5551 * 5.1.1. CredentialCreationOptions Extension
 5552 * 5.1.3. Create a new credential - PublicKeyCredential's
 5553 [[Create]](options) method
 5554 * 5.4. Options for Credential Creation (dictionary
 5555 MakePublicKeyCredentialOptions)
 5556
 5557 #dom-makepublickeycredentialoptions-rpReferenced in:
 5558 * 5.1.3. Create a new credential - PublicKeyCredential's
 5559 [[Create]](options) method (2) (3) (4) (5) (6)
 556c * 5.4. Options for Credential Creation (dictionary
 5561 MakePublicKeyCredentialOptions)
 5562
 5563 #dom-makepublickeycredentialoptions-userReferenced in:
 5564 * 5.1.3. Create a new credential - PublicKeyCredential's
 5565 [[Create]](options) method (2) (3) (4)
 5566 * 5.4. Options for Credential Creation (dictionary
 5567 MakePublicKeyCredentialOptions)

5583 [[Create]](options) method
 5584 * 5.2.1. Information about Public Key Credential (interface
 5585 AuthenticatorAttestationResponse)
 5586 * 7.1. Registering a new credential
 5587
 5588 #authenticatorassertionresponseReferenced in:
 5589 * 4. Terminology
 559c * 5.1. PublicKeyCredential Interface
 5591 * 5.1.4.1. PublicKeyCredential's
 5592 [[DiscoverFromExternalSource]](options) method
 5593 * 5.2.2. Web Authentication Assertion (interface
 5594 AuthenticatorAssertionResponse) (2)
 5595 * 7. Relying Party Operations
 5596
 5597 #dom-authenticatorassertionresponse-authenticatordataReferenced in:
 5598 * 5.1.4.1. PublicKeyCredential's
 5599 [[DiscoverFromExternalSource]](options) method (2)
 560c * 5.2.2. Web Authentication Assertion (interface
 5601 AuthenticatorAssertionResponse)
 5602 * 7.2. Verifying an authentication assertion
 5603
 5604 #dom-authenticatorassertionresponse-signatureReferenced in:
 5605 * 5.1.4.1. PublicKeyCredential's
 5606 [[DiscoverFromExternalSource]](options) method (2)
 5607 * 5.2.2. Web Authentication Assertion (interface
 5608 AuthenticatorAssertionResponse)
 5609 * 7.2. Verifying an authentication assertion
 561c
 5611 #dom-authenticatorassertionresponse-userhandleReferenced in:
 5612 * 5.1.4.1. PublicKeyCredential's
 5613 [[DiscoverFromExternalSource]](options) method
 5614 * 5.2.2. Web Authentication Assertion (interface
 5615 AuthenticatorAssertionResponse)
 5616
 5617 #dictdef-publickeycredentialparametersReferenced in:
 5618 * 5.3. Parameters for Credential Generation (dictionary
 5619 PublicKeyCredentialParameters)
 562c * 5.4. Options for Credential Creation (dictionary
 5621 MakePublicKeyCredentialOptions) (2)
 5622
 5623 #dom-publickeycredentialparameters-typeReferenced in:
 5624 * 5.1.3. Create a new credential - PublicKeyCredential's
 5625 [[Create]](options) method (2)
 5626 * 5.3. Parameters for Credential Generation (dictionary
 5627 PublicKeyCredentialParameters)
 5628
 5629 #dom-publickeycredentialparameters-algReferenced in:
 563c * 5.1.3. Create a new credential - PublicKeyCredential's
 5631 [[Create]](options) method
 5632 * 5.3. Parameters for Credential Generation (dictionary
 5633 PublicKeyCredentialParameters)
 5634
 5635 #dictdef-makepublickeycredentialoptionsReferenced in:
 5636 * 5.1.1. CredentialCreationOptions Extension
 5637 * 5.1.3. Create a new credential - PublicKeyCredential's
 5638 [[Create]](options) method
 5639 * 5.4. Options for Credential Creation (dictionary
 564c MakePublicKeyCredentialOptions)
 5641
 5642 #dom-makepublickeycredentialoptions-rpReferenced in:
 5643 * 5.1.3. Create a new credential - PublicKeyCredential's
 5644 [[Create]](options) method (2) (3) (4) (5) (6) (7)
 5645 * 5.4. Options for Credential Creation (dictionary
 5646 MakePublicKeyCredentialOptions)
 5647
 5648 #dom-makepublickeycredentialoptions-userReferenced in:
 5649 * 5.1.3. Create a new credential - PublicKeyCredential's
 565c [[Create]](options) method (2) (3) (4)
 5651 * 5.4. Options for Credential Creation (dictionary
 5652 MakePublicKeyCredentialOptions)

```

556E * 7.1. Registering a new credential
556F
557C #dom-makepublickeycredentialoptions-challengeReferenced in:
557D * 5.1.3. Create a new credential - PublicKeyCredential's
557E [[Create]](options) method
557F * 5.4. Options for Credential Creation (dictionary
557G MakePublicKeyCredentialOptions)
557H
557I #dom-makepublickeycredentialoptions-pubkeycredparamsReferenced in:
557J * 5.1.3. Create a new credential - PublicKeyCredential's
557K [[Create]](options) method (2)
557L * 5.4. Options for Credential Creation (dictionary
557M MakePublicKeyCredentialOptions)
557N
557O #dom-makepublickeycredentialoptions-timeoutReferenced in:
557P * 5.1.3. Create a new credential - PublicKeyCredential's
557Q [[Create]](options) method (2)
557R * 5.4. Options for Credential Creation (dictionary
557S MakePublicKeyCredentialOptions)
557T
557U #dom-makepublickeycredentialoptions-excludecredentialsReferenced in:
557V * 5.1.3. Create a new credential - PublicKeyCredential's
557W [[Create]](options) method
557X * 5.4. Options for Credential Creation (dictionary
557Y MakePublicKeyCredentialOptions)
557Z
5580 #dom-makepublickeycredentialoptions-authenticatorselectionReferenced
5581 in:
5582 * 5.1.3. Create a new credential - PublicKeyCredential's
5583 [[Create]](options) method (2)
5584 * 5.4. Options for Credential Creation (dictionary
5585 MakePublicKeyCredentialOptions)
5586 * 6.2.1. The authenticatorMakeCredential operation (2)
5587
5588 #dom-makepublickeycredentialoptions-extensionsReferenced in:
5589 * 5.1.3. Create a new credential - PublicKeyCredential's
558A [[Create]](options) method (2)
558B * 5.4. Options for Credential Creation (dictionary
558C MakePublicKeyCredentialOptions)
558D * 9.3. Extending request parameters
558E
558F #dictdef-publickeycredentialentityReferenced in:
5590 * 5.4.1. Public Key Entity Description (dictionary
5591 PublicKeyCredentialEntity) (2)
5592 * 5.4.2. RP Parameters for Credential Generation (dictionary
5593 PublicKeyCredentialRpEntity)
5594 * 5.4.3. User Account Parameters for Credential Generation
5595 (dictionary PublicKeyCredentialUserEntity)
5596
5597 #dom-publickeycredentialentity-nameReferenced in:
5598 * 5.1.3. Create a new credential - PublicKeyCredential's
5599 [[Create]](options) method (2)
559A * 5.4. Options for Credential Creation (dictionary
559B MakePublicKeyCredentialOptions) (2)
559C * 5.4.1. Public Key Entity Description (dictionary
559D PublicKeyCredentialEntity)
559E
559F #dom-publickeycredentialentity-iconReferenced in:
559G * 5.4.1. Public Key Entity Description (dictionary
559H PublicKeyCredentialEntity)
559I
559J #dictdef-publickeycredentialrpentityReferenced in:
559K * 5.4. Options for Credential Creation (dictionary
559L MakePublicKeyCredentialOptions) (2)
559M * 5.4.2. RP Parameters for Credential Generation (dictionary
559N PublicKeyCredentialRpEntity) (2)
559O * 6.2.1. The authenticatorMakeCredential operation
559P
559Q #dom-publickeycredentialrpentity-idReferenced in:
559R * 5.1.3. Create a new credential - PublicKeyCredential's
559S

```

```

5653 * 7.1. Registering a new credential
5654
5655 #dom-makepublickeycredentialoptions-challengeReferenced in:
5656 * 5.1.3. Create a new credential - PublicKeyCredential's
5657 [[Create]](options) method
5658 * 5.4. Options for Credential Creation (dictionary
5659 MakePublicKeyCredentialOptions)
565A
565B #dom-makepublickeycredentialoptions-pubkeycredparamsReferenced in:
565C * 5.1.3. Create a new credential - PublicKeyCredential's
565D [[Create]](options) method (2)
565E * 5.4. Options for Credential Creation (dictionary
565F MakePublicKeyCredentialOptions)
5660
5661 #dom-makepublickeycredentialoptions-timeoutReferenced in:
5662 * 5.1.3. Create a new credential - PublicKeyCredential's
5663 [[Create]](options) method (2)
5664 * 5.4. Options for Credential Creation (dictionary
5665 MakePublicKeyCredentialOptions)
5666
5667 #dom-makepublickeycredentialoptions-excludecredentialsReferenced in:
5668 * 5.1.3. Create a new credential - PublicKeyCredential's
5669 [[Create]](options) method
566A * 5.4. Options for Credential Creation (dictionary
566B MakePublicKeyCredentialOptions)
566C
566D #dom-makepublickeycredentialoptions-authenticatorselectionReferenced
566E in:
566F * 5.1.3. Create a new credential - PublicKeyCredential's
566G [[Create]](options) method (2) (3)
566H * 5.4. Options for Credential Creation (dictionary
566I MakePublicKeyCredentialOptions)
566J * 6.2.1. The authenticatorMakeCredential operation (2)
566K
566L #dom-makepublickeycredentialoptions-extensionsReferenced in:
566M * 5.1.3. Create a new credential - PublicKeyCredential's
566N [[Create]](options) method (2)
566O * 5.4. Options for Credential Creation (dictionary
566P MakePublicKeyCredentialOptions)
566Q * 9.3. Extending request parameters
566R
566S #dictdef-publickeycredentialentityReferenced in:
566T * 5.4.1. Public Key Entity Description (dictionary
566U PublicKeyCredentialEntity) (2)
566V * 5.4.2. RP Parameters for Credential Generation (dictionary
566W PublicKeyCredentialRpEntity)
566X * 5.4.3. User Account Parameters for Credential Generation
566Y (dictionary PublicKeyCredentialUserEntity)
566Z
566A #dom-publickeycredentialentity-nameReferenced in:
566B * 5.1.3. Create a new credential - PublicKeyCredential's
566C [[Create]](options) method (2)
566D * 5.4. Options for Credential Creation (dictionary
566E MakePublicKeyCredentialOptions) (2)
566F * 5.4.1. Public Key Entity Description (dictionary
566G PublicKeyCredentialEntity)
566H
566I #dom-publickeycredentialentity-iconReferenced in:
566J * 5.4.1. Public Key Entity Description (dictionary
566K PublicKeyCredentialEntity)
566L
566M #dictdef-publickeycredentialrpentityReferenced in:
566N * 5.4. Options for Credential Creation (dictionary
566O MakePublicKeyCredentialOptions) (2)
566P * 5.4.2. RP Parameters for Credential Generation (dictionary
566Q PublicKeyCredentialRpEntity) (2)
566R * 6.2.1. The authenticatorMakeCredential operation
566S
566T #dom-publickeycredentialrpentity-idReferenced in:
566U

```

```

5638 [[Create]](options) method (2) (3) (4)
5639 * 5.4. Options for Credential Creation (dictionary
5640 MakePublicKeyCredentialOptions)
5641 * 5.4.2. RP Parameters for Credential Generation (dictionary
5642 PublicKeyCredentialRpEntity)
5643
5644 #dictdef-publickeycredentialuserentityReferenced in:
5645 * 5.4. Options for Credential Creation (dictionary
5646 MakePublicKeyCredentialOptions) (2)
5647 * 5.4.3. User Account Parameters for Credential Generation
5648 (dictionary PublicKeyCredentialUserEntity) (2)
5649 * 6.2.1. The authenticatorMakeCredential operation
5650
5651 #dom-publickeycredentialuserentity-idReferenced in:
5652 * 5.1.3. Create a new credential - PublicKeyCredential's
5653 [[Create]](options) method
5654 * 5.4. Options for Credential Creation (dictionary
5655 MakePublicKeyCredentialOptions)
5656 * 5.4.3. User Account Parameters for Credential Generation
5657 (dictionary PublicKeyCredentialUserEntity)
5658 * 6.2.1. The authenticatorMakeCredential operation
5659
5660 #dom-publickeycredentialuserentity-displaynameReferenced in:
5661 * 5.1.3. Create a new credential - PublicKeyCredential's
5662 [[Create]](options) method
5663 * 5.4. Options for Credential Creation (dictionary
5664 MakePublicKeyCredentialOptions)
5665 * 5.4.3. User Account Parameters for Credential Generation
5666 (dictionary PublicKeyCredentialUserEntity)
5667
5668 #dictdef-authenticatorselectioncriteriaReferenced in:
5669 * 5.4. Options for Credential Creation (dictionary
5670 MakePublicKeyCredentialOptions) (2)
5671 * 5.4.4. Authenticator Selection Criteria (dictionary
5672 AuthenticatorSelectionCriteria) (2)
5673
5674 #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced
5675 in:
5676 * 5.1.3. Create a new credential - PublicKeyCredential's
5677 [[Create]](options) method
5678 * 5.4.4. Authenticator Selection Criteria (dictionary
5679 AuthenticatorSelectionCriteria)
5680
5681 #dom-authenticatorselectioncriteria-requireresidentkeyReferenced in:
5682 * 5.1.3. Create a new credential - PublicKeyCredential's
5683 [[Create]](options) method (2)
5684 * 5.4.4. Authenticator Selection Criteria (dictionary
5685 AuthenticatorSelectionCriteria)
5686 * 6.2.1. The authenticatorMakeCredential operation
5687
5688 #dom-authenticatorselectioncriteria-requireuserverificationReferenced
5689 in:
5690 * 5.1.3. Create a new credential - PublicKeyCredential's
5691 [[Create]](options) method
5692 * 5.4.4. Authenticator Selection Criteria (dictionary
5693 AuthenticatorSelectionCriteria)
5694 * 6.2.1. The authenticatorMakeCredential operation
5695
5696 #enumdef-authenticatorattachmentReferenced in:
5697 * 5.4.4. Authenticator Selection Criteria (dictionary
5698 AuthenticatorSelectionCriteria) (2)
5699 * 5.4.5. Authenticator Attachment enumeration (enum
5700 AuthenticatorAttachment) (2)
5701
5702 #platform-authenticatorsReferenced in:
5703 * 5.1.6. Platform Authenticator Availability - PublicKeyCredential's
5704 isPlatformAuthenticatorAvailable() method (2) (3) (4) (5)
5705 * 5.4.5. Authenticator Attachment enumeration (enum
5706 AuthenticatorAttachment) (2)
5707 * 6.2.1. The authenticatorMakeCredential operation

```

```

5722 * 5.4. Options for Credential Creation (dictionary
5723 MakePublicKeyCredentialOptions)
5724 * 5.4.2. RP Parameters for Credential Generation (dictionary
5725 PublicKeyCredentialRpEntity)
5726
5727 #dictdef-publickeycredentialuserentityReferenced in:
5728 * 5.4. Options for Credential Creation (dictionary
5729 MakePublicKeyCredentialOptions) (2)
5730 * 5.4.3. User Account Parameters for Credential Generation
5731 (dictionary PublicKeyCredentialUserEntity) (2)
5732 * 6.2.1. The authenticatorMakeCredential operation
5733
5734 #dom-publickeycredentialuserentity-idReferenced in:
5735 * 5.1.3. Create a new credential - PublicKeyCredential's
5736 [[Create]](options) method
5737 * 5.4. Options for Credential Creation (dictionary
5738 MakePublicKeyCredentialOptions)
5739 * 5.4.3. User Account Parameters for Credential Generation
5740 (dictionary PublicKeyCredentialUserEntity)
5741 * 6.2.1. The authenticatorMakeCredential operation
5742
5743 #dom-publickeycredentialuserentity-displaynameReferenced in:
5744 * 5.1.3. Create a new credential - PublicKeyCredential's
5745 [[Create]](options) method
5746 * 5.4. Options for Credential Creation (dictionary
5747 MakePublicKeyCredentialOptions)
5748 * 5.4.3. User Account Parameters for Credential Generation
5749 (dictionary PublicKeyCredentialUserEntity)
5750
5751 #dictdef-authenticatorselectioncriteriaReferenced in:
5752 * 5.4. Options for Credential Creation (dictionary
5753 MakePublicKeyCredentialOptions) (2)
5754 * 5.4.4. Authenticator Selection Criteria (dictionary
5755 AuthenticatorSelectionCriteria) (2)
5756
5757 #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced
5758 in:
5759 * 5.1.3. Create a new credential - PublicKeyCredential's
5760 [[Create]](options) method
5761 * 5.4.4. Authenticator Selection Criteria (dictionary
5762 AuthenticatorSelectionCriteria)
5763
5764 #dom-authenticatorselectioncriteria-requireresidentkeyReferenced in:
5765 * 5.1.3. Create a new credential - PublicKeyCredential's
5766 [[Create]](options) method (2)
5767 * 5.4.4. Authenticator Selection Criteria (dictionary
5768 AuthenticatorSelectionCriteria)
5769 * 6.2.1. The authenticatorMakeCredential operation
5770
5771 #dom-authenticatorselectioncriteria-requireuserverificationReferenced
5772 in:
5773 * 5.1.3. Create a new credential - PublicKeyCredential's
5774 [[Create]](options) method (2)
5775 * 5.4.4. Authenticator Selection Criteria (dictionary
5776 AuthenticatorSelectionCriteria)
5777 * 6.2.1. The authenticatorMakeCredential operation
5778
5779 #enumdef-authenticatorattachmentReferenced in:
5780 * 5.4.4. Authenticator Selection Criteria (dictionary
5781 AuthenticatorSelectionCriteria) (2)
5782 * 5.4.5. Authenticator Attachment enumeration (enum
5783 AuthenticatorAttachment) (2)
5784
5785 #platform-authenticatorsReferenced in:
5786 * 5.1.6. Platform Authenticator Availability - PublicKeyCredential's
5787 isPlatformAuthenticatorAvailable() method (2) (3) (4) (5)
5788 * 5.4.5. Authenticator Attachment enumeration (enum
5789 AuthenticatorAttachment) (2)
5790 * 6.2.1. The authenticatorMakeCredential operation

```

```

5708 * 12.1. Registration
5709 * 12.2. Registration Specifically with Platform Authenticator (2)
5710
5711 #roaming-authenticatorsReferenced in:
5712 * 1.1.3. Other use cases and configurations
5713 * 5.4.5. Authenticator Attachment enumeration (enum
5714 AuthenticatorAttachment) (2)
5715 * 12.1. Registration
5716
5717 #platform-attachmentReferenced in:
5718 * 5.4.5. Authenticator Attachment enumeration (enum
5719 AuthenticatorAttachment)
5720
5721 #cross-platform-attachedReferenced in:
5722 * 5.4.5. Authenticator Attachment enumeration (enum
5723 AuthenticatorAttachment) (2)
5724
5725 #dictdef-publickeycredentialrequestoptionsReferenced in:
5726 * 5.1.2. CredentialRequestOptions Extension
5727 * 5.5. Options for Assertion Generation (dictionary
5728 PublicKeyCredentialRequestOptions) (2)
5729 * 7.2. Verifying an authentication assertion
5730
5731 #dom-publickeycredentialrequestoptions-challengeReferenced in:
5732 * 5.1.4.1. PublicKeyCredential's
5733 [[DiscoverFromExternalSource]](options) method
5734 * 5.5. Options for Assertion Generation (dictionary
5735 PublicKeyCredentialRequestOptions) (2)
5736
5737 #dom-publickeycredentialrequestoptions-timeoutReferenced in:
5738 * 5.1.4.1. PublicKeyCredential's
5739 [[DiscoverFromExternalSource]](options) method (2)
5740 * 5.5. Options for Assertion Generation (dictionary
5741 PublicKeyCredentialRequestOptions)
5742
5743 #dom-publickeycredentialrequestoptions-rpidReferenced in:
5744 * 5.1.4.1. PublicKeyCredential's
5745 [[DiscoverFromExternalSource]](options) method (2) (3) (4)
5746 * 5.5. Options for Assertion Generation (dictionary
5747 PublicKeyCredentialRequestOptions)
5748 * 10.1. FIDO Appid Extension (appid)
5749
5750 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:
5751 * 5.1.4.1. PublicKeyCredential's
5752 [[DiscoverFromExternalSource]](options) method (2) (3) (4)
5753 * 5.5. Options for Assertion Generation (dictionary
5754 PublicKeyCredentialRequestOptions)
5755
5756 #dom-publickeycredentialrequestoptions-extensionsReferenced in:
5757 * 5.1.4.1. PublicKeyCredential's
5758 [[DiscoverFromExternalSource]](options) method (2)
5759 * 5.5. Options for Assertion Generation (dictionary
5760 PublicKeyCredentialRequestOptions)
5761
5762 #typedefdef-authenticationextensionsReferenced in:
5763 * 5.1. PublicKeyCredential Interface (2)
5764 * 5.1.3. Create a new credential - PublicKeyCredential's
5765 [[Create]](options) method
5766 * 5.1.4.1. PublicKeyCredential's
5767 [[DiscoverFromExternalSource]](options) method
5768 * 5.4. Options for Credential Creation (dictionary
5769 MakePublicKeyCredentialOptions) (2)
5770 * 5.5. Options for Assertion Generation (dictionary
5771 PublicKeyCredentialRequestOptions) (2)
5772 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5773 CollectedClientData) (2)
5774
5775 #dictdef-collectedclientdataReferenced in:
5776 * 5.1.3. Create a new credential - PublicKeyCredential's
5777 [[Create]](options) method

```

```

5791 * 12.1. Registration
5792 * 12.2. Registration Specifically with Platform Authenticator (2)
5793
5794 #roaming-authenticatorsReferenced in:
5795 * 1.1.3. Other use cases and configurations
5796 * 5.4.5. Authenticator Attachment enumeration (enum
5797 AuthenticatorAttachment) (2)
5798 * 12.1. Registration
5799
5800 #platform-attachmentReferenced in:
5801 * 5.4.5. Authenticator Attachment enumeration (enum
5802 AuthenticatorAttachment)
5803
5804 #cross-platform-attachedReferenced in:
5805 * 5.4.5. Authenticator Attachment enumeration (enum
5806 AuthenticatorAttachment) (2)
5807
5808 #dictdef-publickeycredentialrequestoptionsReferenced in:
5809 * 5.1.2. CredentialRequestOptions Extension
5810 * 5.5. Options for Assertion Generation (dictionary
5811 PublicKeyCredentialRequestOptions) (2)
5812 * 7.2. Verifying an authentication assertion
5813
5814 #dom-publickeycredentialrequestoptions-challengeReferenced in:
5815 * 5.1.4.1. PublicKeyCredential's
5816 [[DiscoverFromExternalSource]](options) method
5817 * 5.5. Options for Assertion Generation (dictionary
5818 PublicKeyCredentialRequestOptions) (2)
5819
5820 #dom-publickeycredentialrequestoptions-timeoutReferenced in:
5821 * 5.1.4.1. PublicKeyCredential's
5822 [[DiscoverFromExternalSource]](options) method (2)
5823 * 5.5. Options for Assertion Generation (dictionary
5824 PublicKeyCredentialRequestOptions)
5825
5826 #dom-publickeycredentialrequestoptions-rpidReferenced in:
5827 * 5.1.4.1. PublicKeyCredential's
5828 [[DiscoverFromExternalSource]](options) method (2) (3) (4)
5829 * 5.5. Options for Assertion Generation (dictionary
5830 PublicKeyCredentialRequestOptions)
5831 * 10.1. FIDO Appid Extension (appid)
5832
5833 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:
5834 * 5.1.4.1. PublicKeyCredential's
5835 [[DiscoverFromExternalSource]](options) method (2) (3) (4)
5836 * 5.5. Options for Assertion Generation (dictionary
5837 PublicKeyCredentialRequestOptions)
5838
5839 #dom-publickeycredentialrequestoptions-extensionsReferenced in:
5840 * 5.1.4.1. PublicKeyCredential's
5841 [[DiscoverFromExternalSource]](options) method (2)
5842 * 5.5. Options for Assertion Generation (dictionary
5843 PublicKeyCredentialRequestOptions)
5844
5845 #typedefdef-authenticationextensionsReferenced in:
5846 * 5.1. PublicKeyCredential Interface (2)
5847 * 5.1.3. Create a new credential - PublicKeyCredential's
5848 [[Create]](options) method (2)
5849 * 5.1.4.1. PublicKeyCredential's
5850 [[DiscoverFromExternalSource]](options) method
5851 * 5.4. Options for Credential Creation (dictionary
5852 MakePublicKeyCredentialOptions) (2)
5853 * 5.5. Options for Assertion Generation (dictionary
5854 PublicKeyCredentialRequestOptions) (2)
5855 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5856 CollectedClientData) (2)
5857
5858 #dictdef-collectedclientdataReferenced in:
5859 * 5.1.3. Create a new credential - PublicKeyCredential's
5860 [[Create]](options) method

```

577f * 5.1.4.1. PublicKeyCredential's
577f [[DiscoverFromExternalSource]](options) method
578c * 5.7.1. Client data used in WebAuthn signatures (dictionary
5781 CollectedClientData) (2)
5782
5783 #client-dataReferenced in:
5784 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
5785 * 6. WebAuthn Authenticator model (2) (3) (4)
5786 * 6.1. Authenticator data (2)
5787 * 7.1. Registering a new credential
5788 * 7.2. Verifying an authentication assertion
5789 * 9. WebAuthn Extensions
579c * 9.4. Client extension processing
5791 * 9.6. Example Extension
5792
5793 #dom-collectedclientdata-challengeReferenced in:
5794 * 5.1.3. Create a new credential - PublicKeyCredential's
5795 [[Create]](options) method
5796 * 5.1.4.1. PublicKeyCredential's
5797 [[DiscoverFromExternalSource]](options) method
5798 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5799 CollectedClientData)
580c * 7.1. Registering a new credential
5801 * 7.2. Verifying an authentication assertion
5802
5803 #dom-collectedclientdata-originReferenced in:
5804 * 5.1.3. Create a new credential - PublicKeyCredential's
5805 [[Create]](options) method
5806 * 5.1.4.1. PublicKeyCredential's
5807 [[DiscoverFromExternalSource]](options) method
5808 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5809 CollectedClientData)
581c * 7.1. Registering a new credential
5811 * 7.2. Verifying an authentication assertion
5812
5813 #dom-collectedclientdata-hashalgorithmReferenced in:
5814 * 5.1.3. Create a new credential - PublicKeyCredential's
5815 [[Create]](options) method
5816 * 5.1.4.1. PublicKeyCredential's
5817 [[DiscoverFromExternalSource]](options) method
5818 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5819 CollectedClientData) (2)
582c * 7.1. Registering a new credential
5821 * 7.2. Verifying an authentication assertion
5822
5823 #dom-collectedclientdata-tokenbindingidReferenced in:
5824 * 5.1.3. Create a new credential - PublicKeyCredential's
5825 [[Create]](options) method
5826 * 5.1.4.1. PublicKeyCredential's
5827 [[DiscoverFromExternalSource]](options) method
5828 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5829 CollectedClientData)
583c * 7.1. Registering a new credential
5831 * 7.2. Verifying an authentication assertion
5832
5833 #dom-collectedclientdata-clientextensionsReferenced in:
5834 * 5.1.3. Create a new credential - PublicKeyCredential's
5835 [[Create]](options) method
5836 * 5.1.4.1. PublicKeyCredential's
5837 [[DiscoverFromExternalSource]](options) method
5838 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5839 CollectedClientData)
584c * 7.1. Registering a new credential
5841 * 7.2. Verifying an authentication assertion
5842 * 9.4. Client extension processing
5843
5844 #dom-collectedclientdata-authenticatorextensionsReferenced in:
5845 * 5.1.3. Create a new credential - PublicKeyCredential's
5846 [[Create]](options) method
5847 * 5.1.4.1. PublicKeyCredential's

5861 * 5.1.4.1. PublicKeyCredential's
5862 [[DiscoverFromExternalSource]](options) method
5863 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5864 CollectedClientData) (2)
5865
5866 #client-dataReferenced in:
5867 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
5868 * 6. WebAuthn Authenticator model (2) (3) (4)
5869 * 6.1. Authenticator data (2)
587c * 7.1. Registering a new credential
5871 * 7.2. Verifying an authentication assertion
5872 * 9. WebAuthn Extensions
5873 * 9.4. Client extension processing
5874 * 9.6. Example Extension
5875
5876 #dom-collectedclientdata-challengeReferenced in:
5877 * 5.1.3. Create a new credential - PublicKeyCredential's
5878 [[Create]](options) method
5879 * 5.1.4.1. PublicKeyCredential's
588c [[DiscoverFromExternalSource]](options) method
5881 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5882 CollectedClientData)
5883 * 7.1. Registering a new credential
5884 * 7.2. Verifying an authentication assertion
5885
5886 #dom-collectedclientdata-originReferenced in:
5887 * 5.1.3. Create a new credential - PublicKeyCredential's
5888 [[Create]](options) method
5889 * 5.1.4.1. PublicKeyCredential's
589c [[DiscoverFromExternalSource]](options) method
5891 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5892 CollectedClientData)
5893 * 7.1. Registering a new credential
5894 * 7.2. Verifying an authentication assertion
5895
5896 #dom-collectedclientdata-hashalgorithmReferenced in:
5897 * 5.1.3. Create a new credential - PublicKeyCredential's
5898 [[Create]](options) method
5899 * 5.1.4.1. PublicKeyCredential's
590c [[DiscoverFromExternalSource]](options) method
5901 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5902 CollectedClientData) (2)
5903 * 7.1. Registering a new credential
5904 * 7.2. Verifying an authentication assertion
5905
5906 #dom-collectedclientdata-tokenbindingidReferenced in:
5907 * 5.1.3. Create a new credential - PublicKeyCredential's
5908 [[Create]](options) method
5909 * 5.1.4.1. PublicKeyCredential's
591c [[DiscoverFromExternalSource]](options) method
5911 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5912 CollectedClientData)
5913 * 7.1. Registering a new credential
5914 * 7.2. Verifying an authentication assertion
5915
5916 #dom-collectedclientdata-clientextensionsReferenced in:
5917 * 5.1.3. Create a new credential - PublicKeyCredential's
5918 [[Create]](options) method
5919 * 5.1.4.1. PublicKeyCredential's
592c [[DiscoverFromExternalSource]](options) method
5921 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5922 CollectedClientData)
5923 * 7.1. Registering a new credential
5924 * 7.2. Verifying an authentication assertion
5925 * 9.4. Client extension processing
5926
5927 #dom-collectedclientdata-authenticatorextensionsReferenced in:
5928 * 5.1.3. Create a new credential - PublicKeyCredential's
5929 [[Create]](options) method
593c * 5.1.4.1. PublicKeyCredential's

```

5848 [[DiscoverFromExternalSource]](options) method
5849 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5850 CollectedClientData)
5851 * 7.1. Registering a new credential
5852 * 7.2. Verifying an authentication assertion
5853
5854 #collectedclientdata-json-serialized-client-dataReferenced in:
5855 * 5.1.3. Create a new credential - PublicKeyCredential's
5856 [[Create]](options) method
5857 * 5.1.4.1. PublicKeyCredential's
5858 [[DiscoverFromExternalSource]](options) method
5859 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
5860 * 5.2.1. Information about Public Key Credential (interface
5861 AuthenticatorAttestationResponse) (2)
5862 * 5.2.2. Web Authentication Assertion (interface
5863 AuthenticatorAssertionResponse)
5864 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5865 CollectedClientData)
5866
5867 #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:
5868 * 5.1.3. Create a new credential - PublicKeyCredential's
5869 [[Create]](options) method (2)
5870 * 5.1.4.1. PublicKeyCredential's
5871 [[DiscoverFromExternalSource]](options) method (2)
5872 * 5.2.1. Information about Public Key Credential (interface
5873 AuthenticatorAttestationResponse)
5874 * 5.2.2. Web Authentication Assertion (interface
5875 AuthenticatorAssertionResponse)
5876 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5877 CollectedClientData)
5878 * 6. WebAuthn Authenticator model
5879 * 6.2.1. The authenticatorMakeCredential operation
5880 * 6.2.2. The authenticatorGetAssertion operation (2)
5881 * 6.3.2. Attestation Statement Formats (2)
5882 * 6.3.4. Generating an Attestation Object
5883 * 7.1. Registering a new credential
5884 * 8.2. Packed Attestation Statement Format (2)
5885 * 8.3. TPM Attestation Statement Format (2)
5886 * 8.4. Android Key Attestation Statement Format (2)
5887 * 8.5. Android SafetyNet Attestation Statement Format
5888 * 8.6. FIDO U2F Attestation Statement Format (2)
5889
5890 #enumdef-publickeycredentialtypeReferenced in:
5891 * 5.1.3. Create a new credential - PublicKeyCredential's
5892 [[Create]](options) method (2)
5893 * 5.3. Parameters for Credential Generation (dictionary
5894 PublicKeyCredentialParameters)
5895 * 5.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
5896 * 5.7.3. Credential Descriptor (dictionary
5897 PublicKeyCredentialDescriptor)
5898 * 6.2.1. The authenticatorMakeCredential operation (2) (3)
5899
5900 #dom-publickeycredentialtype-public-keyReferenced in:
5901 * 5.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
5902
5903 #dictdef-publickeycredentialdescriptorReferenced in:
5904 * 5.4. Options for Credential Creation (dictionary
5905 MakePublicKeyCredentialOptions) (2)
5906 * 5.5. Options for Assertion Generation (dictionary
5907 PublicKeyCredentialRequestOptions) (2) (3)
5908 * 5.7.3. Credential Descriptor (dictionary
5909 PublicKeyCredentialDescriptor)
5910 * 6.2.1. The authenticatorMakeCredential operation
5911 * 6.2.2. The authenticatorGetAssertion operation
5912
5913 #dom-publickeycredentialdescriptor-transportReferenced in:
5914 * 5.1.3. Create a new credential - PublicKeyCredential's
5915 [[Create]](options) method (2)
5916 * 5.1.4.1. PublicKeyCredential's
5917 [[DiscoverFromExternalSource]](options) method (2)

```

```

5931 [[DiscoverFromExternalSource]](options) method
5932 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5933 CollectedClientData)
5934 * 7.1. Registering a new credential
5935 * 7.2. Verifying an authentication assertion
5936
5937 #collectedclientdata-json-serialized-client-dataReferenced in:
5938 * 5.1.3. Create a new credential - PublicKeyCredential's
5939 [[Create]](options) method
5940 * 5.1.4.1. PublicKeyCredential's
5941 [[DiscoverFromExternalSource]](options) method
5942 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
5943 * 5.2.1. Information about Public Key Credential (interface
5944 AuthenticatorAttestationResponse) (2)
5945 * 5.2.2. Web Authentication Assertion (interface
5946 AuthenticatorAssertionResponse)
5947 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5948 CollectedClientData)
5949
5950 #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:
5951 * 5.1.3. Create a new credential - PublicKeyCredential's
5952 [[Create]](options) method (2)
5953 * 5.1.4.1. PublicKeyCredential's
5954 [[DiscoverFromExternalSource]](options) method (2)
5955 * 5.2.1. Information about Public Key Credential (interface
5956 AuthenticatorAttestationResponse)
5957 * 5.2.2. Web Authentication Assertion (interface
5958 AuthenticatorAssertionResponse)
5959 * 5.7.1. Client data used in WebAuthn signatures (dictionary
5960 CollectedClientData)
5961 * 6. WebAuthn Authenticator model
5962 * 6.2.1. The authenticatorMakeCredential operation
5963 * 6.2.2. The authenticatorGetAssertion operation (2)
5964 * 6.3.2. Attestation Statement Formats (2)
5965 * 6.3.4. Generating an Attestation Object
5966 * 7.1. Registering a new credential
5967 * 8.2. Packed Attestation Statement Format (2)
5968 * 8.3. TPM Attestation Statement Format (2)
5969 * 8.4. Android Key Attestation Statement Format (2)
5970 * 8.5. Android SafetyNet Attestation Statement Format
5971 * 8.6. FIDO U2F Attestation Statement Format (2)
5972
5973 #enumdef-publickeycredentialtypeReferenced in:
5974 * 5.1.3. Create a new credential - PublicKeyCredential's
5975 [[Create]](options) method (2)
5976 * 5.3. Parameters for Credential Generation (dictionary
5977 PublicKeyCredentialParameters)
5978 * 5.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
5979 * 5.7.3. Credential Descriptor (dictionary
5980 PublicKeyCredentialDescriptor)
5981 * 6.2.1. The authenticatorMakeCredential operation (2) (3)
5982
5983 #dom-publickeycredentialtype-public-keyReferenced in:
5984 * 5.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
5985
5986 #dictdef-publickeycredentialdescriptorReferenced in:
5987 * 5.4. Options for Credential Creation (dictionary
5988 MakePublicKeyCredentialOptions) (2)
5989 * 5.5. Options for Assertion Generation (dictionary
5990 PublicKeyCredentialRequestOptions) (2) (3)
5991 * 5.7.3. Credential Descriptor (dictionary
5992 PublicKeyCredentialDescriptor)
5993 * 6.2.1. The authenticatorMakeCredential operation
5994 * 6.2.2. The authenticatorGetAssertion operation
5995
5996 #dom-publickeycredentialdescriptor-transportReferenced in:
5997 * 5.1.3. Create a new credential - PublicKeyCredential's
5998 [[Create]](options) method (2)
5999 * 5.1.4.1. PublicKeyCredential's
6000 [[DiscoverFromExternalSource]](options) method (2)

```

```

5918 #dom-publickeycredentialdescriptor-typeReferenced in:
5919 * 5.1.4.1. PublicKeyCredential's
5920 [[DiscoverFromExternalSource]](options) method
5921 * 5.7.3. Credential Descriptor (dictionary
5922 PublicKeyCredentialDescriptor)
5923
5924 #dom-publickeycredentialdescriptor-idReferenced in:
5925 * 5.1.4.1. PublicKeyCredential's
5926 [[DiscoverFromExternalSource]](options) method
5927 * 5.7.3. Credential Descriptor (dictionary
5928 PublicKeyCredentialDescriptor)
5929
5930 #enumdef-authenticatortransportReferenced in:
5931 * 5.7.3. Credential Descriptor (dictionary
5932 PublicKeyCredentialDescriptor)
5933 * 5.7.4. Authenticator Transport enumeration (enum
5934 AuthenticatorTransport)
5935
5936 #dom-authenticatortransport-usbReferenced in:
5937 * 5.7.4. Authenticator Transport enumeration (enum
5938 AuthenticatorTransport)
5939
5940 #dom-authenticatortransport-nfcReferenced in:
5941 * 5.7.4. Authenticator Transport enumeration (enum
5942 AuthenticatorTransport)
5943
5944 #dom-authenticatortransport-bleReferenced in:
5945 * 5.7.4. Authenticator Transport enumeration (enum
5946 AuthenticatorTransport)
5947
5948 #typedefdef-cosealgorithmidentifierReferenced in:
5949 * 5.1.3. Create a new credential - PublicKeyCredential's
5950 [[Create]](options) method
5951 * 5.3. Parameters for Credential Generation (dictionary
5952 PublicKeyCredentialParameters)
5953 * 5.7.5. Cryptographic Algorithm Identifier (typedef
5954 COSEAlgorithmIdentifier)
5955 * 6.2.1. The authenticatorMakeCredential operation
5956 * 6.3.1. Attested credential data
5957 * 8.2. Packed Attestation Statement Format
5958 * 8.3. TPM Attestation Statement Format
5959
5960 #attestation-signatureReferenced in:
5961 * 4. Terminology
5962 * 6. WebAuthn Authenticator model (2) (3)
5963 * 6.3. Attestation
5964 * 8.6. FIDO U2F Attestation Statement Format
5965
5966 #assertion-signatureReferenced in:
5967 * 6. WebAuthn Authenticator model (2)
5968 * 6.2.2. The authenticatorGetAssertion operation (2) (3)
5969
5970 #authenticator-dataReferenced in:
5971 * 5.2.1. Information about Public Key Credential (interface
5972 AuthenticatorAttestationResponse) (2)
5973 * 5.2.2. Web Authentication Assertion (interface
5974 AuthenticatorAssertionResponse)
5975 * 6. WebAuthn Authenticator model (2)
5976 * 6.1. Authenticator data (2) (3) (4) (5) (6) (7) (8) (9)
5977 * 6.1.1. Signature Counter Considerations (2)
5978 * 6.2.1. The authenticatorMakeCredential operation
5979 * 6.2.2. The authenticatorGetAssertion operation
5980 * 6.3. Attestation (2)
5981 * 6.3.1. Attested credential data
5982 * 6.3.2. Attestation Statement Formats (2)
5983 * 6.3.4. Generating an Attestation Object
5984 * 6.3.5.3. Attestation Certificate Hierarchy
5985 * 7.1. Registering a new credential (2)
5986 * 8.5. Android SafetyNet Attestation Statement Format
5987

```

```

6001 #dom-publickeycredentialdescriptor-typeReferenced in:
6002 * 5.1.4.1. PublicKeyCredential's
6003 [[DiscoverFromExternalSource]](options) method
6004 * 5.7.3. Credential Descriptor (dictionary
6005 PublicKeyCredentialDescriptor)
6006
6007 #dom-publickeycredentialdescriptor-idReferenced in:
6008 * 5.1.4.1. PublicKeyCredential's
6009 [[DiscoverFromExternalSource]](options) method
6010 * 5.7.3. Credential Descriptor (dictionary
6011 PublicKeyCredentialDescriptor)
6012
6013 #enumdef-authenticatortransportReferenced in:
6014 * 5.7.3. Credential Descriptor (dictionary
6015 PublicKeyCredentialDescriptor)
6016 * 5.7.4. Authenticator Transport enumeration (enum
6017 AuthenticatorTransport)
6018
6019 #dom-authenticatortransport-usbReferenced in:
6020 * 5.7.4. Authenticator Transport enumeration (enum
6021 AuthenticatorTransport)
6022
6023 #dom-authenticatortransport-nfcReferenced in:
6024 * 5.7.4. Authenticator Transport enumeration (enum
6025 AuthenticatorTransport)
6026
6027 #dom-authenticatortransport-bleReferenced in:
6028 * 5.7.4. Authenticator Transport enumeration (enum
6029 AuthenticatorTransport)
6030
6031 #typedefdef-cosealgorithmidentifierReferenced in:
6032 * 5.1.3. Create a new credential - PublicKeyCredential's
6033 [[Create]](options) method
6034 * 5.3. Parameters for Credential Generation (dictionary
6035 PublicKeyCredentialParameters)
6036 * 5.7.5. Cryptographic Algorithm Identifier (typedef
6037 COSEAlgorithmIdentifier)
6038 * 6.2.1. The authenticatorMakeCredential operation
6039 * 6.3.1. Attested credential data
6040 * 8.2. Packed Attestation Statement Format
6041 * 8.3. TPM Attestation Statement Format
6042
6043 #attestation-signatureReferenced in:
6044 * 4. Terminology
6045 * 6. WebAuthn Authenticator model (2) (3)
6046 * 6.3. Attestation
6047 * 8.6. FIDO U2F Attestation Statement Format
6048
6049 #assertion-signatureReferenced in:
6050 * 6. WebAuthn Authenticator model (2)
6051 * 6.2.2. The authenticatorGetAssertion operation (2) (3)
6052
6053 #authenticator-dataReferenced in:
6054 * 5.2.1. Information about Public Key Credential (interface
6055 AuthenticatorAttestationResponse) (2)
6056 * 5.2.2. Web Authentication Assertion (interface
6057 AuthenticatorAssertionResponse)
6058 * 6. WebAuthn Authenticator model (2)
6059 * 6.1. Authenticator data (2) (3) (4) (5) (6) (7) (8) (9)
6060 * 6.1.1. Signature Counter Considerations (2)
6061 * 6.2.1. The authenticatorMakeCredential operation
6062 * 6.2.2. The authenticatorGetAssertion operation
6063 * 6.3. Attestation (2)
6064 * 6.3.1. Attested credential data
6065 * 6.3.2. Attestation Statement Formats (2)
6066 * 6.3.4. Generating an Attestation Object
6067 * 6.3.5.3. Attestation Certificate Hierarchy
6068 * 7.1. Registering a new credential (2)
6069 * 8.5. Android SafetyNet Attestation Statement Format
6070

```

598E * 9.5. Authenticator extension processing
598F * 9.6. Example Extension (2)
5990 * 10.6. User Verification Index Extension (uvi)
5991 * 10.7. Location Extension (loc)
5992 * 10.8. User Verification Method Extension (uvm)
5993
5994 #rpidhashReferenced in:
5995 * 7.2. Verifying an authentication assertion
5996
5997 #flagsReferenced in:
5998 * 6.1. Authenticator data
5999
6000 #signcountReferenced in:
6001 * 6.1.1. Signature Counter Considerations (2)
6002 * 7.2. Verifying an authentication assertion (2) (3)
6003
6004 #attestedcredentialdataReferenced in:
6005 * 5.1.3. Create a new credential - PublicKeyCredential's
6006 [[Create]](options) method
6007 * 6.1. Authenticator data (2)
6008 * 6.2.1. The authenticatorMakeCredential operation
6009 * 6.2.2. The authenticatorGetAssertion operation
6010 * 7.1. Registering a new credential (2)
6011 * 8.3. TPM Attestation Statement Format
6012 * 8.4. Android Key Attestation Statement Format
6013 * 8.6. FIDO U2F Attestation Statement Format
6014
6015 #authdataextensionsReferenced in:
6016 * 6.1. Authenticator data
6017 * 6.2.1. The authenticatorMakeCredential operation
6018 * 6.2.2. The authenticatorGetAssertion operation
6019
6020 #signature-counterReferenced in:
6021 * 6.1. Authenticator data
6022 * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6) (7) (8)
6023 (9) (10)
6024 * 6.2.1. The authenticatorMakeCredential operation (2) (3) (4)
6025 * 6.2.2. The authenticatorGetAssertion operation (2)
6026 * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6)
6027
6028 #authenticatormakecredentialReferenced in:
6029 * 4. Terminology (2) (3)
6030 * 5.1.3. Create a new credential - PublicKeyCredential's
6031 [[Create]](options) method (2)
6032 * 6. WebAuthn Authenticator model
6033 * 6.2.3. The authenticatorCancel operation (2)
6034 * 9. WebAuthn Extensions
6035 * 9.2. Defining extensions
6036
6037 #authenticatorgetassertionReferenced in:
6038 * 4. Terminology (2) (3)
6039 * 5.1.4.1. PublicKeyCredential's
6040 [[DiscoverFromExternalSource]](options) method (2) (3) (4) (5)
6041 * 6. WebAuthn Authenticator model
6042 * 6.1. Authenticator data
6043 * 6.1.1. Signature Counter Considerations (2) (3)
6044 * 6.2.3. The authenticatorCancel operation (2)
6045 * 9. WebAuthn Extensions
6046 * 9.2. Defining extensions
6047
6048 #authenticatorcancelReferenced in:
6049 * 5.1.3. Create a new credential - PublicKeyCredential's
6050 [[Create]](options) method (2) (3)
6051 * 5.1.4.1. PublicKeyCredential's
6052 [[DiscoverFromExternalSource]](options) method (2) (3)
6053
6054 #attestation-objectReferenced in:
6055 * 4. Terminology
6056 * 5. Web Authentication API
6057 * 5.2.1. Information about Public Key Credential (interface)

6071 * 9.5. Authenticator extension processing
6072 * 9.6. Example Extension (2)
6073 * 10.6. User Verification Index Extension (uvi)
6074 * 10.7. Location Extension (loc)
6075 * 10.8. User Verification Method Extension (uvm)
6076
6077 #rpidhashReferenced in:
6078 * 7.2. Verifying an authentication assertion
6079
6080 #flagsReferenced in:
6081 * 6.1. Authenticator data
6082
6083 #signcountReferenced in:
6084 * 6.1.1. Signature Counter Considerations (2)
6085 * 7.2. Verifying an authentication assertion (2) (3)
6086
6087 #attestedcredentialdataReferenced in:
6088 * 5.1.3. Create a new credential - PublicKeyCredential's
6089 [[Create]](options) method
6090 * 6.1. Authenticator data (2)
6091 * 6.2.1. The authenticatorMakeCredential operation
6092 * 6.2.2. The authenticatorGetAssertion operation
6093 * 7.1. Registering a new credential (2)
6094 * 8.3. TPM Attestation Statement Format
6095 * 8.4. Android Key Attestation Statement Format
6096 * 8.6. FIDO U2F Attestation Statement Format
6097
6098 #authdataextensionsReferenced in:
6099 * 6.1. Authenticator data
6100 * 6.2.1. The authenticatorMakeCredential operation
6101 * 6.2.2. The authenticatorGetAssertion operation
6102
6103 #signature-counterReferenced in:
6104 * 6.1. Authenticator data
6105 * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6) (7) (8)
6106 (9) (10)
6107 * 6.2.1. The authenticatorMakeCredential operation (2) (3) (4)
6108 * 6.2.2. The authenticatorGetAssertion operation (2)
6109 * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6)
6110
6111 #authenticatormakecredentialReferenced in:
6112 * 4. Terminology (2) (3)
6113 * 5.1.3. Create a new credential - PublicKeyCredential's
6114 [[Create]](options) method (2)
6115 * 6. WebAuthn Authenticator model
6116 * 6.2.3. The authenticatorCancel operation (2)
6117 * 9. WebAuthn Extensions
6118 * 9.2. Defining extensions
6119
6120 #authenticatorgetassertionReferenced in:
6121 * 4. Terminology (2) (3)
6122 * 5.1.4.1. PublicKeyCredential's
6123 [[DiscoverFromExternalSource]](options) method (2) (3) (4) (5)
6124 * 6. WebAuthn Authenticator model
6125 * 6.1. Authenticator data
6126 * 6.1.1. Signature Counter Considerations (2) (3)
6127 * 6.2.3. The authenticatorCancel operation (2)
6128 * 9. WebAuthn Extensions
6129 * 9.2. Defining extensions
6130
6131 #authenticatorcancelReferenced in:
6132 * 5.1.3. Create a new credential - PublicKeyCredential's
6133 [[Create]](options) method (2) (3)
6134 * 5.1.4.1. PublicKeyCredential's
6135 [[DiscoverFromExternalSource]](options) method (2) (3)
6136
6137 #attestation-objectReferenced in:
6138 * 4. Terminology
6139 * 5. Web Authentication API
6140 * 5.2.1. Information about Public Key Credential (interface)

605E AuthenticatorAttestationResponse) (2)
 605F * 5.4. Options for Credential Creation (dictionary)
 606C MakePublicKeyCredentialOptions) (2)
 * 6.2.1. The authenticatorMakeCredential operation (2)
 6062 * 6.3. Attestation (2) (3)
 6063 * 6.3.1. Attested credential data
 6064 * 6.3.4. Generating an Attestation Object (2)
 6065 * 7.1. Registering a new credential
 6066
 6067 #attestation-statementReferenced in:
 6068 * 4. Terminology
 6069 * 5.2.1. Information about Public Key Credential (interface
 607C AuthenticatorAttestationResponse) (2) (3)
 6071 * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8)
 6072 * 6.3.2. Attestation Statement Formats (2) (3)
 6073 * 7.1. Registering a new credential
 6074
 6075 #attestation-statement-formatReferenced in:
 6076 * 5.2.1. Information about Public Key Credential (interface
 6077 AuthenticatorAttestationResponse)
 6078 * 5.7.4. Authenticator Transport enumeration (enum
 6079 AuthenticatorTransport)
 608C * 6.2.1. The authenticatorMakeCredential operation
 6081 * 6.3. Attestation (2) (3) (4) (5) (6) (7)
 6082 * 6.3.2. Attestation Statement Formats (2) (3) (4)
 6083 * 6.3.4. Generating an Attestation Object
 6084 * 7.1. Registering a new credential
 6085
 6086 #attestation-typeReferenced in:
 6087 * 6.3. Attestation (2) (3) (4) (5) (6)
 6088 * 6.3.2. Attestation Statement Formats
 6089
 609C #attested-credential-dataReferenced in:
 6091 * 6.1. Authenticator data (2) (3) (4) (5)
 6092 * 6.2.1. The authenticatorMakeCredential operation
 6093 * 6.3. Attestation (2)
 6094 * 6.3.1. Attested credential data
 6095 * 6.3.3. Attestation Types
 6096
 6097 #aaguidReferenced in:
 6098 * 7.1. Registering a new credential
 6099 * 8.2. Packed Attestation Statement Format
 610C * 8.3. TPM Attestation Statement Format
 6101
 6102 #credentialidlengthReferenced in:
 6103 * 6.1. Authenticator data
 6104
 6105 #credentialidReferenced in:
 6106 * 5.1.3. Create a new credential - PublicKeyCredential's
 6107 [[Create]](options) method
 6108 * 6.1. Authenticator data
 6109 * 7.1. Registering a new credential
 6110
 6111 #credentialpublickeyReferenced in:
 6112 * 6.1. Authenticator data
 6113 * 7.1. Registering a new credential
 6114 * 8.2. Packed Attestation Statement Format
 6115 * 8.3. TPM Attestation Statement Format
 6116 * 8.4. Android Key Attestation Statement Format
 6117
 6118 #signing-procedureReferenced in:
 6119 * 6.3.2. Attestation Statement Formats
 612C * 6.3.4. Generating an Attestation Object
 6121
 6122 #authenticator-data-for-the-attestationReferenced in:
 6123 * 8.2. Packed Attestation Statement Format
 6124 * 8.3. TPM Attestation Statement Format
 6125 * 8.4. Android Key Attestation Statement Format (2)
 6126 * 8.5. Android SafetyNet Attestation Statement Format
 6127 * 8.6. FIDO U2F Attestation Statement Format

6141 AuthenticatorAttestationResponse) (2)
 6142 * 5.4. Options for Credential Creation (dictionary)
 6143 MakePublicKeyCredentialOptions) (2)
 6144 * 6.2.1. The authenticatorMakeCredential operation (2)
 6145 * 6.3. Attestation (2) (3)
 6146 * 6.3.1. Attested credential data
 6147 * 6.3.4. Generating an Attestation Object (2)
 6148 * 7.1. Registering a new credential
 6149
 615C #attestation-statementReferenced in:
 6151 * 4. Terminology
 6152 * 5.2.1. Information about Public Key Credential (interface
 6153 AuthenticatorAttestationResponse) (2) (3)
 6154 * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8)
 6155 * 6.3.2. Attestation Statement Formats (2) (3)
 6156 * 7.1. Registering a new credential
 6157
 6158 #attestation-statement-formatReferenced in:
 6159 * 5.2.1. Information about Public Key Credential (interface
 616C AuthenticatorAttestationResponse)
 6161 * 5.7.4. Authenticator Transport enumeration (enum
 6162 AuthenticatorTransport)
 6163 * 6.2.1. The authenticatorMakeCredential operation
 6164 * 6.3. Attestation (2) (3) (4) (5) (6) (7)
 6165 * 6.3.2. Attestation Statement Formats (2) (3) (4)
 6166 * 6.3.4. Generating an Attestation Object
 6167 * 7.1. Registering a new credential
 6168
 6169 #attestation-typeReferenced in:
 617C * 6.3. Attestation (2) (3) (4) (5) (6)
 6171 * 6.3.2. Attestation Statement Formats
 6172
 6173 #attested-credential-dataReferenced in:
 6174 * 6.1. Authenticator data (2) (3) (4) (5)
 6175 * 6.2.1. The authenticatorMakeCredential operation
 6176 * 6.3. Attestation (2)
 6177 * 6.3.1. Attested credential data
 6178 * 6.3.3. Attestation Types
 6179
 618C #aaguidReferenced in:
 6181 * 7.1. Registering a new credential
 6182 * 8.2. Packed Attestation Statement Format
 6183 * 8.3. TPM Attestation Statement Format
 6184
 6185 #credentialidlengthReferenced in:
 6186 * 6.1. Authenticator data
 6187
 6188 #credentialidReferenced in:
 6189 * 5.1.3. Create a new credential - PublicKeyCredential's
 619C [[Create]](options) method
 6191 * 6.1. Authenticator data
 6192 * 7.1. Registering a new credential
 6193
 6194 #credentialpublickeyReferenced in:
 6195 * 6.1. Authenticator data
 6196 * 7.1. Registering a new credential
 6197 * 8.2. Packed Attestation Statement Format
 6198 * 8.3. TPM Attestation Statement Format
 6199 * 8.4. Android Key Attestation Statement Format
 620C
 6201 #signing-procedureReferenced in:
 6202 * 6.3.2. Attestation Statement Formats
 6203 * 6.3.4. Generating an Attestation Object
 6204
 6205 #authenticator-data-for-the-attestationReferenced in:
 6206 * 8.2. Packed Attestation Statement Format
 6207 * 8.3. TPM Attestation Statement Format
 6208 * 8.4. Android Key Attestation Statement Format (2)
 6209 * 8.5. Android SafetyNet Attestation Statement Format
 621C * 8.6. FIDO U2F Attestation Statement Format

6128 #authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:
6129 nced in:
6130 * 8.2. Packed Attestation Statement Format
6131 * 8.3. TPM Attestation Statement Format
6132 * 8.4. Android Key Attestation Statement Format (2)
6133 * 8.6. FIDO U2F Attestation Statement Format
6134
6135 #basic-attestationReferenced in:
6136 * 6.3.5.1. Privacy
6137
6138 #self-attestationReferenced in:
6139 * 4. Terminology (2) (3) (4)
6140 * 6.3. Attestation (2)
6141 * 6.3.2. Attestation Statement Formats
6142 * 6.3.3. Attestation Types
6143 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
6144 Compromise
6145 * 7.1. Registering a new credential (2) (3)
6146 * 8.2. Packed Attestation Statement Format (2)
6147 * 8.6. FIDO U2F Attestation Statement Format
6148
6149 #privacy-caReferenced in:
6150 * 6.3.5.1. Privacy
6151
6152 #elliptic-curve-based-direct-anonymous-attestationReferenced in:
6153 * 6.3.5.1. Privacy
6154
6155 #ecdaaReferenced in:
6156 * 6.3.2. Attestation Statement Formats
6157 * 6.3.3. Attestation Types
6158 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
6159 Compromise
6160 * 7.1. Registering a new credential
6161 * 8.2. Packed Attestation Statement Format (2)
6162 * 8.3. TPM Attestation Statement Format (2)
6163
6164 #attestation-statement-format-identifierReferenced in:
6165 * 6.3.2. Attestation Statement Formats
6166 * 6.3.4. Generating an Attestation Object
6167
6168 #identifier-of-the-ecdaa-issuer-public-keyReferenced in:
6169 * 7.1. Registering a new credential
6170 * 8.2. Packed Attestation Statement Format
6171 * 8.3. TPM Attestation Statement Format (2)
6172
6173 #ecdaa-issuer-public-keyReferenced in:
6174 * 6.3.2. Attestation Statement Formats
6175 * 6.3.5.1. Privacy
6176 * 7.1. Registering a new credential
6177 * 8.2. Packed Attestation Statement Format (2) (3)
6178
6179 #registration-extensionReferenced in:
6180 * 5.1.3. Create a new credential - PublicKeyCredential's
6181 [[Create]](options) method
6182 * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
6183 * 9.6. Example Extension
6184 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
6185 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
6186 * 10.4. Authenticator Selection Extension (authnSel)
6187 * 10.5. Supported Extensions Extension (exts)
6188 * 10.6. User Verification Index Extension (uvi)
6189 * 10.7. Location Extension (loc)
6190 * 10.8. User Verification Method Extension (uvm)
6191 * 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
6192 (6) (7)
6193
6194 #authentication-extensionReferenced in:
6195 * 5.1.4.1. PublicKeyCredential's
6196 [[DiscoverFromExternalSource]](options) method
6197

6211 #authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:
6212 nced in:
6213 * 8.2. Packed Attestation Statement Format
6214 * 8.3. TPM Attestation Statement Format
6215 * 8.4. Android Key Attestation Statement Format (2)
6216 * 8.6. FIDO U2F Attestation Statement Format
6217
6218 #basic-attestationReferenced in:
6219 * 6.3.5.1. Privacy
6220
6221 #self-attestationReferenced in:
6222 * 4. Terminology (2) (3) (4)
6223 * 6.3. Attestation (2)
6224 * 6.3.2. Attestation Statement Formats
6225 * 6.3.3. Attestation Types
6226 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
6227 Compromise
6228 * 7.1. Registering a new credential (2) (3)
6229 * 8.2. Packed Attestation Statement Format (2)
6230 * 8.6. FIDO U2F Attestation Statement Format
6231
6232 #privacy-caReferenced in:
6233 * 6.3.5.1. Privacy
6234
6235 #elliptic-curve-based-direct-anonymous-attestationReferenced in:
6236 * 6.3.5.1. Privacy
6237
6238 #ecdaaReferenced in:
6239 * 6.3.2. Attestation Statement Formats
6240 * 6.3.3. Attestation Types
6241 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
6242 Compromise
6243 * 7.1. Registering a new credential
6244 * 8.2. Packed Attestation Statement Format (2)
6245 * 8.3. TPM Attestation Statement Format (2)
6246
6247 #attestation-statement-format-identifierReferenced in:
6248 * 6.3.2. Attestation Statement Formats
6249 * 6.3.4. Generating an Attestation Object
6250
6251 #identifier-of-the-ecdaa-issuer-public-keyReferenced in:
6252 * 7.1. Registering a new credential
6253 * 8.2. Packed Attestation Statement Format
6254 * 8.3. TPM Attestation Statement Format (2)
6255
6256 #ecdaa-issuer-public-keyReferenced in:
6257 * 6.3.2. Attestation Statement Formats
6258 * 6.3.5.1. Privacy
6259 * 7.1. Registering a new credential
6260 * 8.2. Packed Attestation Statement Format (2) (3)
6261
6262 #registration-extensionReferenced in:
6263 * 5.1.3. Create a new credential - PublicKeyCredential's
6264 [[Create]](options) method
6265 * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
6266 * 9.6. Example Extension
6267 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
6268 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
6269 * 10.4. Authenticator Selection Extension (authnSel)
6270 * 10.5. Supported Extensions Extension (exts)
6271 * 10.6. User Verification Index Extension (uvi)
6272 * 10.7. Location Extension (loc)
6273 * 10.8. User Verification Method Extension (uvm)
6274 * 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
6275 (6) (7)
6276
6277 #authentication-extensionReferenced in:
6278 * 5.1.4.1. PublicKeyCredential's
6279 [[DiscoverFromExternalSource]](options) method
6280

619E * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
619F * 9.6. Example Extension
620C * 10.1. FIDO Appid Extension (appid)
6201 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
6202 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
6203 * 10.6. User Verification Index Extension (uvi)
6204 * 10.7. Location Extension (loc)
6205 * 10.8. User Verification Method Extension (uvm)
6206 * 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
6207 (6)
6208
6209 #client-extensionReferenced in:
621C * 5.1.3. Create a new credential - PublicKeyCredential's
6211 [[Create]](options) method
6212 * 5.1.4.1. PublicKeyCredential's
6213 [[DiscoverFromExternalSource]](options) method
6214 * 5.6. Authentication Extensions (typedef AuthenticationExtensions)
6215 * 9. WebAuthn Extensions
6216 * 9.2. Defining extensions
6217 * 9.4. Client extension processing
6218
6219 #authenticator-extensionReferenced in:
622C * 5.1.3. Create a new credential - PublicKeyCredential's
6221 [[Create]](options) method
6222 * 5.1.4.1. PublicKeyCredential's
6223 [[DiscoverFromExternalSource]](options) method
6224 * 5.6. Authentication Extensions (typedef AuthenticationExtensions)
6225 * 9. WebAuthn Extensions (2) (3)
6226 * 9.2. Defining extensions (2)
6227 * 9.3. Extending request parameters
6228 * 9.5. Authenticator extension processing
6229
623C #extension-identifierReferenced in:
6231 * 5.1. PublicKeyCredential Interface
6232 * 5.1.3. Create a new credential - PublicKeyCredential's
6233 [[Create]](options) method
6234 * 5.1.4.1. PublicKeyCredential's
6235 [[DiscoverFromExternalSource]](options) method
6236 * 6.1. Authenticator data
6237 * 6.2.1. The authenticatorMakeCredential operation (2)
6238 * 6.2.2. The authenticatorGetAssertion operation (2)
6239 * 9. WebAuthn Extensions (2)
624C * 9.2. Defining extensions
6241 * 9.3. Extending request parameters
6242 * 9.4. Client extension processing (2)
6243 * 9.5. Authenticator extension processing (2)
6244 * 9.6. Example Extension
6245 * 10.5. Supported Extensions Extension (exts) (2)
6246 * 10.7. Location Extension (loc)
6247 * 11.2. WebAuthn Extension Identifier Registrations
6248
6249 #client-extension-inputReferenced in:
625C * 9. WebAuthn Extensions (2) (3)
6251 * 9.2. Defining extensions
6252 * 9.3. Extending request parameters (2) (3) (4) (5) (6)
6253 * 9.4. Client extension processing (2) (3) (4)
6254 * 9.6. Example Extension
6255
6256 #authenticator-extension-inputReferenced in:
6257 * 6.2.1. The authenticatorMakeCredential operation
6258 * 6.2.2. The authenticatorGetAssertion operation
6259 * 9. WebAuthn Extensions (2) (3) (4) (5)
626C * 9.2. Defining extensions
6261 * 9.3. Extending request parameters (2) (3)
6262 * 9.4. Client extension processing
6263 * 9.5. Authenticator extension processing (2) (3)
6264
6265 #client-extension-processingReferenced in:
6266 * 5.1. PublicKeyCredential Interface
6267 * 5.1.3. Create a new credential - PublicKeyCredential's

6281 * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
6282 * 9.6. Example Extension
6283 * 10.1. FIDO Appid Extension (appid)
6284 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
6285 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
6286 * 10.6. User Verification Index Extension (uvi)
6287 * 10.7. Location Extension (loc)
6288 * 10.8. User Verification Method Extension (uvm)
6289 * 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
629C (6)
6291
6292 #client-extensionReferenced in:
6293 * 5.1.3. Create a new credential - PublicKeyCredential's
6294 [[Create]](options) method
6295 * 5.1.4.1. PublicKeyCredential's
6296 [[DiscoverFromExternalSource]](options) method
6297 * 5.6. Authentication Extensions (typedef AuthenticationExtensions)
6298 * 9. WebAuthn Extensions
6299 * 9.2. Defining extensions
630C * 9.4. Client extension processing
6301
6302 #authenticator-extensionReferenced in:
6303 * 5.1.3. Create a new credential - PublicKeyCredential's
6304 [[Create]](options) method
6305 * 5.1.4.1. PublicKeyCredential's
6306 [[DiscoverFromExternalSource]](options) method
6307 * 5.6. Authentication Extensions (typedef AuthenticationExtensions)
6308 * 9. WebAuthn Extensions (2) (3)
6309 * 9.2. Defining extensions (2)
631C * 9.3. Extending request parameters
6311 * 9.5. Authenticator extension processing
6312
6313 #extension-identifierReferenced in:
6314 * 5.1. PublicKeyCredential Interface
6315 * 5.1.3. Create a new credential - PublicKeyCredential's
6316 [[Create]](options) method (2)
6317 * 5.1.4.1. PublicKeyCredential's
6318 [[DiscoverFromExternalSource]](options) method
6319 * 6.1. Authenticator data
632C * 6.2.1. The authenticatorMakeCredential operation (2)
6321 * 6.2.2. The authenticatorGetAssertion operation (2)
6322 * 9. WebAuthn Extensions (2)
6323 * 9.2. Defining extensions
6324 * 9.3. Extending request parameters
6325 * 9.4. Client extension processing (2)
6326 * 9.5. Authenticator extension processing (2)
6327 * 9.6. Example Extension
6328 * 10.5. Supported Extensions Extension (exts) (2)
6329 * 10.7. Location Extension (loc)
633C * 11.2. WebAuthn Extension Identifier Registrations
6331
6332 #client-extension-inputReferenced in:
6333 * 9. WebAuthn Extensions (2) (3)
6334 * 9.2. Defining extensions
6335 * 9.3. Extending request parameters (2) (3) (4) (5) (6)
6336 * 9.4. Client extension processing (2) (3) (4)
6337 * 9.6. Example Extension
6338
6339 #authenticator-extension-inputReferenced in:
634C * 6.2.1. The authenticatorMakeCredential operation
6341 * 6.2.2. The authenticatorGetAssertion operation
6342 * 9. WebAuthn Extensions (2) (3) (4) (5)
6343 * 9.2. Defining extensions
6344 * 9.3. Extending request parameters (2) (3)
6345 * 9.4. Client extension processing
6346 * 9.5. Authenticator extension processing (2) (3)
6347
6348 #client-extension-processingReferenced in:
6349 * 5.1. PublicKeyCredential Interface
635C * 5.1.3. Create a new credential - PublicKeyCredential's

```

626E [[Create]](options) method (2)
626F * 5.1.4.1. PublicKeyCredential's
6270 [[DiscoverFromExternalSource]](options) method (2)
6271 * 9. WebAuthn Extensions (2) (3) (4)
6272 * 9.2. Defining extensions
6273
6274 #client-extension-outputReferenced in:
6275 * 5.1. PublicKeyCredential Interface
6276 * 5.1.3. Create a new credential - PublicKeyCredential's
6277 [[Create]](options) method (2)
6278 * 5.1.4.1. PublicKeyCredential's
6279 [[DiscoverFromExternalSource]](options) method (2)
6280 * 9. WebAuthn Extensions (2) (3)
6281 * 9.2. Defining extensions (2) (3)
6282 * 9.4. Client extension processing (2) (3)
6283 * 9.6. Example Extension
6284
6285 #authenticator-extension-processingReferenced in:
6286 * 6.2.1. The authenticatorMakeCredential operation
6287 * 6.2.2. The authenticatorGetAssertion operation
6288 * 9. WebAuthn Extensions
6289 * 9.2. Defining extensions
6290 * 9.5. Authenticator extension processing
6291
6292 #authenticator-extension-outputReferenced in:
6293 * 6.1. Authenticator data
6294 * 9. WebAuthn Extensions (2) (3)
6295 * 9.2. Defining extensions (2) (3)
6296 * 9.4. Client extension processing
6297 * 9.5. Authenticator extension processing
6298 * 9.6. Example Extension
6299 * 10.5. Supported Extensions Extension (exts)
6300 * 10.6. User Verification Index Extension (uvi)
6301 * 10.7. Location Extension (loc)
6302 * 10.8. User Verification Method Extension (uvm)
6303
6304 #typedefdef-authenticatorselectionlistReferenced in:
6305 * 10.4. Authenticator Selection Extension (authnSel)
6306
6307 #typedefdef-aaguidReferenced in:
6308 * 10.4. Authenticator Selection Extension (authnSel)
6309

```

```

6351 [[Create]](options) method (2)
6352 * 5.1.4.1. PublicKeyCredential's
6353 [[DiscoverFromExternalSource]](options) method (2)
6354 * 9. WebAuthn Extensions (2) (3) (4)
6355 * 9.2. Defining extensions
6356
6357 #client-extension-outputReferenced in:
6358 * 5.1. PublicKeyCredential Interface
6359 * 5.1.3. Create a new credential - PublicKeyCredential's
6360 [[Create]](options) method (2) (3)
6361 * 5.1.4.1. PublicKeyCredential's
6362 [[DiscoverFromExternalSource]](options) method (2)
6363 * 9. WebAuthn Extensions (2) (3)
6364 * 9.2. Defining extensions (2) (3)
6365 * 9.4. Client extension processing (2) (3)
6366 * 9.6. Example Extension
6367
6368 #authenticator-extension-processingReferenced in:
6369 * 6.2.1. The authenticatorMakeCredential operation
6370 * 6.2.2. The authenticatorGetAssertion operation
6371 * 9. WebAuthn Extensions
6372 * 9.2. Defining extensions
6373 * 9.5. Authenticator extension processing
6374
6375 #authenticator-extension-outputReferenced in:
6376 * 6.1. Authenticator data
6377 * 9. WebAuthn Extensions (2) (3)
6378 * 9.2. Defining extensions (2) (3)
6379 * 9.4. Client extension processing
6380 * 9.5. Authenticator extension processing
6381 * 9.6. Example Extension
6382 * 10.5. Supported Extensions Extension (exts)
6383 * 10.6. User Verification Index Extension (uvi)
6384 * 10.7. Location Extension (loc)
6385 * 10.8. User Verification Method Extension (uvm)
6386
6387 #typedefdef-authenticatorselectionlistReferenced in:
6388 * 10.4. Authenticator Selection Extension (authnSel)
6389
6390 #typedefdef-aaguidReferenced in:
6391 * 10.4. Authenticator Selection Extension (authnSel)
6392

```